TED UNIVERSITY

CMPE492/SENG492

Computer Engineering Senior Project

Software Engineering Senior Project

PyroGuard Fire Detection System

Low Level Design Report

10.03.2024

Team Members:

Ecem Sıla Gök (1118505616)

Ece Selin Adıgüzel (1988345282)

Zeynep Beyza Uçar (1004263330)

Doruk Aydoğan (1354737307)

Table of Contents

1. Introduction to Low-Level Design for PyroGuard Fire Detection System

The Low-Level Design (LLD) document for the PyroGuard Fire Detection System serves as a comprehensive guide for developers and engineers, detailing the implementation specifics that bridge the conceptual framework provided in the High-Level Design (HLD) with real-world application. This phase is critical for translating theoretical designs into practical, executable models that adhere to the outlined architectural principles, design patterns, and functional requirements established in the HLD. The LLD focuses on defining the system's components, classes, interfaces, data structures, and algorithms in detail, ensuring a robust foundation for the development, testing, and maintenance of the PyroGuard system. This document aims to address the nuances of object design trade-offs, establish interface documentation guidelines, adhere to engineering standards such as UML and IEEE, and clarify definitions, acronyms, and abbreviations used throughout the design and development process.

1.1 Object Design Trade-offs

In designing the PyroGuard Fire Detection System, several critical trade-offs must be carefully balanced to achieve a system that is efficient, reliable, and scalable while remaining maintainable and user-friendly.

Deep Learning Model Complexity vs. Execution Speed: The choice of deep learning models for fire detection and analysis involves balancing the complexity and accuracy of the model against the system's ability to process data in real-time. A more complex model may offer higher detection accuracy but could also lead to slower processing times, which is unacceptable in a system where real-time detection is crucial. The trade-off here involves selecting or designing models that provide an optimal balance, ensuring that the system can make timely and accurate fire detection decisions without significant delays.

Scalability vs. Initial Complexity: Designing a system to be scalable from the outset often introduces additional complexity in the initial phases of development. For PyroGuard, scalability is essential to accommodate future growth, such as integrating more sensor types, covering larger geographical areas, or handling increased data volumes from additional camera feeds. The trade-off involves implementing a flexible and modular architecture that can grow without necessitating complete redesigns, while managing the initial complexity this flexibility introduces.

Customizability of Notifications vs. User Experience Simplicity: The notification subsystem must strike a balance between offering sufficient customization options to meet diverse user needs and maintaining a simple, intuitive user interface. This balance ensures that in critical situations, users can quickly understand and act upon the information provided by the system without being overwhelmed by unnecessary complexity or configurability.

1.2 Interface Documentation Guidelines

Clear and comprehensive interface documentation is crucial for the development and future integration of the PyroGuard system. The following guidelines ensure that all system interfaces are well-documented, understandable, and usable:

Standardized Documentation Format: Adopt a uniform documentation format across all interfaces to ensure consistency. This format should include the interface name, purpose, input parameters, output results, possible error states, and exception handling mechanisms. Providing a standardized format helps developers quickly familiarize themselves with different parts of the system.

Detailed Usage Examples: Supplement the documentation with practical examples demonstrating how to use each interface. These examples should cover common use cases and scenarios, illustrating the steps required to perform specific tasks or achieve certain outcomes within the system.

Version Control and Change Logs: Maintain accurate version control for all documented interfaces, including a detailed change log for each version. This approach facilitates backward compatibility and helps developers understand the evolution of the interface over time, making it easier to adapt to or integrate with the system.

1.3 Engineering Standards (e.g., UML and IEEE)

The PyroGuard Fire Detection System's design and documentation processes adhere to internationally recognized engineering standards to ensure high quality, consistency, and interoperability:

Unified Modeling Language (UML): Extensively use UML diagrams, including class diagrams to describe data structures and relationships, sequence diagrams to outline interactions between system

components, and activity diagrams to depict the workflow of the system. UML provides a standardized way to visualize the system's architecture and behavior, facilitating clear communication among developers and stakeholders.

IEEE Standards for Software Documentation: Follow the IEEE standards for documenting software requirements, design, testing procedures, and maintenance guidelines. These standards ensure that all aspects of the PyroGuard system are thoroughly and consistently documented, supporting effective development, testing, and long-term maintenance.

1.4 Definitions, Acronyms, and Abbreviations

To ensure that the document is accessible and understandable to all stakeholders, including developers, engineers, and end-users, a comprehensive section is dedicated to explaining the definitions, acronyms, and abbreviations used throughout the documentation:

- Deep Learning (DL): A subset of machine learning that utilizes neural networks with many layers (deep architectures) to analyze various types of data, including images and videos. In PyroGuard, DL models are used for the accurate detection of fire and smoke in video feeds. [1]
- Role-Based Access Control (RBAC): A method for restricting system access to authorized users based on their roles within the organization. RBAC facilitates the management of user permissions, ensuring that users can only access the information and functionalities relevant to their roles. [2]
- Application Programming Interface (API): A set of protocols, routines, and tools for building software applications, allowing different software entities to communicate with each other. PyroGuard's APIs enable integration with external systems, such as emergency response systems and user notification services. [3]
- Internet of Things (IoT): The network of physical devices, vehicles, and other items embedded with electronics, software, sensors, and network connectivity, enabling these objects to collect and exchange data. PyroGuard leverages IoT technologies to integrate with various sensors and devices for enhanced fire detection capabilities. [4]

- Graphical User Interface (GUI): The interface through which users interact with the PyroGuard system, designed to be user-friendly and intuitive, enabling quick and easy access to system functionalities, including live video feeds, system alerts, and historical data analysis. [5]

This detailed introduction to the Low-Level Design of the PyroGuard Fire Detection System establishes a solid foundation for the subsequent design and development phases, ensuring that the system is built on a well-documented, carefully considered, and standards-compliant basis.

2. Packages

2.1 System

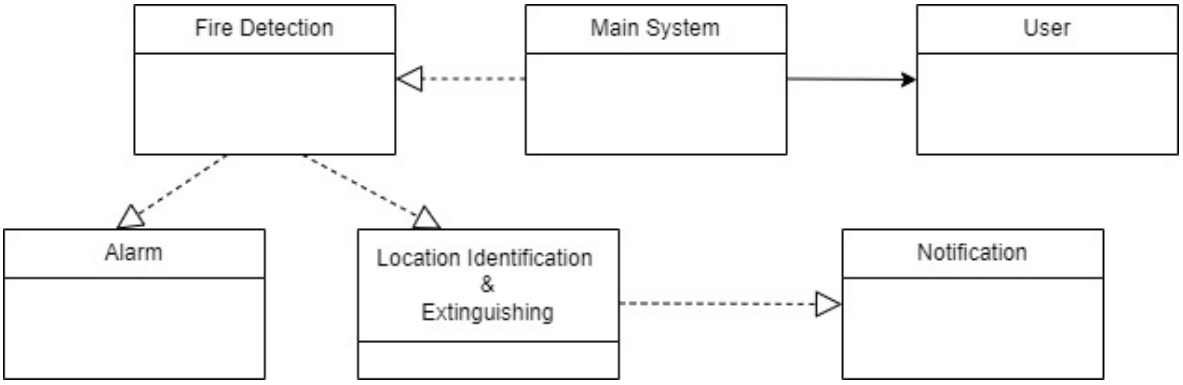


Figure 1 System Package

Main System: This class is responsible for controlling the entire process from frame analysis to the user interface. It coordinates all other classes and combines their feedback to deliver overall results.

Fire Detection: This class is the main class of our fire detection model. Camera connection is also provided from this class. After the frames received from the camera are processed in the model, data is distributed to other classes in the pipeline depending on whether there is a detection or not. Therefore, it contains *camera* and *deepLearningModel* attributes. Its functions are *detectFlames(), analyzeFootage()* and *activateAlarm().*

Alarm: This class allows the alarm to sound when the fire detection occurs. It includes the *activateAlarm()* function which enables the alarm to beep actively, and the *overrideAlarm(pin)* function which allows an outsider to silence the alarm with a pin.

User: This class allows the user to log in and out of the system in the simplest way. This job is provided by the handleLogin() function. It also provides telemetry of the required area. displayLocations(), locationAccess() provide telemetry.

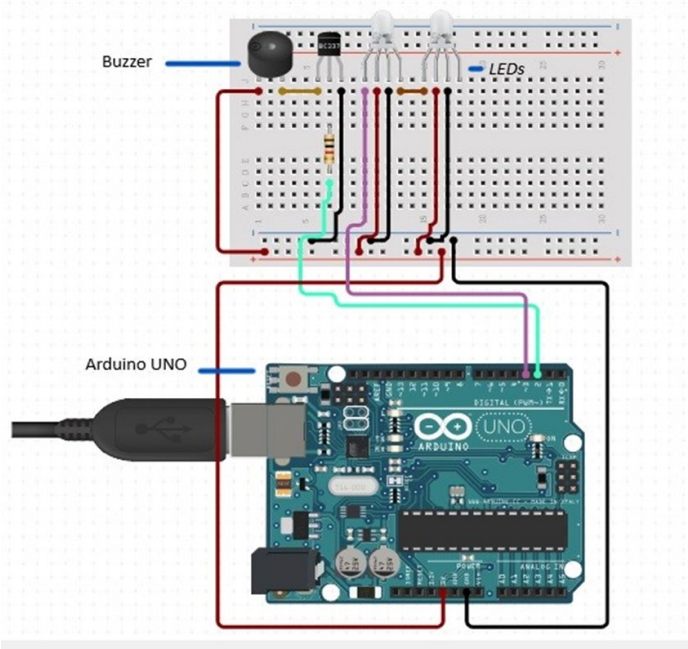
Location Identification & Extinguishing: This class allows extracting the necessary location information from the detected frame in case of fire detection. It contains the *determineLocation() function* along with the frame attribute.After the location information of the fire is obtained, via

Location Identification package, this class responsible for finding the fire extinguisher which is closest to the fire. It provides this with euclidean distance measurements. It includes *calculateDistance()* and *calculateEuclidean()* functions.

Notification: After all the necessary information about the fire (i.e. location, closest extinguisher, alarm) is obtained with other packages, it ensures that reliable and necessary information is collected and notified. It has extinguisherLocation and fireLocation attributes. This data is sent to the User via the *sendNotification()* function.

2.2 Alarm System

Arduino will be used as a prototype of the alarm unit in our project. It contains basically the same components of the real alarm devices. Python Serial library will be used for the communication between our program and the Arduino device. This library provides using Serial Communication Protocol.



Buzzer: It will provide the sound component of the alarm.

LEDs: It will provide the light component of the alarm

Arduino UNO: Basic microcontroller board

Bread Board: The part where all components in the software and alarm system are brought together.

3. Class Interfaces

3.1 User Interface

3.1.1 User

class User
Serves as the foundational structure for all user profiles within the PyroGuard system. It encompasses fundamental attributes such as password, user name and email information.
Attributes
username: holds the username of the user
password: holds the password of the user
email: holds the email address of the user
Methods
Getter and setter methods

3.1.2 Login Screen

class LoginScreen
Provides a user interface for verifying user credentials and managing accounts. UserDatabase object ensures secure storage and retrieval of user credentials. Getter methods enable seamless interaction with the UserDatabase instance and user data.
Attributes
user_database_object: database object to verify user credentials or enter data into the database system for registered user
user_object: holds user's credentials
Methods
displayLoginScreen(): displays the login screen to allow users to enter their credentials

getUserCredentials(): prompts the user to input their username and password for authentication
validateUserCredentials(): validates the entered username and password
displayRegistrationScreen(): displays the registration screen for new users to create an account
getUserRegistrationDetails(): prompts the user to input their desired username, password, and email for registration
createUserAccount(): creates a new user account with the provided credentials

3.1.3 User Database

class UserDatabase
Stores user information for authentication purposes in a software system.
Attributes
user_object: stores user information for authentication
Methods
Constructor(user_database: User): Initializes the UserDatabase and user_object.
addUser(): adds a new user to the user database
authenticateUser(): authenticates a user's credentials for login
updateUser(): updates a user's password or email
deleteUser(): deletes a user from the user database

3.1.4 Control Panel

class ControlPanel(LoginScreen)
Serves as a central hub for managing system settings, camera configurations, and user accounts within the software interface.
Attributes
camera__object: holds the camera object for camera settings
camera_count: holds the number of cameras connected to the system
ip_addresses: holds the IP addresses of the cameras
Methods
Getter and setter methods
displayControlPanel(): displays the control panel after successful login
showCameraSettings(): allows the user to adjust camera settings such as brightness and contrast
viewDetectionHistory(): allows the user to view past detections and events
manageUserAccount(): allows the user to manage their account settings, such as updating password or email
logOut(): logs the user out of the system and returns to the login screen
displayFeed(): displays the camera feed on the user interface after getting the ip camera number and ip addresses to user
showAlert(): shows an alert on the user interface
updateStatus(): updates the system status on the user interface

3.2 Camera

class Camera
Serves as an interface for managing camera operations and settings within the PyroGuard system. It provides functionalities to control the camera feed, extract frames, and adjust various camera settings, including resolution, exposure, brightness, and contrast.
Attributes
resolution: stores the resolution of the camera feed
frame: holds the stores the frame to be processed by the fire detection model
frame_rate: stores the frame rate of the camera feed
exposure: stores the exposure settings of the camera
brightness: stores the brightness level of the camera
contrast: stores the contrast level of the camera
Methods
startFeed(): initiates the camera feed to start capturing frames
stopFeed(): terminates the camera feed to stop capturing frames
extractFrames(): retrieves a single frame from the camera feed
adjustSettings(settings: CameraSettings): adjusts the camera settings

3.3 Fire Detection Interface

3.3.1 Fire Class

class Fire
Serves as the foundational structure for all user profiles within the PyroGuard system. It encompasses fundamental attributes such as password, user name and email information.
Attributes
active: a boolean flag indicating whether the fire detection is activated
location: stores the location information of the detected fire
frame_file: stores the file path of the frame containing the detected fire
Methods
Getter and setter methods

3.3.2 Fire Detection Class

class FireDetection(Camera)
Utilizes deep learning models to analyze camera frames and identify instances of fire.
Attributes
deep_learning_model: uploads the deep learning model used for fire detection
Methods
detectFire(frame: Frame): applies the specified deep learning model to detect and classify objects, including fire, in a given frame

3.3.3 Location Identification Class

class LocationIdentification
Provides functionality for identifying the location of fire incidents and nearby fire extinguishers within the PyroGuard system.
Attributes
camera_location: stores the location of the camera
fire_object: stores information about the detected fire object
fire_extinguish_location: stores the location of the nearest fire extinguisher
location_detected: a boolean flag indicating whether the location has been detected or not
Methods
detectFirePlacement(): detects the placement of the fire based on camera location and fire object information
detectFireExtinguisher(): detects the location of the nearest fire extinguisher based on camera location
calculateDistance(location1, location2): calculates the distance between two locations
checkActiveFire(): checks if there is an active fire detected in the vicinity

3.3.4 Database Management Class

class ImageDatabase
Manages the storage and organization of captured images within the PyroGuard system. It handles the saving of images from both the camera feed and detected fire instances, while also facilitating the deletion of old data based on user-defined retention periods.
Attributes
image_path: the path where images will be saved
fire_images_path: the path where images of detected fires will be saved

retention_period: the time period after which old data will be deleted, in days
fire_object: stores information about the detected fire object
camera_object: stores information about the camera
Methods
saveImage(image: Image): saves the captured image to the specified image_path
saveFireImage(image: Image): saves the image of the detected fire to the fire_images_path
deleteOldData(): deletes old data from the image_path based on the retention_period

3.4 Alarm Activation Interface

3.4.1 Alarm Class

class Alarm
Serves as a utility within the PyroGuard system for managing alarm functionality. This class enables setting the alarm status, determining the duration of alarm activation, and establishing connectivity with the Arduino device.
Attributes
active: a boolean flag indicating whether the alarm is active or not
duration: the duration for which the alarm will sound, in seconds
arduino: an object representing the connection to the Arduino device
security_pin: a security pin required to authenticate the connection to the Arduino device
Methods
isActive(): returns the current status of the alarm (active or inactive)
setActive(status: bool): sets the status of the alarm (active or inactive)

getDuration(): returns the duration for which the alarm will sound
setDuration(duration: int): sets the duration for which the alarm will sound
getArduino(): returns the object representing the connection to the Arduino device
setArduino(arduino_obj): sets the object representing the connection to the Arduino device
getSecurityPin(): returns the security pin required for authentication
setSecurityPin(pin: str): sets the security pin required for authentication
triggerArduino(duration): triggers the Arduino device to activate the alarm

3.5 Notification System Interface

3.5.1 Notification Class

class Notification
Serves as the foundational structure for all user profiles within the PyroGuard system. It encompasses fundamental attributes such as password, user name and email information.
Attributes
user: an object representing the user, containing user-related information including email
fire_location: stores information about the detected fire location
nearest_extinguisher_location: stores information about the nearest fire extinguisher location
location_identification: an object of the LocationIdentification class
Methods
sendEmail(): sends an email notification to the user's email address with information about the detected fire location and nearest extinguisher location

4. Glossary

- UI: User Interface - A graphical or textual interface that allows users to interact with software or a system.

- LED: Light Emitting Diode - A semiconductor device that emits light when an electric current passes through it.

- IEEE: Institute of Electrical and Electronics Engineers - An international professional organization for electrical, electronic, and information technologies.

- UML: Unified Modeling Language - A standardized graphical language used for modeling software systems.

- API: Application Programming Interface - A set of defined methods and tools for enabling communication between software applications.

- IoT: Internet of Things - A concept that refers to the interconnection of physical devices over the internet, enabling them to exchange data.

- DL: Deep Learning - A machine learning technique that involves training artificial neural networks on large datasets to learn representations of data.

- RBAC: Role-Based Access Control - An access control method that restricts system access based on users' roles and associated permissions.

- Arduino UNO: Arduino platform's most popular and widely used model. Used as a microcontroller board to interact with sensors, motors, and other electronic components.

- Serial Communication Protocol: A communication protocol that enables serial data communication between two devices. Communication between Arduino and a computer is typically done using this protocol.

5. References

[1] Deep Learning (DL):

- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016.
- LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." Nature 521.7553 (2015): 436-444.

[2] Role-Based Access Control (RBAC):

- Sandhu, Ravi S., et al. "Role-based access control models." Computer 29.2 (1996): 38-47.
- Ferraiolo, David F., et al. "Proposed NIST standard for role-based access control." ACM Transactions on Information and System Security (TISSEC) 4.3 (2001): 224-274.

[3] Application Programming Interface (API):

- Fielding, Roy Thomas. "Architectural Styles and the Design of Network-based Software Architectures." PhD diss., University of California, Irvine, 2000.
- Richardson, Leonard, and Sam Ruby. "RESTful web services." O'Reilly Media, Inc., 2008.

[4] Internet of Things (IoT):

- Ashton, Kevin. "That 'Internet of Things' Thing." RFID journal 22.7 (2009): 97-114.
- Atzori, Luigi, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A survey." Computer networks 54.15 (2010): 2787-2805.

[5] Graphical User Interface (GUI):

- Johnson, Jeff, and Jodie Moraru. "Designing with the mind in mind: Simple guide to understanding user interface design rules." Elsevier, 2013.
- Shneiderman, Ben. "Designing the user interface: Strategies for effective human-computer interaction." Pearson Education India, 2010.

ACM Code of Ethics:

- Association for Computing Machinery (ACM). "ACM Code of Ethics and Professional Conduct." Retrieved from: https://www.acm.org/code-of-ethics

UML Diagram Types and Examples:

- Creately. "UML Diagram Types and Examples." Retrieved from: https://creately.com/blog/diagrams/uml-diagram-types-examples/