Dokumentacja - Winda

Filip Jędrzejewski

2 Czerwca 2024

1 Opis modelowanego systemu

Winda to system transportu pionowego, który głównie umożliwia przemieszczanie się osób między różnymi poziomami budynku. Działanie windy opiera się na kilku kluczowych elementach:

1. **Panel przycisków:** Pasażerowie wybierają piętro docelowe, naciskając odpowiedni przycisk na panelu (w windzie lub poza nią - przyciski na poszczególnych piętrach).
2. **Sterowanie:** Sygnały z przycisków są przekazywane do sterownika windy, który analizuje żądania i podejmuje decyzje dotyczące ruchu kabiny.
3. **Silnik:** Silnik elektryczny porusza kabiną windy w odpowiednim kierunku.
4. **Kabina:** Kabina windy porusza się w szybie windowym przewożąc pasażerów.
5. **Drzwi:** Drzwi kabiny i drzwi zewnętrzne (piętra) otwierają się i zamykają automatycznie, umożliwiając bezpieczne wejście i wyjście pasażerów.
6. **Wyświetlacz:** Wyświetlacz informuje pasażerów o aktualnym położeniu kabiny.

Cykl działania przykładowej windy:

1. Pasażer naciska przycisk, wskazując piętro docelowe.
2. Sterownik windy odbiera sygnał i analizuje go, uwzględniając aktualne położenie kabiny i inne żądania.
3. Sterownik wysyła polecenie do silnika, aby rozpocząć ruch w odpowiednim kierunku.
4. Silnik napędza kabinę poruszającą się w szybie windowym.
5. Gdy kabina osiągnie żądane piętro, sterownik wysyła polecenie do silnika, aby zatrzymać windę.

6. Sterownik wysyła polecenie do drzwi kabiny i drzwi piętra, aby się otworzyły.

7. Pasażerowie wchodzą lub wychodzą z windy.

8. Po określonym czasie lub po naciśnięciu przycisku zamknięcia drzwi, sterownik wysyła polecenie zamknięcia drzwi.

9. Po zamknięciu drzwi, winda jest gotowa do przyjęcia kolejnego żądania (cykl się powtarza).

2 Spis komponentów

2.1 Komponenty typu `data`

2.1.1 ElevatorState

Kod:

```
data ElevatorState
    properties
        Data_Model::Enumerators => ("Idle", "MovingUp", "MovingDown",
                                    "DoorOpening", "DoorClosing");
        Data_Model::Data_Representation => Integer;
        Data_Size => 4 Bytes;
end ElevatorState;

data implementation ElevatorState.impl
end ElevatorState.impl;
```

`ElevatorState` to enumerator, który reprezentuje możliwe stany windy. Komponent ten jest używany do śledzenia aktualnego stanu windy w systemie. W podanym modelu zdefiniowano pięć stanów:

- **Idle:** winda jest bezczynna, nie porusza się i ma zamknięte drzwi.
- **MovingUp:** Winda porusza się w górę.
- **MovingDown:** Winda porusza się w dół.
- **DoorOpening:** Drzwi windy są otwierane.
- **DoorClosing:** Drzwi windy są zamykane.

2.1.2 ElevatorAction

Kod:

```
data ElevatorAction
end ElevatorAction;

data implementation ElevatorAction.impl
    subcomponents
        targetFloor: data FloorNumber;
        direction: data MotorCommand;
end ElevatorAction.impl;
```

`ElevatorAction` jest typem danych służącym do przekazywania informacji o żądanej akcji windy. W implementacji `ElevatorAction.impl`, widzimy, że składa się z dwóch subkomponentów:

- **targetFloor:** Określa piętro docelowe, na które winda ma się udać.
- **direction:** Określa kierunek ruchu windy

2.1.3 FloorNumber

Kod:

```
data FloorNumber
    properties
        Data_Model::Data_Representation => Integer;
        Data_Size => 4 Bytes;
end FloorNumber;

data implementation FloorNumber.impl
end FloorNumber.impl;
```

`FloorNumber` to typ danych reprezentujący numer piętra w budynku. W kontekście windy, `FloorNumber` jest używany do określania piętra docelowego, aktualnego piętra windy oraz do wyświetlania informacji o piętrze na panelu sterowania i wyświetlaczu w kabinie.

2.1.4 ButtonType

Kod:

```
data ButtonType
    properties
        Data_Model::Enumerators => ("CallUp", "CallDown", "Cabin");
        Data_Model::Data_Representation => Integer;
        Data_Size => 4 Bytes;
end ButtonType;

data implementation ButtonType.impl
end ButtonType.impl;
```

`ButtonType` to enumerator definiujący rodzaje przycisków obecnych w systemie windy. Ten typ danych jest używany w połączeniu z typem `FloorNumber` w strukturze `ButtonPress`, aby jednoznacznie określić, który przycisk został naciśnięty i na którym piętrze. Zawiera trzy wartości:

- **CallUp:** Przycisk wywołania windy na wyższe piętro, umieszczony na zewnątrz kabiny.
- **CallDown:** Przycisk wywołania windy na niższe piętro, umieszczony na zewnątrz kabiny.
- **Floor:** Przycisk wyboru konkretnego piętra, znajdujący się wewnątrz kabiny windy.

2.1.5 ButtonPress

Kod:

```
data ButtonPress
    properties
        Data_Size => 8 Bytes;
end ButtonPress;

data implementation ButtonPress.impl
    subcomponents
        floor: data FloorNumber;
        button_type: data ButtonType;
end ButtonPress.impl;
```

`ButtonPress` to struktura reprezentująca zdarzenie naciśnięcia przycisku w windzie. Zawiera ona dwa elementy:

- **floor:** Przechowuje numer piętra, na którym przycisk został naciśnięty (lub na jakie ma się udać).
- **button_type:** Określa rodzaj naciśniętego przycisku (góra, dół lub piętro).

Te informacje są kluczowe dla systemu sterowania windą, ponieważ umożliwiają mu określenie, na które piętro winda powinna się udać i w jakim kierunku.

2.1.6 MotorCommand

Kod:

```
data MotorCommand
    properties
        Data_Model::Enumerators => ("Up", "Down", "Stop");
        Data_Model::Data_Representation => Integer;
        Data_Size => 4 Bytes;
```

```
end MotorCommand;

data implementation MotorCommand.impl
end MotorCommand.impl;
```

`MotorCommand` to enumerator definiujący możliwe polecenia sterujące dla silnika windy. Kontroler silnika interpretuje to polecenie i steruje silnikiem windy zgodnie z nim. Zawiera trzy wartości:

- **Up:** Polecenie ruchu windy w górę.
- **Down:** Polecenie ruchu windy w dół.
- **Stop:** Polecenie zatrzymania windy

2.1.7 DoorCommand

Kod:

```
data DoorCommand
    properties
        Data_Model::Enumerators => ("Open", "Close");
        Data_Model::Data_Representation => Integer;
        Data_Size => 4 Bytes;
end DoorCommand;

data implementation DoorCommand.impl
end DoorCommand.impl;
```

`DoorCommand` to enumerator określający możliwe polecenia dla drzwi windy. Zawiera dwie wartości:

- **Open:** Polecenie otwarcia drzwi windy.
- **Close:** Polecenie zamknięcia drzwi windy.

2.1.8 DoorState

Kod:

```
data DoorState
    properties
        Data_Model::Enumerators => ("Opened", "Closed", "Opening", "Closing"
        Data_Model::Data_Representation => Integer;
        Data_Size => 4 Bytes;
end DoorState;

data implementation DoorState.impl
end DoorState.impl;
```

`DoorState` to enumerator reprezentujący możliwe stany drzwi windy. Zawiera on cztery wartości:

- **Opened:** Drzwi są całkowicie otwarte.
- **Closed:** Drzwi są całkowicie zamknięte.
- **Opening:** Drzwi są w trakcie otwierania.
- **Closing:** Drzwi są w trakcie zamykania.

2.2 Komponenty typu `thread`

2.2.1 ButtonPanelThread

Kod:

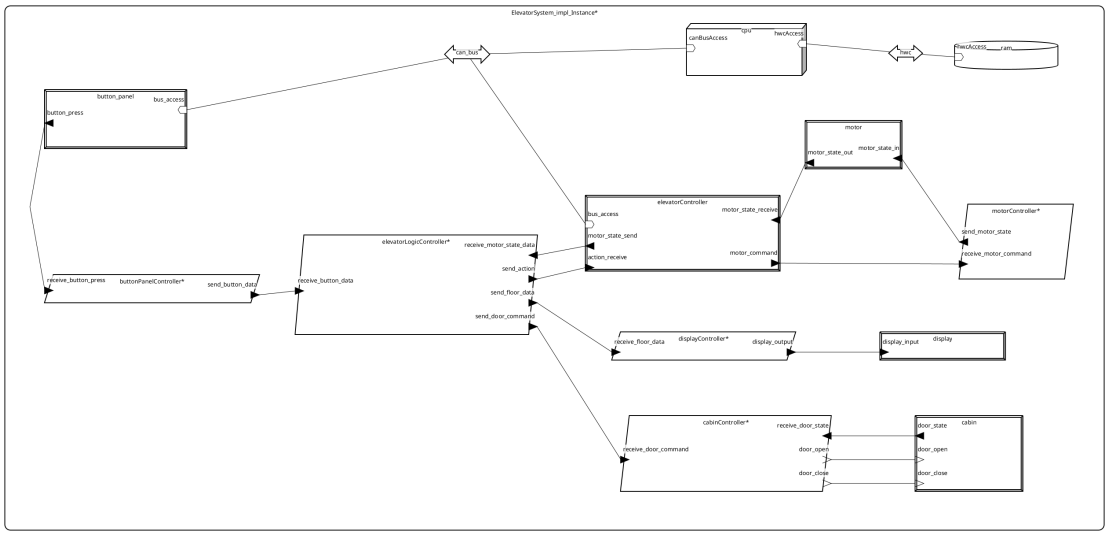
```
thread ButtonPanelThread
    features
        button_press_in: in event data port ButtonPress;
        button_data_out: out data port ButtonPress;
    properties
        SEI::MIPSBudget => 15.0 mips;
end ButtonPanelThread;

thread implementation ButtonPanelThread.impl
end ButtonPanelThread.impl;
```

`ButtonPanelThread` jest wątkiem odpowiedzialnym za obsługę panelu przycisków w windzie. Jego głównym zadaniem jest odczytywanie sygnałów z przycisków i przekazywanie informacji o naciśnięciach do `ButtonPanelController`. Wątek `ButtonPanelThread` jest kluczowym elementem systemu windy, ponieważ umożliwia pasażerom komunikowanie swoich żądań do systemu.

2.3 Schemat rozwiązania

W celu rozwiązania zadania stworzono następujący diagram `Orange`:


ElevatorSystem_impl_Instance*
can_bus
canBusAccess
cpu
hwcAccess
ram
hwcAccess_____ram
button_panel
bus_access
button_press
motor
motor_state_out
motor_state_in
elevatorController
bus_access
motor_state_receive
motor_state_send
action_receive
motor_command
motorController*
send_motor_state
receive_motor_command
elevatorLogicController*
receive_motor_state_data
send_action
send_floor_data
send_door_command
receive_button_data
receive_button_press
buttonPanelController*
send_button_data
receive_floor_data
displayController*
display_output
display_input
display
cabinController*
receive_door_state
receive_door_command
door_open
door_close
door_state
cabin
door_open
door_close