Protocol Validation - MPSP assignment

Luc Veldhuis - 2538227 Marta Rozek - 2643093

October 2018

1 Introduction

For this assignment, we are asked to create a protocol specification for a simplified version of the Movable Patient Support Platform (MPSP) invented by Philips. We are given a specification of the behaviour in natural language and we are asked to identify the interactions with the outside world and translate the specification into requirements. Then we will express these requirements in terms of the actions. We also have to think about the architecture of the system. Once we have extracted all of these properties from the specification, we have to model this protocol in the μCRL2 toolset and use μ-calculus to verify our requirements.

2 Interactions with the outside world

We start by identifying the following sets:

- Buttons = {up, down, undock, reset, resume, stop}
- SenseHorizontal = {initial, middle, outermost }
- SenseVertical = {lowest, below, standard, above, highest }
- DirectionHorizontal = {inwards, outwards}
- DirectionVertical = {upwards, downwards}

With these sets, we can identify the following actions:

- button press(b $\in$ Buttons). A button from the set Buttons is pressed.
- button release(b $\in$ Buttons). A button from the set Buttons is released.
- motor vertical stop. The vertical motor stops moving.
- motor horizontal stop. The horizontal motor stops moving.
- brake vertical. The vertical brake is applied.
- unbrake vertical. The vertical brake is released.
- brake horizontal. The horizontal brake is applied.
- unbrake horizontal. The horizontal brake is released.

- sense horizontal(s ∈ SenseHorizontal). The sensor senses the horizontal position from the set SenseHorizontal.
- sense vertical(s ∈ SenseVertical). The sensor senses the vertical position from the set SenseVertical.
- move horizontal(d ∈ DirectionHorizontal). The horizontal motor moves in a direction from the set DirectionHorizontal.
- move vertical(d ∈ DirectionVertical). The vertical motor moves in a direction from the set DirectionVertical.
- sense dock. The sensor senses a docking action.
- undock. The spring mechanism undocks the MPSP from the scanner.
- calibrate. The standard height is set.
- uncalibrate. The standard height is reset.
- emergency. The MPSP enters emergency mode.
- unemergency. The MPSP exists emergency mode.

We do not have a docking action, because this is not an action of the system, but a manual action that operator of the machine performs, which we are only able to sense. The calibrate action sets the 'standard' value, which the sensor reads from the vertical position. The uncalibrate action resets this value. We also could have modelled some more actions, for example a 'sense nothing' action, if the sensor does not sense anything, but this would make our specification only more complicated, without adding necessary information to functions.

3 Global requirements

From the natural language specification we identified the following requirements and expressed them in the actions which we have just listed:

MV1 Horizontal movement is only allowed when the MPSP is docked.

```
[(!sense_dock)*.(sense_dock.(!undock)*.undock.(!sense_dock)
   *)*.move_horizontal(inwards)] false && [(!sense_dock)*.(
   sense_dock.(!undock)*.undock.(!sense_dock)*)*.
   move_horizontal(outwards)] false
```

MV2 When the MPSP is uncalibrated, the up and down buttons can only be used to move the bed up and down.

```
[(!calibrate)*.(calibrate.(!uncalibrate)*.uncalibrate.(!
   calibrate)*)*.(move_horizontal(inwards) ||
   move_horizontal(outwards))] false
```

MV3 The bed is not allowed to move above some highest and below some lowest position.

```
[true*.sense_vertical(v_upper).brake_vertical.move_vertical
   (up)] false &&
[true*.sense_vertical(v_lowest).brake_vertical.
   move_vertical(down)] false
```

POS1 When the MPSP is undocked, the bed must always be in the initial position.

```
[(!sense_dock)*.(sense_dock.(!undock)*.undock.(!sense_dock)
   *)*.(sense_horizontal(h_middle) || sense_horizontal(
   h_outer))] false
```

BR1 When the MPSP is undocked, the horizontal brake must always be applied.

```
[(!sense_dock)*.(sense_dock.(!undock)*.undock.(!sense_dock)
   *)*.unbrake_horizontal] false
```

BR2 The vertical brake must always be applied while the vertical motor is off.

```
[true*.motor_vertical_stop.brake_vertical.(!move_vertical(
   up) && !move_vertical(down))*.unbrake_vertical] false
```

The brake must also always be released when the motor is running, which is the next requirement. This is a contradiction: it may not be released when the motor is running, but it may also not be released when the motor is not running. So we decided that the very first action after stopping a motor is applying the brake, and the first action after starting the motor is releasing the brake. So we verify this requirement by checking that the brake may never be released somewhere before starting the motor.

BR3 When starting a motor, the break must be released.

```
[true*.(move_horizontal(inwards) || move_horizontal(
   outwards)).(!unbrake_horizontal)] false
&&
[true*.(move_vertical(up) || move_vertical(down)).(!
   unbrake_vertical)] false
```

This is intertwined with the previous requirement. We have to fix some order in which we start the motors and apply the brakes.

BR4 When a motor is running, the break must never be applied unless the motor is being stopped.

```
[true*.(move_vertical(up) || move_vertical(down)).(!
   motor_vertical_stop)*.brake_vertical.(!
   motor_vertical_stop)]false &&
[true*.(move_horizontal(inwards) || move_horizontal(
   outwards)).(!motor_horizontal_stop)*.brake_horizontal.(!
   motor_horizontal_stop)]false
```

EM1 The stop button puts the MPSP in emergency mode.

```
[true*.but_release(b_stop)]<(!unemergency)*.emergency> true
```

To model this liveness requirement correctly, we added the ‘emergency’ and ‘unemergency’ actions to the set of possible actions.

EM2 In emergency mode, the horizontal brake must be released.

```
[true*.emergency]<(!unemergency)*.unbrake_horizontal> true
```

EM3 In emergency mode, the horizontal motor must never be active.

```
forall d: DH.[true*.emergency.(!unemergency)*.(
   move_horizontal(d))] false
```

EM4 The resume button puts the MPSP back to normal operating mode.

```
[true*.but_release(b_resume)]<(!emergency)*.unemergency>
   true
```

EM5 The emergency mode can only be activated when docked.

```
[(!sense_dock)*.(sense_dock.(!undock)*.undock.(!sense_dock)
   *)*.emergency] false
```

Because in the emergency mode the horizontal brake is released, this may never happen when the MPSP is undocked, because otherwise the bed might fall over.

UND1 The MPSP should never undock if the bed is not in the initial position. The actions `sense_vertical(v_upper)` and `sense_vertical(v_above)` are here because if the bed is manually docked while above the middle height, it's immediately undocked. See also MVDCK1.

```
[true*.sense_dock.(!(sense_horizontal(h_initial) ||
   sense_vertical(v_upper) || sense_vertical(v_above)))*.
   undock] false
```

Because we do not control the docking action, if we sense that somebody attempted to dock the MPSP while in an incorrect height, it should immediately undock.

UND2 The MPSP should never undock if the bed is not docked.

```
[(!sense_dock)*.(sense_dock.(!undock)*.undock.(!sense_dock)
   *)*.undock] false
```

CAL1 Before use, the MPSP must be calibrated by setting the standard height.

```
[(!calibrate)*.(calibrate.(!uncalibrate)*.uncalibrate.(!
   calibrate)*)*.(move_horizontal(inwards) ||
   move_horizontal(outwards))] false
```

Here we model the 'before use' part of the requirement by checking if it is calibrated before it starts moving inwards.

CAL2 The MPSP can only be uncalibrated when undocked.

```
[true*.sense_dock.(!undock)*.uncalibrate] false
```

We do not want to be able to uncalibrate when there is a patient in the scanner. To prevent accidents, we allow only for uncalibrating when the MPSP is undocked.

MVDCK1 The MPSP can only be docked if it is at the standard height or below it.

```
[true*.sense_dock.(sense_vertical(v_upper) ||
   sense_vertical(v_above)).!undock] false
```

MVDCK2 When the up button is pressed and the bed is below standard height, the bed will move up.

```
[true*.but_press(b_up).(sense_vertical(v_lowest) ||
   sense_vertical(v_below))]<(!sense_vertical(v_mid))*.
   move_vertical(up)> true
```

MVDCK3 If the up button is pressed at a non-standard height, the MPSP does not move into the scanner.

```
[true*.!sense_vertical(v_mid).sense_horizontal(h_initial).
   move_horizontal(inwards)] false
```

MVDCK4 When the MPSP is at the standard position and the up button is pressed, the bed moves inside the scanner.

```
[true*.but_press(b_up).sense_vertical(v_mid)]<(!
   sense_horizontal(h_outer))*.move_horizontal(inwards)>
   true
```

MVDCK5 When the MPSP is at the standard height and at a non-initial position and the down button is pressed, the bed moves outwards.

```
[true*.but_press(b_down).sense_vertical(v_mid).(
   sense_horizontal(h_outer) || sense_horizontal(h_middle))
   ]<(!sense_horizontal(h_initial))*.move_horizontal(
   outwards)> true
```

MVDCK6 When the bed is at the initial position and the down button is pressed, the bed will subsequently move downwards.

```
[true*.sense_horizontal(h_initial).(!(sense_horizontal(
   h_middle)))*.but_press(b_down)]<(!sense_vertical(
   v_lowest))*.move_vertical(down)> true
```

MVDCK7 While the MPSP is docked and calibrated, the bed cannot be moved above the standard height.

```
[true*.calibrate.(!undock)*.sense_vertical(v_mid).
   move_vertical(up)] false
```

We use the fact that we can only calibrate when we are docked and only uncalibrate when we are undocked to prevent this formula from becoming very large.

BTN Only 1 button can be pressed at the same time.

```
forall b1, b2: BTN.[but_press(b1).(!but_release(b1))*.
   but_press(b2)] false &&
forall b3, b4: BTN.[but_release(b3).(!but_press(b3))*.
   but_release(b4)] false
```

DEAD Some action should always be possible.

```
[true*]<true> true
```

This is a standard deadlock-free requirement.

There is a separate calibrate routine, which is executed when the MPSP needs to be calibrated:

INIT1 The MPSP can only be calibrated when it's docked.

```
[(!sense_dock)*.(sense_dock.(!undock)*.undock.(!sense_dock)
   *)*.calibrate] false
```

INIT2 When the MPSP is uncalibrated and the up or down button is pressed, the bed is moved.

```
[(!calibrate)*.(calibrate.(!uncalibrate)*.uncalibrate.(!
   calibrate)*)*.but_press(b_up)]<!sense_vertical(v_upper).
   move_vertical(up)> true
&&
[(!calibrate)*.(calibrate.(!uncalibrate)*.uncalibrate.(!
   calibrate)*)*.but_press(b_down)]<!sense_vertical(
   v_lowest).move_vertical(down)> true
```

INIT3 When the MPSP is undocked and the reset button is pressed, standard height is forgotten and the bed is uncalibrated.

```
[(!sense_dock)*.(sense_dock.(!undock)*.undock.(!sense_dock)
   *)*.but_release(b_reset)]<uncalibrate> true
```

4 Architecture

For the architecture, we use 7 separate components.

- The controller. All buttons are located on this controller. The logic is also implemented in this component.
- The horizontal brake. The horizontal brake and vertical brake should be able to work independently, so we made these separate components.
- The vertical brake.
- The horizontal motor. The horizontal motor and vertical motor should be able to work independently, so we made these separate components.
- The vertical motor.
- The undock spring mechanism. This system activates the spring mechanism to undock the MPSP.
- The sensor. This is able to sense the position of the bed, vertical and horizontal, and detect if the MPSP is docked or not.

Only the controller communicates with the other components. The other components do not communicate with each other. Whenever there is no action specified for a button press, we assume that it is ignored. For example: pressing the reset button in the emergency mode. We did not add all of these possibilities because this would make the protocol less readable.

5 Verifying the requirements

We verified all requirements by checking if the corresponding μ-calculus rules were true in our constructed protocol description using the μCRL tool-set. We used a script which verified all rules after one another, so we could easily see which rule failed.