Software Architecture Description For YOLO

Version 1.1

Group 8

Aral Yekta Yarımca, 2376093

Abdurrahman Said Gürbüz, 2380483

Table of Contents

List of Figures

List of Tables

Revision History

Version	Date	Explanation
1.0	22.05.2022	The introduction, references, glossary and the context diagram with its explanation are written.
1.1	05.06.2022	The whole SAD document is finished.

Table 1: Revision History

1. Introduction

1.1 Purpose and objectives of the YOLO social robot

The aim of the YOLO system is to stimulate the creativity of children while they are playing games. Generally, children tend to play games consisting of storytelling and developing new ideas. Hence, YOLO is designed to act like a realistic character to these stories which can behave in different ways in the story to stimulate the children's creativity.

1.2 Scope

- The system shall have a user interface which the users (children) use to interact with the system. This interface is mainly composed of visual and sensory data transmission. The users touch and/or move the robot to interact with it.
- The system shall have a software interface which uses the current state of the robot, the current state of the story being told and the input of the user (child) to calculate the next behavior of the robot.
- The system shall have a programmer interface which the programmers and developers use to write new programs on a computer and define new behaviors. These programs are then needed to be uploaded to the robot. This way, new functionalities and new playstyles can be added to the robot.
- The system shall have a communication interface with the computer which is used to send and receive data. This data can be a newly written program to be uploaded to the robot and/or the recorded movement data to be processed by the machine learning model.
- The system shall have a data process interface implemented by a machine learning model. This interface gets the recorded movement data and then maps it to a movement pattern.

1.3 Stakeholders and their concerns

There are four main stakeholders of the system: the users (children), parents and educators, programmers and researchers.

The users are the children who play with the robot. They are expected to not have any physical disabilities as they need to move around with the robot. They are also expected to be in an age where their brain development supports storytelling and understanding social cues (such as recognizing contrasting and mirroring behaviors). Their concern is small-scale and light-weight design of the robot which won't scare them and will fit the size of their hands.

The parents and the educators prepare the robot for the children to play. They need to first upload the default software to the robot and then turn the robot on. They also need to be sure that the robot is connected to the wi-fi. Therefore, they are expected to have a basic knowledge and sufficiency in technology, computers and network connections. Their concern is the ease of assembling the robot's components.

The programmers write new programs and define new behaviors for the robot. They write their programs on an external computer and then upload them to the Raspberry Pi on the robot using the wi-fi connection between them. They are expected to have knowledge of Raspberry Pi and a scripting language called Python. Their concern is an API that enables them to generate new behaviors.

The researchers use the robot to observe children's play with it. Therefore, they should be able to operate the robot, prepare it for their own experiment setups, and possibly write new programs for it. Hence, just like the parents and the educators, they are expected to have a basic knowledge and sufficiency in technology, computers and network connections, and possibly like the programmers, Raspberry Pi and Python. Their concern is a robot that is cheaper than off-the-shelf robots and does not require special transportation services to be used.

2. References

This document is written according to the specifications written in the document below:

"ISO/IEC/IEEE Systems and software engineering -- Architecture description," in *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)* , vol., no., pp.1-46, 1 Dec. 2011, doi: 10.1109/IEEESTD.2011.6129467.

Other sources:

Alves-Oliveira, Patrícia & Arriaga, Patricia & Paiva, Ana & Hoffman, Guy. (2021). Children as Robot Designers. 399-408. 10.1145/3434073.3444650.

Alves-Oliveira, Patrícia & Arriaga, Patricia & Paiva, Ana & Hoffman, Guy. (2019). Guide to build YOLO, a creativity-stimulating robot for children. HardwareX. 6. e00074. 10.1016/j.ohx.2019.e00074.

Alves-Oliveira, Patrícia & Gomes, Samuel & Chandak, Ankita & Arriaga, Patricia & Hoffman, Guy & Paiva, Ana. (2020). Software architecture for YOLO, a creativity-stimulating robot. 11. 100461. 10.1016/j.softx.2020.100461.

3. Glossary

Term	Definition
Wi-fi	Wireless internet connection
YOLO	The name of the system to be developed, it stands for “Your Own Living Object”
Python	A scripting language which can be used to develop computer programs
LED	Light emitting diode, it is used to emit different colors of light
KNN Algorithm	A machine learning algorithm called “K-Nearest Neighbor”
Raspberry Pi	A small single board computer used as the main computation source
Mbps	A unit to measure connection speed which stands for “megabits per second”
GB	A unit to measure computer memory which stands for “gigabyte”
API	Application Programming Interface
TCP/IP	Transmission Control Protocol/Internet Protocol

Table 2: Glossary

4. Architectural Views

4.1 Context View

4.1.1 Stakeholders’ use of this view

There are four main stakeholders of the system: the users (children), parents and educators, programmers and researchers.

The users (children) use this view to understand the general structure of their interaction with the robot. Parents and educators use it to supervise their children’s interaction with the robot. Programmers use it to plan the new behavior of the robot that they will define. Researchers use it to understand the overview of the interaction between children and the robot to plan their studies and experiments.

4.1.2 Context Diagram

YOLO is not a part of a larger system. However, it interacts with 2 external entities. The child and the programmer. The child plays with YOLO, providing physical contact and movement pattern data to it. And YOLO provides the child LED lights and (movement) imitation. The programmer develops new software for YOLO.

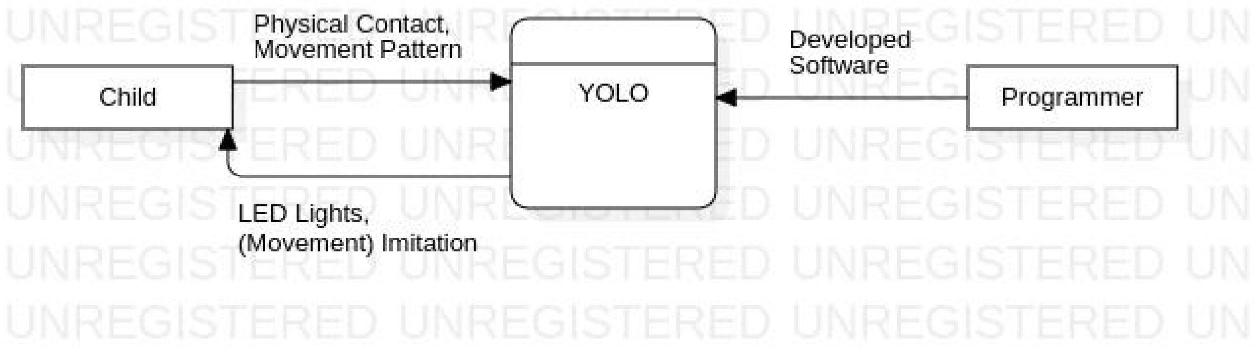


Figure 1: Context Diagram for YOLO

4.1.3 External Interfaces

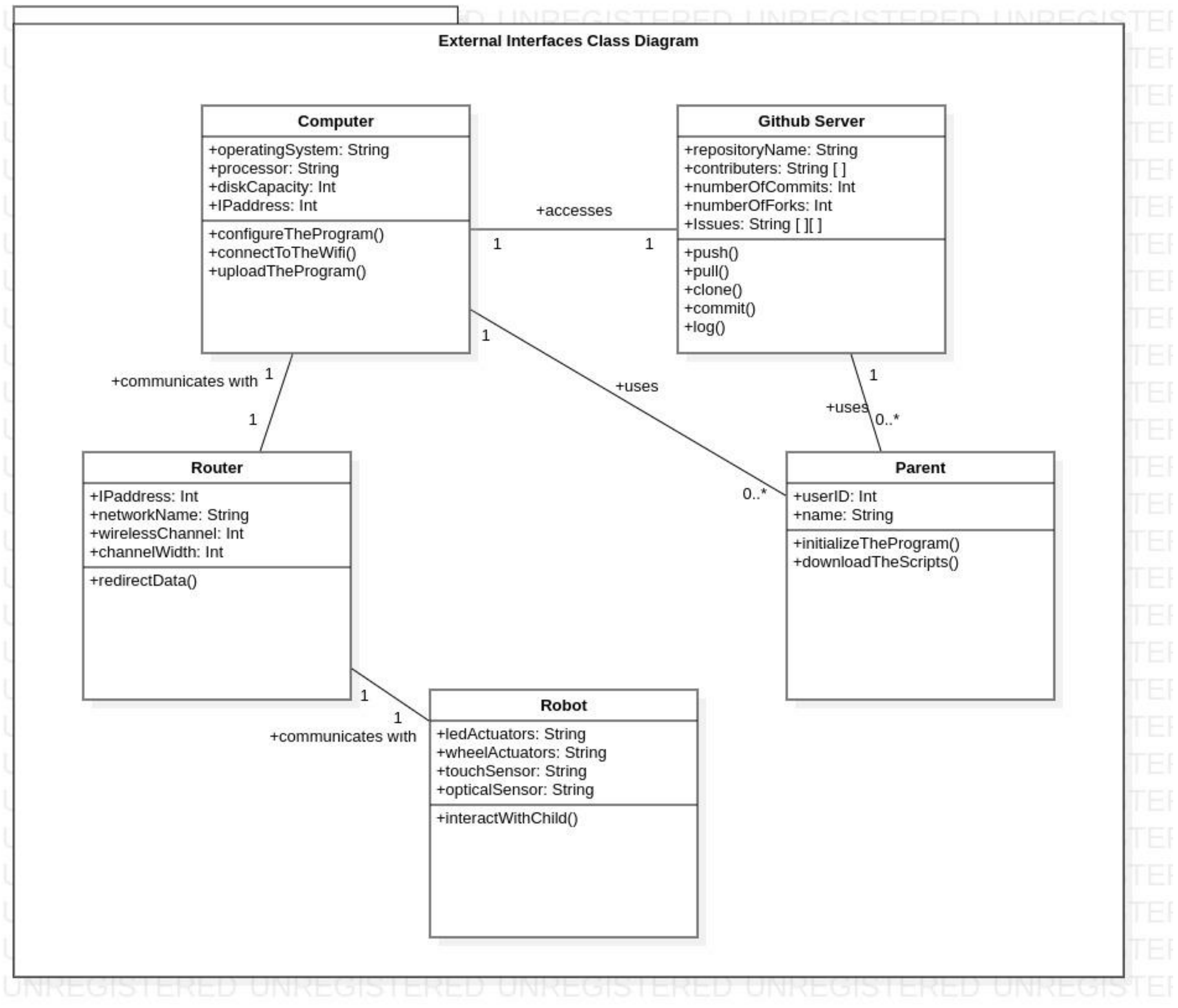


Figure 2: External Interfaces Class Diagram for YOLO

Operation	Description
configureTheProgram	Modify the robot software to make further configurations or to redefine behaviors
connectToTheWifi	The computer connects to the router through wifiç
uploadTheProgram	The computer uploads the program to the robot.
push	Push the code changes from local to the Github server.
pull	Pull the code changes from the Github server to local.
clone	Download the code from the Github server to local.

commit	Turn the new code changes to a commit to be further pushed onto the server.
log	Log the commits done to the repository.
redirectData	Redirect the data coming from one end of the connection to the other end.
initializeTheProgram	Initialize and set up the robot software onto the computer of the robot.
downloadTheScripts	Download the robot software from the web.
interactWithChild	Interact with the child, react to their movement according to the social mode and the story arc.

Table 3: External Interface Operation Descriptions

4.1.4 Interaction Scenarios

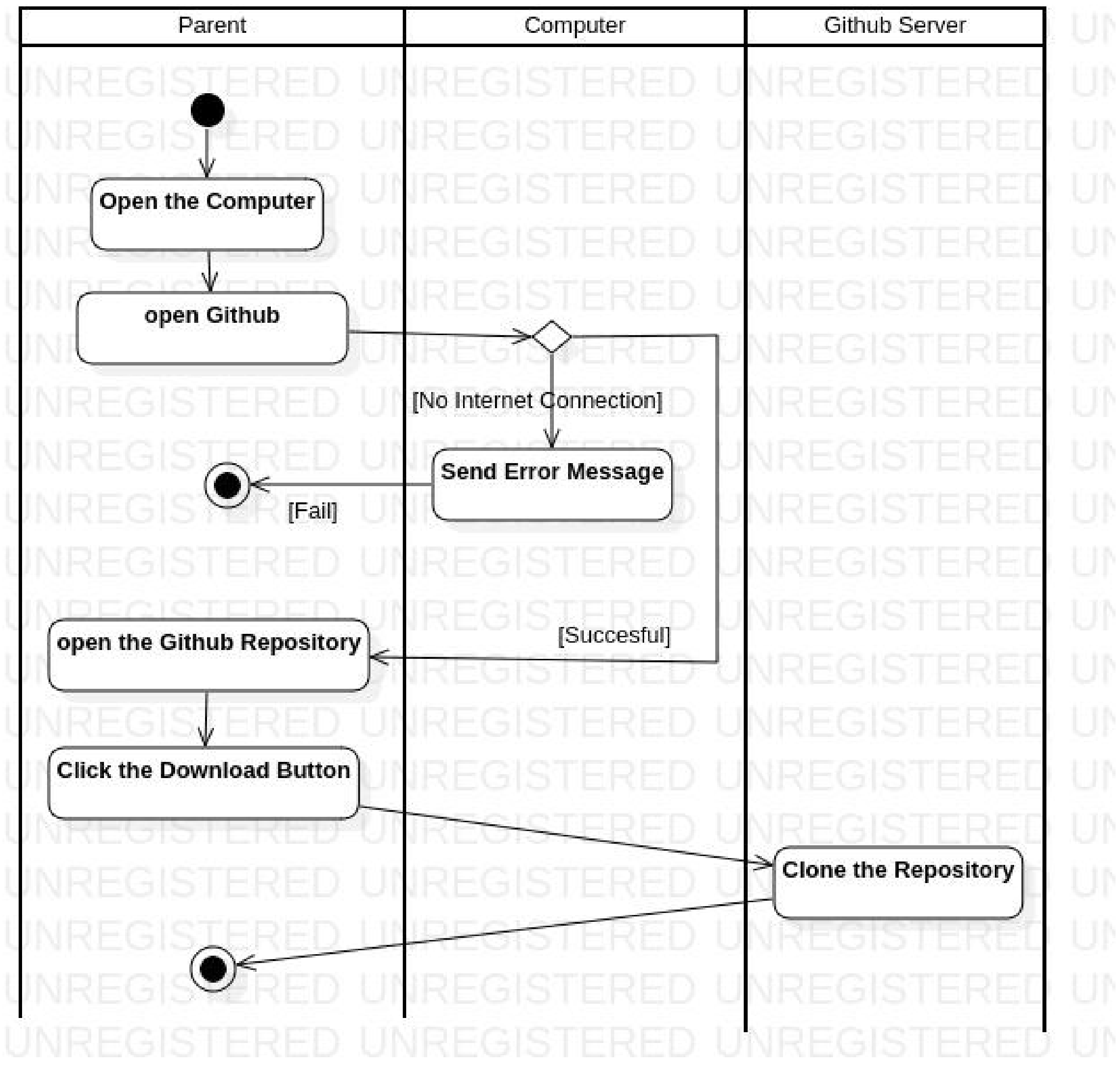


Figure 3: Activity Diagram for Interaction Between the Parent and Github Server External Interfaces

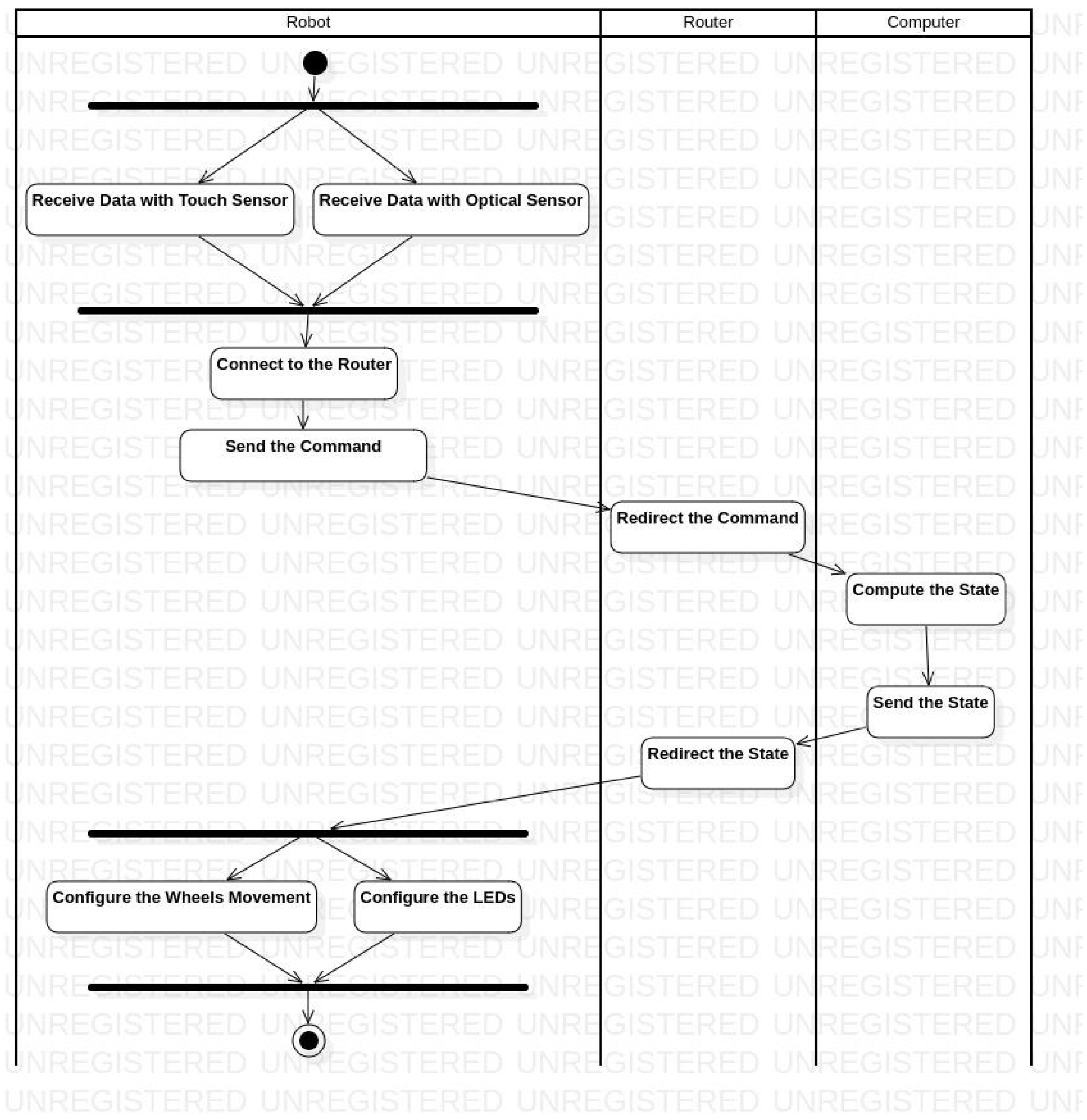


Figure 4: Activity Diagram for Interaction Between the Robot and the Computer External Interfaces

4.2 Functional View

4.2.1 Stakeholders' use of this view

There are four main stakeholders of the system: the users (children), parents, and educators, programmers, and researchers.

The users (children) use the context view to understand the limits and the capabilities of the robot and how to operate it. Parents and educators use it to be able to prepare the robot for the children to play. Programmers use it to understand how to define new behaviors for the robot and reprogram it. Researchers use it to understand the capabilities and the characteristics of the robot and evaluate how they can use it in their research.

4.2.2 Component Diagram

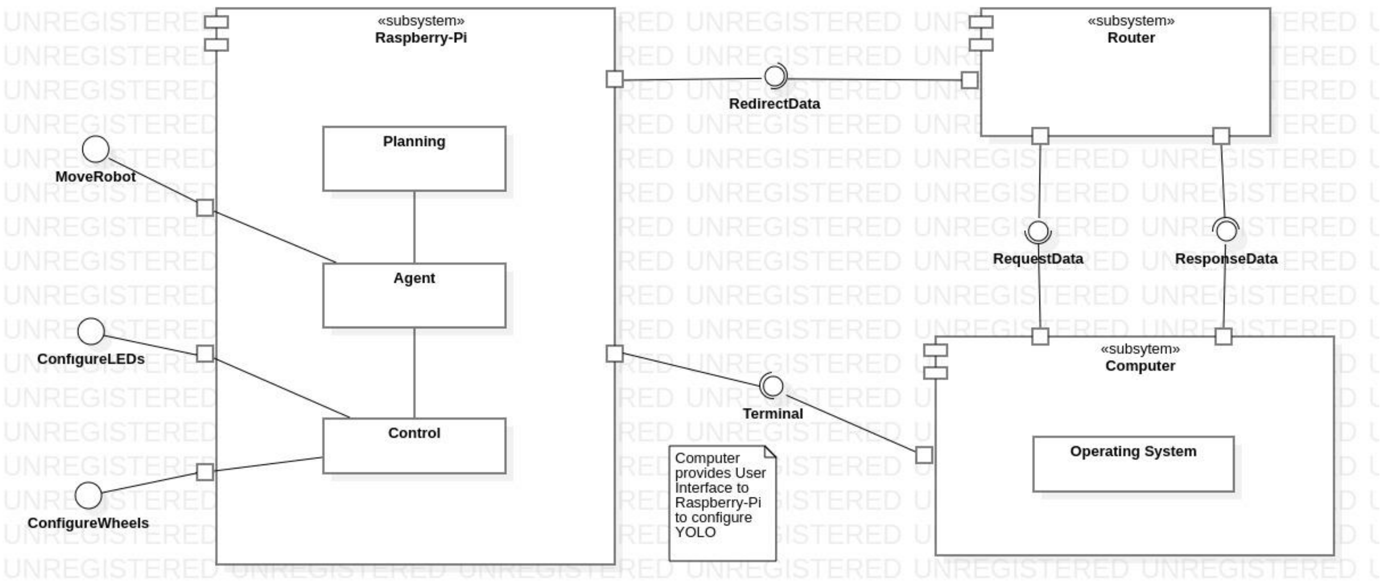


Figure 5: Component Diagram for YOLO

- There are three subsystems in the YOLO, Raspberry-Pi, Router, and Computer.
- Raspberry-Pi consists of three parts, Planning, Agent, and Control.
- Agent provides an interface to the child, namely MoveRobot. It is responsible for the movement of the robot according to the interaction of the child.
- Control provides interfaces to the child, namely ConfigureLEDs, and ConfigureWheels. It provides configuring LEDs and Wheels according to the current behavior and the current story arc of the robot.
- Router requires an interface to redirect the data to the computer and Raspberry Pi provides an interface to redirect the data. Therefore, there is an assembly interface between the router and Raspberry Pi named RedirectData.
- The computer has a part named operating system which is an interface between the application program and the hardware.
- The computer requires an interface to request the data to process over it. And the router provides an interface to redirect the data to the computer. Therefore, there is an assembly interface between the computer and the router named RequestData.
- The router requires an interface to redirect the data to Raspberry Pi. And the computer provides an interface to give a response to the data that it requests. Therefore, there is an assembly interface between the router and the computer named ResponseData.
- There is an assembly interface between Raspberry Pi and the computer which is Terminal. Raspberry Pi requires and user interface to configure the program, and the computer provides graphical user interfaces via terminal. The programmer can configure the program and send it to Raspberry Pi using the terminal.

4.2.3 Internal Interfaces

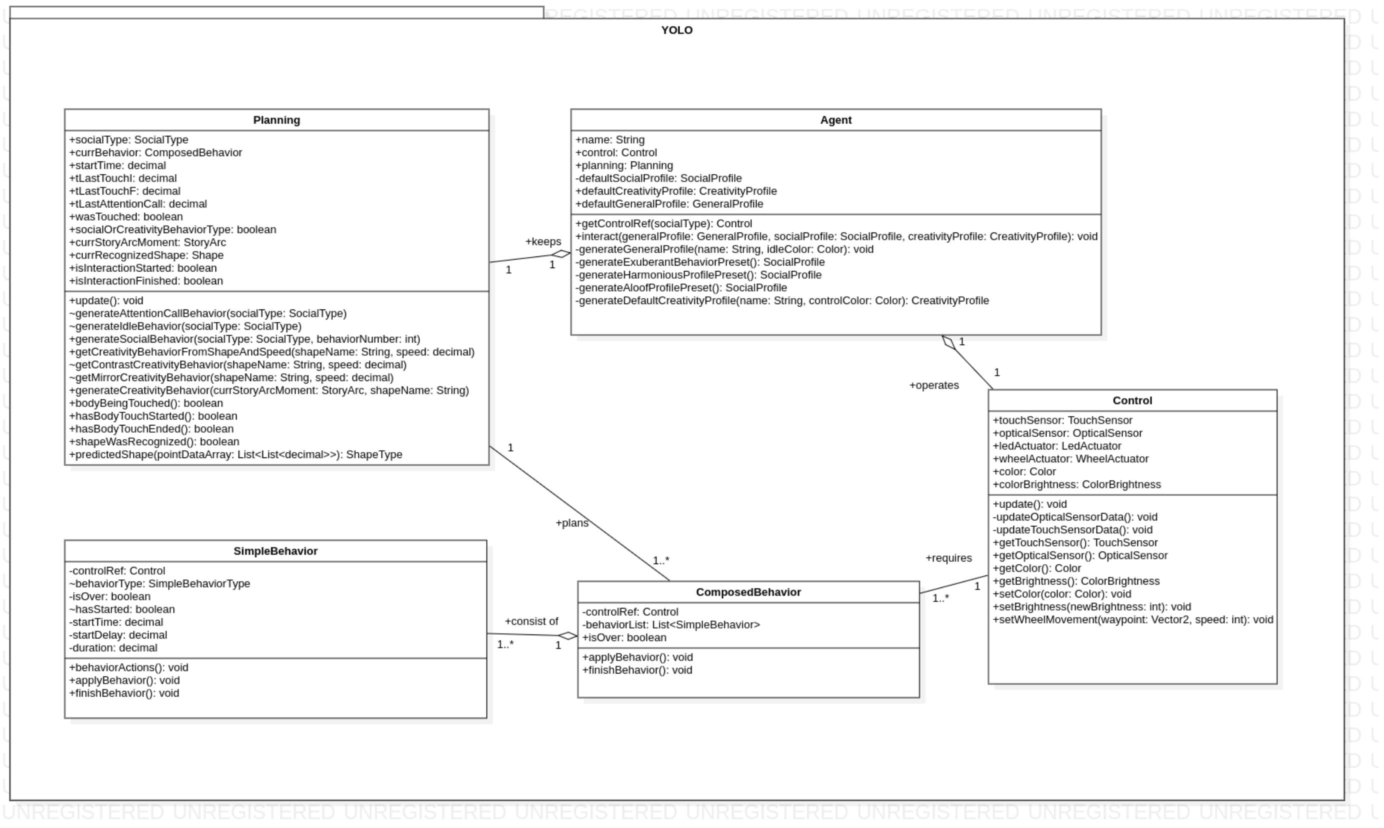

YOLO
Planning
+socialType: SocialType
+currBehavior: ComposedBehavior
+startTime: decimal
+tLastTouchI: decimal
+tLastTouchF: decimal
+tLastAttentionCall: decimal
+wasTouched: boolean
+socialOrCreativityBehaviorType: boolean
+currStoryArcMoment: StoryArc
+currRecognizedShape: Shape
+isInteractionStarted: boolean
+isInteractionFinished: boolean
+update(): void
~generateAttentionCallBehavior(socialType: SocialType)
~generateIdleBehavior(socialType: SocialType)
+generateSocialBehavior(socialType: SocialType, behaviorNumber: int)
+getCreativityBehaviorFromShapeAndSpeed(shapeName: String, speed: decimal)
~getContrastCreativityBehavior(shapeName: String, speed: decimal)
~getMirrorCreativityBehavior(shapeName: String, speed: decimal)
+generateCreativityBehavior(currStoryArcMoment: StoryArc, shapeName: String)
+bodyBeingTouched(): boolean
+hasBodyTouchStarted(): boolean
+hasBodyTouchEnded(): boolean
+shapeWasRecognized(): boolean
+predictedShape(pointDataArray: List<List<decimal>>): ShapeType
+keeps
1
1
Agent
+name: String
+control: Control
+planning: Planning
-defaultSocialProfile: SocialProfile
+defaultCreativityProfile: CreativityProfile
+defaultGeneralProfile: GeneralProfile
+getControlRef(socialType): Control
+interact(generalProfile: GeneralProfile, socialProfile: SocialProfile, creativityProfile: CreativityProfile): void
-generateGeneralProfile(name: String, idleColor: Color): void
-generateExuberantBehaviorPreset(): SocialProfile
-generateHarmoniousProfilePreset(): SocialProfile
-generateAloofProfilePreset(): SocialProfile
-generateDefaultCreativityProfile(name: String, controlColor: Color): CreativityProfile
1
1
+operates
Control
+touchSensor: TouchSensor
+opticalSensor: OpticalSensor
+ledActuator: LedActuator
+wheelActuator: WheelActuator
+color: Color
+colorBrightness: ColorBrightness
+update(): void
-updateOpticalSensorData(): void
-updateTouchSensorData(): void
+getTouchSensor(): TouchSensor
+getOpticalSensor(): OpticalSensor
+getColor(): Color
+getBrightness(): ColorBrightness
+setColor(color: Color): void
+setBrightness(newBrightness: int): void
+setWheelMovement(waypoint: Vector2, speed: int): void
1
+plans
1..*
+requires
1
1..*
SimpleBehavior
-controlRef: Control
~behaviorType: SimpleBehaviorType
-isOver: boolean
~hasStarted: boolean
-startTime: decimal
-startDelay: decimal
-duration: decimal
+behaviorActions(): void
+applyBehavior(): void
+finishBehavior(): void
+consist of
1..*
1
ComposedBehavior
-controlRef: Control
-behaviorList: List<SimpleBehavior>
+isOver: boolean
+applyBehavior(): void
+finishBehavior(): void

Figure 6: Internal Interface Class Diagram for YOLO

Operation	Description
Planning::update	Updates the current composed behavior depending on the generated behavior.
generateAttentionCallBehavior	The robot plans the behavior to call for attention.
generateIdleBehavior	The robot plans the introverted idle behavior.
generateSocialBehavior	The robot plans the extroverted social behavior.
getCreativityBehaviorFromShapeAndSpeed	The robot plans its next action from the movement data provided by the child.
getContrastCreativityBehavior	The robot plans its action contrasting with the child.
getMirrorCreativityBehavior	The robot plans its action mirroring the child.
generateCreativityBehavior	The robot decides on a contrasting or a mirroring action depending on the story arc.
bodyBeingTouched	The robot checks whether it is being touched or not.
hasBoudyTouchStarted	If the robot is being touched, it checks whether this is the initial physical contact.

hasBodyTouchEnded	If the robot is being touched, it checks whether this is the final physical contact.
shapeWasRecognized	The robot checks whether it recognized the movement shape that the child provided or not.
predictedShape	The robot recognizes the movement shape that the child provides.
getControlRef	Creates the member variable named “control”.
interact	Depending on the general, social and creativity profiles of the robot, it interacts with the child.
generateGeneralProfile	Creates a general profile for the robot.
generateExuberantBehaviorPreset	Creates an exuberant social profile for the robot.
generateHarmoniousProfilePreset	Creates a harmonious social profile for the robot.
generateAloofProfilePreset	Creates an aloof social profile for the robot.
generateDefaultCreativityProfile	Creates a default creativity profile for the robot.
Control::update	Updates the data read by all of the sensors.
updateOpticalSensorData	Updates the data read by the optical sensor.
updateTouchSensorData	Updates the data read by the touch sensor.
getTouchSensor	Gets the touch sensor class object.
getOpticalSensor	Gets the optical sensor class object.
getColor	Get the current color that is being displayed.
getBrightness	Get the brightness level that is being displayed.
setColor	Set the color that is being displayed.
setBrightness	Set the brightness that is being displayed.
setWheelMovement	Set the motor actuators and thus the wheel movement.
behaviorActions	Keep executing the behavior until it is finished
SimpleBehavior::applyBehavior	Start to execute the simple behavior.
SimpleBehavior::finishBehavior	Finish executing the simple behavior.
ComposedBehavior::applyBehavior	Start to execute the composed behavior.
ComposedBehavior::finishBehavior	Finish executing the composed behavior.

Table 4: Internal Interface Operation Descriptions

4.2.4 Interaction Patterns

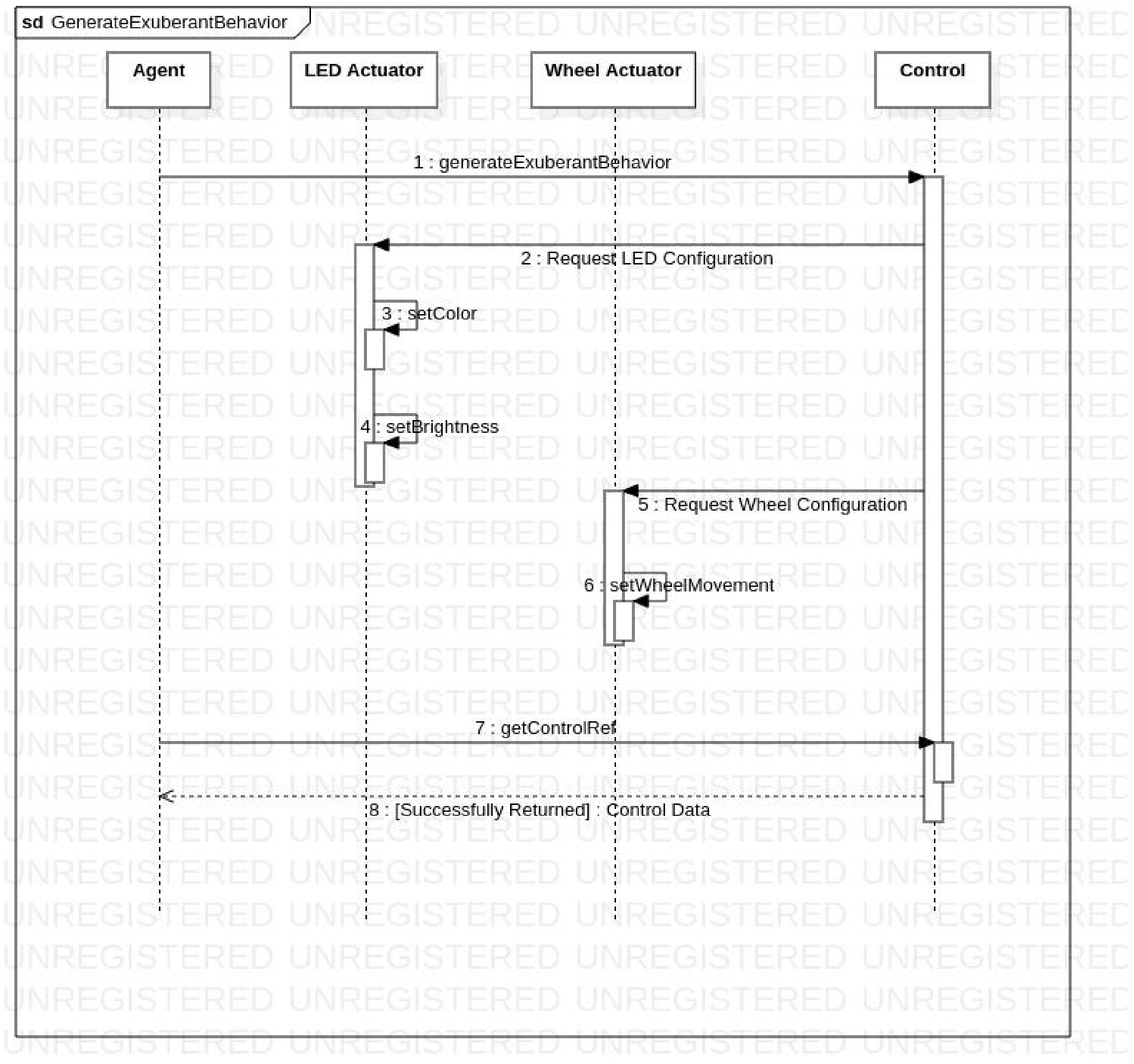

sd GenerateExuberantBehavior
Agent
LED Actuator
Wheel Actuator
Control
1 : generateExuberantBehavior
2 : Request LED Configuration
3 : setColor
4 : setBrightness
5 : Request Wheel Configuration
6 : setWheelMovement
7 : getControlRef
8 : [Successfully Returned] : Control Data

Figure 7: Sequence Diagram for Interaction Between the Agent and the Control Internal Interfaces

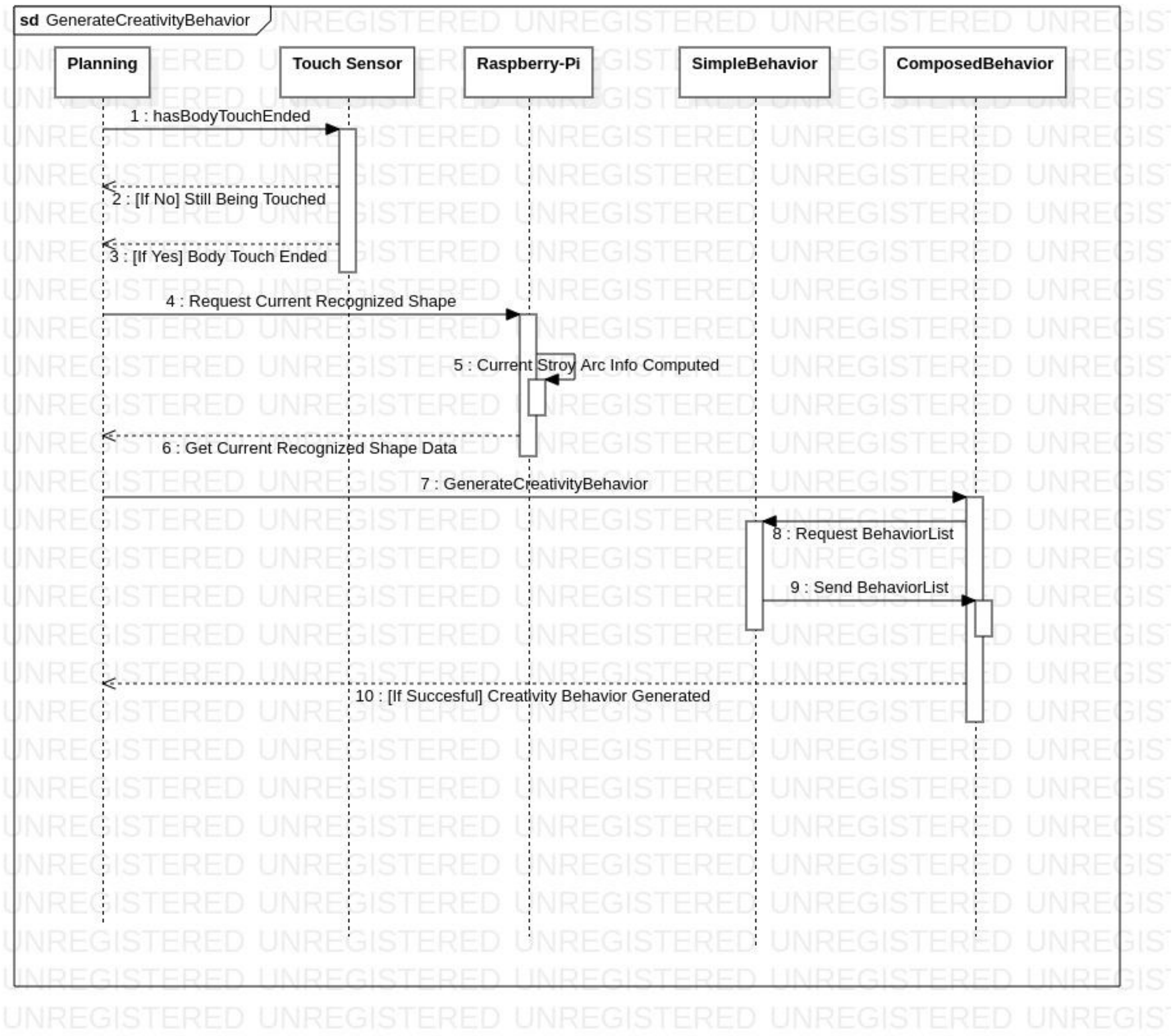

sd GenerateCreativityBehavior
Planning
Touch Sensor
Raspberry-Pi
SimpleBehavior
ComposedBehavior
1 : hasBodyTouchEnded
2 : [If No] Still Being Touched
3 : [If Yes] Body Touch Ended
4 : Request Current Recognized Shape
5 : Current Stroy Arc Info Computed
6 : Get Current Recognized Shape Data
7 : GenerateCreativityBehavior
8 : Request BehaviorList
9 : Send BehaviorList
10 : [If Succesful] Creativity Behavior Generated

Figure 8: Sequence Diagram for the Planning and the Composed Behavior Internal Interfaces

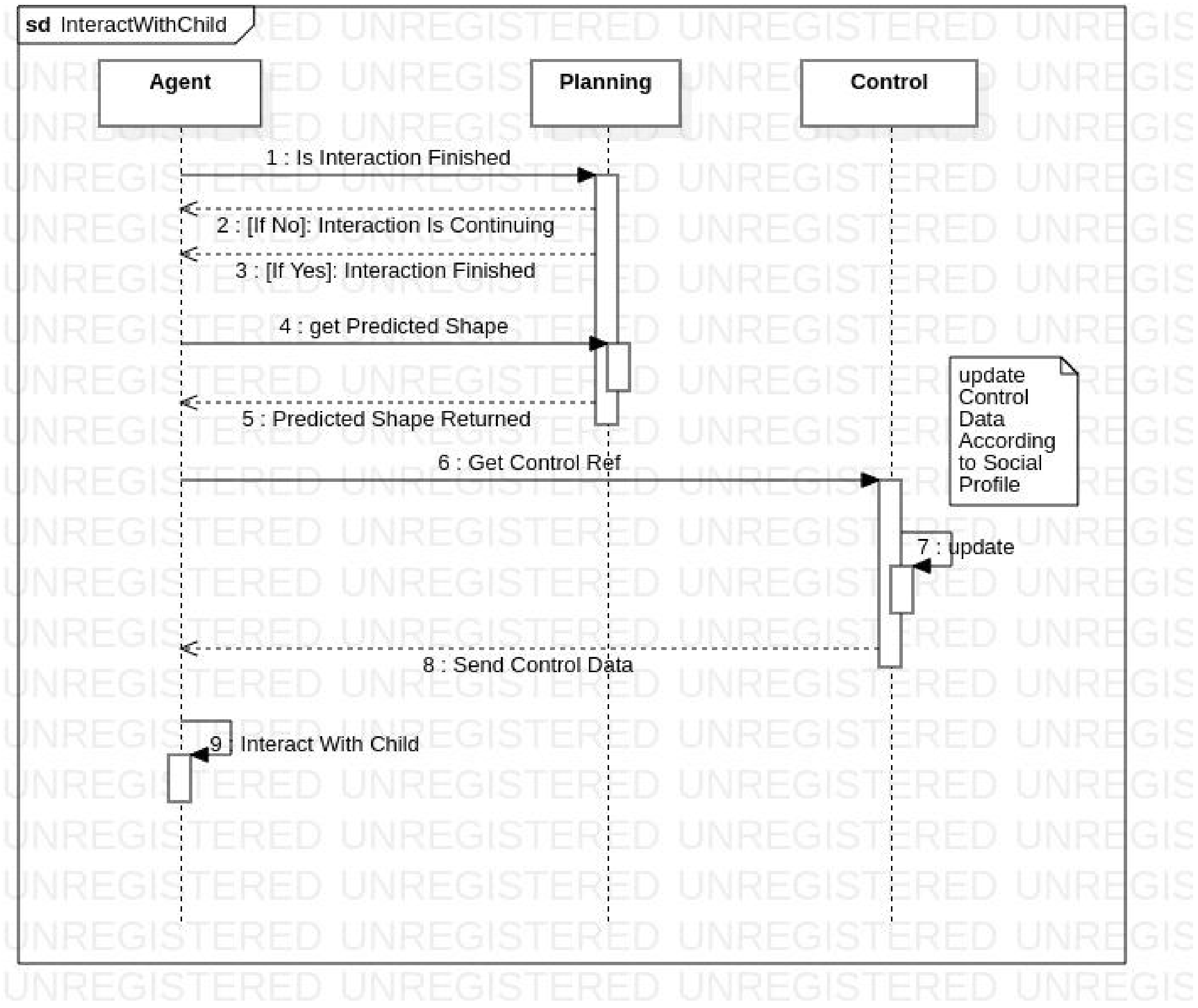

sd InteractWithChild
Agent
Planning
Control
1 : Is Interaction Finished
2 : [If No]: Interaction Is Continuing
3 : [If Yes]: Interaction Finished
4 : get Predicted Shape
5 : Predicted Shape Returned
6 : Get Control Ref
update Control Data According to Social Profile
7 : update
8 : Send Control Data
9 : Interact With Child
UNREGISTERED

Figure 9: Sequence Diagram for Interaction Between the Agent and the Planning Internal Interfaces

4.3 Information View

4.3.1 Stakeholders' use of this view

There are four main stakeholders of the system: the users (children), parents and educators, programmers and researchers.

The users (children) use this view to understand how the robot operates and tailor their behavior according to it while they are interacting with the robot. Parents and educators use this view to understand how the robot functions, what data it stores, and what dangers it could impose to their children. Programmers use this view to decide on their approach to programming new behaviors for the robot. They understand the limits and the functionalities of the robot to understand their boundaries when reprogramming the robot. Researchers use this view to decide on the experiments that they can do with the robot by understanding how the robot operates and what data it stores to ultimately decide on what they can measure with it.

4.3.2 Database Class Diagram

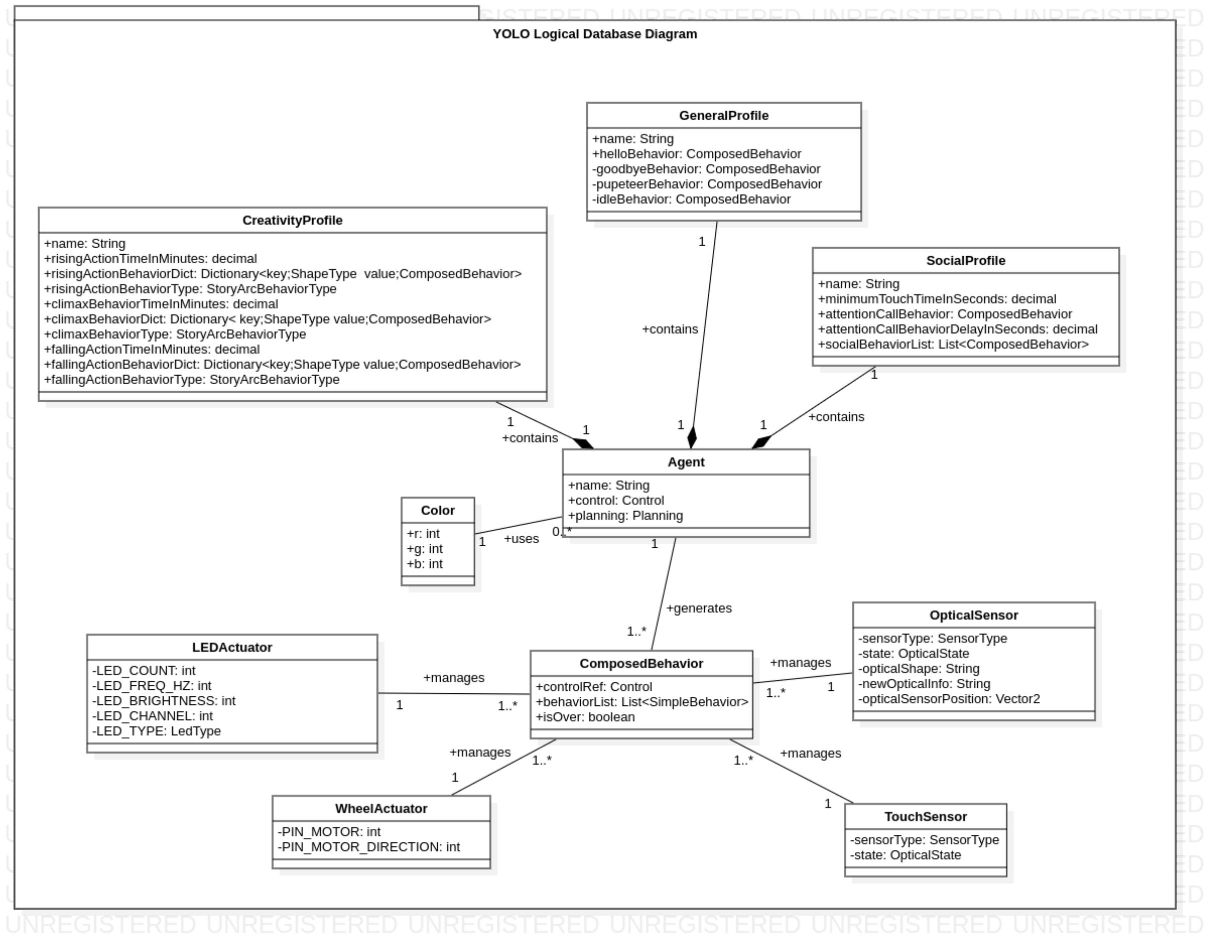

YOLO Logical Database Diagram
GeneralProfile
+name: String
+helloBehavior: ComposedBehavior
-goodbyeBehavior: ComposedBehavior
-pupeteerBehavior: ComposedBehavior
-idleBehavior: ComposedBehavior
CreativityProfile
+name: String
+risingActionTimeInMinutes: decimal
+risingActionBehaviorDict: Dictionary<key;ShapeType value;ComposedBehavior>
+risingActionBehaviorType: StoryArcBehaviorType
+climaxBehaviorTimeInMinutes: decimal
+climaxBehaviorDict: Dictionary< key;ShapeType value;ComposedBehavior>
+climaxBehaviorType: StoryArcBehaviorType
+fallingActionTimeInMinutes: decimal
+fallingActionBehaviorDict: Dictionary<key;ShapeType value;ComposedBehavior>
+fallingActionBehaviorType: StoryArcBehaviorType
SocialProfile
+name: String
+minimumTouchTimeInSeconds: decimal
+attentionCallBehavior: ComposedBehavior
+attentionCallBehaviorDelayInSeconds: decimal
+socialBehaviorList: List<ComposedBehavior>
1
+contains
1
1
+contains
1
1
+contains
1
1
Agent
+name: String
+control: Control
+planning: Planning
Color
+r: int
+g: int
+b: int
1
+uses
0..*
1
+generates
1..*
OpticalSensor
-sensorType: SensorType
-state: OpticalState
-opticalShape: String
-newOpticalInfo: String
-opticalSensorPosition: Vector2
LEDActuator
-LED_COUNT: int
-LED_FREQ_HZ: int
-LED_BRIGHTNESS: int
-LED_CHANNEL: int
-LED_TYPE: LedType
+manages
1
1..*
ComposedBehavior
+controlRef: Control
+behaviorList: List<SimpleBehavior>
+isOver: boolean
+manages
1..*
1
+manages
1..*
1
1..*
+manages
1
WheelActuator
-PIN_MOTOR: int
-PIN_MOTOR_DIRECTION: int
TouchSensor
-sensorType: SensorType
-state: OpticalState

Figure 10: Database Requirements Class Diagram

- When the robot starts working, the agent is created with a unique name. Also led actuator, optical sensor, wheel actuator, and touch sensor objects are created.
- CreativityProfile, generalProfile, and social profiles are weak entities, so they shall be created when an agent exists.
- Some agents can use the color objects, therefore, the color object shall be created for those cases.
- If an agent is not using the color object for generating creativity behavior, the color object shall not be created.
- Every agent contains exactly one creativityProfile, one generalProfile and one socialProfile.
- When the agent exists it shall generate one or more composed behaviors, thus one or more composed behaviors object shall be created.
- Every composed behavior generated by the agent shall manage the led actuator, optical sensor, wheel actuator, and touch sensor objects.
- Each led actuator, optical sensor, wheel actuator, and touch sensor can be managed by one or more composed behaviors.
- Each composed behavior shall contain one or more simple behavior.
- When the agent object is deleted, the creativity profile, general profile, and social profile are also deleted.
- The system shall have exactly one active agent object at any time.
- The information that the led actuator, optical sensor, wheel actuator, and touch sensor contain is only accessible and changeable by a composed behavior object that actively manages those objects.

- The information that creativityProfile, generalProfile, and SocialProfile contain is accessible outside the agent object since they are all stored in a public state.

4.3.3 Operations on Data

Operation	Description
Planning::update	Create: Read: Update: GeneralProfile, ComposedBehavior Delete:
generateAttentionCallBehavior	Create: ComposedBehavior Read: SocialProfile Update: GeneralProfile Delete:
generateIdleBehavior	Create: ComposedBehavior Read: SocialProfile Update: GeneralProfile Delete:
generateSocialBehavior	Create: ComposedBehavior Read: SocialProfile Update: GeneralProfile Delete:
getCreativityBehaviorFromShapeAndSpeed	Create: Read: OpticalSensor Update: CreativityProfile Delete:
getContrastCreativityBehavior	Create: ComposedBehavior Read: SocialProfile Update: CreativityProfile Delete:
getMirrorCreativityBehavior	Create: ComposedBehavior Read: SocialProfile Update: CreativityProfile Delete:
generateCreativityBehavior	Create: ComposedBehavior Read: SocialProfile Update: CreativityProfile Delete:
bodyBeingTouched	Create: Read: TouchSensor Update: Delete:
hasBoudyTouchStarted	Create: Read: TouchSensor Update:

	Delete:
hasBodyTouchEnded	Create: Read: TouchSensor Update: Delete:
shapeWasRecognized	Create: Read: CreativityProfile Update: Delete:
predictedShape	Create: Read: OpticalSensor Update: CreativityProfile Delete:
getControlRef	Create: Read: Update: Agent Delete:
interact	Create: Read: GeneralProfile, SocialProfile, CreativityProfile Update: LEDActuator, WheelActuator Delete:
generateGeneralProfile	Create: GeneralProfile Read: Update: Delete:
generateExuberantBehaviorPreset	Create: SocialProfile Read: Update: Delete:
generateHarmoniousProfilePreset	Create: SocialProfile Read: Update: Delete:
generateAloofProfilePreset	Create: SocialProfile Read: Update: Delete:
generateDefaultCreativityProfile	Create: CreativityProfile Read: Update: Delete:
Control::update	Create: Read: Update: OpticalSensor, TouchSensor

	Delete:
updateOpticalSensorData	Create: Read: Update: OpticalSensor Delete:
updateTouchSensorData	Create: Read: Update: TouchSensor Delete:
getTouchSensor	Create: Read: TouchSensor Update: Delete:
getOpticalSensor	Create: Read: OpticalSensor Update: Delete:
getColor	Create: Read: Color Update: Delete:
getBrightness	Create: Read: LEDActuator Update: Delete:
setColor	Create: Read: Update: Color Delete:
setBrightness	Create: Read: Update: LEDActuator Delete:
setWheelMovement	Create: Read: Update: WheelActuator Delete:
behaviorActions	Create: Read: ComposedBehavior Update: ComposedBehavior Delete:
SimpleBehavior::applyBehavior	Create: Read: Update: LEDActuator, WheelActuator

	Delete:
SimpleBehavior::finishBehavior	Create: Read: Update: LEDActuator, WheelActuator, ComposedBehavior Delete:
ComposedBehavior::applyBehavior	Create: Read: Update: LEDActuator, WheelActuator Delete:
ComposedBehavior::finishBehavior	Create: Read: Update: LEDActuator, WheelActuator, ComposedBehavior Delete:

Table 5: CRUD Operations

4.4 Deployment View

4.4.1 Stakeholders' use of this view

There are four main stakeholders of the system: the users (children), parents and educators, programmers and researchers.

Parents and educators use this view to understand how they can set the robot up and how to upload new programs to it. Programmers use this view to understand how the file structure of the code that they write for the robot should be. Researchers use this view to understand the hardware properties of the robot which may be important for their experiments.

4.4.2 Deployment Diagram

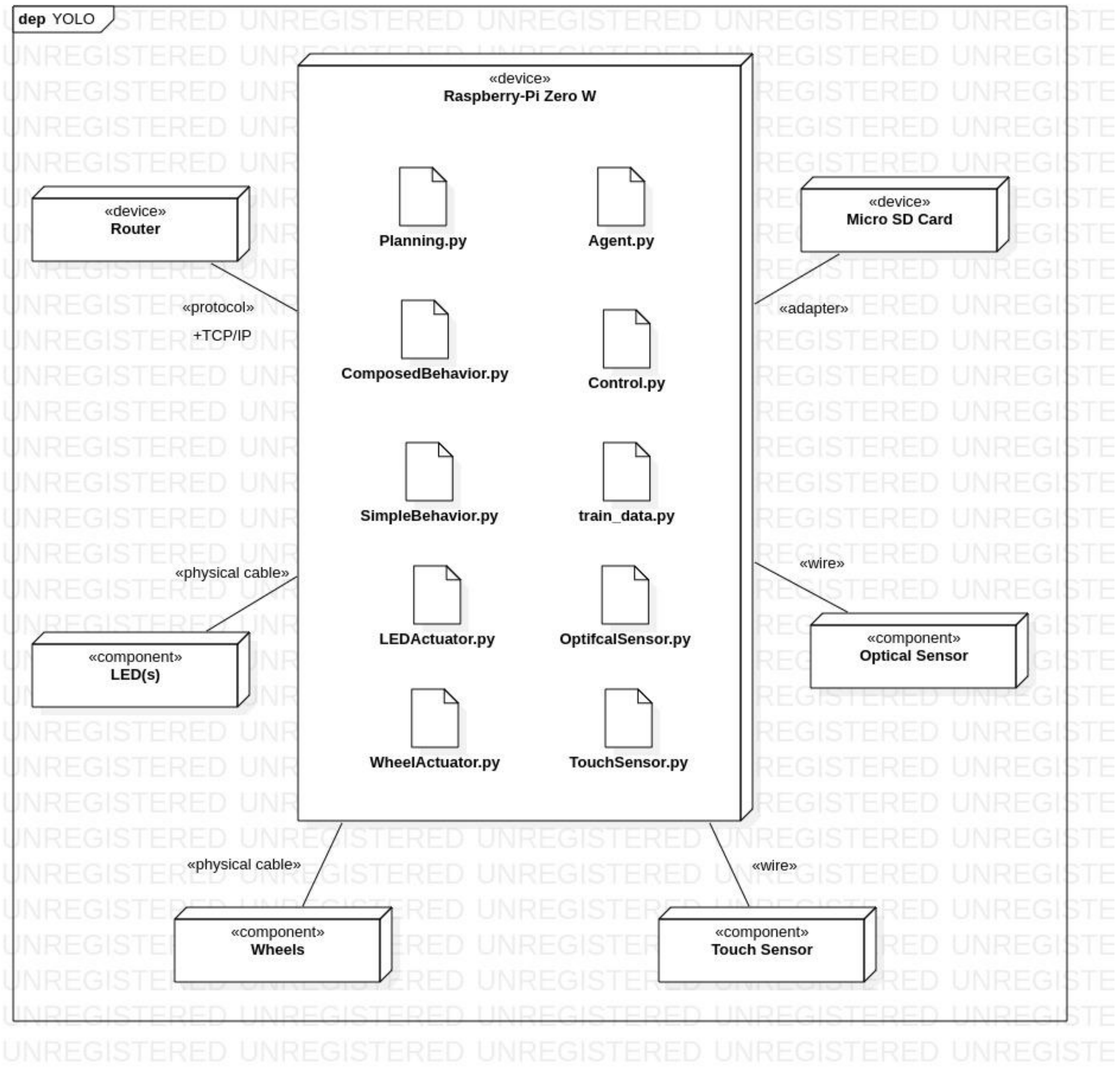


Figure 11: Deployment Diagram for YOLO

- All scripts and machine learning-related data reside on **Raspberry-Pi Zero W**.
- **Planning.py** coordinates the behaviors through the moments of the interaction.
- **Agent.py** is responsible for initializing the robot, given the social and creativity profiles. Agent encapsulates all the functionalities of our robot and commands it to act through the method interact.
- **SimpleBehavior.py** represents a behavior using directly the actuators of the robot.
- **ComposedBehavior.py** represents a behavior composed of several simple behaviors. When it is updated, started, and stopped all the simple behaviors are updated, started, and stopped accordingly. The behavior is over only when all simple behaviors are over.
- **Control.py** coordinates the actions related to the sensor and actuator classes (TouchSensor, OpticalSensor, LEDActuator, WheelActuator).
- **TouchSensor.py** allows the extraction of data from the touch sensor, positioned in the front of the robot.
- **OpticalSensor.py** allows the extraction of data from the optical sensor, positioned in the bottom of the robot.
- **LEDActuator.py** allows the direct manipulation of the robot’s LED lights.

- **WheelActuator.py** allows the direct manipulation of the robot's wheel motors. The considered movement space is a uniform grid.
- **Train_data.py** contains the training data required for machine learning algorithms.
- **Micro SD Card** that has 32GB disk capacity provides external storage for the robot and it is connected to the Raspberry Pi via the adapter.
- **Optical Sensor** detects the movement direction of the robot and it is connected to Raspberry Pi via the wire.
- **Touch Sensor** recognizes the physical contact of the child and it is connected to Raspberry Pi via the wire.
- **Wheels** provide navigation to the robot and they are connected to the Raspberry Pi via the physical cables.
- **LEDs** provide lights according to the current behavior of the robot and they are connected to the Raspberry Pi via the physical cables.
- **Router** redirects the signals that contain movement data requested by the computer and it has a network connection with Raspberry Pi according to the TCP/IP protocol.

4.5 Design Rationale

4.5.1 Context View

The rationale behind the context view is keeping the scope of the system as self-sufficient as possible with minimal external guidance (except its interaction with the child). The external entities aside from the children are only needed for the initial setup of the system. The children and the system exchange data throughout their interactions. The system processes the information gathered and reacts accordingly. The primary concern of the context view is the interaction between the system and the children as this is the main goal of the system.

4.5.2 Functional View

The rationale behind the functional view is defining the major operations in a way that they are consistent, coherent, and not interdependent. Not being interdependent means that the operations are done and completed within an element, instead of making use of multiple elements throughout the operations. This helps the modularity and the simplicity of the functionality. They are also loosely coupled, meaning that the elements are not strictly dependent on each other.

4.5.3 Information View

The rationale behind the information view is storing the minimal amount of personal data while offering the full functionality. YOLO only records the information relevant with the playing activity and not anything else. It deals with information ownership by using references to objects instead of holding copies of them. The information is only kept in the local memory of the robot, it is not uploaded to anywhere else. This helps with information security.

4.5.4 Deployment View

The rationale behind the deployment view is keeping the overall software and hardware structure simple and modular. Each module (component) has its own file and all the other modules using them import them beforehand. The hardware modules are kept simple enough so that they are supported by the operating system and the external hardware required to communicate with them. It is provided that the wi-fi module (router) is able to connect to virtually any house-grade wi-fi module. A simple Raspberry-Pi computer is being used as an operating system. This helps in the portability of the system while still keeping the operating system universal enough so that any Python code can be run on it. Aside from Python, no other software framework is needed as this would make programming and reprogramming the robot easier.

5. Novelty

YOLO could also take the child's face expressions into account when planning the mirroring and/or the contrasting behavior. To further explain, when the robot is in the mirroring mode, the happiness level of the child positively influences the intensity of the movement (Happier expression means faster and more exaggerated movement). When the robot is in the contrasting mode, it negatively influences the intensity of the movement (Happier expression means slower and less exaggerated movement). The system achieves this by computing the happiness level out of the face expression.

5.1 Context View

In the context view, the data that the child provides to YOLO now also includes the face expression data and the imitation that YOLO provides to the child takes the face expression into account as well. The external interface diagram did not change as the interfaces that the system provides to the external entities did not change. The new context diagram is below:

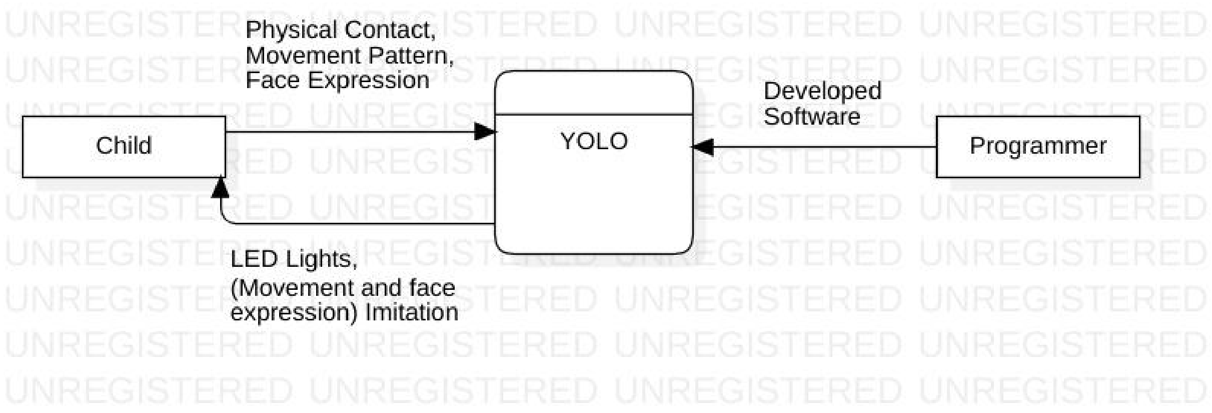
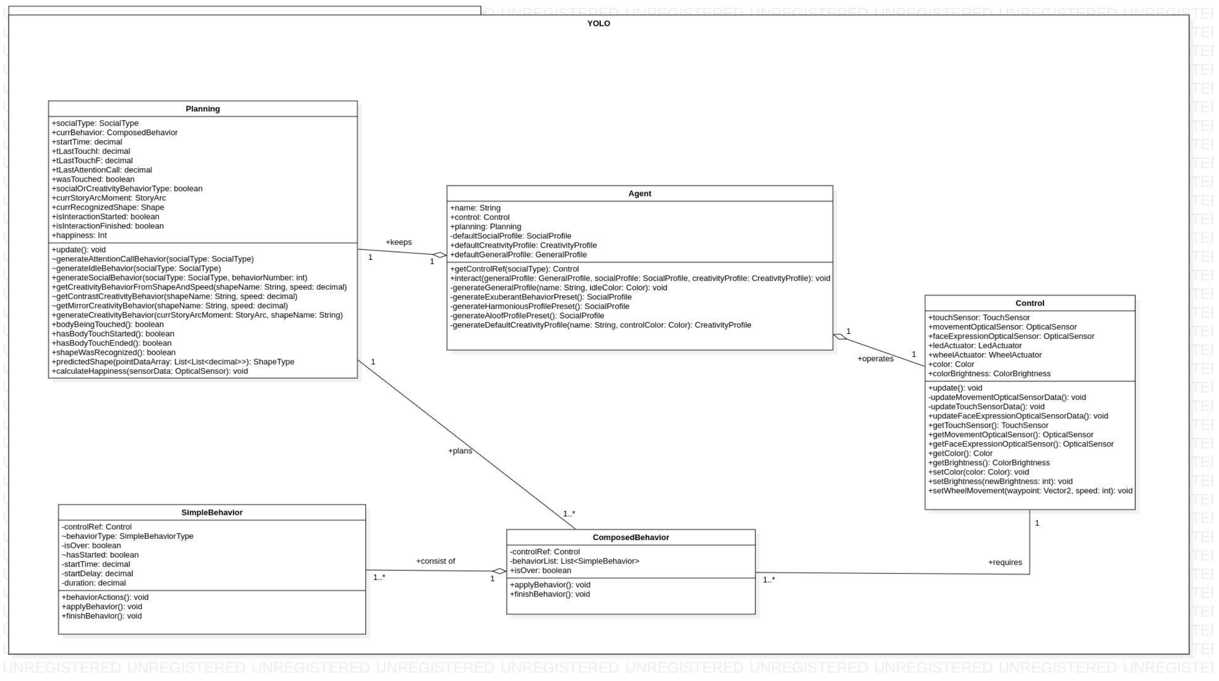


Figure 12: The New Context Diagram for YOLO

5.2 Functional View

In the functional view, the internal interface diagram now includes the member variables and the operations related with the retrieval and the processing of the face expression data. The component diagram did not change as the interfaces that the system provides to the external entities did not change. The new internal interface diagram is below:



YOLO
Planning
+socialType: SocialType
+currBehavior: ComposedBehavior
+startTime: decimal
+tLastTouchI: decimal
+tLastTouchF: decimal
+tLastAttentionCall: decimal
+wasTouched: boolean
+socialOrCreativityBehaviorType: boolean
+currStoryArcMoment: StoryArc
+currRecognizedShape: Shape
+isInteractionStarted: boolean
+isInteractionFinished: boolean
+happiness: Int
+update(): void
~generateAttentionCallBehavior(socialType: SocialType)
~generateIdleBehavior(socialType: SocialType)
+generateSocialBehavior(socialType: SocialType, behaviorNumber: int)
+getCreativityBehaviorFromShapeAndSpeed(shapeName: String, speed: decimal)
~getContrastCreativityBehavior(shapeName: String, speed: decimal)
~getMirrorCreativityBehavior(shapeName: String, speed: decimal)
+generateCreativityBehavior(currStoryArcMoment: StoryArc, shapeName: String)
+bodyBeingTouched(): boolean
+hasBodyTouchStarted(): boolean
+hasBodyTouchEnded(): boolean
+shapeWasRecognized(): boolean
+predictedShape(pointDataArray: List<List<decimal>>): ShapeType
+calculateHappiness(sensorData: OpticalSensor): void
+keeps
1
1
Agent
+name: String
+control: Control
+planning: Planning
-defaultSocialProfile: SocialProfile
+defaultCreativityProfile: CreativityProfile
+defaultGeneralProfile: GeneralProfile
+getControlRef(socialType): Control
+interact(generalProfile: GeneralProfile, socialProfile: SocialProfile, creativityProfile: CreativityProfile): void
-generateGeneralProfile(name: String, idleColor: Color): void
-generateExuberantBehaviorPreset(): SocialProfile
-generateHarmoniousProfilePreset(): SocialProfile
-generateAloofProfilePreset(): SocialProfile
-generateDefaultCreativityProfile(name: String, controlColor: Color): CreativityProfile
1
+operates
1
Control
+touchSensor: TouchSensor
+movementOpticalSensor: OpticalSensor
+faceExpressionOpticalSensor: OpticalSensor
+ledActuator: LedActuator
+wheelActuator: WheelActuator
+color: Color
+colorBrightness: ColorBrightness
+update(): void
-updateMovementOpticalSensorData(): void
-updateTouchSensorData(): void
+updateFaceExpressionOpticalSensorData(): void
+getTouchSensor(): TouchSensor
+getMovementOpticalSensor(): OpticalSensor
+getFaceExpressionOpticalSensor(): OpticalSensor
+getColor(): Color
+getBrightness(): ColorBrightness
+setColor(color: Color): void
+setBrightness(newBrightness: int): void
+setWheelMovement(waypoint: Vector2, speed: int): void
1
+plans
1..*
1
+requires
SimpleBehavior
-controlRef: Control
~behaviorType: SimpleBehaviorType
-isOver: boolean
~hasStarted: boolean
-startTime: decimal
-startDelay: decimal
-duration: decimal
+behaviorActions(): void
+applyBehavior(): void
+finishBehavior(): void
+consist of
1..*
1
ComposedBehavior
-controlRef: Control
-behaviorList: List<SimpleBehavior>
+isOver: boolean
+applyBehavior(): void
+finishBehavior(): void
1..*

Figure 13: The New Internal Interface Class Diagram for YOLO

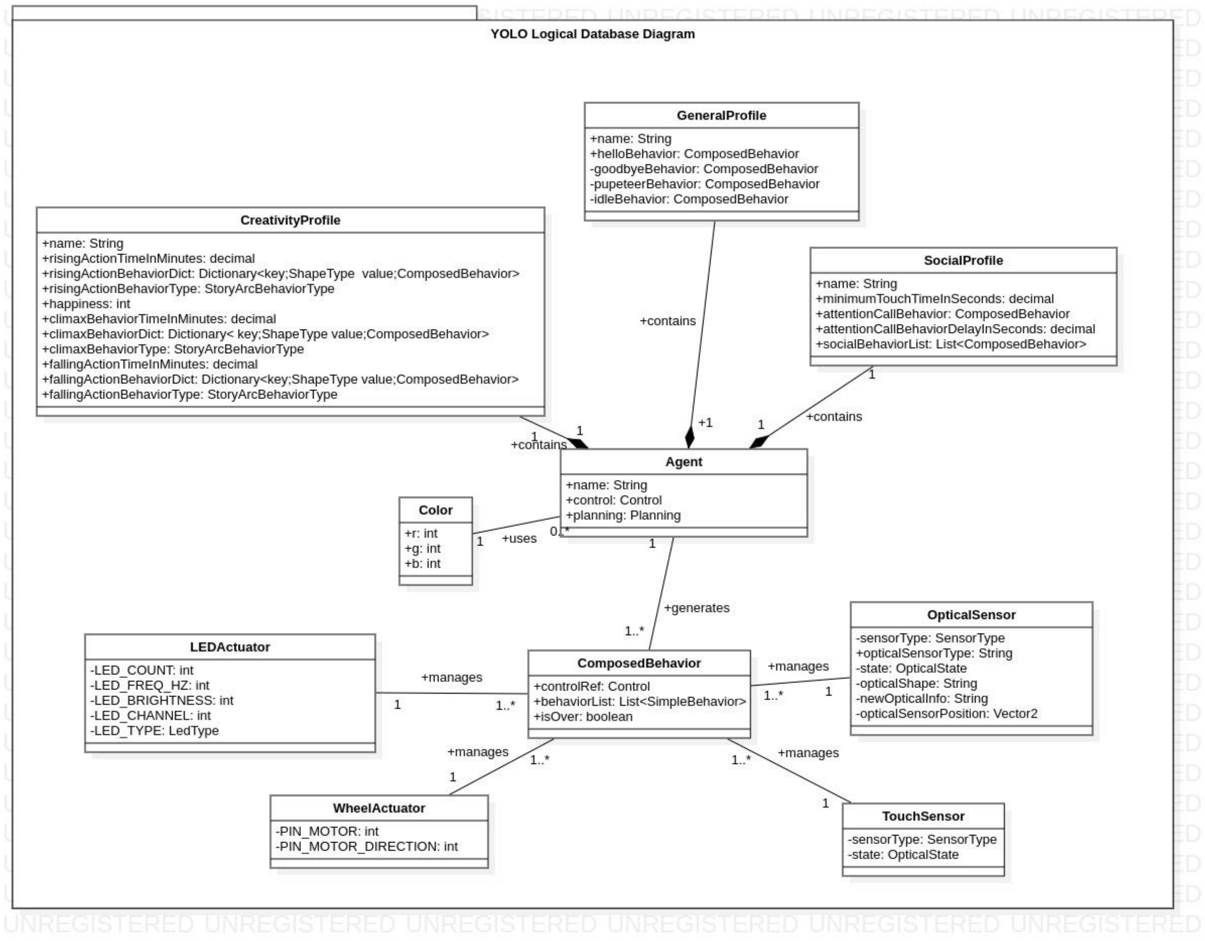
The descriptions of the newly added operations are below:

Operation	Description
updateMovementOpticalSensorData	Updates the data read by the optical sensor related with the movement.
updateFaceExpressionOpticalSensorData	Updates the data read by the optical sensor related with the face expression of the child.
getMovementOpticalSensor	Gets the optical sensor class object related with the movement.
getFaceExpressionOpticalSensor	Gets the optical sensor class object related with the face expression of the child.
calculateHappiness	Calculates the happiness level of the child from their face expression

Table 6: The Added Internal Interface Operation Descriptions

5.3 Information View

In the information view, the database requirements class diagram changed to include the happiness level derived from the face expression of the child in CreativityProfile and the member variable to differentiate between the movement and the face expression optical sensors in OpticalSensor. The new database requirements class diagram is given below:



YOLO Logical Database Diagram
GeneralProfile
+name: String
+helloBehavior: ComposedBehavior
-goodbyeBehavior: ComposedBehavior
-pupeteerBehavior: ComposedBehavior
-idleBehavior: ComposedBehavior
CreativityProfile
+name: String
+risingActionTimeInMinutes: decimal
+risingActionBehaviorDict: Dictionary<key;ShapeType value;ComposedBehavior>
+risingActionBehaviorType: StoryArcBehaviorType
+happiness: int
+climaxBehaviorTimeInMinutes: decimal
+climaxBehaviorDict: Dictionary< key;ShapeType value;ComposedBehavior>
+climaxBehaviorType: StoryArcBehaviorType
+fallingActionTimeInMinutes: decimal
+fallingActionBehaviorDict: Dictionary<key;ShapeType value;ComposedBehavior>
+fallingActionBehaviorType: StoryArcBehaviorType
SocialProfile
+name: String
+minimumTouchTimeInSeconds: decimal
+attentionCallBehavior: ComposedBehavior
+attentionCallBehaviorDelayInSeconds: decimal
+socialBehaviorList: List<ComposedBehavior>
+contains
+1
+contains
1
+contains
Agent
+name: String
+control: Control
+planning: Planning
Color
+r: int
+g: int
+b: int
+uses
0..*
+generates
1..*
LEDActuator
-LED_COUNT: int
-LED_FREQ_HZ: int
-LED_BRIGHTNESS: int
-LED_CHANNEL: int
-LED_TYPE: LedType
+manages
ComposedBehavior
+controlRef: Control
+behaviorList: List<SimpleBehavior>
+isOver: boolean
OpticalSensor
-sensorType: SensorType
+opticalSensorType: String
-state: OpticalState
-opticalShape: String
-newOpticalInfo: String
-opticalSensorPosition: Vector2
WheelActuator
-PIN_MOTOR: int
-PIN_MOTOR_DIRECTION: int
TouchSensor
-sensorType: SensorType
-state: OpticalState

Figure 14: The New Database Requirements Class Diagram for YOLO

Here are the newly added operations on data:

Operation	Description
updateMovementOpticalSensorData	Create: Read: Update: OpticalSensor Delete:
updateFaceExpressionOpticalSensorData	Create: Read: Update: OpticalSensor Delete:
getMovementOpticalSensor	Create:

	Read: OpticalSensor Update: Delete:
getFaceExpressionOpticalSensor	Create: Read: OpticalSensor Update: Delete:
calculateHappiness	Create: Read: OpticalSensor Update: CreativityProfile Delete:

Table 8: CRUD Operations of the Newly Added Operations

5.4 Deployment View

In the deployment view, the deployment diagram now includes an additional optical sensor designed to recognize the facial expression data. The new deployment diagram is given below:

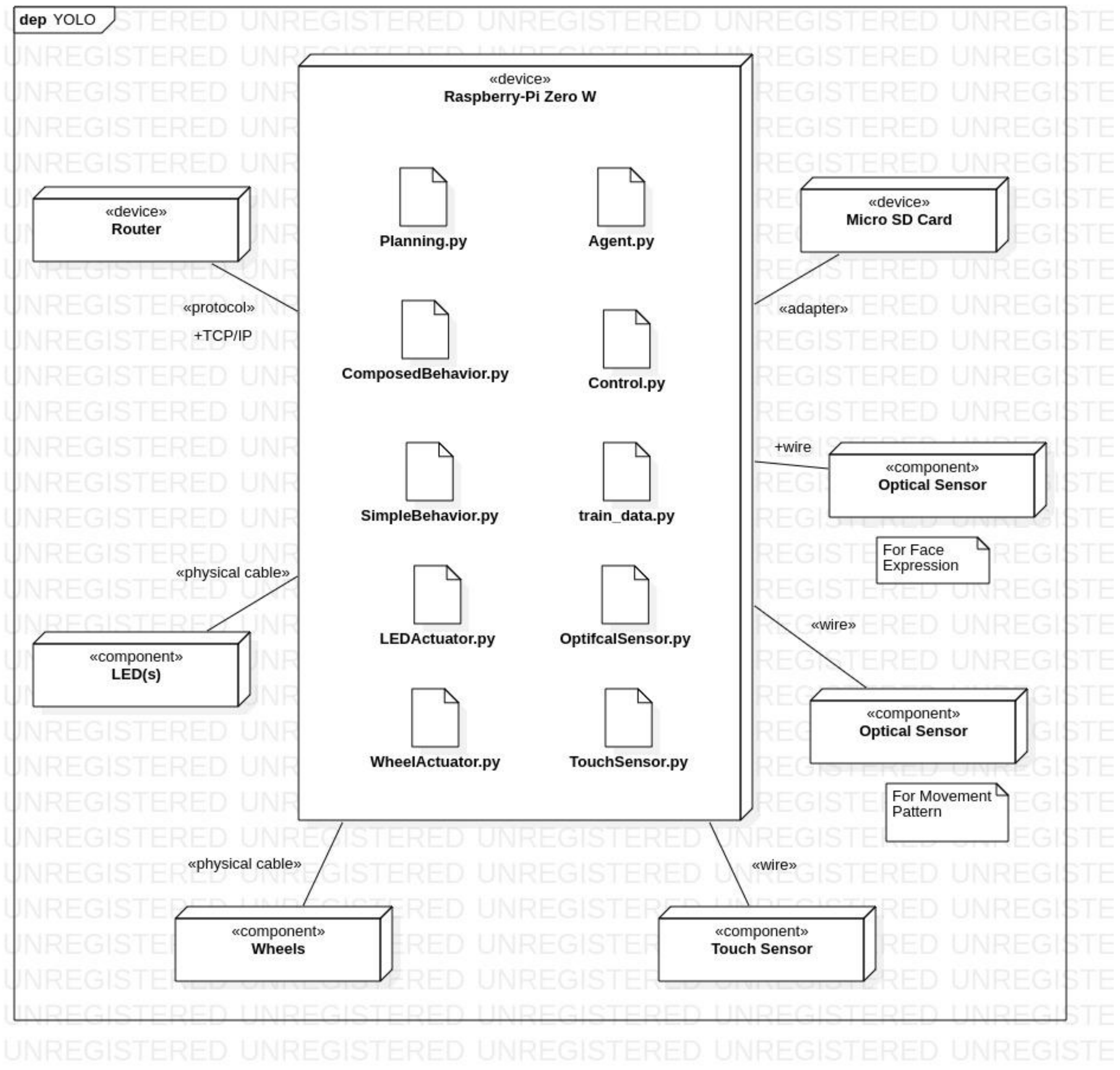


Figure 15: The New Deployment Diagram For YOLO