Protocol Validation Assignment 2018

Lai Wenchen, Will Bergen

Overview

This report covers our proposed architecture for a Mobile Patent Support Platform (MPSP). An MPSP is movable bed apparatus consisting of a bed capable of being actuated by two motors. One motor allows the bed to move horizontally, while the other allows vertical movement. Both motors have individually actuable brakes, which must be active anytime their respective motor is not on. The MPSP is capable of being 'docked' with a scanning unit. After such a docking, the MPSP may be 'calibrated', which saves the bed's current height, and allows the up and down buttons to be used to move the bed into and out of the attached scanner. Additionally, there is an 'emergency' mode, in which the bed is manually-movable horizontally, while no vertical movement, or any other action save exiting emergency mode, is possible.

A user interacts with the MPSP via a controller which has the following buttons: up, down, reset, undock, resume, stop. The up and down buttons which actuate the motors may each also be held down, causing continuous actuation of the associated motor until they are released. The reset button controls calibration: if the scanner is docked, reset saves the bed's current height, while if it is not docked, reset clears or 'uncalibrated' the saved height. Stop puts the MSPS into the emergency mode described above which can then be exited by use of the resume button.

Finally, the MPSP has seven sensors which detect the following positions of the MPSP's bed:

- Its rightmost position
- Its leftmost position
- Its maximum height
- Its minimum height
- Above the calibrated height
- Below the calibrated height

Section 1.

We have identified the following global requirements, or rules, that the whole system must abide by. Their Rx labels correspond to their modal validation formulas:

1. (R1) Horizontal movement is only allowed when the MPSP is docked.
2. (R2) The MPSP may never be undocked from an attached scanner if the bed is not in the rightmost position.
3. When the MPSP is not docked to a scanner, the horizontal brake must always be applied (covered by our formulation of R1, see Appendix 1).
4. (R4) Whenever a motor is on, its brake must be off.
5. (R5) Liveness of Locks: lock acquisition by any caller must eventually be followed by a release from the same caller. (We will describe our locks in the following sections)
6. (R6) When the MPSP is calibrated, its bed is at the calibrated height, and up is pressed on the controller, the bed moves into the scanner (‘leftward’ from the perspective of our design). Conversely, if the same conditions hold, but instead of up, down is pressed, the bed moves out of the scanner (‘rightward’).
7. (R7) If the MPSP is docked and calibrated the bed may not move above the calibrated height.
8. (R8) When the MPSP is either undocked or uncalibrated, the up/down buttons only cause the bed to move up or down, never right or left.
9. (R9) The bed may never move above or below two predetermined limits: a maximum and minimum height.
10. (R10) Deadlock Freeness

We have further assumed the following given the underspecified nature of the original requirements:

1. Pressing reset button while the system is docked and calibrated has no effect. ‘Recalibrating’ the MPSP requires undocking.
2. Only a single motor may be moving at any given time and in a single direction only.

Section 2.

The concise building blocks used in our system. These describe the interactions of the system with the outside world.

1. Read up released
    a. The ‘UP’ button on the console is released: {up_released}
2. Read down released
    a. The ‘DOWN’ button on the console is released: {down_released}
3. Read B:{all buttons} pressed
    a. The ‘B’ button on the console is pressed
    b. {up_pressed, down_pressed, stop_pressed, resume_pressed, reset_pressed, undock_pressed}
4. Actuate vertical motor {up/down}
    a. Starts the vertical motor s.t. Bed moves in direction specified
5. Stop vertical motor
6. Actuate horizontal motor {right/left}
    a. Starts the horizontal motor s.t. bed moves in direction specified
7. Stop horizontal motor
8. Apply/release horizontal brake
9. Apply/release vertical brake

Further, we internally maintain a notion of the following system properties:

1. Docking:
    a. The MPSP is aware of whether it is docked or not, and may either ‘dock’ with, or ‘undock’ from a scanner given the conditions described in the overview.
    b. In our terms: {do_dock}, while its reciprocal is represented as a state resulting from undock_pressed.
    c. We’ll denote these two states as DOCKED, !DOCKED
2. Emergency Mode:
    a. The MPSP is aware of being in either emergency mode or not, as a result of reset and resume button presses.
    b. We’ll denote these two states as EMERGENCY, !EMERGENCY
3. Sensor Activations:
    a. The MPSP’s sensors are all binary on/off sensors, set on when the MPSP’s bed performs a specific action ie. moves below the saved height, or when the bed reaches a particular state, ie. moves fully out of the scanner.
    b. In our terms: {min_reached, max_reached, right_reached, left_reached, above_calibrated_detected, below_calibrated_detected}
    c. These actions allow the MPSP to internally be aware of where the bed is, resulting in the additional conditions we’ll denote as RIGHTMOST and LEFTMOST.

d. The bed being at the calibrated height is represented by both above_calibrated_detected and below_calibrated_detected sensors being off.

Finally, as will be described more fully in (section 4) our system also has the following lock related actions: acquire_sensor_lock, release_sensor_lock, acquire_control_lock and release_control_lock.

Section 3.

Translation of our global requirements (Section 1) in terms of our interactions (Section 2).

1. (R1) Horizontal movement is only allowed when the MPSP is docked.
    a. If !DOCKED, horizontal movement is not possible.
2. (R2) The MPSP may never be undocked from an attached scanner if the bed is not in the rightmost position.
    a. If !RIGHTMOST undocking not possible
3. (R3) When the MPSP is not docked to a scanner, the horizontal brake must always be applied.
    a. If !DOCKED, horizontal brake must be on
    b. This requirement is covered by R1
4. (R4) Whenever a motor is on, its brake must be off.
    a. If the horizontal motor is moving, horizontal brake must be off
    b. If the vertical motor is moving, vertical brake must be off
5. (R5) Liveness of Locks: lock acquisition by any caller must eventually be followed by a release from the same caller.
6. (R6) When the MPSP is calibrated, its bed is at the calibrated height, and up is pressed on the controller, the bed moves into the scanner (‘leftward’ from the perspective of our design). Conversely, if the same conditions hold, but instead of up, down is pressed, the bed moves out of the scanner (‘rightward’).
    a. If CALIBRATED and sensor detects neither above_calibrated_detected nor below_calibrated_detected and up_pressed occurs, the bed moves to the left (into the scanner).
    b. Same conditions, but down_pressed occurs, bed moves right (out of scanner)
7. (R7) If the MPSP is docked and calibrated the bed may not move above the calibrated height.
    a. If CALIBRATED and DOCKED, up_pressed occuring may never move the bed above saved height.
8. (R8) When the MPSP is either undocked or uncalibrated, the up/down buttons only cause the bed to move up or down, never right or left.

    a. If !DOCKED or !CALIBRATED, up/down_pressed actions may never result in horizontal movement.
9. (R9) The bed may never move above or below two predetermined limits: a maximum and minimum height.
    a. up/down_pressed actions may never cause the bed to move above or below maximum/minimum heights detected via sensors.
10. (R10) Deadlock Freeness

Section 4.

Depiction of system architecture and design

Brief overview of design:

We have chosen to model this system using four separate controllers, or processes: a global state process (GlobalMod), a motor actuator process (MotorMod), a sensor detector process (SensorMod) and a controller process (ControlMod). All of our processes except MotorMod take a data structure we call Global_State as a parameter, which is maintained in GlobalMod. SensorMod and ControlMod each communicate with GlobalMod, and update the Global_State maintained there each time they receive input ie. via an action taken or a sensor activation. ControlMod and SensorMod are synchronized by requiring both to acquire a lock on GlobalMod, implemented as an action, before either is able to update the Global_State in GlobalMod. Finally, MotorMod is simply a dummy process which accepts a Global_State and sets the motors and brake accordingly.

Our Global_State structure is as follows:

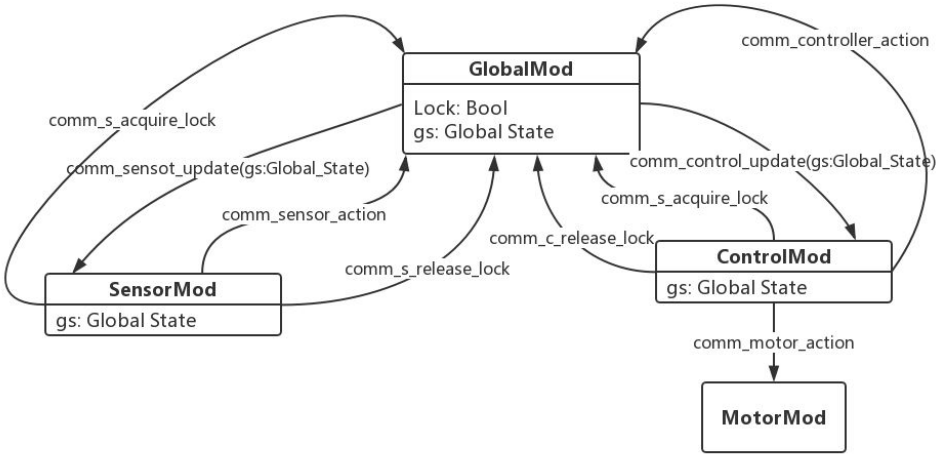
```
sort Global_State = struct global_state(
        sens_min_h: Bool,           % = false [sense min height]
        sens_max_h: Bool,           % = false [sense max height]
        sens_left: Bool,            % = false
        sens_right: Bool,           % = false
        docked: Bool,               % = false
        calibrated: Bool,           % = false
        emergency: Bool,    % = false
        h_above_cal: Bool,          % = false [bed is above the calibrated height]
        h_below_cal: Bool,          % = false [bed is below the calibrated height]

        mot_vert_down: Bool,        % = false
        mot_vert_up: Bool,          % = false
        mot_hor_down: Bool,         % = false [left]
        mot_hor_up: Bool,           % = false [right]
        brake_vert: Bool,   % = true
```

```
        brake_hor: Bool                % = true
    );
```

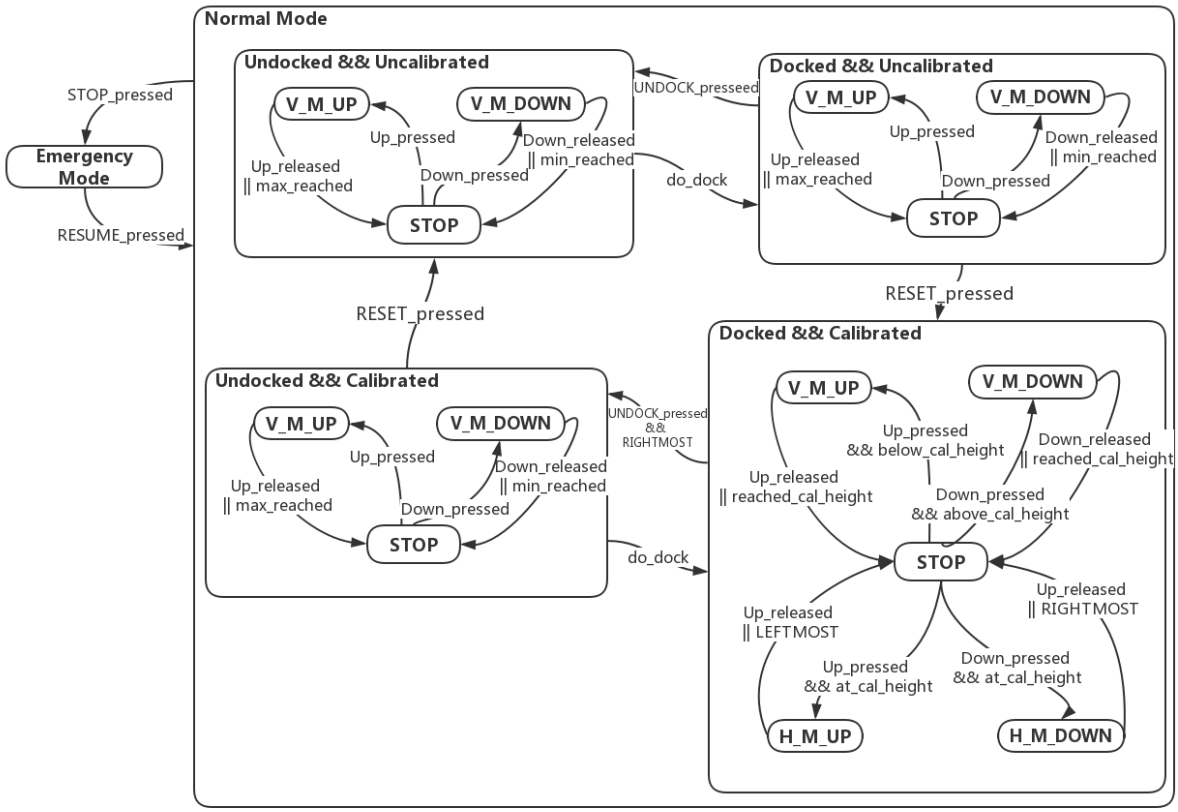
*The commented notes to the right indicate default/initial values, additional clarifications

Figure 1. Architecture of system, indicating how communication and locking works. (Included as *Architecture.png*)



In Figure 1, the communication between our four processes is depicted. MotorMod simply receives communication from ControlMod, indicating updates in the states of the two motors and brakes. Before SensorMod or ControlMod can take an action they must first 'acquire' GlobalMod's lock, via the comm_{control,sensor}_acquire_lock communicating actions. Once the lock has been acquired, the process receives the current Global_State from GlobalMod via the comm_{sensor,control}_update actions. The locking process may then communicate updates in the system's state to GlobalMod by using communicating actions which exist for each of the properties in Global_State ie. becoming 'docked' uses comm_docked(true). After communicating an update to GlobalMod, the locking process releases its lock.

Figure 2. High-level state transition diagram, indicating availability of actions depending on internal state. (included as file *States.png*)



In Figure 2, the abstract states of the system which arise from our Global_State view are depicted. At the top of the hierarchy is the distinction between the highly restrictive Emergency mode and 'Normal' Mode. In Normal mode, the MPSP's ability to be either docked or undocked, combined with its ability to be either calibrated or uncalibrated produce four general abstract conditions within which, system interactions behave differently. For instance, when the MPSP is uncalibrated and undocked, the up_pressed and down_pressed actions will never produce horizontal movement, whereas they may if the MPSP is docked and calibrated.

Following these considerations, we have added a third assumption to the two already given in Section 1:

- Our SensorMod mandates that only a one of the following pairs be 'on' ie. -eq 1 at any given time:
    a. (sens_min_h, sens_max_h)
    b. (sens_left, sens_right)
    c. (h_above_cal, h_below_cal)

Section 5.

The mCRL2 specification can be found in the file *assign.mcrl2*. For brevity, it is not copied in this document.

Section 6.

Our uCalc formulas can be found in the folder *properties* and in Appendix 1. A simple script to evaluate all of them against our mCRL2 spec is included as *validate.sh*

Appendix 1.

- R1.mcf
  - If !DOCKED, horizontal movement is not possible.
  - [true* . comm_control_update(global_state((true||false),(true||false),(true||false),(true||false),false,(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false)))]<comm_action_state(global_state((true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),false))>false
- R2.mcf
  - If !RIGHTMOST undocking not possible
  - [true* . comm_control_update(global_state((true||false), (true||false), (true||false), false, (true||false), (true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false))) . undock_pressed]false
- R4a.mcf
  - If horizontal motor moving, horizontal brake must be off
  - [true* .comm_action_state(global_state((true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false), (true||false), false, false, (true||false), false))]false
- R4b.mcf
  - If vertical motor moving, vertical brake must be off
  - [true* .comm_action_state(global_state((true||false),(true||false),(true||false),(true||false)

,(true||false),(true||false),(true||false),(true||false),(true||false),false, false, (true||false), (true||false), false, (true||false)))]false

- R5.mcf
  - Lock Liveness
  - [acquire_c_lock]<true*.release_c_lock>true
  - &&
  - [acquire_s_lock]<true*.release_s_lock>true
- R6a.mcf
  - If DOCKED && CALIBRATED && at the saved height && RIGHTMOST, down_pressed not available.
  - [true* . comm_control_update(global_state((true||false),(true||false),(true||false),true, true, true, false, false, false, (true||false),(true||false),(true||false),(true||false), (true||false),(true||false)))]<down_pressed>false
- R6b.mcf
  - If DOCKED && CALIBRATED && at the saved height && LEFTMOST, up_pressed not available.
  - [true* . comm_control_update(global_state((true||false),(true||false),true,(true||false), true, true, false, false, false, (true||false),(true||false),(true||false),(true||false), (true||false),(true||false)))]<up_pressed>false
- R6c.mcf
  - If !RIGHTMOST && DOCKED && CALIBRATED && at saved height, down_pressed results in horizontal movement outward (rightward)
  - [true* . comm_control_update(global_state((true||false),(true||false),(true||false),false,true ,true,(true||false),false,false,(true||false),(true||false),(true||false),(true||false),(true| |false),(true||false))) . down_pressed . comm_action_state(global_state((true||false),(true||false),(true||false),(true||false), (true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||fals e),true,(true||false),(true||false),false))]true
- R6d.mcf
  - If !LEFTMOST && DOCKED && CALIBRATED && at saved height, up_pressed results in horizontal movement inward (leftward)
  - [true* . comm_control_update(global_state((true||false),(true||false),false,(true||false),true ,true,(true||false),false,false,(true||false),(true||false),(true||false),(true||false),(true| |false),(true||false))) . up_pressed . comm_action_state(global_state((true||false),(true||false),(true||false),(true||false), (true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||fals e),(true||false),true,(true||false),false))]true
- R7.mcf

    - If CALIIBRATED && DOCKED, up_pressed may never move the bed above saved height.
    - [true* . comm_control_update(global_state((true||false),(true||false),(true||false),(true||false),true, true,(true||false),false, false,(true||false),(true||false),(true||false),(true||false),(true||false),(true||false)))]<true . comm_action_state(global_state((true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),true,(true||false),(true||false),(true||false),(true||false)))>false
- R8.mcf
    - If !DOCKED && !CALIBRATED, up/down_pressed actions never result in horizontal movement.
    - [true* . comm_control_update(global_state((true||false),(true||false),(true||false),(true||false),false, false,(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false)))]<true . (comm_action_state(global_state((true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),true,(true||false),(true||false),false)) || comm_action_state(global_state((true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),true,(true||false),false)))>false
- R9.mcf
    - up/down_pressed actions may never result in the bed moving above or below to two predefined maximum and minimum limits.
    - [true*. comm_control_update(global_state(true,(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false), (true||false), (true||false), (true||false), (true||false), (true||false), (true||false)))]<true . comm_action_state(global_state((false||true),(false||true),(false||true),(false||true),(false||true),(false||true),(false||true),(false||true),(false||true),(false||true),true,(false||true),(false||true),(false||true),(false||true),(false||true)))>false
    - &&
    - [true*. comm_control_update(global_state((true||false),true,(true||false),(true||false),(true||false),(true||false),(true||false),(true||false),(true||false), (true||false), (true||false), (true||false), (true||false), (true||false), (true||false)))]<true . comm_action_state(global_state((false||true),(false||true),(false||true),(false||true),(false||true),(false||true),(false||true),(false||true),(false||true),(true||false),true,(false||true),(false||true),(false||true),(false||true),(false||true)))>false
- R10.mcf

- Deadlock freeness
- [true*]<true>true