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 => 0.8 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.

2.2.2 ElevatorLogicThread

Kod:

```
thread ElevatorLogicThread
    features
        button_data_in: in data port ButtonPress;
        action_data_out: out data port ElevatorAction;
    properties
        SEI::MIPSBudget => 0.8 mips;
end ElevatorLogicThread;

thread implementation ElevatorLogicThread.impl
end ElevatorLogicThread.impl;
```

`ElevatorLogicThread` jest wątkiem odpowiedzialnym za podejmowanie podstawowych decyzji dotyczących działania windy. Jego głównym zadaniem jest analiza żądań pasażerów (`ButtonPress`) i generowanie akcji (`ElevatorAction`) w odpowiedzi na te żądania. Wątek ten bierze pod uwagę aktualny stan windy (np. piętro, kierunek ruchu) oraz informacje o naciśniętych przyciskach, aby zdecydować, na które piętro winda powinna się udać.

2.2.3 ElevatorLogicTransformThread

Kod:

```
thread ElevatorLogicTransformThread
    features
        action_data_in: in data port ElevatorAction;
        motor_state_in: in data port ElevatorState;
        motor_state_out: out data port MotorCommand;
        floor_data_out: out data port FloorNumber;
        door_command_out: out data port DoorCommand;
    properties
        SEI::MIPSBudget => 0.8 mips;
end ElevatorLogicTransformThread;

thread implementation ElevatorLogicTransformThread.impl
end ElevatorLogicTransformThread.impl;
```

`ElevatorLogicTransformThread` jest wątkiem odpowiedzialnym za przekształcanie akcji windy (`ElevatorAction`) na konkretne polecenia sterujące dla silnika (`MotorCommand`) i drzwi (`DoorCommand`). Dodatkowo, wątek ten wysyła informację do kontrolera wyświetlacza (`DisplayController`).

2.2.4 MotorControlThread

Kod:

```
thread MotorControlThread
    features
        motor_state_in: in data port MotorCommand;
        motor_state_out: out data port ElevatorState;
        motor_command_out: out data port MotorCommand;
    properties
        SEI::MIPSBudget => 0.8 mips;
end MotorControlThread;

thread implementation MotorControlThread.impl
end MotorControlThread.impl;
```

`MotorControlThread` jest wątkiem odpowiedzialnym za bezpośrednie sterowanie silnikiem windy. Odbiera on polecenia od `MotorController`, które określają, czy silnik ma się poruszać w górę, w dół, czy też zatrzymać. Na podstawie tych poleceń, wątek generuje odpowiednie sygnały sterujące dla silnika i monitoruje jego stan.

2.2.5 DisplayThread

Kod:

```
thread DisplayThread
    features
        floor_data_in: in data port FloorNumber;
        display_output: out data port FloorNumber;
    properties
        SEI::MIPSBudget => 0.8 mips;
end DisplayThread;

thread implementation DisplayThread.impl
end DisplayThread.impl;
```

`DisplayThread` jest wątkiem odpowiedzialnym za aktualizację wyświetlacza windy. Odbiera on informację o aktualnym piętrze od `DisplayController` i wyświetla ją na panelu w kabinie windy. Wątek ten działa w sposób ciągły, aktualizując wyświetlacz za każdym razem, gdy winda zmienia piętro. Jego zadaniem jest informowanie, za pomocą wyświetlacza (`Display`), pasażerów o aktualnym położeniu windy.

2.2.6 CabinControlThread

Kod:

```
thread CabinControlThread
    features
        door_command_in: in data port DoorCommand;
        door_state_in: in data port DoorState;
        door_open: out event port;
        door_close: out event port;
    properties
        SEI::MIPSBudget => 0.8 mips;
end CabinControlThread;

thread implementation CabinControlThread.impl
end CabinControlThread.impl;
```

`CabinControlThread` jest wątkiem odpowiedzialnym za sterowanie drzwiami kabiny windy. Odbiera on polecenia od `CabinController`, które określają, czy drzwi mają być otwarte czy zamknięte. Wątek ten generuje odpowiednie sygnały sterujące dla mechanizmu drzwi. Działanie tego wątku jest kluczowe dla bezpieczeństwa i wygody pasażerów, zapewniając prawidłowe funkcjonowanie drzwi windy.

2.3 Komponenty typu `process`

2.3.1 ButtonPanelController

Kod:

```
process ButtonPanelController
    features
        receive_button_press: in data port ButtonPress;
        send_button_data: out data port ButtonPress;
end ButtonPanelController;

process implementation ButtonPanelController.impl
    subcomponents
        buttonPanelThread: thread ButtonPanelThread.impl;
    connections
        c1: port receive_button_press -> buttonPanelThread.button_press_in;
        c2: port buttonPanelThread.button_data_out -> send_button_data;
end ButtonPanelController.impl;
```

`ButtonPanelController` jest procesem odpowiedzialnym za zarządzanie panelem przycisków w windzie. Jego głównym zadaniem jest odbiór informacji o naciśniętych przyciskach z panelu i przekazanie tych danych do kontrolera logiki windy (`ElevatorLogicController`). Ten proces umożliwia pasażerom wybór piętra docelowego.

2.3.2 ElevatorLogicController

Kod:

```
process ElevatorLogicController
    features
        receive_button_data: in data port ButtonPress;
        receive_motor_state_data: in data port ElevatorState;
        send_floor_data: out data port FloorNumber;
        send_door_command: out data port DoorCommand;
        send_action: out data port MotorCommand;
end ElevatorLogicController;

process implementation ElevatorLogicController.impl
    subcomponents
        logicThread: thread ElevatorLogicThread.impl;
        transformThread: thread ElevatorLogicTransformThread.impl;
    connections
        c1: port receive_button_data -> logicThread.button_data_in;
        c2: port receive_motor_state_data -> transformThread.motor_state_in;
        c3: port logicThread.action_data_out -> transformThread.action_data_
        c5: port transformThread.floor_data_out -> send_floor_data;
        c6: port transformThread.door_command_out -> send_door_command;
        c7: port transformThread.motor_state_out -> send_action;
end ElevatorLogicController.impl;
```

`ElevatorLogicController` jest centralnym procesem odpowiedzialnym za logikę sterowania windą. Odbiera on informacje o naciśniętych przyciskach oraz aktualny stan windy. Na podstawie tych danych, podejmuje decyzje dotyczące kierunku ruchu windy, piętra docelowego oraz stanu drzwi, wysyłając odpowiednie polecenia do kolejnych komponentów.

2.3.3 DisplayController

Kod:

```
process DisplayController
    features
        receive_floor_data: in data port FloorNumber;
        display_output: out data port FloorNumber;
end DisplayController;

process implementation DisplayController.impl
    subcomponents
        displayThread: thread DisplayThread.impl;
    connections
        c1: port receive_floor_data -> displayThread.floor_data_in;
        c2: port displayThread.display_output -> display_output;
end DisplayController.impl;
```

`DisplayController` jest procesem odpowiedzialnym za zarządzanie wyświetlaczem windy. Jego głównym zadaniem jest odbieranie informacji o aktualnym piętrze od `ElevatorLogicController` i przekazywanie ich do wątku `DisplayThread`, który zajmuje się fizyczną aktualizacją wyświetlacza.

2.3.4 CabinController

Kod:

```
process CabinController
    features
        receive_door_command: in data port DoorCommand;
        receive_door_state: in data port DoorState;
        door_open: out event port;
        door_close: out event port;
end CabinController;

process implementation CabinController.impl
    subcomponents
        cabinThread: thread CabinControlThread.impl;
    connections
        c1: port receive_door_command -> cabinThread.door_command_in;
        c2: port receive_door_state -> cabinThread.door_state_in;
        c3: port cabinThread.door_open -> door_open;
        c4: port cabinThread.door_close -> door_close;
end CabinController.impl;
```

`CabinController` jest procesem odpowiedzialnym za zarządzanie drzwiami kabiny windy. Zapewnia, że drzwi otwierają się i zamykają tylko w odpowiednich momentach, na przykład gdy winda jest zatrzymana na piętrze, oraz że drzwi są bezpiecznie zamknięte podczas jazdy windy.

2.3.5 MotorController

Kod:

```
process MotorController
    features
        receive_motor_command: in data port MotorCommand;
        send_motor_state: out data port ElevatorState;
end MotorController;

process implementation MotorController.impl
    subcomponents
        motorThread: thread MotorControlThread.impl;
    connections
        c1: port receive_motor_command -> motorThread.motor_state_in;
        c2: port motorThread.motor_state_out -> send_motor_state;
end MotorController.impl;
```

`MotorController` jest procesem pełniącym rolę pośrednika między logiką sterowania windą, a silnikiem. Jego głównym zadaniem jest odbieranie poleceń ruchu i przekazywanie ich do wątku `MotorControlThread`, który bezpośrednio steruje silnikiem.

2.4 Komponenty typu `bus`

2.4.1 CANBus

Kod:

```
bus CANBus
    properties
        SEI::GrossWeight => 0.075kg;
        SEI::BandWidthCapacity => 1.0 Mbps;
end CANBus;

bus implementation CANBus.impl
end CANBus.impl;
```

`CANBus` jest magistralą komunikacyjną typu CAN (Controller Area Network). Służy ona do wymiany danych między różnymi komponentami systemu windy. Magistrala CAN zapewnia niezawodną i szybką komunikację w czasie rzeczywistym, co jest kluczowe dla prawidłowego działania systemu windy.

2.4.2 HWConnection

Kod:

```
bus HWConnection
    properties
        SEI::GrossWeight => 0.075kg;
        SEI::BandWidthCapacity => 1000.0 Mbps;
end HWConnection;

bus implementation HWConnection.impl
end HWConnection.impl;
```

`HWConnection` reprezentuje magistralę sprzętową, która umożliwia szybką komunikację między procesorem (`CPU`) a pamięcią RAM (`RAM`). Jest to wewnętrzne połączenie o wysokiej przepustowości, które służy do przesyłania informacji między tymi dwoma komponentami systemu.

2.5 Komponenty typu `memory`

2.5.1 RAM

Kod:

```
memory RAM
    features
        hwcAccess: requires bus access HWConnection;
    properties
        SEI::GrossWeight => 0.025kg;
end RAM;

memory implementation RAM.impl
end RAM.impl;
```

`RAM` to komponent reprezentujący pamięć operacyjną systemu windy. Jest to miejsce, w którym przechowywane są dane tymczasowe, instrukcje programu oraz zmienne wykorzystywane przez procesor (`CPU`) podczas działania windy.

2.6 Komponenty typu `processor`

2.6.1 CPU

Kod:

```
processor CPU
    features
        canBusAccess: requires bus access CANBus;
        hwcAccess: requires bus access HWConnection;
    properties
        Scheduling_Protocol => (Round_Robin_Protocol);
        Clock_Period => 1 ms;
        Timing_Properties::Processor_Capacity => 1.0 MIPS;
        SEI::MIPSCapacity => 1.2 mips;
        SEI::GrossWeight => 0.05kg;
end CPU;

processor implementation CPU.impl
end CPU.impl;
```

`CPU` to centralny element obliczeniowy systemu windy, odpowiedzialny za wykonywanie instrukcji oprogramowania sterującego windą. Jest to kluczowy komponent, który przetwarza dane wejściowe z różnych czujników i przycisków, podejmuje decyzje dotyczące ruchu windy (kierunek, prędkość, zatrzymanie) oraz steruje innymi urządzeniami, takimi jak: silnik, drzwi i wyświetlacz.

2.7 Komponenty typu `device`

2.7.1 ButtonPanel

Kod:

```
device ButtonPanel
    features
        button_press: out data port ButtonPress;
        bus_access: requires bus access CANBus;
    properties
        SEI::GrossWeight => 1.0kg;
end ButtonPanel;

device implementation ButtonPanel.impl
end ButtonPanel.impl;
```

`ButtonPanel` jest urządzeniem, które reprezentuje panel przycisków wewnątrz i na zewnątrz windy. Zadaniem tego panelu jest umożliwienie pasażerom wydawanie poleceń windzie, takich jak wybór piętra docelowego czy wezwanie windy na dane piętro.

2.7.2 Cabin

Kod:

```
device Cabin
    features
        door_state: out data port DoorState;
        door_open: in event port;
        door_close: in event port;
    properties
        SEI::GrossWeight => 150.0kg;
end Cabin;

device implementation Cabin.impl
end Cabin.impl;
```

`Cabin` reprezentuje kabinę windy, czyli przestrzeń, w której przebywają pasażerowie podczas jazdy.

2.7.3 Motor

Kod:

```
device Motor
    features
        motor_state_in: in data port ElevatorState;
        motor_state_out: out data port ElevatorState;
    properties
        SEI::GrossWeight => 5.0kg;
end Motor;

device implementation Motor.impl
end Motor.impl;
```

`Motor` to urządzenie, które reprezentuje silnik elektryczny napędzający windę. Silnik ten jest odpowiedzialny za ruch kabiny w górę i w dół szybu windowego.

2.7.4 Display

Kod:

```
device Display
    features
        display_input: in data port FloorNumber;
end Display;

device implementation Display.impl
end Display.impl;
```

`Display` jest urządzeniem odpowiedzialnym za prezentowanie informacji pasażerom windy. Jego głównym zadaniem jest wyświetlanie aktualnego piętra, na którym znajduje się kabina.

2.7.5 ElevatorController

Kod:

```
device ElevatorController
    features
        action_receive: in data port MotorCommand;
        motor_command: out data port MotorCommand;
        motor_state_receive: in data port ElevatorState;
        motor_state_send: out data port ElevatorState;
        bus_access: requires bus access CANBus;
    properties
        SEI::GrossWeight => 0.5kg;
        Period => 100ms;
        Dispatch_Protocol => Periodic;
end ElevatorController;

device implementation ElevatorController.impl
end ElevatorController.impl;
```

`ElevatorController` to urządzenie pełniące rolę głównego sterownika sprzętowego windy. Ten komponent jest kluczowy dla prawidłowego działania windy, ponieważ integruje logikę sterowania z fizycznymi elementami windy, takimi jak silnik i drzwi.

2.8 Komponenty typu `system`

2.8.1 ElevatorSystem

Kod:

```
system implementation ElevatorSystem.impl
    subcomponents
        button_panel: device ButtonPanel.impl;
        cabin: device Cabin.impl;
        motor: device Motor.impl;
        elevatorController: device ElevatorController.impl;
        display: device Display.impl;

        cpu: processor CPU.impl;
        ram: memory RAM.impl;
        can_bus: bus CANBus.impl;
        hwc: bus HWConnection.impl;
```

```
    buttonPanelController: process ButtonPanelController.impl;
    elevatorLogicController: process ElevatorLogicController.impl;
    displayController: process DisplayController.impl;
    cabinController: process CabinController.impl;
    motorController: process MotorController.impl;



connections
    -- ButtonPanel connections
    c1: bus access button_panel.bus_access -> can_bus;
    c2: port button_panel.button_press -> buttonPanelController.receive_

    -- ElevatorController connections
    c3: bus access elevatorController.bus_access -> can_bus;
    c4: port elevatorController.motor_command -> motorController.receive
    c5: port motorController.send_motor_state -> motor.motor_state_in;
    c6: port motor.motor_state_out -> elevatorController.motor_state_rec
    c7: port elevatorController.motor_state_send -> elevatorLogicControl
    c8: port elevatorLogicController.send_action -> elevatorController.a

    -- CPU connections
    c9: bus access cpu.hwcAccess -> hwc;
    c10: bus access cpu.canBusAccess -> can_bus;

    -- RAM connection
    c11: bus access ram.hwcAccess -> hwc;

    -- Process connections
    c12: port buttonPanelController.send_button_data -> elevatorLogicCon
    c13: port elevatorLogicController.send_floor_data -> displayControlle
    c14: port elevatorLogicController.send_door_command -> cabinControlle
    c15: port cabin.door_state -> cabinController.receive_door_state;

    -- Cabin connections
    c16: port cabinController.door_open -> cabin.door_open;
    c17: port cabinController.door_close -> cabin.door_close;

    -- Display connection
    c18: port displayController.display_output -> display.display_input;

properties
    -- Bindings
    Actual_Processor_Binding => (reference(cpu)) applies to buttonPanelC
    Actual_Processor_Binding => (reference(cpu)) applies to elevatorLogi
    Actual_Processor_Binding => (reference(cpu)) applies to displayContr
```

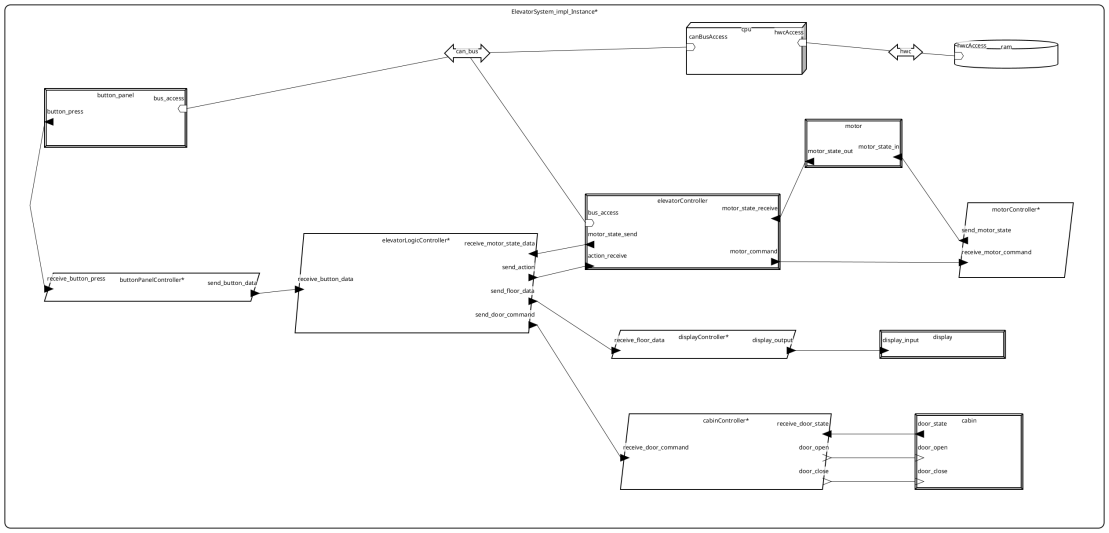
```
        Actual_Processor_Binding => (reference(cpu)) applies to cabinControll
        Actual_Processor_Binding => (reference(cpu)) applies to motorControll

        --Other
        SEI::MIPSCapacity => 2.4 mips;

end ElevatorSystem.impl;
```

`ElevatorSystem` reprezentuje cały system windy jako całość. Jest to komponent typu system, który integruje wszystkie elementy składowe windy. Model `ElevatorSystem` definiuje hierarchiczną strukturę systemu windy, określając zależności i połączenia między poszczególnymi komponentami, co umożliwia analizę i weryfikację poprawności działania całego systemu.

3 Schemat systemu



4 Analiza systemu

4.1 Statystyki systemu

Elementy modelu	Liczba
Komponenty ogółem	21
Komponenty `bus`	2
Komponenty `device`	5
Komponenty `memory`	1
Komponenty `process`	5
Komponenty `processor`	1
Komponenty `system`	1
Komponenty `thread`	6
Połączenia	19

4.2 Całkowita masa systemu

Wyniki analizy *Sum of weights*:

```
[L] Sum of weights / gross weight is 156.975 kg (no limit specified).
```