Decoupling UI Logic in Embedded Systems: Technical Design of R-MVP-based Thermostat Software

Alexander Menzel
Department of Electrical Engineering
Fulda University of Applied Sciences
Fulda, Germany
alexander.menzel@et.hs-fulda.de

***Abstract*—Text...**

***Index Terms*—keyword1, keyword2, keyword3, keyword4, keyword5**

I. Introduction

Heating private living spaces is one of the most significant sources of CO_2 emissions. In Germany, a substantial portion of annual greenhouse gas emissions originates from this sector [1] [2]. While intelligent heating control and smart home systems offer an average energy saving potential of between 8 and 19% [3], the market is currently dominated by proprietary solutions. Consequently, there is a lack of open-domain projects that can serve as a foundation for research and development of smart heating controllers.

This paper presents the software development for a radiator thermostat prototype, realized as part of the interdisciplinary "MiraTherm Radiator Thermostat" project at Fulda University of Applied Sciences. The overarching project aims to create a complete device, encompassing mechanics, electronics, and control algorithms.

The primary objective of this work is to establish a solid software foundation for the thermostat's microcontroller-based hardware. While the long-term vision includes control algorithms and wireless connectivity (e.g., Matter-over-Thread), the central contribution of this paper lies in the architectural design of the application and its User Interface (UI). Specifically, we propose the application of the Routed-Model-View-Presenter (R-MVP) design pattern in order to decouple UI logic from hardware drivers and core system logic. This approach is used to implement basic consumer functions considering constraints of an embedded system and of the C programming language.

II. Background

This section provides the necessary context for this work. First, the domain of smart radiator thermostats is introduced. Then, the Model-View-Presenter (MVP) design pattern, which forms the architectural basis for the software, is described.

A. Smart Radiator Thermostats

A radiator thermostat is a device mounted on a heating radiator valve to control the flow of hot water, thereby regulating the room temperature. Traditional thermostatic radiator valves (TRVs) use a wax or liquid-filled capsule that expands and contracts with temperature changes, mechanically adjusting valve position. Digital radiator thermostats replace this mechanism with an electronically controlled system, typically consisting of a microcontroller unit (MCU), a DC motor with a gearbox for valve actuation, a user interface (buttons or rotary encoder, display), and a motor driver circuit.

Smart radiator thermostats extend the digital concept with wireless connectivity, such as Wi-Fi, Bluetooth Low Energy (BLE), or emerging standards like Matter-over-Thread, enabling integration into smart home ecosystems and remote control via mobile applications. Common consumer functions of such devices include manual temperature adjustment, mode selection (e.g., auto, manual, boost, vacation), weekly schedule programming, an open-window detection function, and a periodic valve decalcification program [**?**]. They are most commonly battery-powered and are expected to operate for extended periods, typically around two years on two AA batteries [**?**].

The market for smart radiator thermostats is currently dominated by proprietary, closed-source solutions from manufacturers such as eQ-3, tado, and Bosch. While these products offer mature consumer features, their proprietary nature limits their utility as a foundation for academic research and open development of smart heating control strategies. The MiraTherm project addresses this gap by developing an open-domain radiator thermostat prototype. The software presented in this paper uses the eQ-3 eqiva Bluetooth Smart Radiator Thermostat (CC-RT-M-BLE-EQ) [**?**] as a functional reference for defining its scope.

B. MVP Pattern

The Model-View-Presenter (MVP) pattern is a derivative of the classic Model-View-Controller (MVC) pattern originating from Smalltalk-80 [**?**]. MVP was formalized by Potel at Taligent, Inc. as a generalization of MVC that decomposes

an application's design into two fundamental domains: *Data Management* and *User Interface* [**?**].

In the Data Management domain, Potel identifies three subquestions that a developer must address:

1) *What is my data?* — answered by the **Model**, which encapsulates the application's data and its access methods.
2) *How do I specify my data?* — answered by **Selections**, which represent abstractions for identifying subsets of the model's data.
3) *How do I change my data?* — answered by **Commands**, which represent operations that can be performed on selections.

In the User Interface domain, three further questions are posed:

4) *How do I display my data?* — answered by the **View**, which renders a representation of the model's data. Views need not be graphical.
5) *How do events map into changes in my data?* — answered by **Interactors**, which handle user-initiated actions such as mouse clicks, keystrokes, or physical input like turning a dial.
6) *How do I put it all together?* — answered by the **Presenter**, which coordinates all other elements by providing the business logic that maps user gestures onto the appropriate commands for manipulating the model.

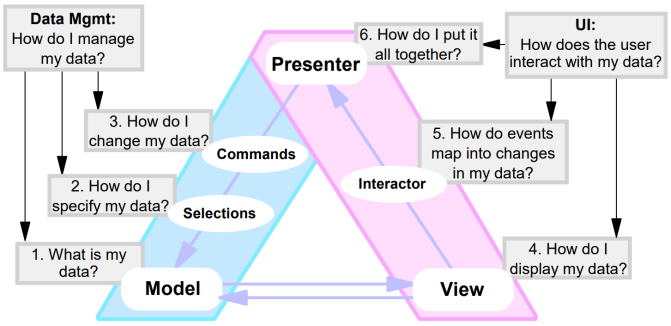

Fig. 1. The six design questions of the MVP programming model and their corresponding abstractions, after Potel [**?**].

A key insight from Potel's work is that these abstractions need not all be employed at once. In the "Building an Application" section of [**?**], Potel demonstrates using a calculator example that a developer can start with only a subset of the MVP elements—such as the Model, View, and Presenter—and incrementally introduce Selections, Commands, and Interactors as the application evolves and requires their respective benefits, such as undo/redo capability, scriptability, or advanced input handling. This incremental adoption is particularly relevant for the embedded context of this work, where the constrained environment and the C programming language favor a minimal yet well-structured architecture. Accordingly, the design presented in this paper deliberately omits the Command, Selection, and Interactor abstractions, retaining only the core Model, View, and Presenter triad as the foundation of its UI architecture.

The primary benefit of the MVP pattern that motivates its use in this project is the clean separation between the Model and the View-Presenter (Presentation). This separation enables independent development and testing of UI logic and data management, facilitates reuse of models with different presentations, and allows the underlying data structures to be modified without affecting the presentation code [**?**].

III. Requirements

Text...

IV. Technical Design

Text...

A. Overall Architecture

Text...

B. R-MVP Pattern

Text...

V. Implementation

Text...

VI. Verification and Results

Text...

A. Test Environment

Text...

B. Results

Text...

VII. Conclusion

Text...

References

[1] Statistisches Bundesamt (Deutschland). (2025, 07) CO2-Emissionen beim Heizen binnen 20 Jahren um 12% gesunken. [Online]. Available: https://www.destatis.de/DE/Presse/Pressemitteilungen/Zahl-der-Woche/2024/PD24_05_p002.html

[2] Umweltbundesamt (Deutschland). (2022, 03) Treibhausgasemissionen stiegen 2021 um 4,5%. [Online]. Available: https://www.umweltbundesamt.de/presse/pressemitteilungen/treibhausgasemissionen-stiegen-2021-um-45-prozent

[3] M. Kersken, H. Sinnesbichler, and H. Erhorn, "Analyse der Einsparpotenziale durch Smarthome– und intelligente Heizungsregelungen," *Bauphysik*, vol. 40, no. 5, pp. 276–285, 2018.