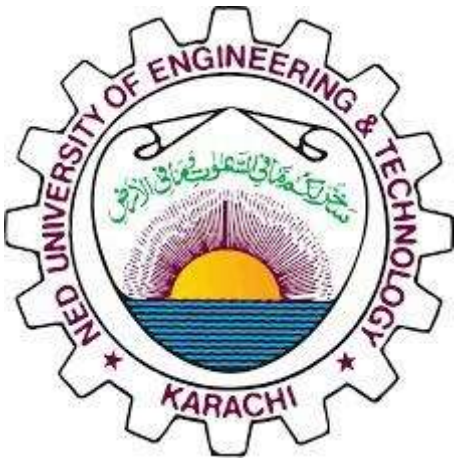


NED UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Formal Methods in Software Engineering

Formal Specification Document for

"HEV Radiator Controller System"

Group

- Bisma Shuja (SE-21024)
- Uzair Asif (SE-21032)

Table of Contents

HEV Radiator Controller System

1. Problem Statement: HEV Radiator Controller System

Background: A critical vehicle where a well-designed and effective radiator system is essential is a hybrid electric vehicle (HEV). In these vehicles, the thermal management system, which includes the radiator, is crucial for maintaining optimal operating temperatures for various components, especially the electric drivetrain components and the battery. To control the radiator system effectively, a software solution is required. The radiator system is equipped with a temperature sensor and a fan that can be activated or deactivated based on predefined temperature thresholds.

Requirements: Design and implement a software solution for controlling the vehicle radiator system. The system should be able to:

1. Set the temperature of the radiator system within valid bounds.
2. Activate the radiator fan when the temperature exceeds a specified maximum threshold.
3. Deactivate the radiator fan when the temperature goes below a specified minimum threshold.
4. Provide a mechanism to query the current status of the fan.

Specifications:

- The radiator system has a temperature sensor providing real-time temperature values.
- The fan should activate when the temperature exceeds a maximum limit of 80.0 degrees Celsius.
- The fan should deactivate when the temperature drops below a minimum limit of 20.0 degrees Celsius.
- Ensure that the temperature is within the valid bounds of 20.0 to 80.0 degrees Celsius.
- The system should prevent activating the fan if the temperature is below the minimum threshold and prevent deactivating the fan if the temperature is above the maximum threshold.
- Implement an enumeration (**Signal**) to represent the possible states of the fan (e.g., HIGH and LOW).

Functionalities:

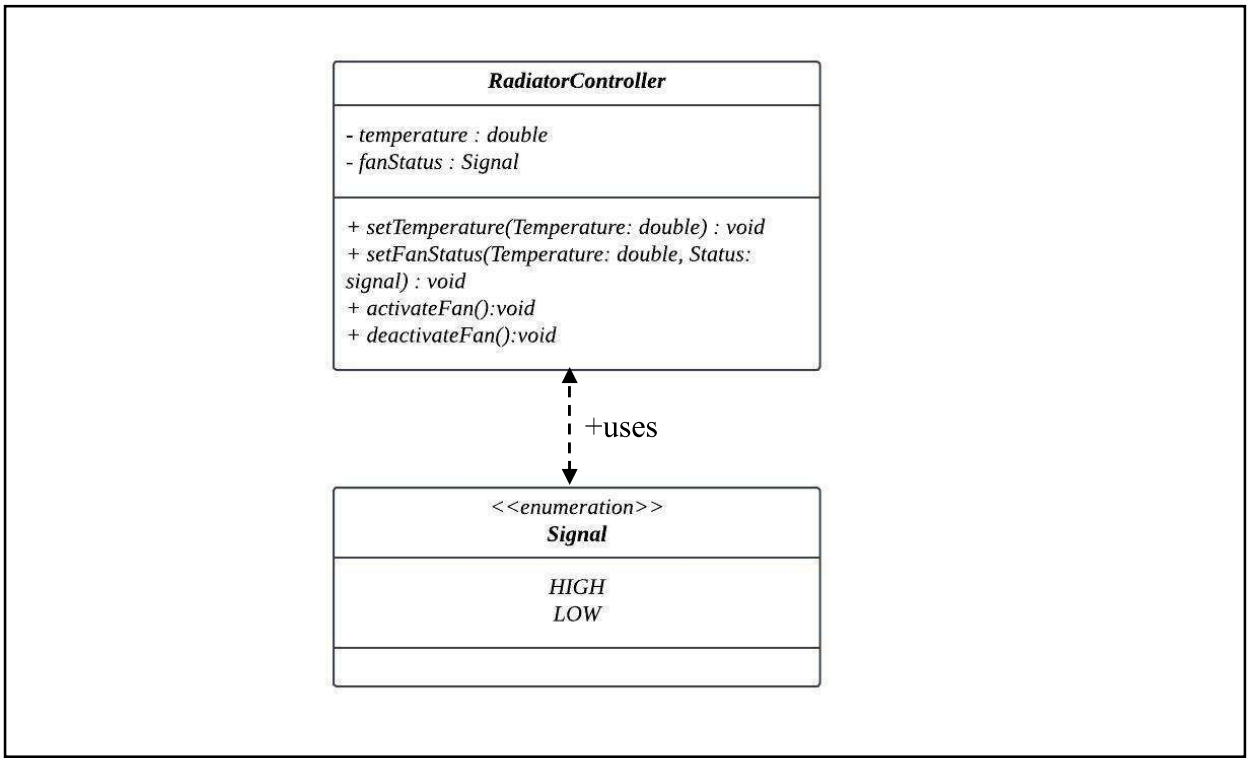
- **Set Temperature:**
    - Input: Temperature value.
    - Preconditions: The input temperature should be within the valid bounds.
    - Post conditions: The system temperature is updated, and the fan status is adjusted accordingly.
- **Set Fan Status:**
    - Preconditions: The current temperature should be within the valid bounds and not be null.
    - Post conditions: The fan status is set according to the current temperature.
- **Activate Fan:**
    - Preconditions: The current temperature is above the maximum threshold.
    - Post conditions: The fan is activated.
- **Deactivate Fan:**
    - Preconditions: The current temperature is below the minimum threshold.
    - Post conditions: The fan is deactivated.

Constraints:

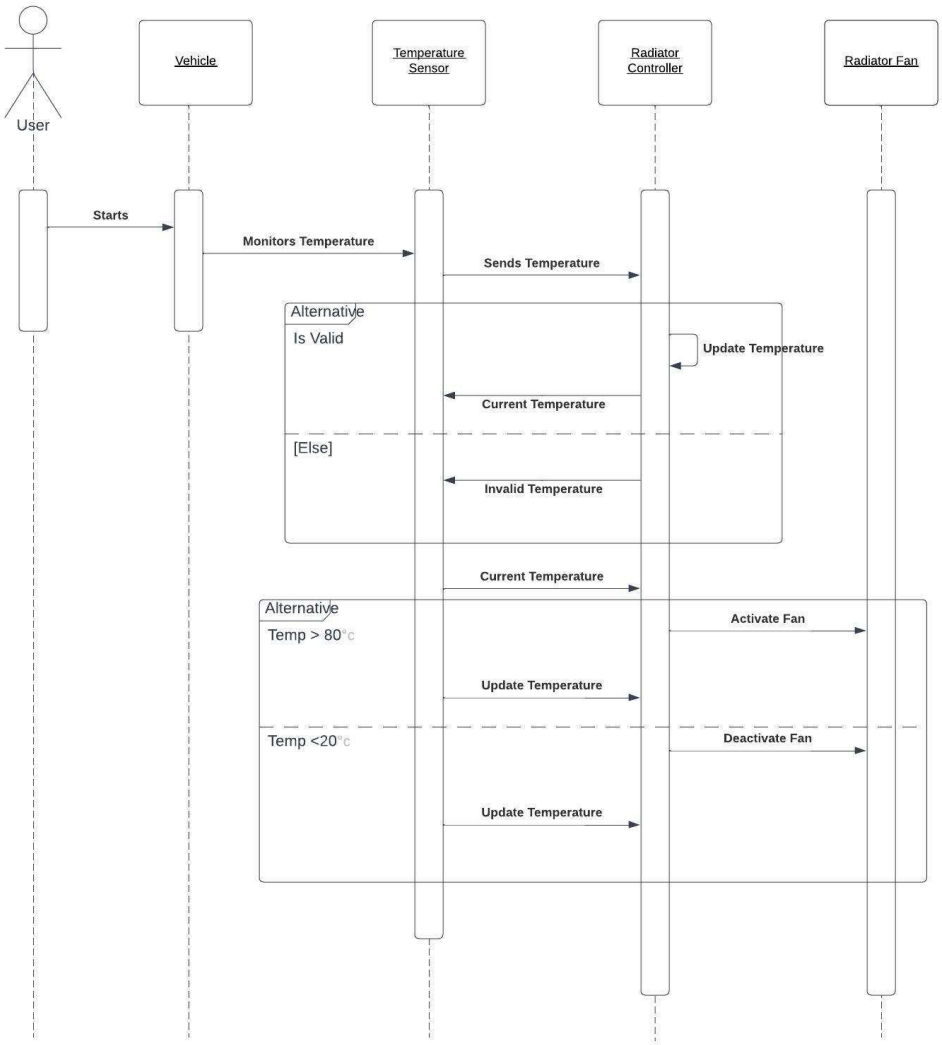
- Ensure the software adheres to the specified temperature thresholds and follows the pre and post-conditions for each operation.
- Implement proper error handling for invalid inputs and ensure the system maintains its invariants.

2. 4 + 1 Architectural View

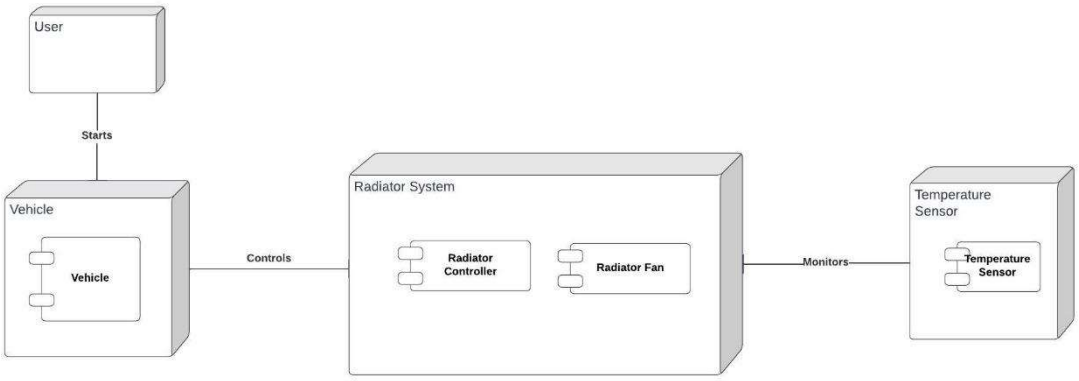
2.1. Logical View



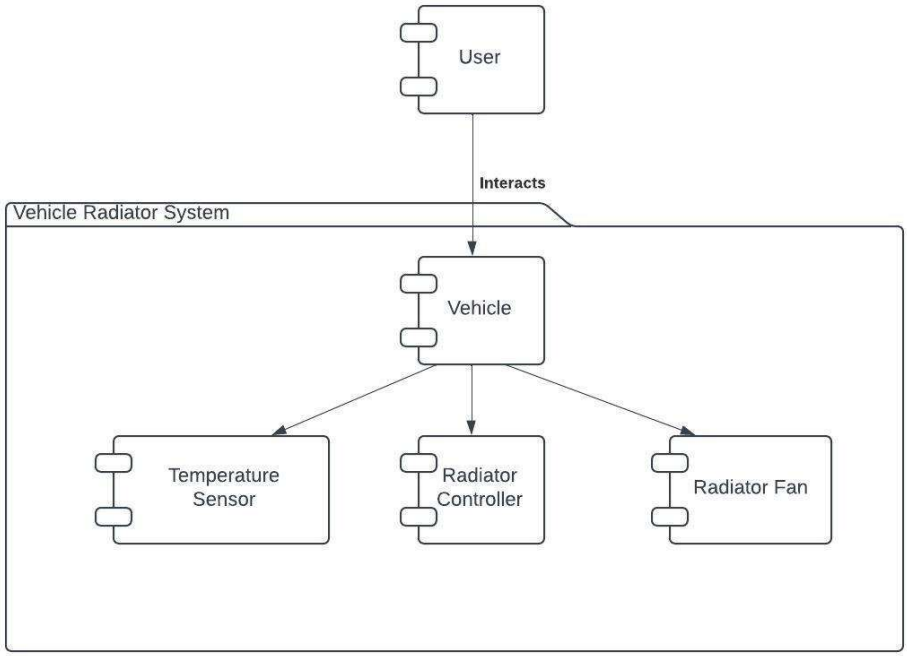
2.2. Process View



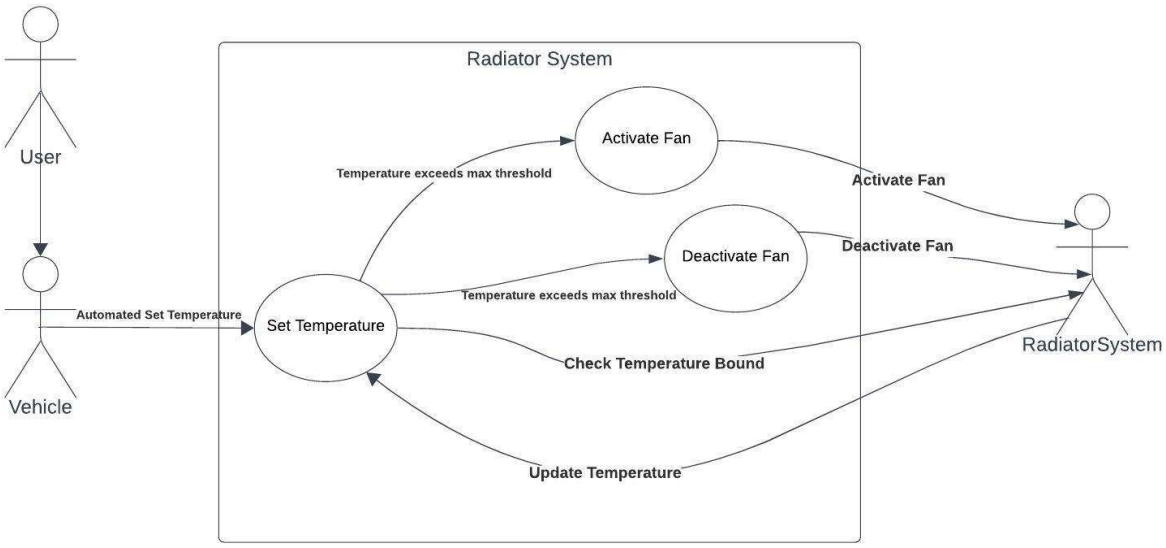
2.3. Physical View



2.4. Development View



2.5. +1 Scenarios



3. The complete specification of *RadiatorController* System

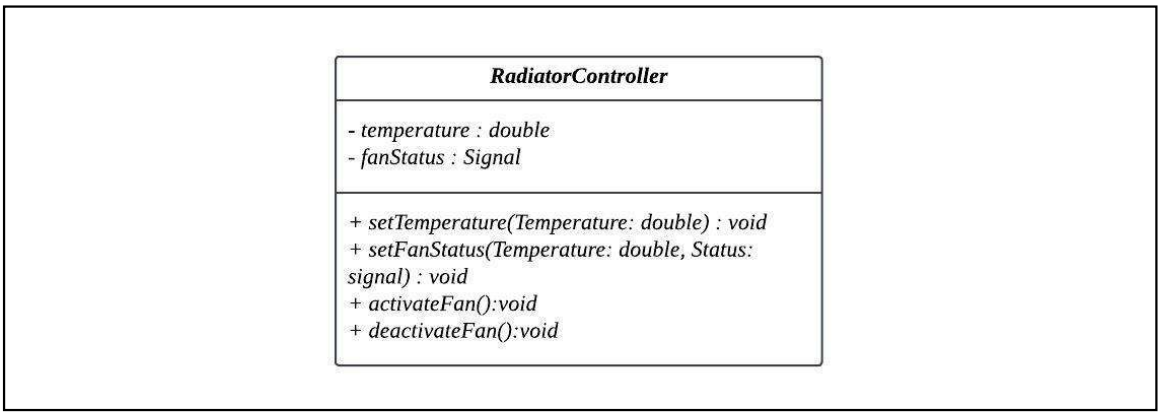


Figure 1 The specification of the *RadiatorController*

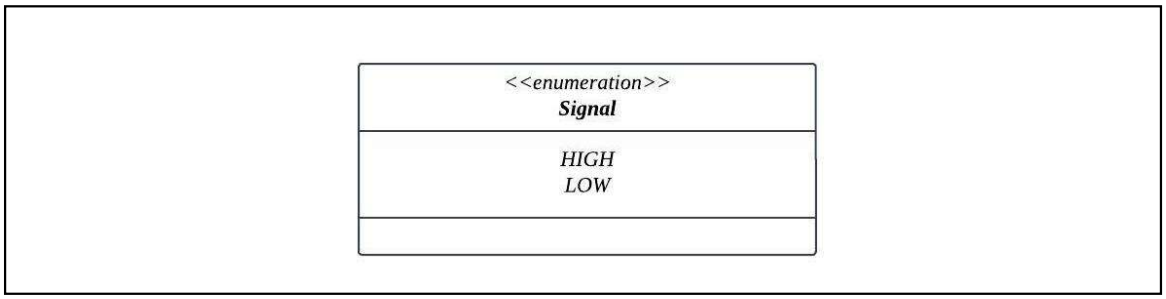


Figure 2 UML specification of the *Signal* type

```
types
  Temperature = ℝ;
  Signal = <HIGH> | <LOW>

values
  MIN_TEMPERATURE: Temperature = 20.0;
  MAX_TEMPERATURE: Temperature = 80.0;

state RadiatorController of
  temperature : [Temperature]
  fanStatus : [Signal]

  -- Temperature and fan status must be in range or equal to nil
  inv mk-RadiatorController (t, f) (inRange(t) ∨ = nil) ∧ (inRange(f) ∨ = nil)

  -- Temperature and fan status are undefined when the system is initialized
  init mk-RadiatorController (t, f) t = nil ∧ f = nil
end
```

functions
inRange:(val: Temperature) result: B
pre True
post result ⇔ MIN_TEMPERATURE ≤ val ≤ MAX_TEMPERATURE;

operations
– –an operation that sets the temperature of the system
setTemperature: (temp : Temperature)
ext wr temperature:[Temperature]
pre inRange(temp) ∧ temperature = **nil**
post temperature = temp ∧ setFanStatus (temp, fanStatus);

– – an operation that records the temperature and signals the fan to high or low
setFanStatus (temp:Temperature , status : Signal)
ext rd temperature:[Temperature]
wr fanStatus : [Signal]
pre inRange(temp) ∧ temperature ≠ **nil**
post (temp ≥ MAX_TEMPERATURE ∧ activateFan()) ∨ (temp ≤ MIN_TEMPERATURE ∧ deactiavteFan())

– –an operation that changes the status of fan from low to high
activateFan()
ext wr fanStatus : [Signal]
pre fanStaus = <LOW>
post fanStatus = <HIGH>

– –an operation that changes the status of fan from high to low
deactivateFan()
ext wr fanStatus : [Signal]
pre fanStaus = <HIGH>
post fanStatus = <LOW>

4. Java Implementation

Code:

```
// Enum for Signal
enum Signal {
    HIGH,
    LOW
}

// Class representing the state of RadiatorController
class RadiatorController {
    private Double temperature;
    private Signal fanStatus;

// Constructor
    public RadiatorController() {
        this.temperature = null;
        this.fanStatus = null;
    }

    // Invariant
    public boolean inRange(Double val) {
        return val != null && (val >= 20.0 && val <= 80.0);
    }

    // Initialization
    public RadiatorController(Double temp, Signal status) {
        assert inRange(temp) : "Temperature should be in the range or equal to
nil";
        assert status != null : "Fan status should be in the range or equal to
nil";
        this.temperature = temp;
        this.fanStatus = status;
    }

    // Function to set temperature
    public void setTemperature(Double temp) {
        assert inRange(temp) : "Temperature should be in the range or equal to
nil";
        assert temperature == null : "Temperature is already set";
        this.temperature = temp;
        setFanStatus(temp, fanStatus);
    }

    // Function to set fan status
```

```
    private void setFanStatus(Double temp, Signal status) {
        assert inRange(temp) : "Temperature should be in the range or equal to
nil";
        assert temperature != null : "Temperature should be set before setting
fan status";

        if (temp >= 80.0) {
            activateFan();
        } else if (temp <= 20.0) {
            deactivateFan();
        }
    }

    // Function to activate fan
    private void activateFan() {
        assert fanStatus == Signal.LOW : "Fan is already activated";
        fanStatus = Signal.HIGH;
    }

    // Function to deactivate fan
    private void deactivateFan() {
        assert fanStatus == Signal.HIGH : "Fan is already deactivated";
        fanStatus = Signal.LOW;
    }

    // Getter for temperature
    public Double getTemperature() {
        return temperature;
    }

    // Getter for fan status
    public Signal getFanStatus() {
        return fanStatus;
    }
}

public class Main {
    public static void main(String[] args) {
        RadiatorController controller = new RadiatorController();

        // Example: Set temperature and print status
        Double temp = 100.0;
        controller.setTemperature(temp);
        printStatus(controller);
    }
```

```
    private static void printStatus(RadiatorController controller) {
        System.out.println("Temperature: " + controller.getTemperature());

        if (controller.getTemperature() != null) {
            if (controller.getTemperature() >= 20.0 &&
controller.getTemperature() <= 80.0) {
                System.out.println("Fan Status: Normal");
            } else {
                System.out.println("Fan Status: " + controller.getFanStatus());
            }
        } else {
            System.out.println("Fan Status: Not available");
        }

        System.out.println();
    }
}
```

Output:

5. Testing Class

Code:

```
import java.util.Scanner;

class RadiatorControllerTest {
    public static void main(String[] args) {
        RadiatorController controller = new RadiatorController();
        Scanner scanner = new Scanner(System.in);

        int choice;
        do {
            System.out.println("");
            System.out.println("=== Radiator Controller Menu ===");
            System.out.println("1. Set Temperature");
            System.out.println("2. Print Status");
            System.out.println("0. Exit");
            System.out.print("Enter your choice: ");

            choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    setTemperature(controller, scanner);
                    break;
                case 2:
                    printStatus(controller);
                    break;
                case 0:
                    System.out.println("Exiting the program. Goodbye!");
                    break;
                default:
                    System.out.println("Invalid choice. Please enter a valid
option.");
                    break;
            }

        } while (choice != 0);

        scanner.close();
    }
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

User@DESKTOP-4PIM1JR MINGW64 /d/PROJECTS/A-05 FMS VDM (main)
$  /usr/bin/env C:\\Program\ Files\\Microsoft\\jdk-17.0.8.7-hotspot\\bin\\java.exe -XX:+ShowCodeDetailsInExcept
ionMessages -cp C:\\Users\\User\\AppData\\Roaming\\Code\\User\\workspaceStorage\\e4f4c961b16e54b8561e8b91623ac5
3d\\redhat.java\\jdt_ws\\A-05\ FMS\ VDM_c0e13638\\bin RadiatorControllerTest

=== Radiator Controller Menu ===
1. Set Temperature
2. Print Status
0. Exit
Enter your choice: 1

Enter temperature: 15

=== Radiator Controller Menu ===
1. Set Temperature
2. Print Status
0. Exit
Enter your choice: 2

=== Radiator Controller Status ===
Temperature: 15.0
Fan Status: LOW


=== Radiator Controller Menu ===
1. Set Temperature
2. Print Status
0. Exit
Enter your choice: 1

Enter temperature: 67

=== Radiator Controller Menu ===
1. Set Temperature
2. Print Status
0. Exit
Enter your choice: 2

=== Radiator Controller Status ===
Temperature: 67.0
Fan Status: Normal


=== Radiator Controller Menu ===
1. Set Temperature
2. Print Status
0. Exit
Enter your choice: 1

Enter temperature: 160

=== Radiator Controller Menu ===
1. Set Temperature
2. Print Status
0. Exit
Enter your choice: 2

=== Radiator Controller Status ===
Temperature: 160.0
Fan Status: HIGH

=== Radiator Controller Menu ===
1. Set Temperature
2. Print Status
0. Exit
Enter your choice: 5
Invalid choice. Please enter a valid option.
```

```
=== Radiator Controller Menu ===
1. Set Temperature
2. Print Status
0. Exit
Enter your choice: 0
Exiting the program. Goodbye!
```