Software Architecture Description

for

YOLO

Version 1.0

Prepared by
Muhammad Haseeb MOTAN
2398071
Tayyip ÖZTÜRK
2380806

Group: 27

Date: 8 June 2022

Table of Contents

List of Figures

List of Tables

1 Introduction

This document encapsulates the Software Design Description (SDD) of YOLO (Your Own Living Object), which is an autonomous robotic toy made for enhancing the creativity of children and their parents during their free play activities. The official document for YOLO can be accessed from here[1].

1.1 Purpose and objectives of the YOLO social robot

The purpose of this project is creating new story-lines to users of YOLO, which are ordinarily children, their parents and developers of the project. In this wise, children have the opportunity of developing their social and analytical skills via interacting a social robot and experiencing different behaviors of robot during their free play activities.

1.2 Scope

The target audience of this project is primarily children. In a manner suitable for the purposes of the system, YOLO can be used in various environments and for different purposes:

- Education
- Free play activities
- Physical and/or mental rehabilitation of children

The system has been designed and developed in accordance with the capabilities that it be needed to serve in these fields and purposes. As a system, YOLO (Your Own Living Object) includes different subsystems and parts, which enables it to interact with user in a manner of social interaction. Thus it has been equipped with a compatible layers of hardware including:

- Raspberry-Pi W Zero

[1] https://www.researchgate.net/publication/340367036˙YOLO˙-˙Your˙Own˙Living˙Object

- Optical sensor and touch sensor
- Motors, their drivers and dockings
- Wheels and wheels layer
- Batteries and their layer
- Controller boards and their layer
- Glowing fibers layer and their layer
- A shell that envelops the hardware system of the system and protects it against damage.

Those components also gives the ability to the system interact with user by combining movements, light states and animations. Moreover, the system is equipped with a software that infers user's behaviour pattern and act differently for each user, besides developing YOLO over the API provided and creating new behaviour profiles manually.
From stakeholder's point of view, YOLO is designed and developed to be used by various organizations and communities, such as:

- Educational Institutions
- Healthcare Institutions
- Families

Although YOLO approximately has a $200 expense of development per unit and can be considered expensive, it brings many benefits for these communities, however, a medical treatment including physical and mental rehabilitation of a child can not be measured by money. Moreover, this expenditure is nothing compared to its value and compared to the expenditure of other techniques used by stakeholders to get similar results.

1.3 Stakeholders and their concerns

Users

The users of YOLO are generally children which are in middle school ages or younger and their parents. However, anyone can experience this robot for their free play activities or physical/-mental rehabilitation.

- For a child user, interaction-feedback speed is an important concern and the diversity of free-play profiles is another concern.
- For a parent user, primary concerns are safety and durability of the device during an free-play activity.

Developers

Developers are also using YOLO for their software and system development projects. Developer should be interested in programming with Python and working on Raspbian Stretch Lite OS.

- For a developer, availability of the device to be programmable is a main concern. YOLO should be programmed and run different profiles with different behaviours.
- API compatibility and connection stability is a relevant concern to the main concern of a developer.

2 References

1. https://www.researchgate.net/publication/340367036_YOLO_-_Your_Own_Living_Object
2. https://github.com/patricialvesoliveira/YOLO-Software

3 Glossary

Name	Definition
YOLO	Your Own Living Robot
Stakeholder	User/Developer
User	Child or parent who uses a device or program
Developer	A person or company that creates new products, especially computer products such as software
Profile	Data specific to a particular user
Wi-Fi	Wireless Fidelity
API	Application Programming Interface
SD Card	Secure Digital Card
IP	Internet Protocol
LAN	Local Area Network
HTTP	Hypertext Transfer Protocol
ML	Machine Learning
DBMS	Database Management Systems
Sensor	Device that converts real-world data to computer understandable digital data
Actuator	A component of a machine that is responsible for moving and controlling a mechanism or system
LED	Light Emitting Diode

Table 3.1: Glossary

4 Architectural Views

4.1 Context View

4.1.1 Stakeholders' use of this view

This view is used by stakeholders to understand the way external entities interact with YOLO and vice versa.

4.1.2 Context Diagram

The system context diagram of YOLO indicates the external entities interacting with the system, and their relations with it. Over these relations, YOLO can fulfill its objectives. Entities are shown in the Figure 4.1 below.

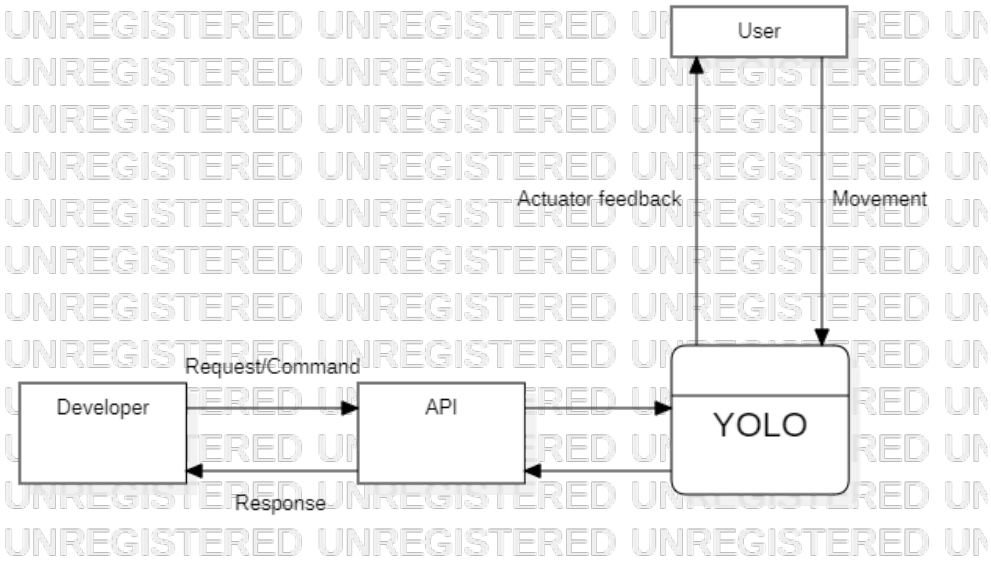

Figure 4.1: System Context Diagram

As it can be observed in System Context Diagram above, a user can interact with YOLO by

making movements with it and gets feedback from actuators of the device.
A developer can send commands or requests over the API provided to YOLO and get responses from the device again over the API.

4.1.3 External Interfaces

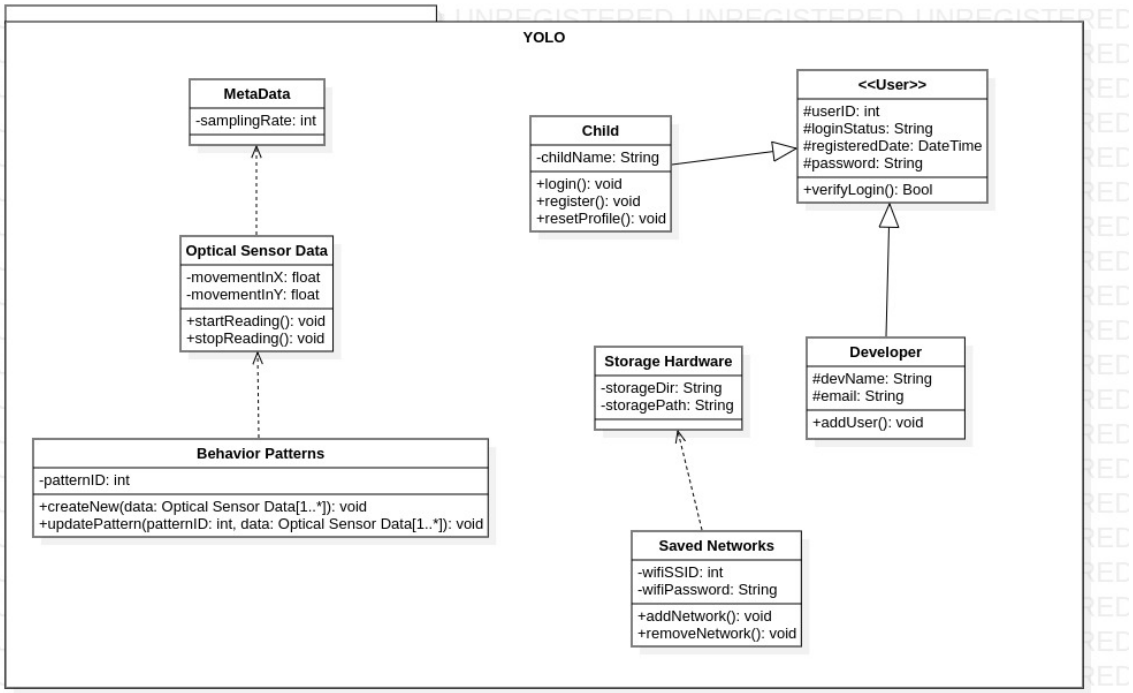

Figure 4.2: External Interfaces Class Diagram

- Child and developer both fall under the category User.
- Network connection details are stored so that the device is able to connect promptly whenever a saved network is in range.
- Behavior patterns are dependent on optical sensor data (shape drawn) which is dependent on the frequency at which images are taken i.e. the sampling rate.

Operation	**Description**
login	Logs the stakeholder into the system
register	Allows the stakeholder to create a new account
resetProfile	Resets the profile back to default settings and erases profile data
verifyLogin	Returns true if the entered credentials are correct
addUser	Allows the developer to create a new account with administrator privileges
addNetwork	Adds a network to the list of saved networks
removeNetwork	Removes a network from the list of saved networks
startReading	Start tracking movements upon stimulus
stopReading	Stop tracking movements after a period of inactivity
createNew	create a new behaviour pattern corresponding the the recent movement
updatePattern	update a similar behaviour pattern

Table 4.1: Operation Descriptions

4.1.4 Interaction scenarios

Interaction scenarios and their activity diagrams for YOLO:

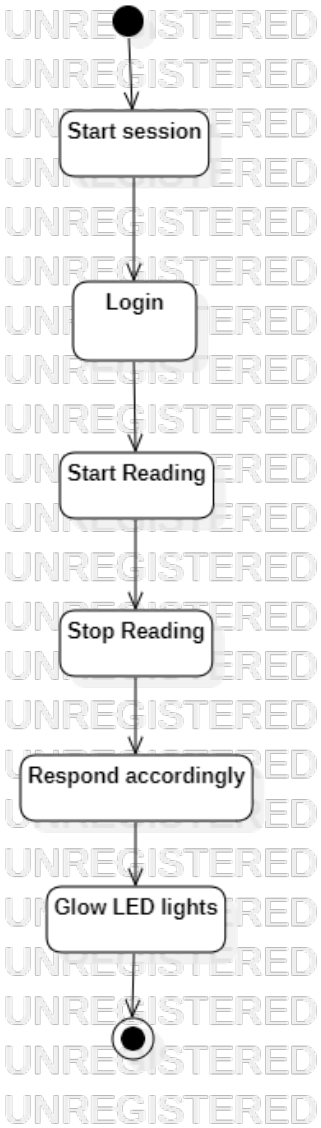

Figure 4.3: Activity Diagram For Getting User Input Regarding Movement

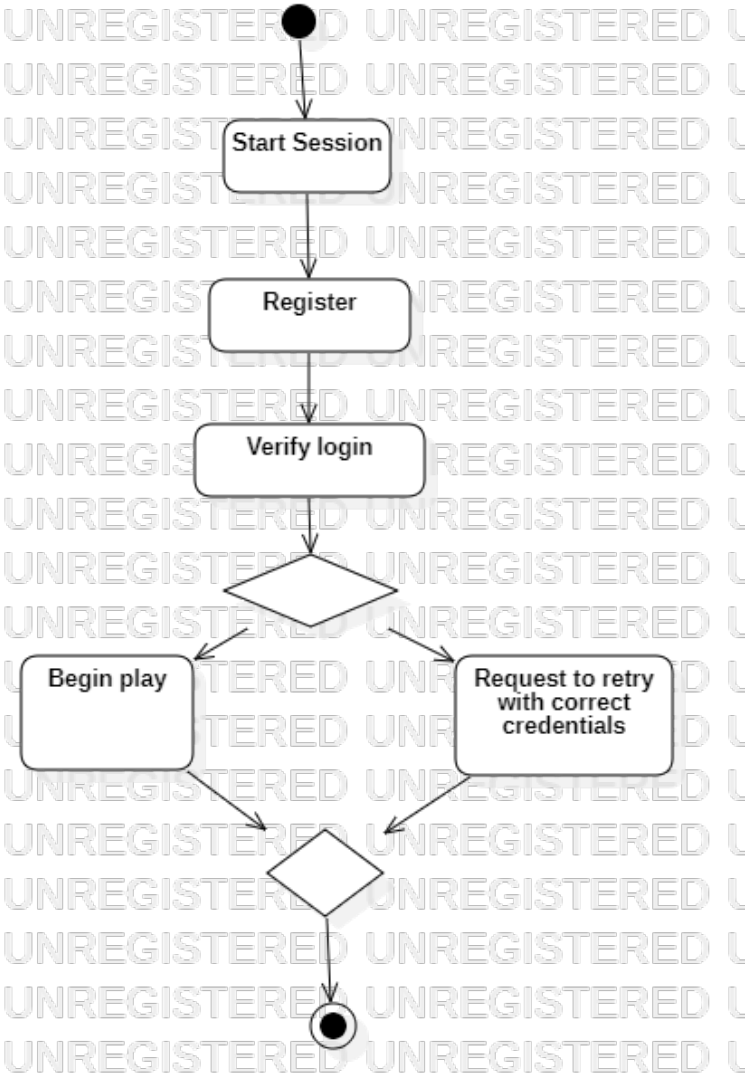

Figure 4.4: Activity Diagram For User Creating New Profile

4.2 Functional View

4.2.1 Stakeholders' use of this view

Stakeholders use this view as a guide to how different components work together. This can be further used to identify reusable, manageable or swappable subsystems. Those subsystems' characteristics and procedures are explained and demonstrated to give an ease to develop and use the whole system.

4.2.2 Component Diagram

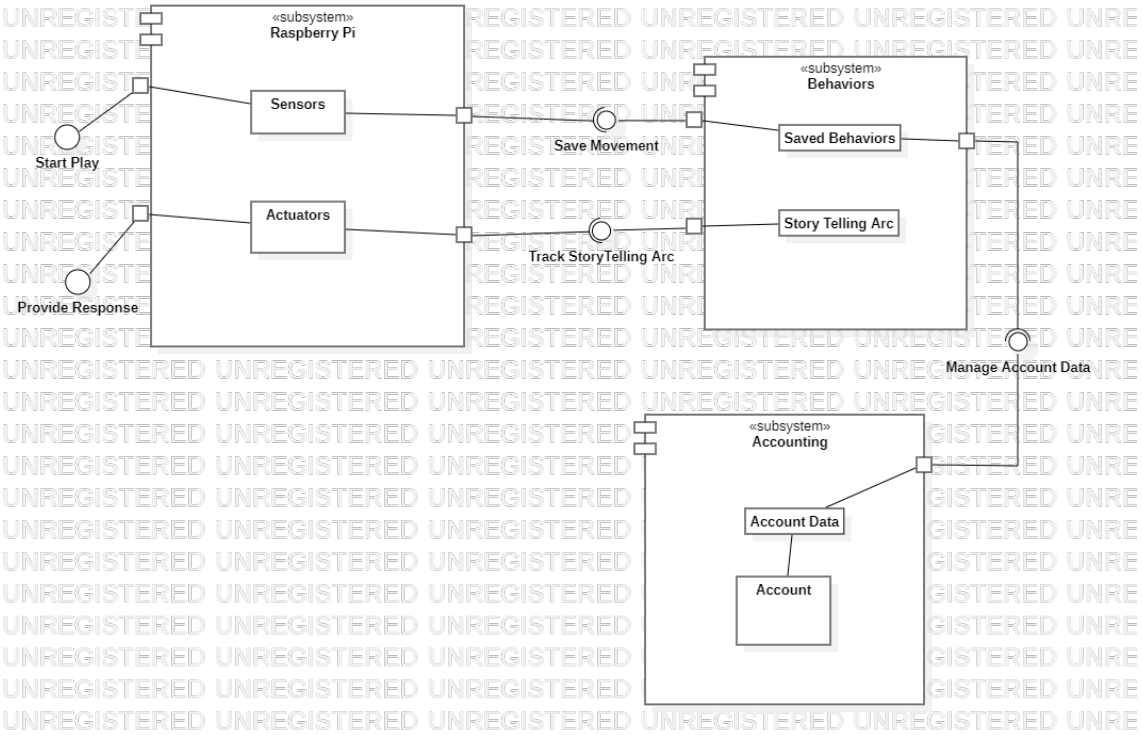

Figure 4.5: Component Diagram

- Raspberry Pi is used for starting play activity. It recognizes the start play command via its sensors.
- Raspberry Pi is used for gathering input data via its sensors and processing them during the free play activity.
- Raspberry Pi processes the data gathered from sensors to provide response via its actuators.
- Raspberry Pi provides response to the user via its actuators and can track story telling arc via responding them with actuators.
- Behaviours subsystem gets account data from accounting subsystem.

- Behaviours subsystem saves behaviours according to the account data taken.
- Behaviours subsystem creates movement data and provides it to Raspberry Pi subsystem.
- Behaviours subsystem creates new Story Telling Arc and can edit story telling arc behaviours.
- Accounting subsystem manages accounts and their data gathered.
- Accounting subsystem provides account data to behaviours subsystem to save behaviours for each account.

4.2.3 Internal Interfaces

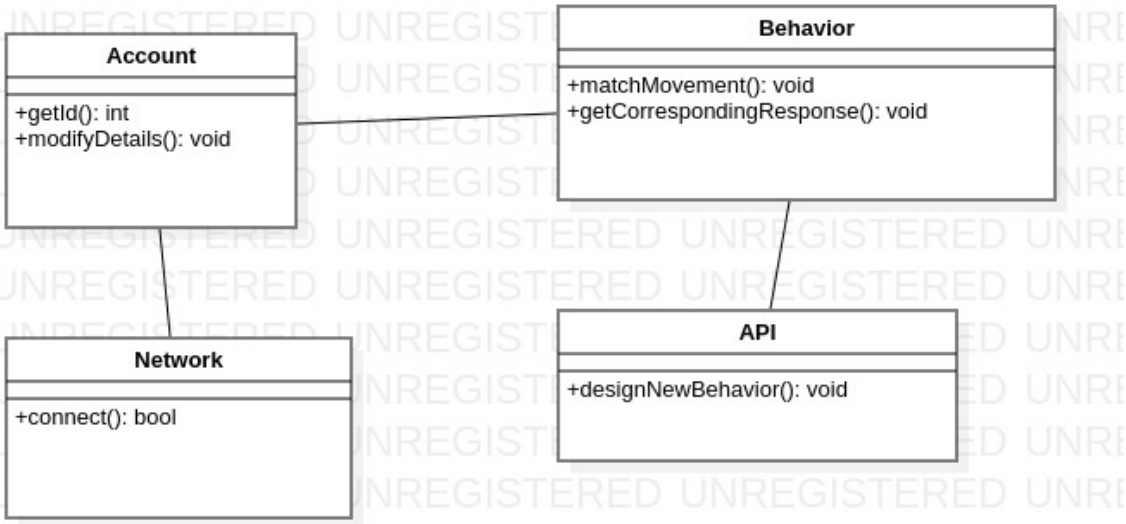

Figure 4.6: Internal Interfaces Class Diagram

In internal interfaces, there are four different interface.

- Network interface has one function which establishes the connection between YOLO and router.
- Account interface has two functions which are getId() and modifyDetails().
- int getId() functions returns an identity number which is unique to its user.
- void modifyDetails() function changes the initialized values belong to an account.

- Behaviour interface has two functions which are matchMovement() and getCorrespondingResponse().
- void matchMovement() function recognizes a movement and matches it with a user by processing the data.
- void getCorrespondingResponse() function gives a response according to the movement recognized.
- API interface has one function which is designNewBehaviour().
- void designNewBehaviour() function helps developer to create new behavior for YOLO. Via this functionality, YOLO can also be pushed to create new behaviour.

4.2.4 Interaction patterns

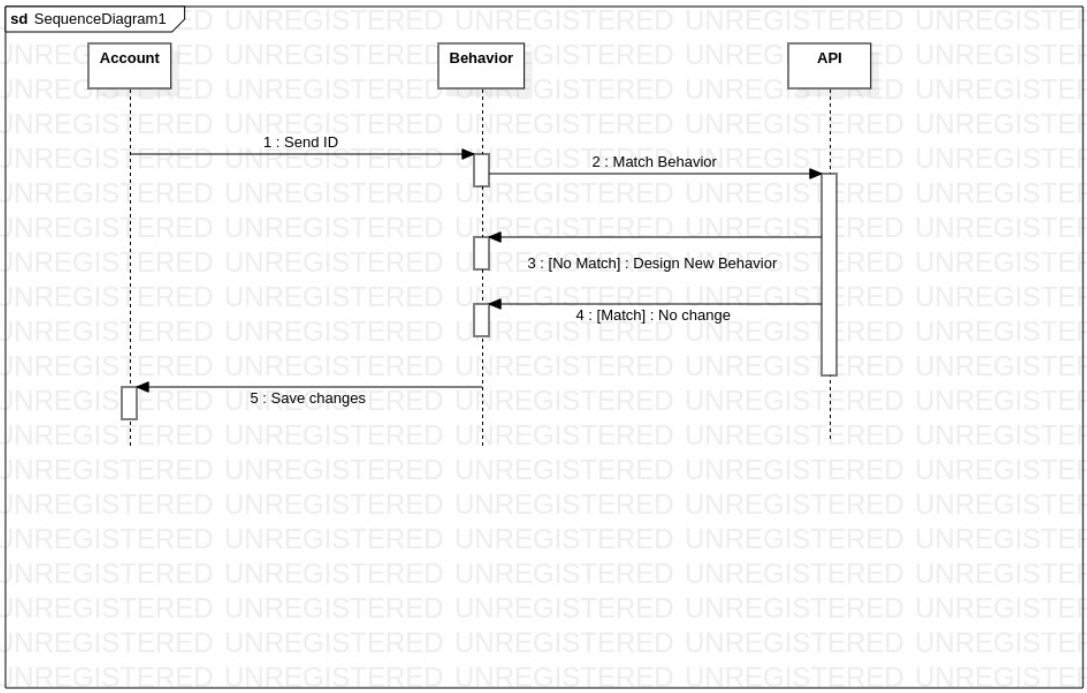

Figure 4.7: Sequence Diagram for Modify User Details

For modifying user details, account interface sends id of the user to the behaviour interface. Behaviour interface checks if a behaviour is matched with this user by requesting it from the API. According to the API response if there is no previous behaviour matched, new behaviour is

designed and created for the user. Else, no change is made. After that, behaviour interface saves the changes to the account of the user.

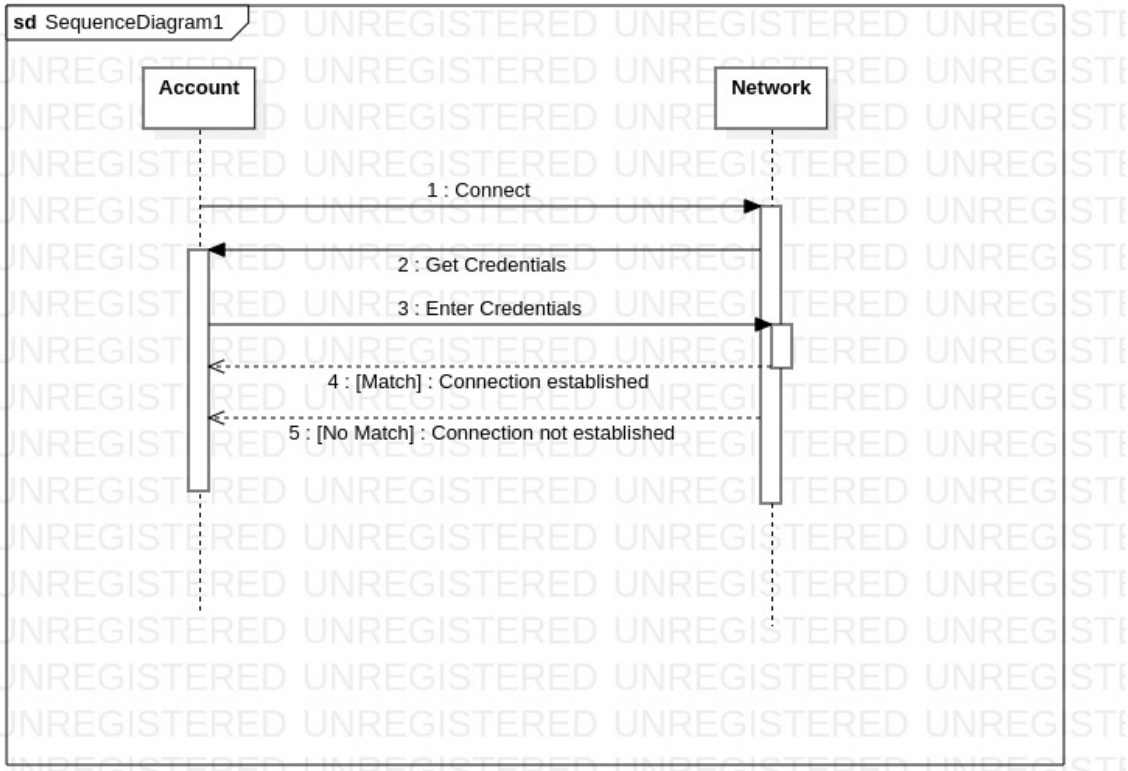

Figure 4.8: Sequence Diagram for Connect before Starting Play

For connecting to the YOLO before play, account interface connects to the network and get credentials for the user. Account interface then enters its credentials and if credentials are appropriate, connection is established. Else, connection is not established.

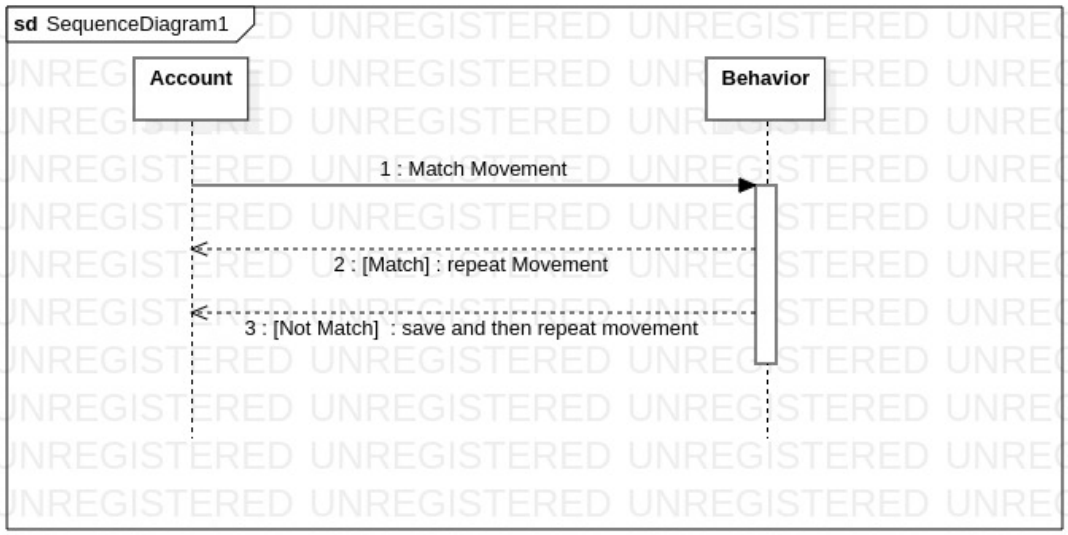

Figure 4.9: Sequence Diagram for Proceeding Play

For proceeding to a play activity, account interfaces tries to match movement. If movement is matched with a previous one in behaviour interface, movement is repeated immediately by the response of behaviour interface. Else, the movement is first saved for the user and then repeated.

4.3 Information View

4.3.1 Stakeholders' use of this view

This view helps stakeholders examine the database operations involved. Components involved are expressed in terms of data manipulation, data transmission and data collection. Via these information, changes and upgrades can be implemented more appropriately and time consumed for development decreases and efficiency increases by this view.

4.3.2 Database Class Diagram

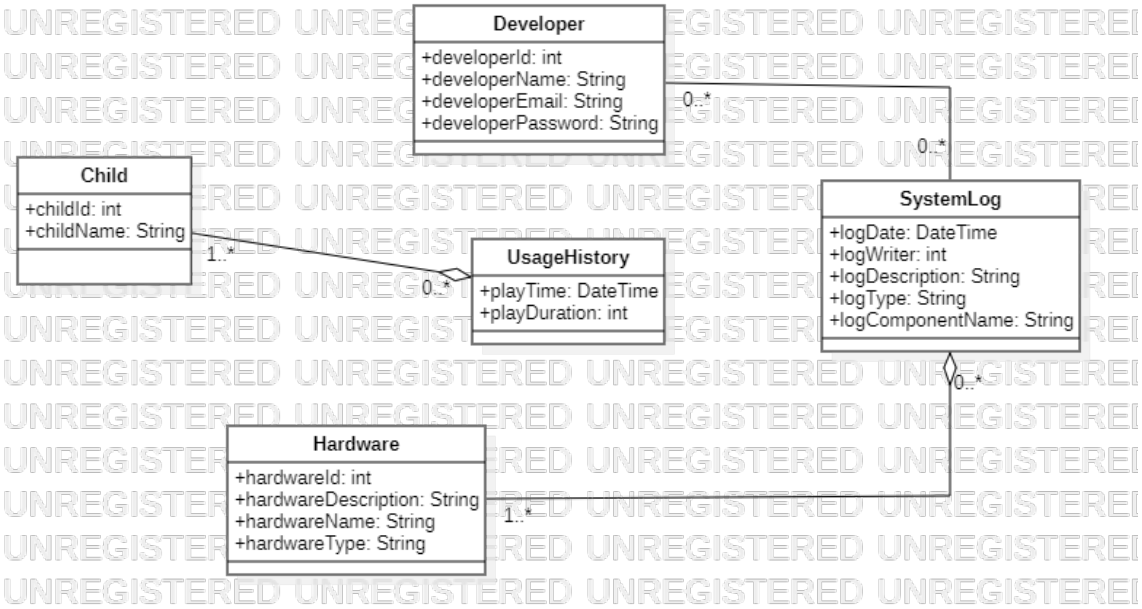

Figure 4.10: Database Class Diagram

Database attributes and their descriptions are below for given classes in Figure 4.10: Database Class Diagram.

Name	**Description**
int Child::childId	Unique ID number for each child user
String Child::childName	Name of child user
DateTime UsageHistory::playTime	Total play time of the YOLO
int UsageHistory::playDuration	Play time from turning on the YOLO
int Developer::developerID	Unique ID number for each developer of YOLO
String Developer::developerName	Name of developer
String Developer::developerEmail	E-mail address of developer
String Developer::developerPassword	Password of developer for authentication
DateTime SystemLog::logDate	Date of saved log
int SystemLog::logWriter	Information who was the writer of log
String SystemLog::logDescription	Explains why this log is written
String SystemLog::logType	Class of log among categories of logs
String SystemLog::logComponentName	Component name why log is written for
int Hardware::hardwareId	Unique ID for hardware part of the YOLO
String Hardware::hardwareDescription	Explanation about what hardware is used for
String Hardware::hardwareName	Name of hardware
String Hardware::hardwareType	Class of hardware among categories of hardwares

Table 4.2: Database Attribute Descriptions

Database operations and their descriptions are below for given classes in Figure 4.10: Database Class Diagram.

Operation	**Description**
createChild	Creates new child object
removeChild	Removes the child object with given childId
getChild	Returns the child object with given childId
createUsageHistory	Creates new UsageHistory object
removeUsageHistory	Removes the UsageHistory object belongs to user with given childId
setPlayTime	Starts recording playTime for new user. It is also called by createUsage-History operation. Sets playTime to zero initially
unsetPlayTime	Sets playTime to zero
setPlayDuration	Starts recording playDuration from turning on the YOLO
unsetPlayDuration	Sets playDuration to zero when YOLO is turned off
getPlayTime	Returns the playTime of user with given childId
getPlayDuration	Returns the playDuration of user with given childId
addDurationToPlayTime	Adds the playDuration to the playTime attribute
createDeveloper	Creates new Developer object
removeDeveloper	Removes the Developer object with given developerId
getDeveloper	Returns the Developer object with given developerId
loginDeveloper	Logins to the API interface with given developerId and developerPass-word credentials
logoutDeveloper	Logouts from the API interface for given developerId
updatePassword	Updates the developerPassword of the Developer object with given de-veloperId
createLog	Creates new SystemLog object
removeLog	Removes the SystemLog object with given logDate
getLog	Returns the SystemLog object with given logDate
createHardware	Creates new Hardware object
removeHardware	Removes the Hardware object with given hardwareId
getHardware	Returns the Hardware object with given hardwareId

Table 4.3: Database Operation Descriptions

4.3.3 Operations on Data

Operation	CRUD Operations
createChild	Create: Child Read: Update: Delete:
removeChild	Create: Read: childId Update: Delete: Child
getChild	Create: Read: childId Update: Delete:
createUsageHistory	Create: UsageHistory Read: Update: Delete:
removeUsageHistory	Create: Read: Update: Delete: UsageHistory
setPlayTime	Create: Read: childId Update: playTime Delete:
unsetPlayTime	Create: Read: childId Update: playTime Delete:

setPlayDuration	Create: Read: childId Update: playDuration Delete:
unsetPlayDuration	Create: Read: childId Update: playDuration Delete:
getPlayTime	Create: Read: childId Update: Delete:
getPlayDuration	Create: Read: childId Update: Delete:
addDurationToPlayTime	Create: Read: childId, playDuration Update: playTime, playDuration Delete:
createDeveloper	Create: Developer Read: Update: Delete:
removeDeveloper	Create: Read: developerId Update: Delete: Developer
getDeveloper	Create: Read: developerId Update: Delete:

loginDeveloper	Create: Read: developerId, developerPassword Update: Delete:
logoutDeveloper	Create: Read: developerId Update: Delete:
updatePassword	Create: Read: developerId Update: developerPassword Delete:
createLog	Create: SystemLog Read: Update: Delete:
removeLog	Create: Read: Update: Delete: SystemLog
getLog	Create: Read: logDate Update: Delete:
createHardware	Create: Hardware Read: Update: Delete:
removeHardware	Create: Read: Update: Delete: Hardware
getHardware	Create: Read: hardwareId Update: Delete:

Table 4.4: Database Attribute Descriptions

4.4 Deployment View

4.4.1 Stakeholders' use of this view

This view gives the stakeholders an idea about how the system looks like. Stakeholders master the deployment and thus, they can maintain the fast and reliable deployment concepts such as CI/CD on periods of development easily.

4.4.2 Deployment Diagram

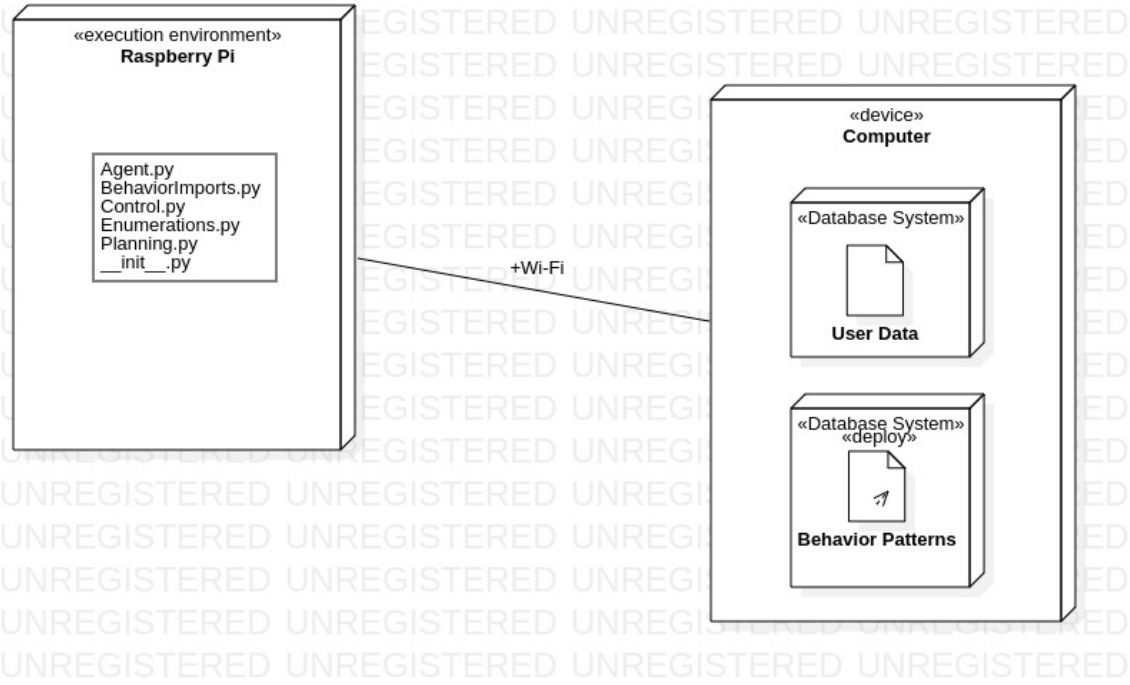

Figure 4.11: Deployment Diagram

- Raspberry Pi contains Python run-time environment and required libraries and classes[2]. to control YOLO.
- Agent.py class implements auxiliary Agent API structs. It generates creativity profile, social profile and general profile for play activity. It connects to the Database System via Wi-Fi and sets/gets behaviour patterns.

[2]https://github.com/patricialvesoliveira/YOLO-Software

- BehaviourImports.py implements behavioral feedbacks such as movements, actuator response. It generates movements and sends data to motors to move YOLO. It gives feedback via Raspberry Pi's actuators. For instance, light blink on light stripes are made in different colors consequently in the HelloBehaviour. These behaviours are exhibited via sending commands via control class operations.
- Control.py implements optical sensor, touch sensor, led actuator, wheel actuator attributes and operations. Setting up sensors and actuators, updating their condition are in Control.py's responsibility. Software-Hardware interaction is mainly implemented in here.
- Enumerations.py contains sensors, actuotors, sensor and actuator states and set their state variances such as color brightness amount, movement direction. Rising edge action decisions and behaviour types are enumerated here.
- Planning.py brings together the class implemented before and it enables YOLO to work appropriately. Autonomous interaction including learning the condition and changing/creating behaviour according to the situation, checking if the interaction has started and updating required variables, responses and behaviours are implemented here. User data acquired from computer's User Data database system is used for these autonomous stages.
- User Data is stored in a database in computer. It stores users of YOLO and data attributes belong to them. Required data is passed to the execution environment over Wi-Fi connection to access variety of methods and attributes related to the current user of YOLO.
- Created behaviour patterns and getting/setting/updating them are made in Behaviour Patterns database. This system interacts with Raspberry Pi execution environment over Wi-Fi connection and make required operations on data to help YOLO work properly.

4.5 Design Rationale

YOLO is firstly designed to be a competitive system in free play activities. For this reason it has been made of solid materials. Besides this materialistic point of view, YOLO has also many effective design for the sake of software architecture.

YOLO's architecture was designed to be compatible with the changes made during the development of the system. For this reason, YOLO was designed in layered architecture. Units, classes, components and the whole system is tongued each other and/or co-operating with each other such that each subsystem works independently and while depending to another subsystem

very well. Alongside the system's context view is particularly very simple such that, a developer can connect to the YOLO as well as a user is interacting YOLO via its sensors and actuators in a very normal way.

YOLO's software system's external interfaces are layered such that, behaviour patterns, optical sensor data and metadata are interacting with each other while hardware and network interfaces and user interface with its sub-interfaces child interface and developer interface are working very well independently from each other besides communicating and co-operating with each other very well. Increasing the functionality here smoothly is the first objective. Subsystems of the YOLO Raspberry Pi, Behaviours subsystem and Accounting subsystems are also providing each other data to run their processes with appropriate inputs and exhibit non-erroneous responses. While external interfaces are co-operating in external frame, in internal frame network, account, behaviour and API co-operation can also be done perfectly, thanks to this architecture.

Abstractions, data type definitions, class implementations, their attributes and operations are also implemented in layered architecture. This construction make stakeholders' work more easier than other type of architectures and decreases construction cost. It not only decreases the construction cost of the system, it also decreases the maintenance, upgrade costs of the system while increasing the efficiency and speed of development process. Thanks to these convenience, YOLO can be viewed informatically simple and elegant.

YOLO's convenience in simplicity and functionality also helps to deployment process of the system. When a new functionality or new information, attribute, method, unit, class or component attached or detached to YOLO, it can be processed reliably and securely with deployment methods correctly and properly.

To conclude the design rationale of YOLO, it has been made to ease the way the system has been developed, maintained, upgraded, and used for whole the community including stakeholders, developers, parents, children. The YOLO has its community with a devotion to the project to ease the way it can be reached, upgrade the practicability of it and support people and encourage them to work with the YOLO.