

Modeling of Mission Critical Systems

Mini Projekt

Hub In Safe Mode

Afleveringsdato: 17. maj 2011

Udarbejdet af

20034665 Klaus Petersen

Indholdsfortegnelse

System Description.....	3
Purpose of the VDM Model.....	3
Hub In Safe Mode Requirements	5
Dictionary	6
Classes and Types.....	6
Operations.....	6
System Buildup	7
Enviroment	7
OperatingPanel.....	7
MainShaftController and Brake.....	8
Hub and Speaker	10
HubController and Mode	10
WindMeasurementController	13
Model Quality.....	14
Internal Consistency	14
External Consistency.....	14

System Description

Figure 1 illustrates a wind turbine and some basic elements of the wind turbine that must be known to understand hub in safe mode. The wind turbine can be brought into a safe mode called hub in safe mode. The purpose of the hub in safe mode is to allow a service technician to physically enter the hub of the wind turbine to carry out maintenance. For the service technician to do so without risking his life, it must be guaranteed that the main shaft connected to the rotor of the wind turbine, is at a complete stand still and securely locked.

To put the wind turbine into hub in safe mode the following must happen:

1. The service technician must enter the wind turbine and via the ground or nacelle operating panel enter hub in safe mode. This will only be possible if the current wind speed is at or below 15m/s.
2. The operator panel must display: "Entering Hub In Safe Mode" until hub in safe mode has been reached.
3. When hub in safe mode has been reached the operator panel must display a hub in safe mode alarm and the door to the hub must be unlocked and opened.

For the wind turbine to enter hub in safe mode, it must be able to stop rotation of the main shaft connected to the rotor on request. To do so the following brake sequence shall be applied:

1. Brake the rotor using the generator and by pitching the rotor blades. The generator is capable of slowing down the main shaft from high speeds, it is not capable of guaranteeing a complete stand still.
2. When the main shaft is below 100 RPM a disc brake must be applied. The purpose of the disc brake is to further slow down the main shaft to a position where it can be locked.
3. When the main shaft is below 1 RPM a lock must be applied. The purpose of the lock is to secure the main shaft in a position where it can be guaranteed not to move.

When the wind turbine has been placed in hub in service mode, the service technician can enter the hub via the open unlocked door to the hub. To leave hub in safe mode the service technician must:

1. Leave the hub
2. Acknowledge the hub in safe mode alarm via the operator panel.
3. System must close and lock the door to the hub.

When the above conditions are met an audible alarm shall sound for one minute. In the unlikely event that someone is still in the hub, that someone has one minute to press the emergency stop in the hub. If the emergency stop is pressed within this minute, leaving hub in safe mode is aborted and the door to the hub is unlocked and opened. If nobody presses the emergency stop button then the hub in safe mode alarm must be removed and both the main shaft lock and all brakes must be released.

Purpose of the VDM Model

The purpose of the model is to clarify the rules governing entering and leaving hub in safe mode.

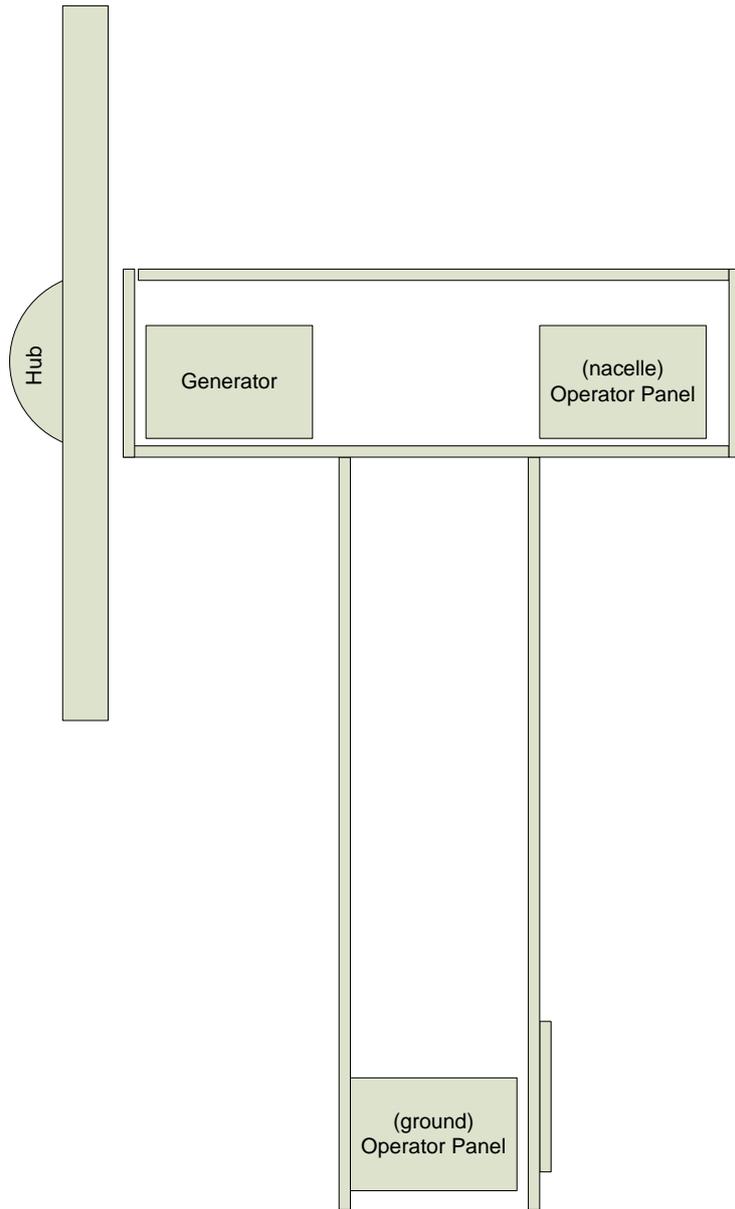


Figure 1

Hub In Safe Mode Requirements

1. A computer-based system is to be developed to manage the rules governing entering and leaving hub in safe mode of the wind turbine.
2. It must only be possible to enter hub in safe mode if current wind speed is at or below 15 m/s.
3. For the wind turbine to enter hub in safe mode the main shaft must be completely stopped and locked.
4. To place the wind turbine in hub in safe mode the following sequence must be applied:
 - a. A service technician must order the wind turbine to hub in safe mode via one of the operating panels.
 - b. The operator panels must display the alarm "Entering Hub In Safe Mode", while entering hub in safe mode.
 - c. When hub in safe mode has been reached the operator panels must display the alarm "Hub In Safe Mode"
 - d. System must unlock and open door to hub.
5. To brake and lock the wind turbine main shaft, the following sequence must be applied:
 - a. Brake rotor by applying generator brake and by pitching blades.
 - b. When main shaft speed is below 100 RPM apply disc brake.
 - c. When main shaft speed is below 1 RPM lock main shaft.
6. For the wind turbine to leave hub in safe mode the following sequence must be applied:
 - a. Acknowledge "Hub In Safe Mode" alarm via one of the operating panels.
 - b. System must close and lock hub door.
 - c. Sound audible alarm for one minute.
 - i. If an emergency stop is pressed during the one minute, abort leaving hub in safe mode, unlock and open door to hub.
 - d. Remove "Hub In Safe Mode" alarm
 - e. Unlock and release brakes on main shaft.
7. The operating panel must at all times display the current mode of the wind turbine.

Dictionary

The following presents classes and main operations of the wind turbine.

Classes and Types

WindTurbine: The entire system

OperatingPanel: The operating panel is used to enter hub in safe mode, leave hub in safe mode and to display current mode of the wind turbine.

Brake: The different types of brakes i.e. generator, pitching blades, disc brake and mechanical lock is represented by different instances of this class.

MainShaftController: The main shaft is represented by a main shaft controller that can brake, lock, release the lock and measure current RPM of the main shaft.

Hub: Controls access to the hub.

Lock: The hub contains a door that can be closed and locked or open and unlocked.

Speaker: The hub contains a speaker that can sound an audible alarm.

HubController: The hub controller can put the hub into a safe mode or exit the hub from the safe mode.

WindMeasurementController: Wind measurement can deliver a current wind speed.

Operations

Enter Hub In Safe Mode: When a service technician orders the turbine to enter hub in safe mode.

Leave Hub In Safe Mode: When a service technician orders the turbine to leave hub in safe mode.

Display Current Mode: For the operating panel to display the current mode of operation.

Apply Brake: Apply brakes to main shaft.

Release Brake: Release brakes on main shaft.

Apply Lock: To lock main shaft into a secure position or to lock access to the hub.

Release Lock: To release the lock of main shaft or to unlock access to the hub

GetRPM: Get the rotational speed of the main shaft in RPM.

GetWindSpeed: Get the current wind speed in m/s.

System Buildup

Figure 2 gives an overview of classes, associations and public operations involved in modeling the hub in safe mode for a wind turbine.

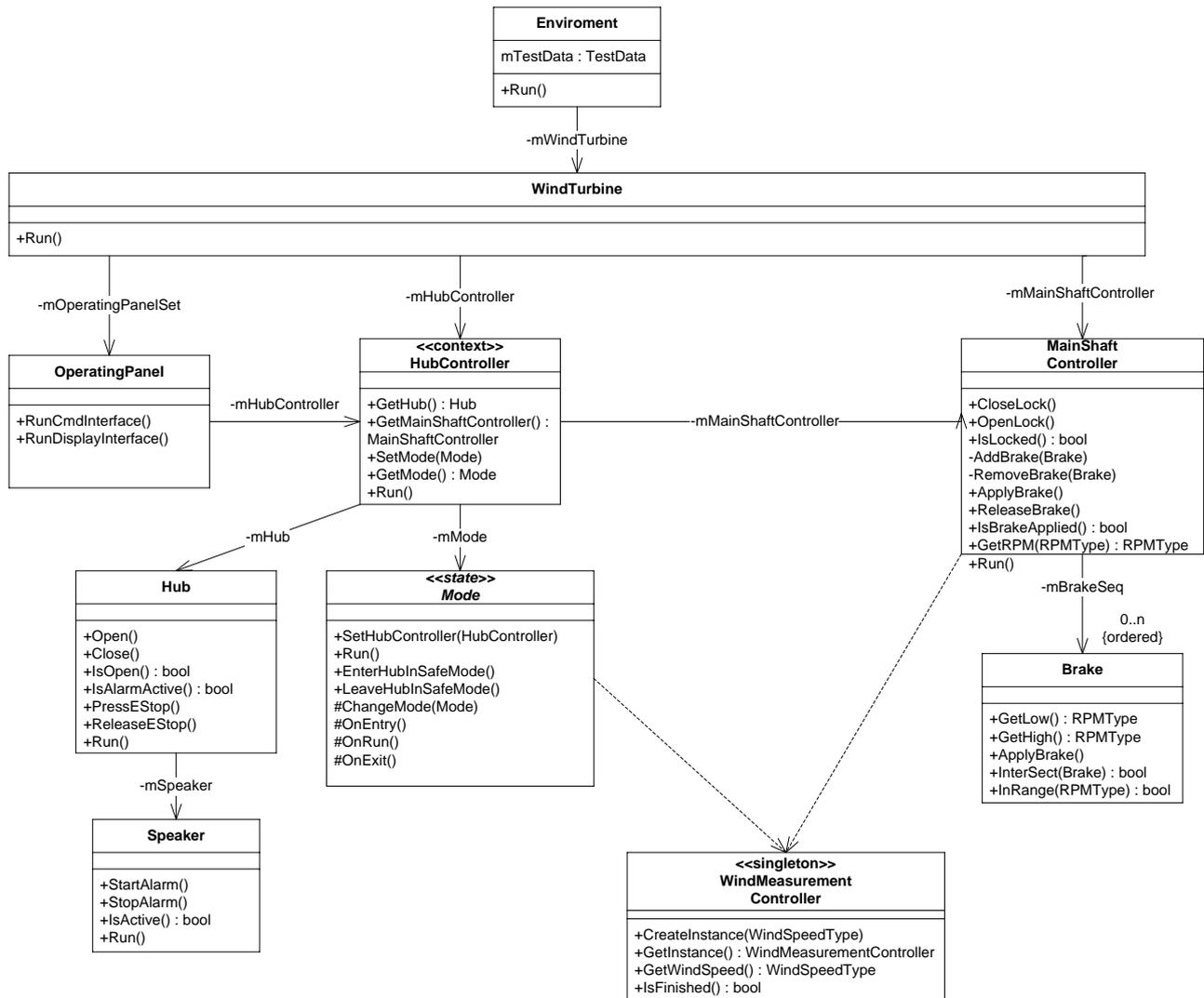


Figure 2

Environment

The Environment class defines a static Run() operation and a sequence of TestData that can be invoked for debug of WindTurbine.

OperatingPanel

A WindTurbine contains two operating panels, but since the purpose of the model is to clarify the rules governing entering and leaving hub in safe mode, this is abstracted away and hence only one instance of OperatingPanel is going to model the user interface to the wind turbine.

The OperatingPanel takes a sequence of CmdType = <E> | <L> | <N> and executes these. The following explains the meaning of CmdType:

- <E> - means service technician has ordered the wind turbine to (E)nter hub in safe mode
- <L> - means service technician has ordered the wind turbine to (L)eave hub in safe mode
- <N> - means (N)othing has been ordered via the operating panel.

An example of a sequence could be [<E>, <N>, <N>, <N>, <L>, <N>, <N>, <N>, <N>].

WindTurbine uses this command interface to the OperatingPanel to set up a test sequence.

MainShaftController and Brake

The MainShaftController controls the main shaft that must be stopped or released by means of brakes and a lock. Both brakes and the lock are modeled by means of class Brake.

A Brake is modeled with a range in which it can effectively brake. This range is provided in the constructor as low limit and a high limit.

```
public Brake : MainShaftController`RPMTType * MainShaftController`RPMTType ==> Brake
Brake(low, high) ==
(
    mLowLimit := low;
    mHighLimit := high
)
```

To apply the brake the Brake class declares the operation ApplyBrake.

```
public ApplyBrake : MainShaftController`RPMTType ==> MainShaftController`RPMTType
ApplyBrake(rpm) ==
(
    if (InRange(rpm)) then
        return mLowLimit
    else
        return rpm
)
```

ApplyBrake takes as input the parameter rpm, and if it is in range defined by low limit and high limit then it returns the low limit. From the callers perspective of this operation it looks like ApplyBrake reduces the supplied rotational speed by braking it from parameter rpm to low limit in time T. Though a very simple model this models a brake.

To model the different types of brakes the following instances of Brake is created

- Blade Pitch - Brake instance with low limit 100 and high limit 500
- Generator - Brake instance with low limit 50 and high limit 100
- Disc Brake - Brake instance with low limit 1 and high limit 50
- Lock - Brake instance with low limit 0 and high limit 1

Rotational speed of the main shaft is modeled as wind speed x 10. As an example a wind speed of 15 m/s equals 150 rpm on the main shaft, this means that the above instances of brakes will be able to reduce rotational speed to 0 in the following sequence.

- Blade pitch will reduce main shaft rotation from 150 RPM to 100 RPM,
- Generator will reduce main shaft rotation from 100 RPM to 50 RPM,

- Disc Brake will reduce main shaft rotation from 50 to 1 RPM,
- Lock will reduce main shaft rotation from 1 to 0 RPM.

To model the above, the brake instances are ordered into a sequence. The brake capable of breaking at the highest speed at index 1 and next highest speeds at index 2 and so forth. To reduce rotational speed the brake operating at the highest speed is applied first. If this First rule is not obeyed the system will not always be able to brake to 0 RPM in time T. The second rule that must be obeyed in the model is that each brake instance must intersect with its neighbor to its low and high limit (except for the brakes at the beginning and end of the sequence).

The rules are checked by adding an invariant to the sequence of brakes in MainShaftController i.e.

```
mBrakeSeq :seq of Brake := [];
    inv BrakeSeqInv (mBrakeSeq);

...
functions
static BrakeSeqInv : seq of Brake -> bool
BrakeSeqInv (brakeSeq) ==
    forall i in set inds brakeSeq &
        i>1 => brakeSeq(i-1).GetLow() = brakeSeq(i).GetHigh();
```

The invariant checks that if two or more brakes are present in the sequence then the low limit of brake at index 1 must equal high limit of brake at index 2 and so forth.

In addition a precondition to the operation AddBrake in MainShaftController is added i.e.

```
operations
Brake ==> ()
AddBrake (brake) ==
    mBrakeSeq := mBrakeSeq ^ [brake]
pre BrakeSeqInv (mBrakeSeq ^ [brake]);
```

The precondition checks that the BrakeSeqInv isn't violated when adding a new brake.

Effectively the precondition and the invariant means that AddBrake has to be called in the correct sequence and intersect correct at the brake borders i.e. the following will work.

```
--Blade Pitch
AddBrake (new Brake (100, 500));
--Generator
AddBrake (new Brake (50, 100));
--Disc Brake
AddBrake (new Brake (1, 50));
--Lock
AddBrake (new Brake (0, 1));
```

As an example of the exist expression the RemoveBrake operation has been added to the MainShaftController. It looks like the following:

```
RemoveBrake (brake) ==
    mBrakeSeq := [mBrakeSeq(i) | i in set inds mBrakeSeq & not mBrakeSeq(i).IsEqual (brake)]
pre (exists i in set inds mBrakeSeq & mBrakeSeq(i).IsEqual (brake)) and
    BrakeSeqInv ([mBrakeSeq(i) | i in set inds mBrakeSeq & not mBrakeSeq(i).IsEqual (brake)]);
```

Notice how the precondition checks that the brake reference exists in `mBrakeSeq` before the brake reference is removed from the sequence.

As noted earlier, the Brake functionality is modeled in a very simple manor and does not take into account that the time to brake is affected by the rotational speed of the main shaft or that each brake may have other different brake properties. This is however not needed as the purpose of the model is to show that we can brake and lock the main shaft under the required circumstances.

Hub and Speaker

The hub is modeled by class `Hub`. It contains operations to model the door and lock that must control access to the hub plus an emergency stop and a speaker.

The speaker when started will be active for $3T$ (the timeunit T is the same as one run of `WindTurbine.Run()`). This is to model the requirement that an audible alarm has to sound for one minute when the hub door is closed.

HubController and Mode

`HubController` contains associations for:

- `MainShaftController`: to be able to order the main shaft to e.g. brake and lock
- `Hub`: to be able to order the hub to e.g. open or close the door to the hub
- `Mode`: the events (modeled as operations `EnterHubInSafeMode` and `LeaveHubInSafeMode`) will behave according to which mode the hub controller is in.

The mode functionality is modeled as a state machine according to the diagram shown in Figure 3.

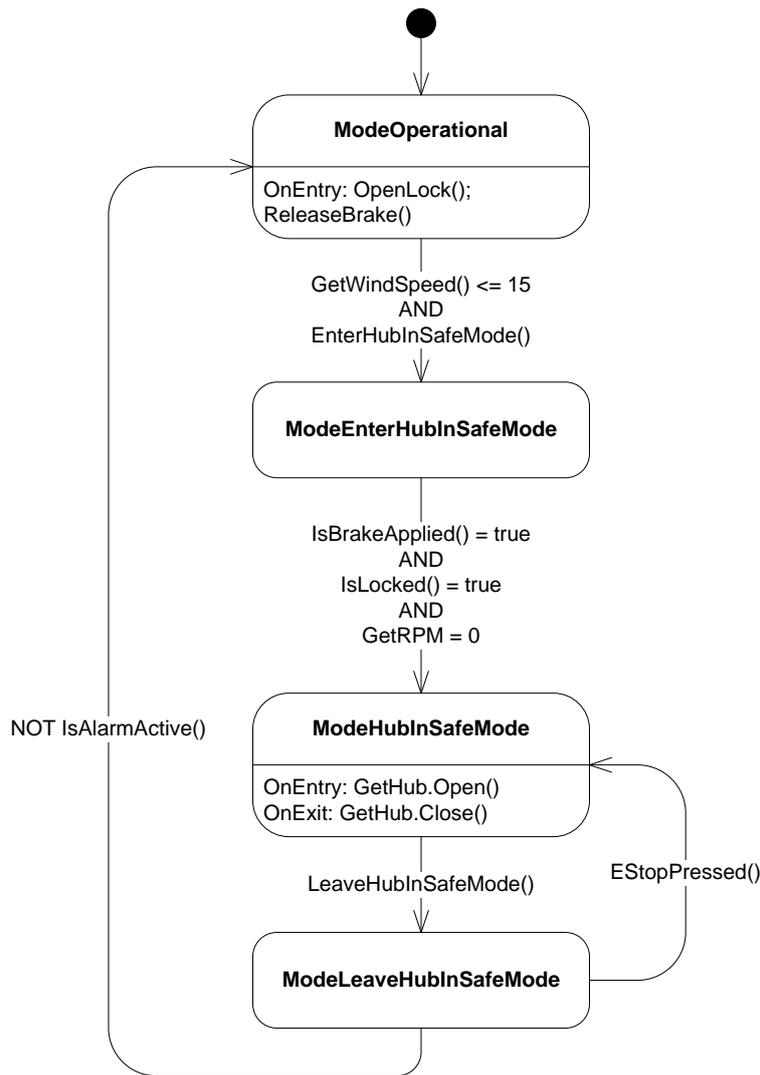


Figure 3

The state machine is implemented using a state pattern. This allows each mode to have an OnEntry, OnRun and OnExit operation. This separation of the state machine into operations is practical e.g. OnEntry to hub in safe mode, where we need to make sure that the hub door is unlocked and open, or OnExit where we need to make sure that the hub door is closed, locked and that an alarm sounds. Furthermore it makes it possible to make an invariant on the instance variable mMode (contains the current mode) that looks like the following (see HubController.vdmp):

```

mMode : Mode;
inv Mode`HubInSafeModeInv (mMode, mMainShaftController.GetRPM(),
mMainShaftController.IsLocked(), mHub.IsOpen());

```

...

```

functions
public static HubInSafeModeInv : Mode * MainShaftController`RPMTType * bool * bool -> bool
HubInSafeModeInv(mode, rpm, isLocked, isOpen) ==

```

```

--if hub in safe mode then RPM must be 0, mainshaft must be locked, hub door must be
unlocked and open
(isofclass(ModeHubInSafeMode, mode) and rpm = 0 and isLocked and isOpen) or
--if not in hub in safe mode hub door must be closed and locked
(not isofclass(ModeHubInSafeMode, mode) and not isOpen);

```

Note that when `mHub.IsOpen()` returns true, the hub is unlocked and open, and vice versa.

To make sure that only the state transitions modeled in Figure 3 are allowed, a precondition on the Mode operation `ChangeMode(Mode)` is specified.

operations

```

protected ChangeMode : Mode ==> ()
ChangeMode(newMode) ==
(
    OnExit();
    newMode.OnEntry();
    mHubController.SetMode(newMode);
)
pre Mode`StateChangeInv(mHubController.GetMode(), newMode) and
let
    mainShaftController = mHubController.GetMainShaftController(),
    hub = mHubController.GetHub()
in
    Mode`HubInSafeModeInv(mHubController.GetMode(), mainShaftController.GetRPM(),
mainShaftController.IsLocked(), hub.IsOpen());

```

...

functions

```

public static StateChangeInv : Mode * Mode -> bool
StateChangeInv(oldMode, newMode) ==
(isofclass(ModeOperational, oldMode) and isofclass(ModeEnterHubInSafeMode, newMode)) or
(isofclass(ModeEnterHubInSafeMode, oldMode) and isofclass(ModeHubInSafeMode, newMode)) or
(isofclass(ModeHubInSafeMode, oldMode) and isofclass(ModeLeaveHubInSafeMode, newMode)) or
(isofclass(ModeLeaveHubInSafeMode, oldMode) and isofclass(ModeOperational, newMode)) or
(isofclass(ModeLeaveHubInSafeMode, oldMode) and isofclass(ModeHubInSafeMode, newMode));

```

Figure 4 shows the class diagram of the classes involved in implementing the modes of the wind turbine.

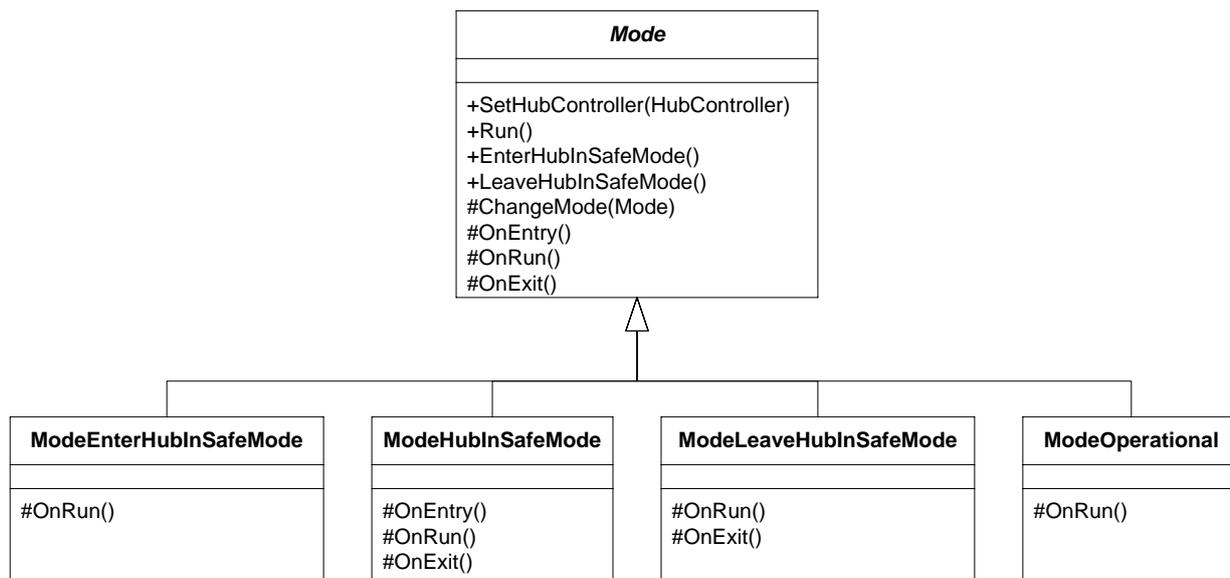


Figure 4

WindMeasurementController

WindMeasurementController models the component in the wind turbine that handles measurement of wind speed. Because wind speed is needed in several places in the model and because there should only be one instance modeling wind speed measurements, WindMeasurementController is modeled as a singleton.

How wind speed is measured is beside the point of the model, the purpose of WindMeasurementController is purely to provide the model with a sequence of wind speeds.

This done through GetWindSpeed, that returns the head of the wind speed sequence i.e.

```

public GetWindSpeed : () ==> WindSpeedType
GetWindSpeed() ==
    return hd mWindSpeedSeq
pre mWindSpeedSeq <> [];
  
```

Note the precondition which checks that the empty sequence is not passed to partial operation hd.

To let WindMeasurementController progress to the next wind speed sample it defines a Run operation.

```

public Run : () ==> ()
Run() ==
(
    if len mWindSpeedSeq >= 1 then
        mWindSpeedSeq := tl mWindSpeedSeq;
);
  
```

As it can be seen the Run operation when executed removes the head of wind speed sequence.

When the wind speed sequence is empty IsFinished will return true and the model will finish execution.

```

public IsFinished : () ==> bool
IsFinished() ==
    return mWindSpeedSeq = [];
  
```

Note that `IsFinished` is used in the `WindTurbine Run` loop.

Model Quality

To gain confidence in the model created internal consistency have been considered and to some degree external consistency through a short checklist on where each requirement is achieved. This is presented in the following subsections.

Internal Consistency

The developed model compiles and generated proof obligations have been considered. To gain further confidence in the model a debug sequence has been added created an run through the `Environment` class.

External Consistency

- R1 A computer-based system is to be developed to manage the rules governing entering and leaving hub in safe mode of the wind turbine
Considered in `WindTurbine` and associated classes and operations.
- R2 It must only be possible to enter hub in safe mode if current wind speed is at or below 15 m/s.
Considered in class `ModeOperational` see `Mode.vdmp`
- R3 For the wind turbine to enter hub in safe mode the main shaft must be completely stopped and locked.
Considered in class `ModeHubInSafeMode` see `Mode.vdmp` and through invariant on `mMode` see `HubController`.
- R4 To place the wind turbine in hub in safe mode the following sequence must be applied:
1. A service technician must order the wind turbine to hub in safe mode via one of the operating panels.
 2. The operator panels must display the alarm “Entering Hub In Safe Mode”, while entering hub in safe mode.
 3. When hub in safe mode has been reached the operator panels must display the alarm “Hub In Safe Mode”
 4. System must unlock and open door to hub.
- Correct sequence is achieved by state machine implemented by `Mode.vdmp`. Valid state changes are checked as precondition on operation `ChangeState` (see `Mode.vdmp`). Other classes involved `OperatingPanel` and `Hub`.*
- R5 To brake and lock the wind turbine main shaft, the following sequence must be applied:
1. Brake rotor by applying generator brake and by pitching blades.
 2. When main shaft speed is below 100 RPM apply disc brake.
 3. When main shaft speed is below 1 RPM lock main shaft.
- Brake sequence is modeled as sequence of `Brake` instances in `MainShaftController`.*
- R6 For the wind turbine to leave hub in safe mode the following sequence must be applied:
1. Acknowledge “Hub In Safe Mode” alarm via one of the operating panels.
 2. System must close and lock hub door.
 3. Sound audible alarm for one minute.
 - a. If an emergency stop is pressed during the one minute, abort leaving hub in safe mode, unlock and open door to hub.

4. Remove “Hub In Safe Mode” alarm
5. Unlock and release brakes on main shaft.

Correct sequence and valid state changes is achieved by state machine implemented by Mode.vdmp. Other classes involved are OperatingPanel, Hub, Speaker and MainShaftController.

- R7 The operating panel must at all times display the current mode of the wind turbine.
Achieved as output to console by OperatingPanel.