Formal Modelling and Safety Analysis of an Embedded Control System for Construction Equipment: an Industrial Case Study using VDM

Takayuki Mori

Newcastle University, UK
Komatsu Ltd., Japan

20 June 2011

Outline

- Background and motivation
- Case study
  - Informal description of control specifications and safety requirements
  - Formal modelling using VDM++
  - Validation and safety analysis
- Conclusions

Background and motivation

- Komatsu Ltd.
  - Construction and mining equipment manufacturer
  - Founded in Komatsu, Japan, in 1921
  - Main products:
    - Bulldozer
    - Hydraulic excavator
    - Wheel loader
    - Dump truck




My main work

- Development of control systems for wheel loaders
  - Control specifications description
  - Software design
  - Implementation
  - Testing

- Currently studying at Newcastle University
- My research interest
  - Applying formal methods to our development activities to make our control software more reliable

Safety for construction equipment

- Safety is a critical factor for construction equipment
- Safety should be ensured even if a fault has occurred in the system

- To ensure safety…
  Failure Mode and Effects Analysis (FMEA)

FMEA process

1. Identify all potential faults (failure modes)
2. Analyse the effects of each fault
3. Estimate the risk of the fault
4. If the risk is not allowable, consider
   - A way to detect the fault
   - Measures to be taken in case the fault has been detected
5. Re-estimate the risk

Motivation

- FMEA is not an easy task
  - Usually, dozens of potential faults in one controller
  - A measure against a fault might affect various parts of the control system
- The research aims to:
  - Describe the specifications of fault detection and measures **formally** (using formal modelling notation VDM++)
  - Check if the specifications are consistent and safety is ensured

Outline

- Background and motivation
- Case study
  - Informal description of control specifications and safety requirements
  - Formal modelling using VDM++
  - Validation and safety analysis
- Conclusions

Modelling target

- A part of a transmission control system for wheel loaders

  “Specifications of detecting the direction lever position”

Modelling target

- Transmission control system

Direction lever
Speed lever

Transmission controller

Vehicle speed
Engine speed
…

Transmission valves

Modelling target

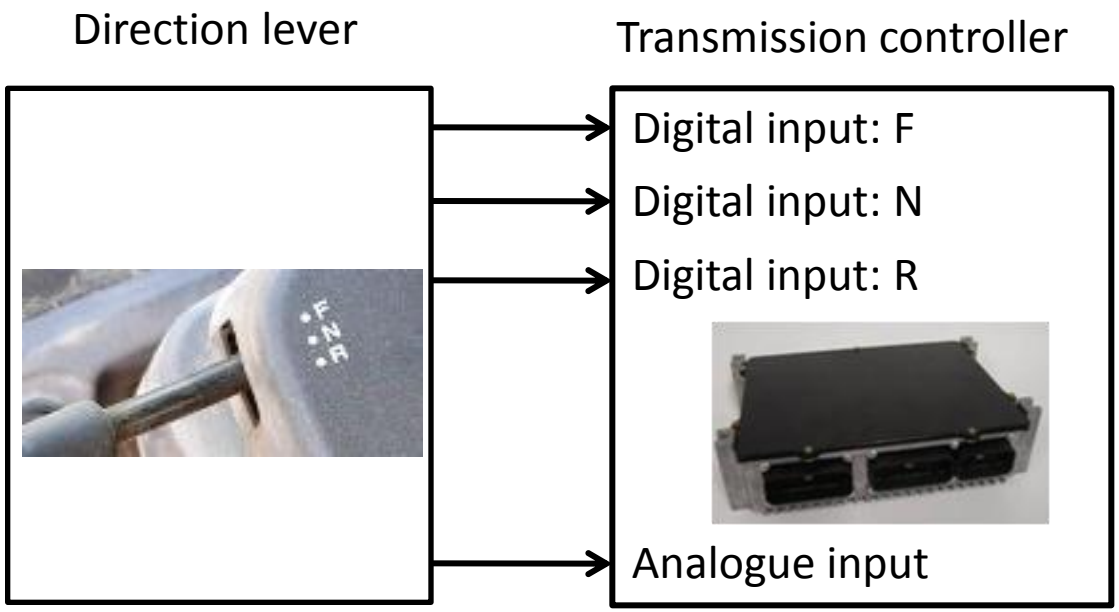
- Detecting the direction lever position

Direction lever

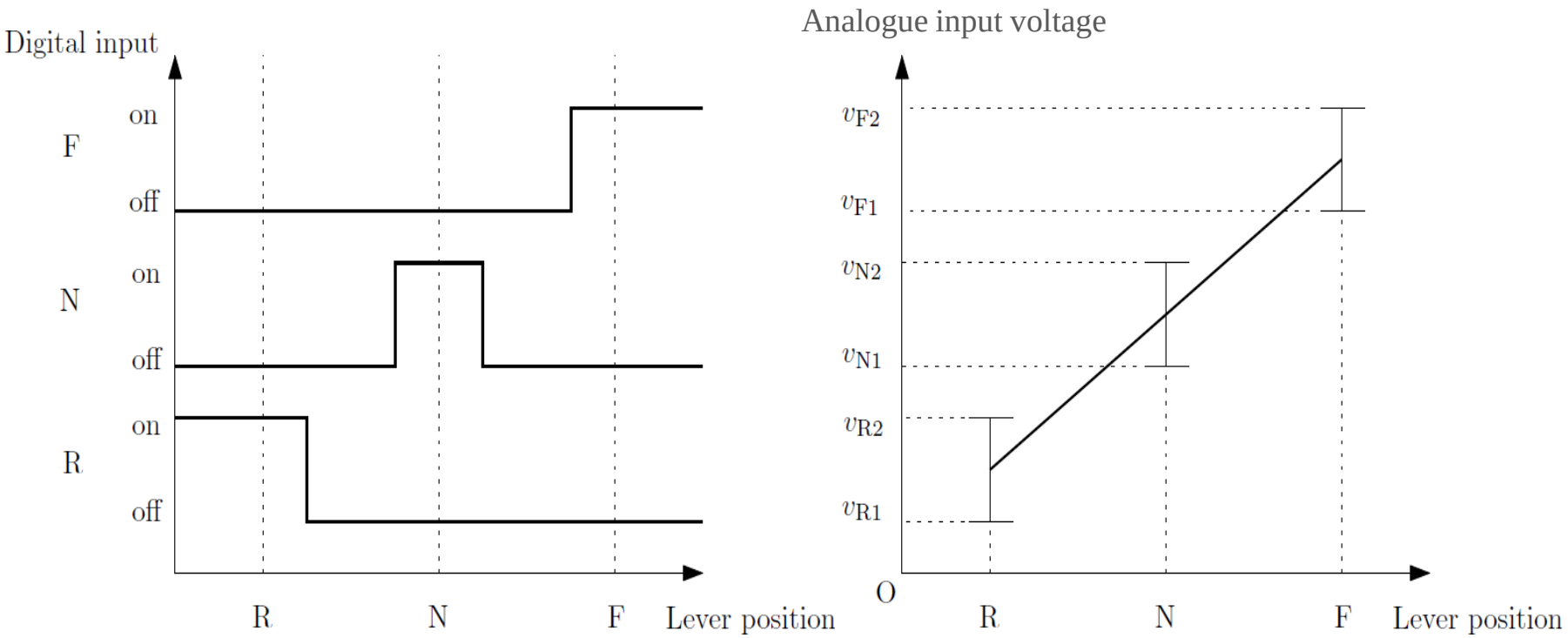
Transmission controller

  - Moving direction is frequently switched
  - Detecting the lever position is crucial for safety
  - The scale and complexity are moderate

System diagram



Electrical characteristics



- Open-circuit of digital input and holding the lever in the middle position cannot be distinguished
- Detected positions by digital and analogue might differ

Control specifications

- Specifications of detecting the direction lever position
  1. Normally, digital input is valid.
  2. If a fault has been detected in digital input, analogue input becomes valid.
  3. If analogue input also has a fault, the lever position is recognised as N.
  4. If digital input has recovered from the fault, digital input should be valid again. However, analogue input remains valid unless the positions detected by digital and analogue input are consistent with each other.

Lever detection by digital input

No.	Digital input signals			Detected lever position
	R	N	F	
1	○			R
2		○		N
3			○	F
4				Undefined. Obey fault detection and measure.
5	○	○		Undefined. Obey fault detection and measure.
6	○		○	
7		○	○	
8	○	○	○	

○ : on, blank: off

Short-circuit to power

Open-circuit or short-circuit to ground or
the direction lever is in the middle position

Fault detection and measures

- Possible Faults of the System
  - Open-circuit, short-circuit or improper operation
- An example

Fault mode	Digital input: open-circuit or short-circuit to ground
Error state	All digital input signals F, N and R are "off".
Fault detecting time	$t_{1\mathrm{f}}$ seconds
Measure before fault confirmation	Keep the detected lever position before the error state.
Measure after fault confirmation	Obey the detected lever position by the analogue input.
Recovery state	Only one digital input signal F, N or R is "on".
Recovery detecting time	$t_{1\mathrm{r}}$ seconds
Measure after fault recovery	Keep obeying the detected lever position by the analogue input until it becomes consistent with that by the digital input. After the consistency, obey the detected lever position by the digital input.

Safety requirements

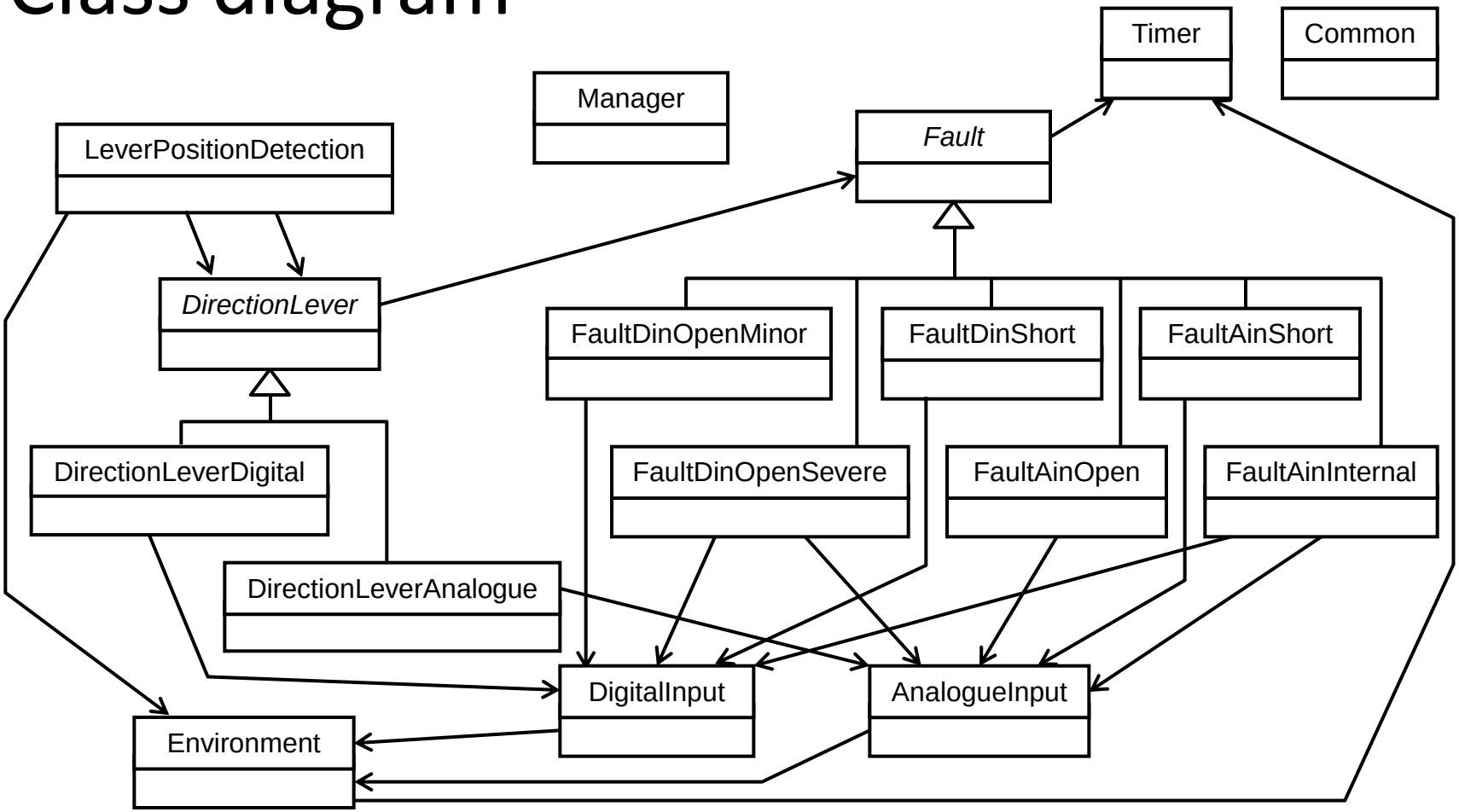
R1: If any fault occurs in the system, the detected position of the direction lever must be consistent with the actual lever position or recognised as neutral (N).

R2: If any fault occurs in the system, the detected position of the direction lever must not change to F or R without lever manipulation by the operator of the vehicle.

Outline

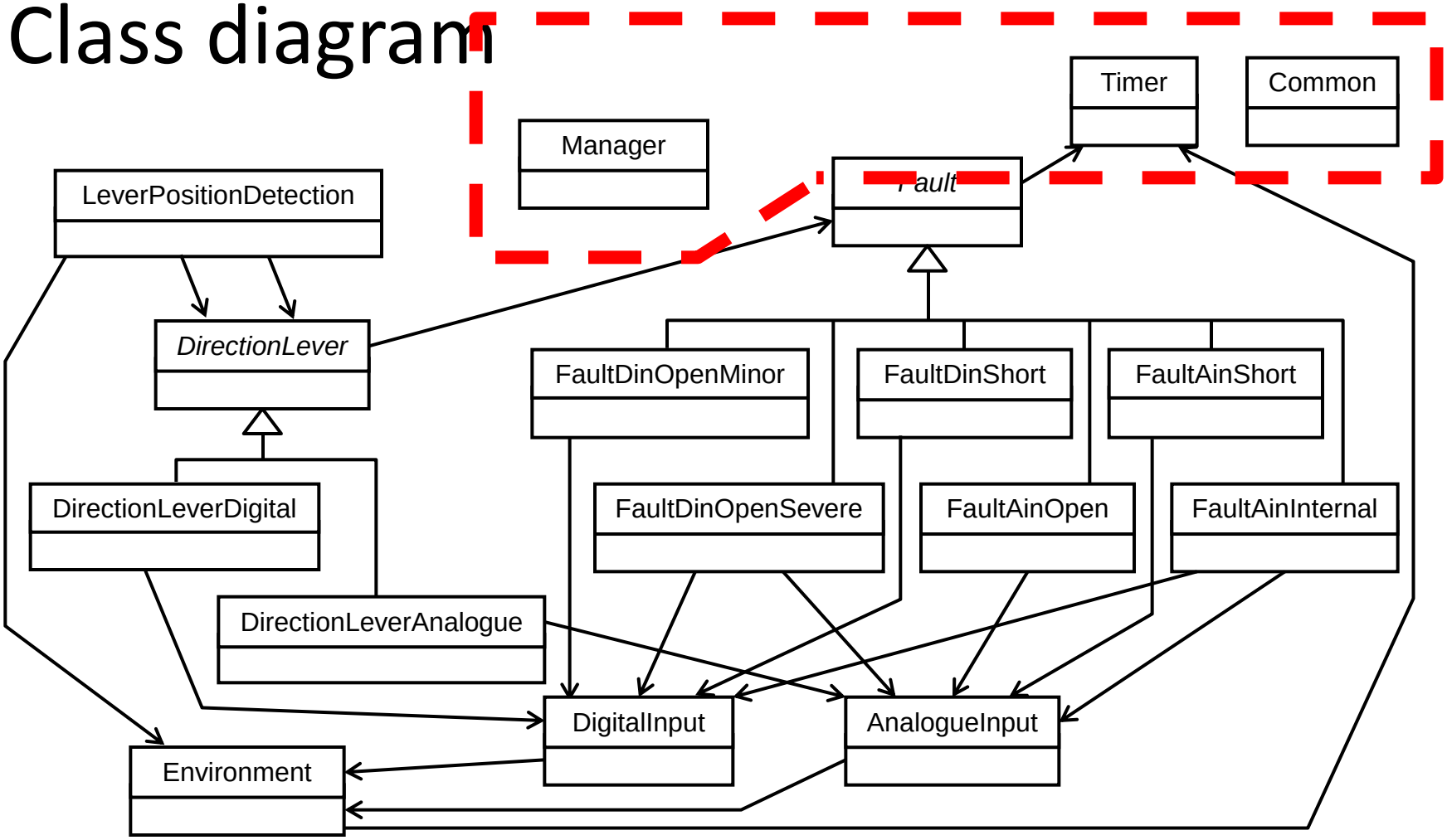
- Background and motivation
- Case study
  - Informal description of control specifications and safety requirements
  - Formal modelling using VDM++
  - Validation and safety analysis
- Conclusions

Class diagram



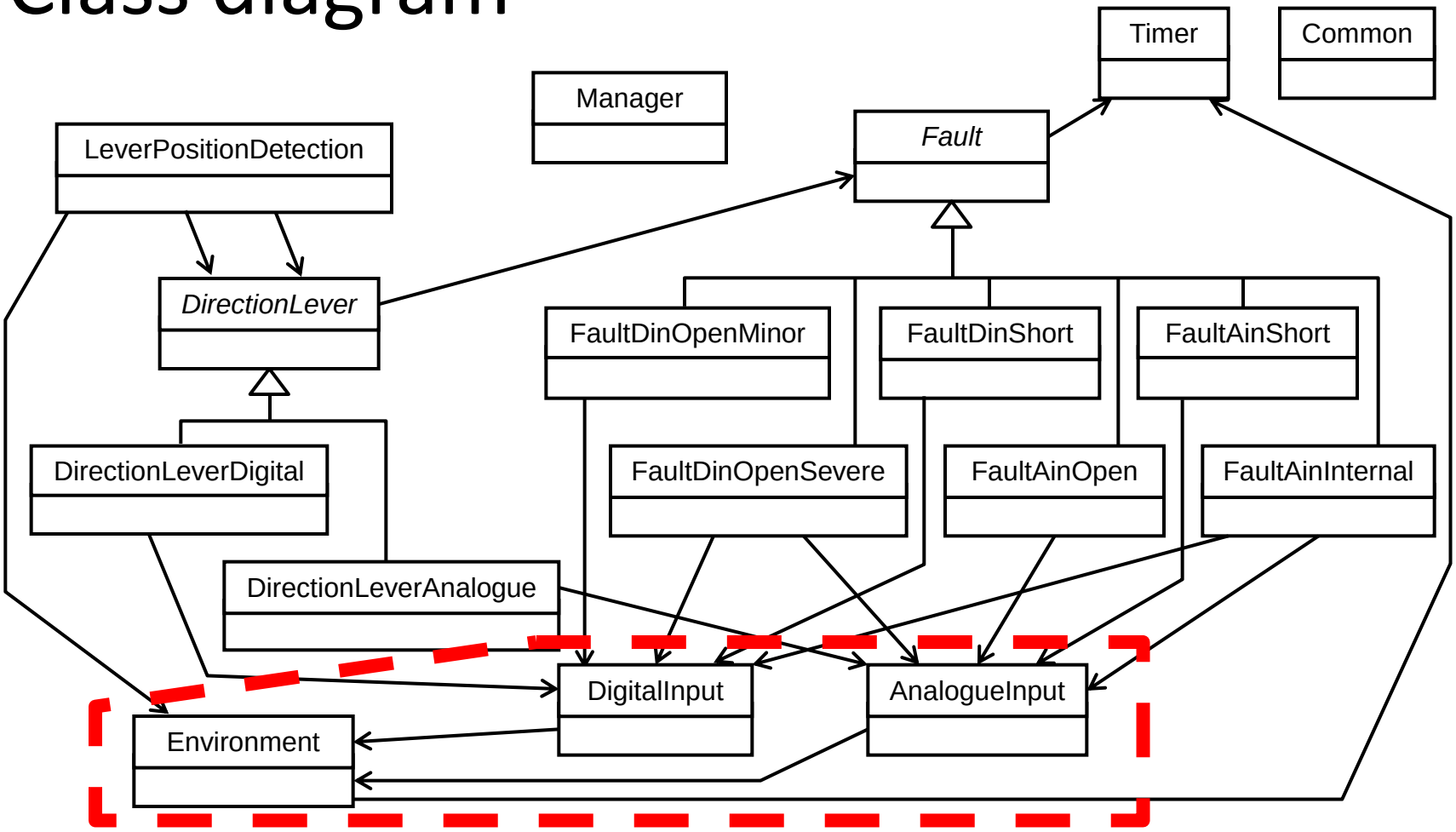
- All association arrows from Manager and inheritance arrows to Common are hidden for legibility

Class diagram



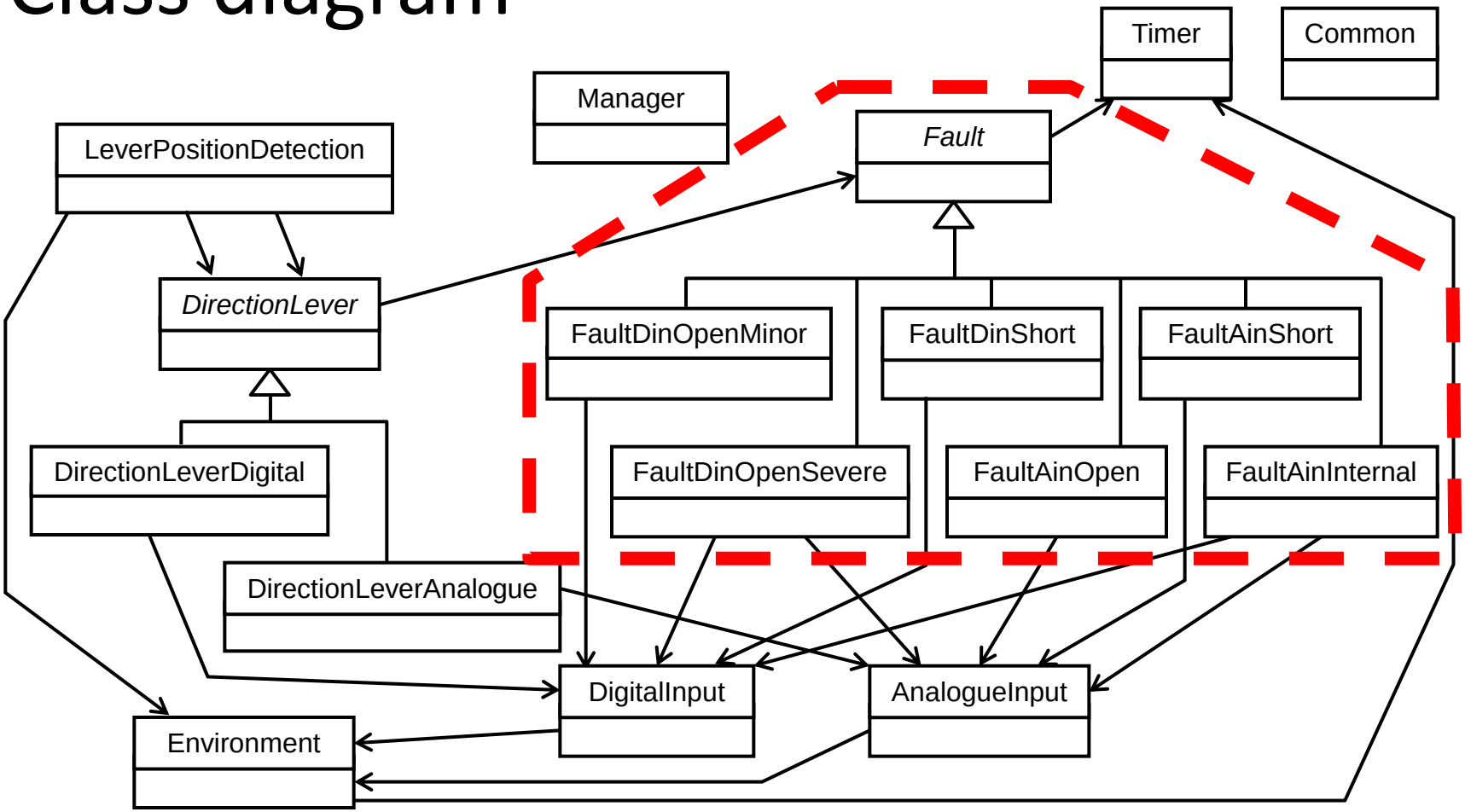
- Managing model execution

Class diagram



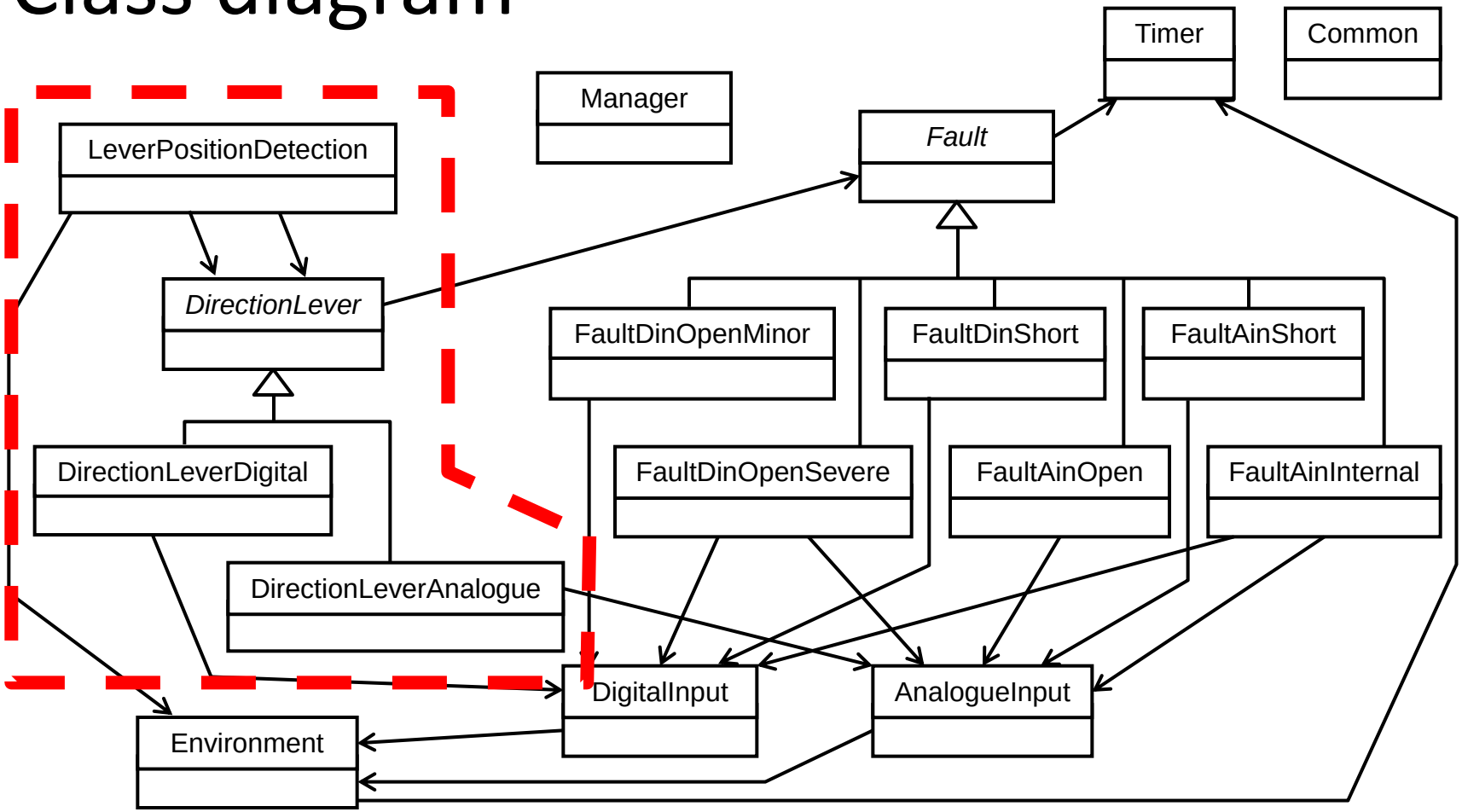
- Input/Output

Class diagram



- Fault detection

Class diagram



- Control logic

Types

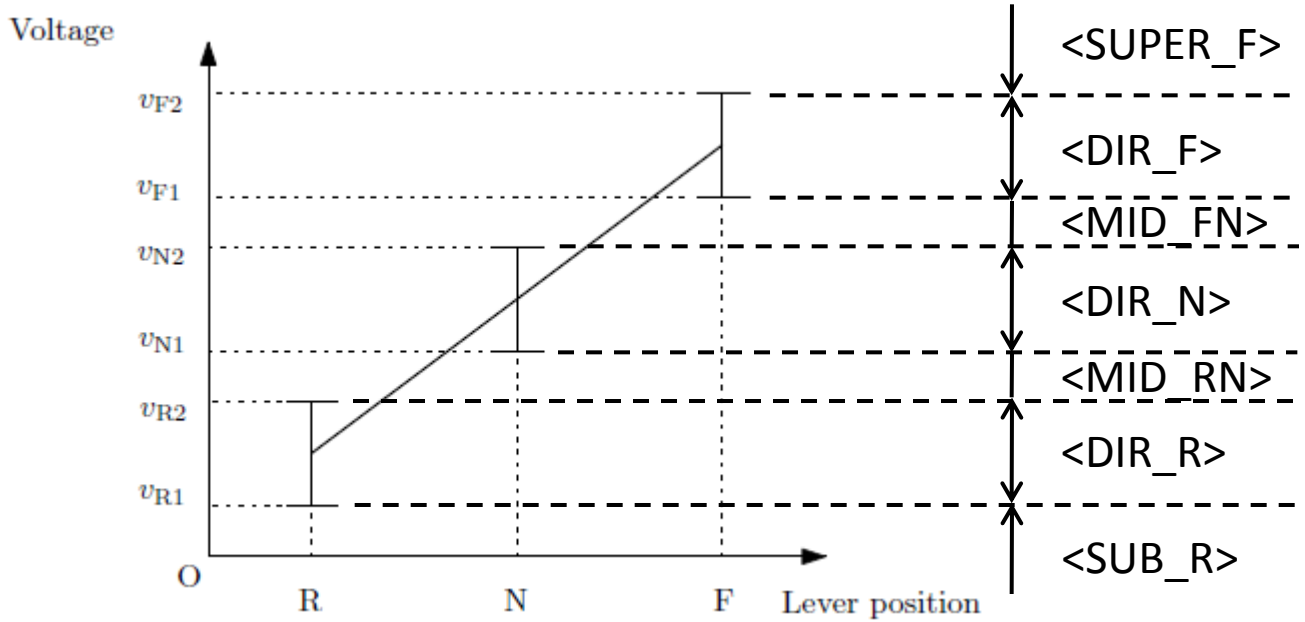
- **Declared in the Common class**

public Time = **nat**;

public Direction = <DIR_F> | <DIR_N> | <DIR_R>;

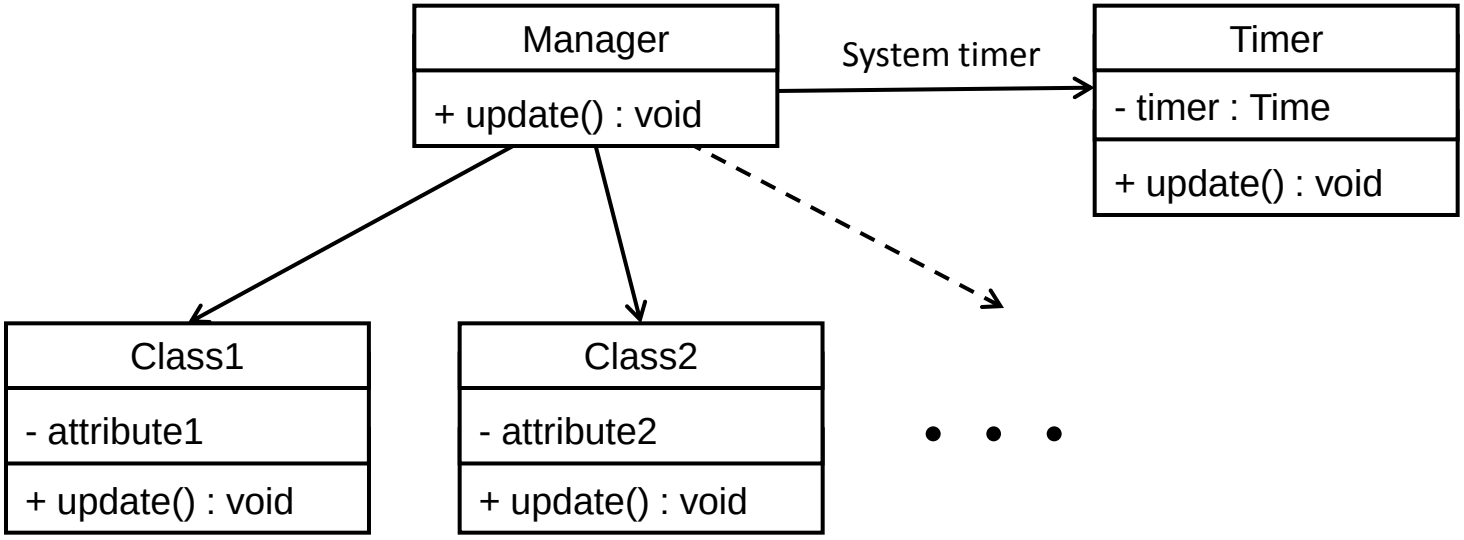
public AinState = Direction
| <SUB_R>
| <MID_RN>
| <MID_FN>
| <SUPER_F>;

Type: AinState

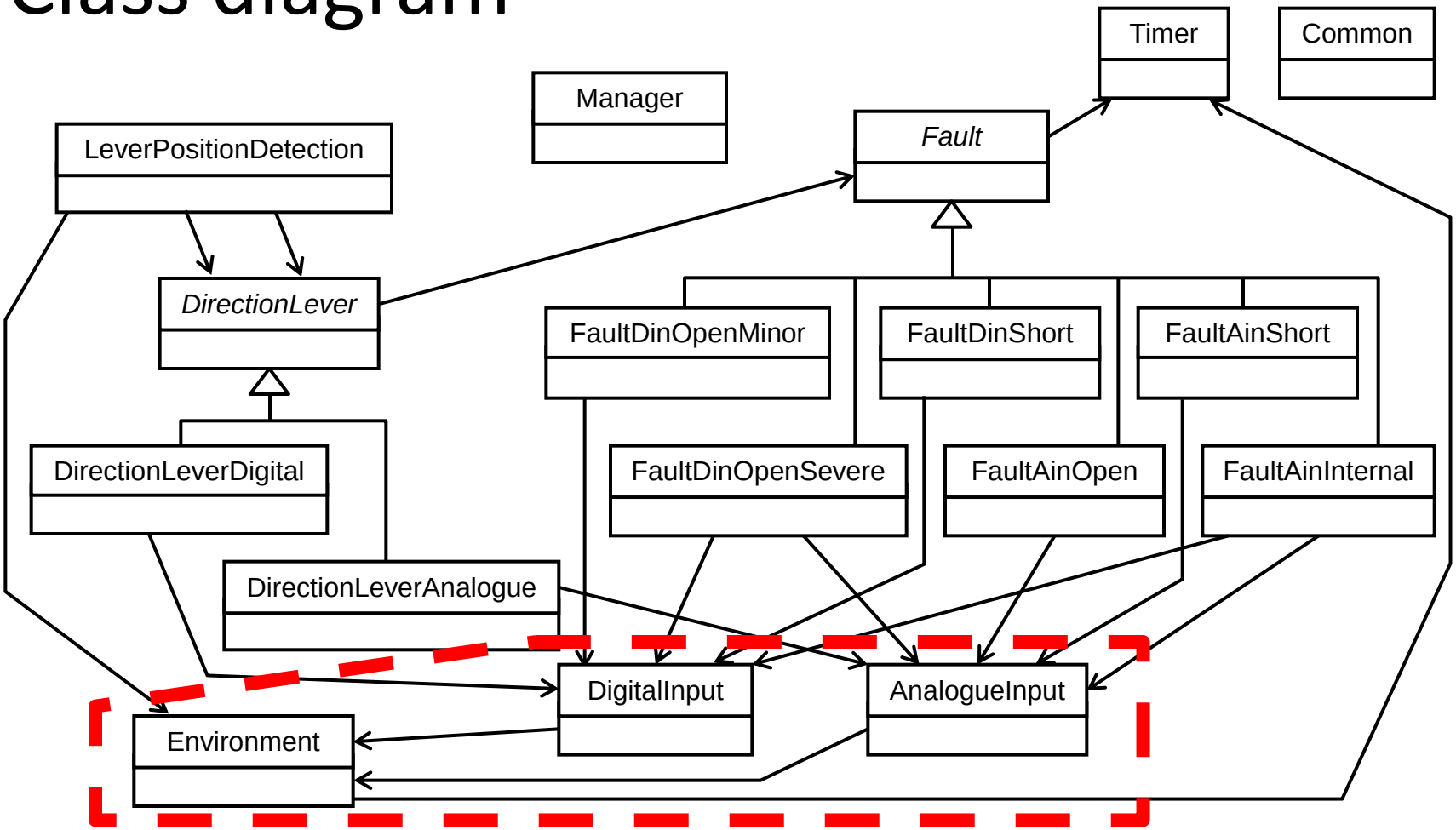


Model execution

- Periodic sequential model
  - Manager class controls the whole model
  - Instantiation and update of objects



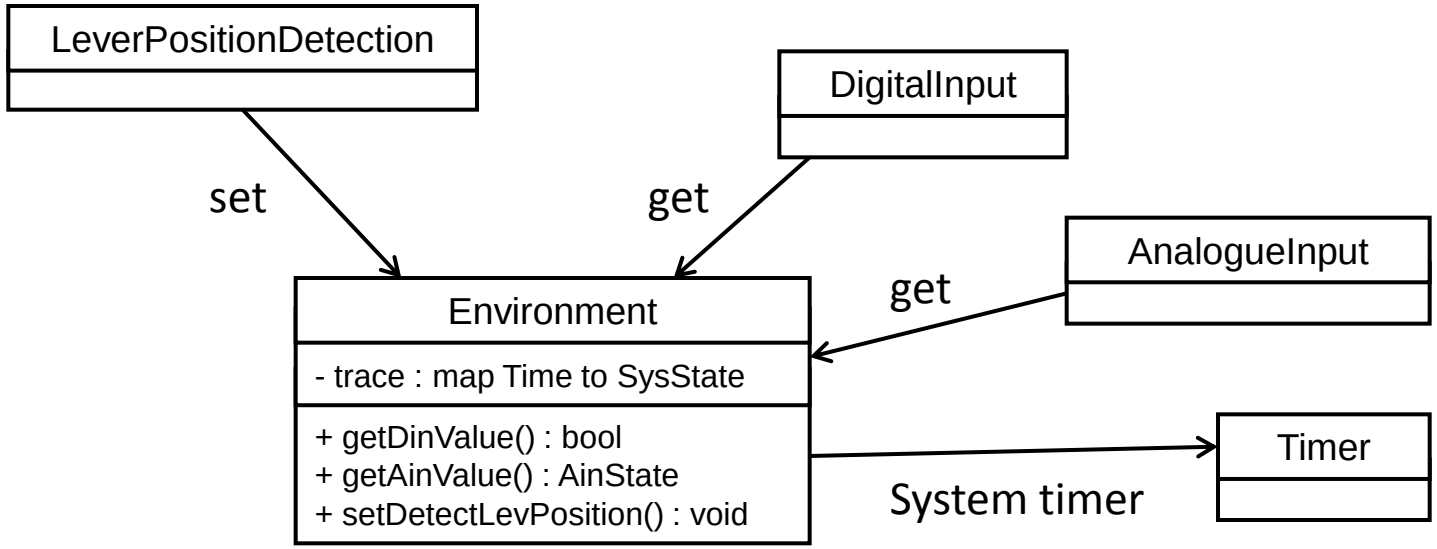
Class diagram



- Input/Output

Environment class

- Components outside the controller
- Provide input to the controller
- Receive output from the controller



Environment class

types

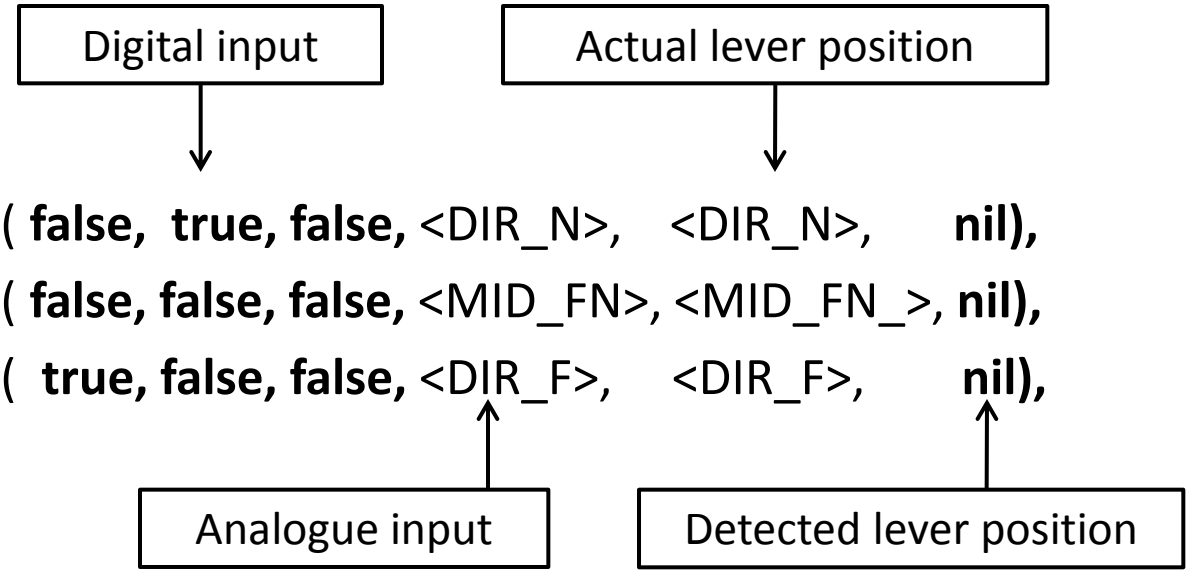
Public SysState :: dinF : **bool**
dinN : **bool**
dinR : **bool**
ain : AinState
levPos : LeverPosition
detectLevPos : [Direction];

instance variables

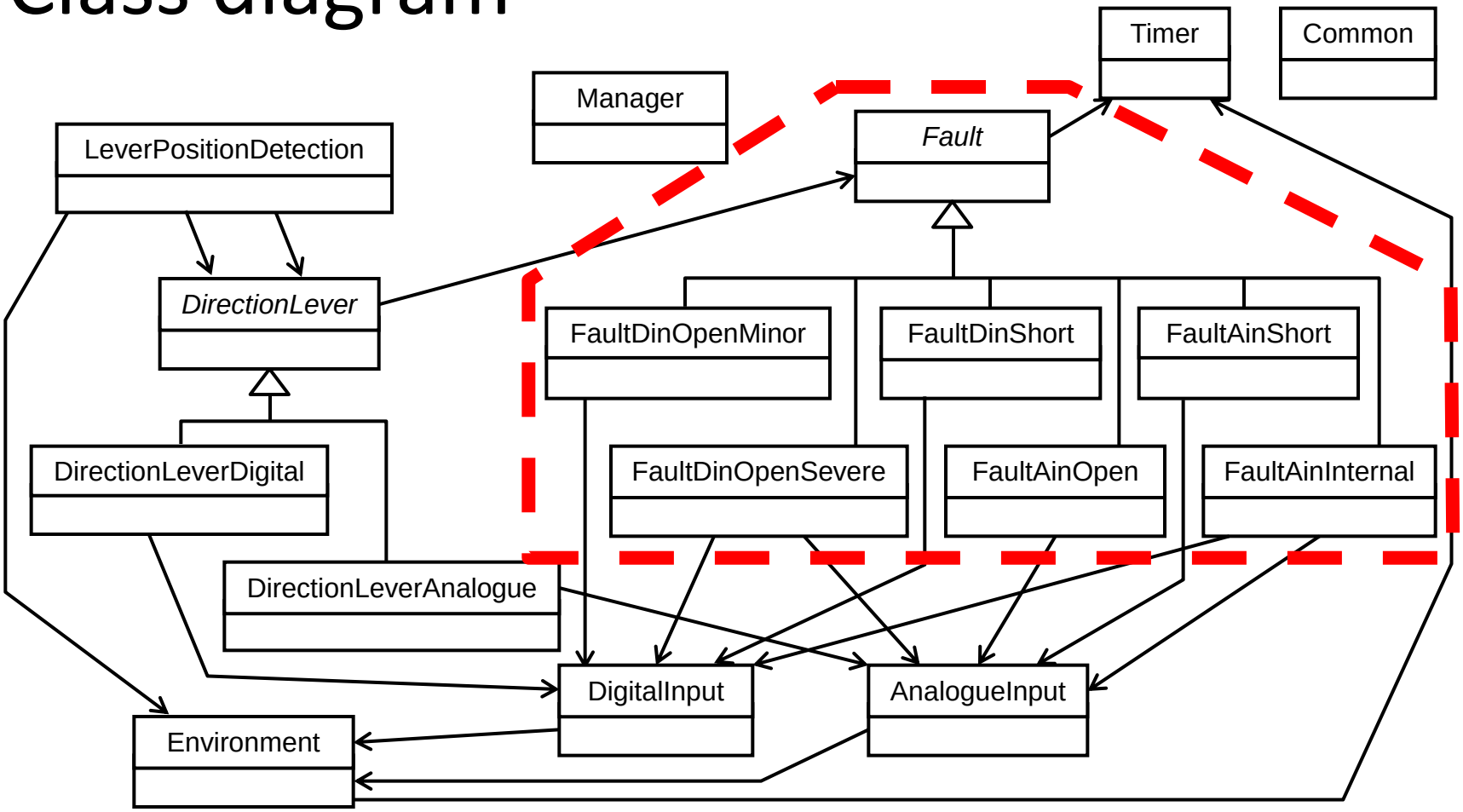
private trace : **map** Time **to** SysState;

Environment class

- An example



Class diagram



- Fault detection

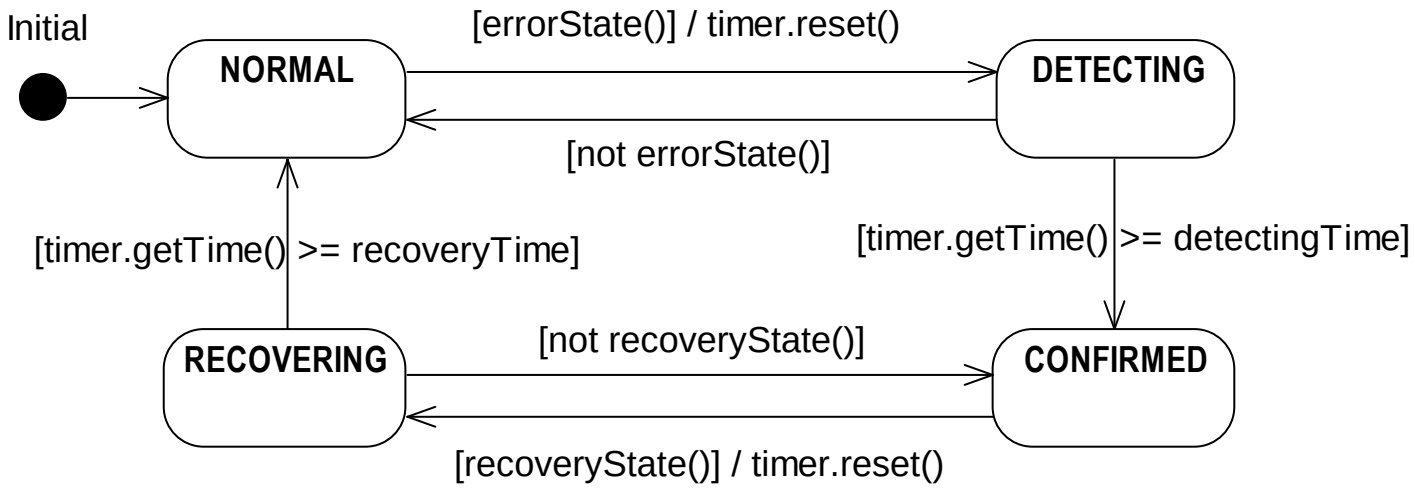
Fault class

- Represent the notion of faults
- Recall the informal description…

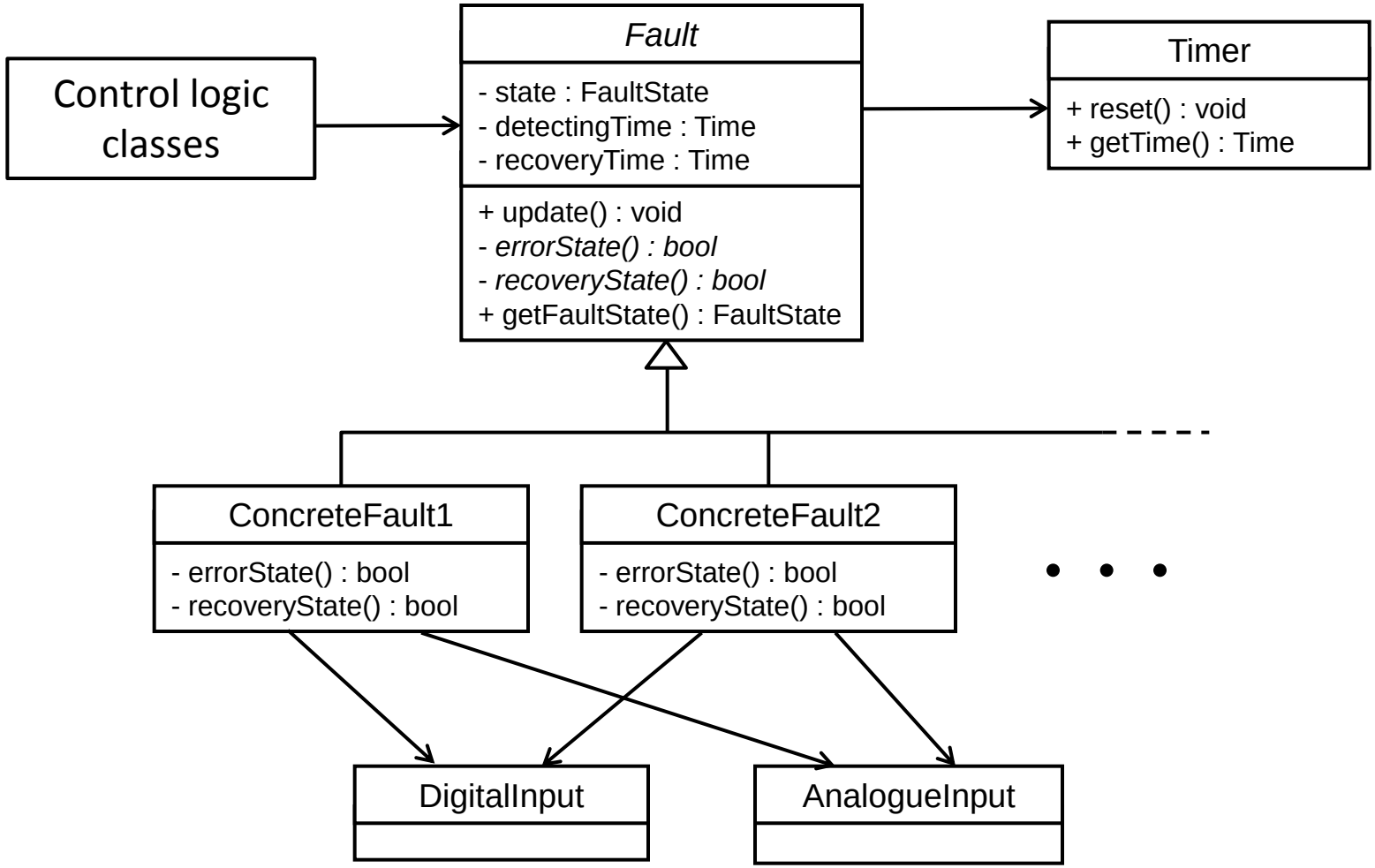
Fault mode	Digital input: open-circuit or short-circuit to ground
Error state	All digital input signals F, N and R are "off".
Fault detecting time	$t_{1\mathrm{f}}$ seconds
Measure before fault confirmation	Keep the detected lever position before the error state.
Measure after fault confirmation	Obey the detected lever position by the analogue input.
Recovery state	Only one digital input signal F, N or R is "on".
Recovery detecting time	$t_{1\mathrm{r}}$ seconds
Measure after fault recovery	Keep obeying the detected lever position by the analogue input until it becomes consistent with that by the digital input. After the consistency, obey the detected lever position by the digital input.

Fault class

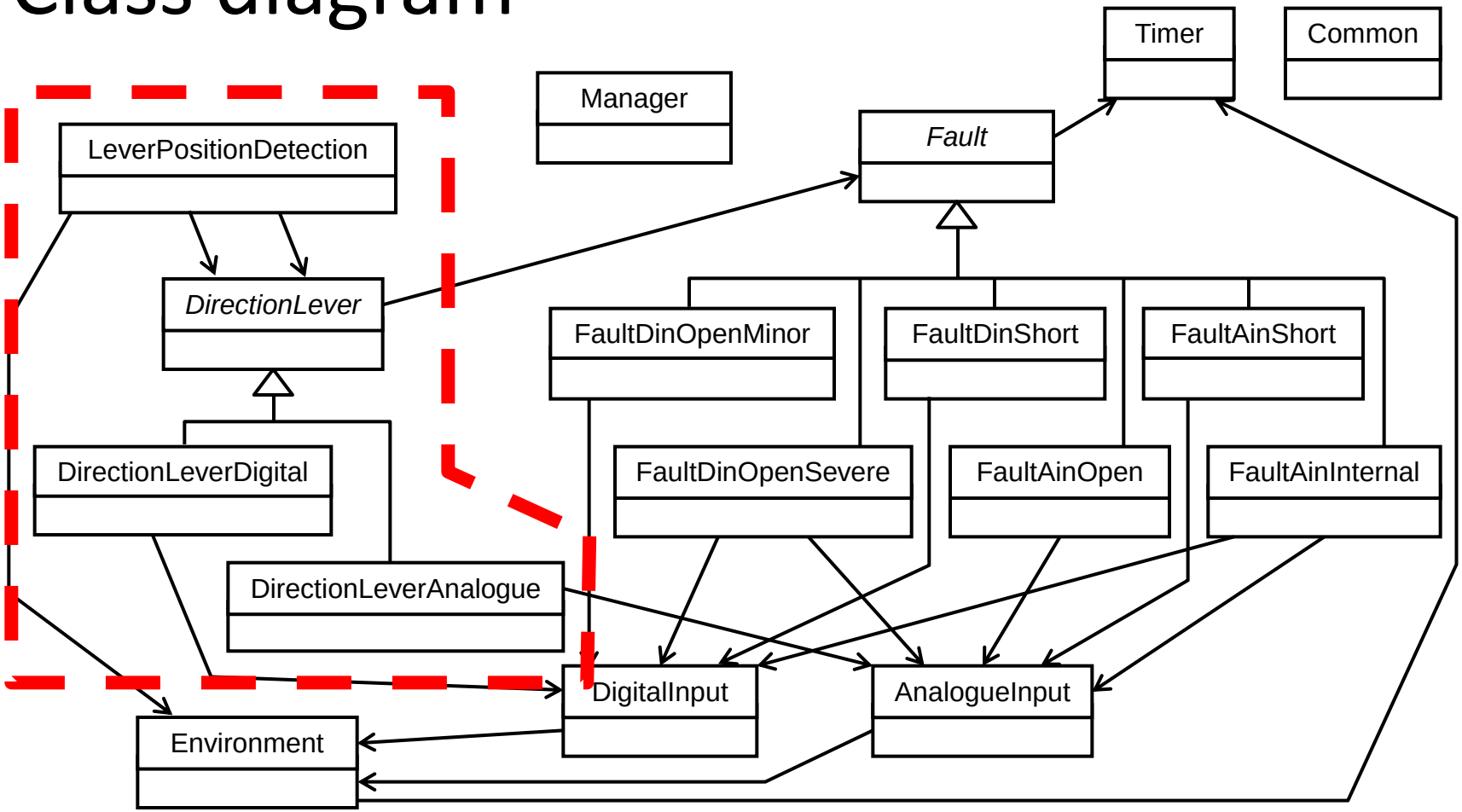
- State diagram of fault



Fault framework

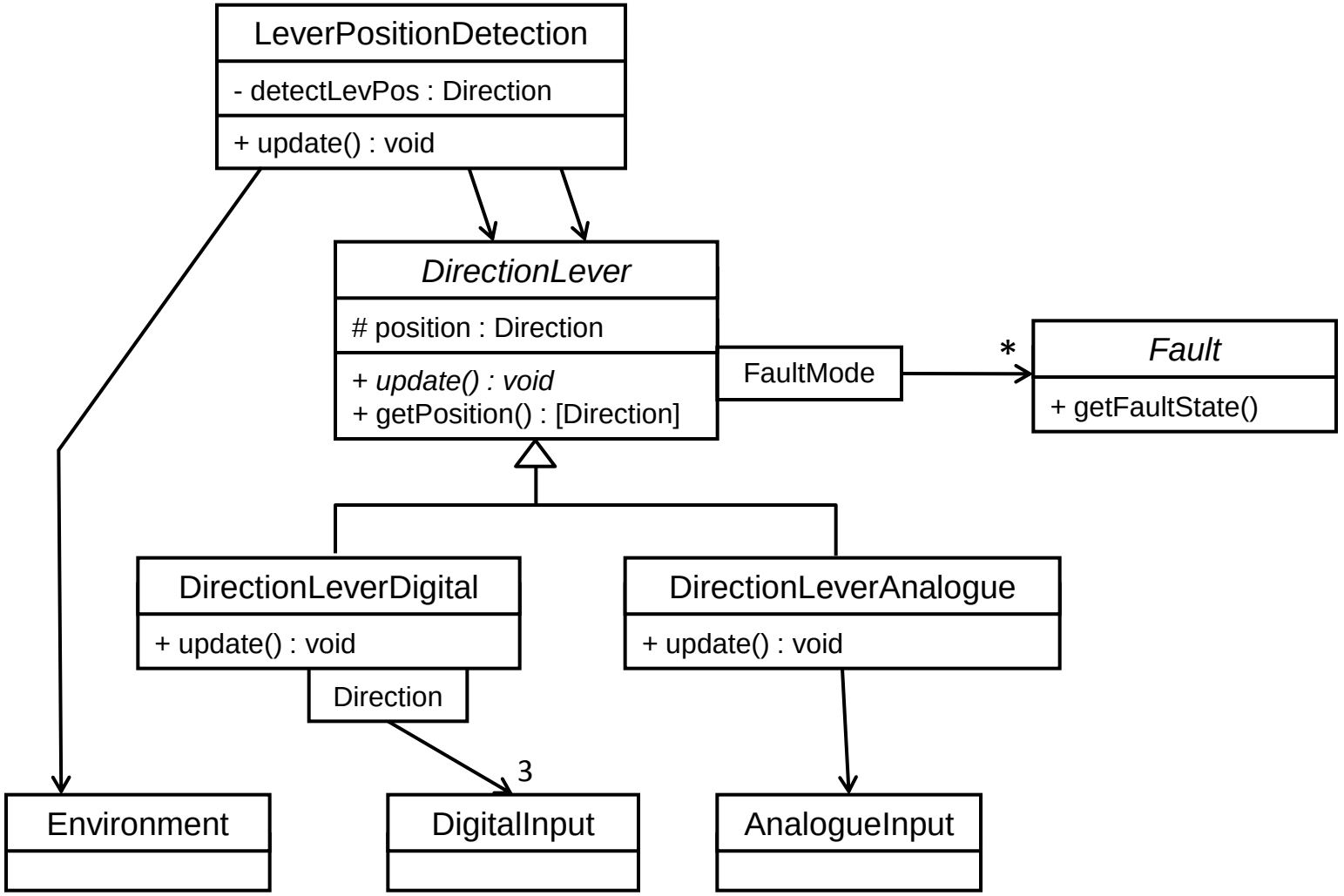


Class diagram



- Control logic

Detection of the lever position



Safety requirements

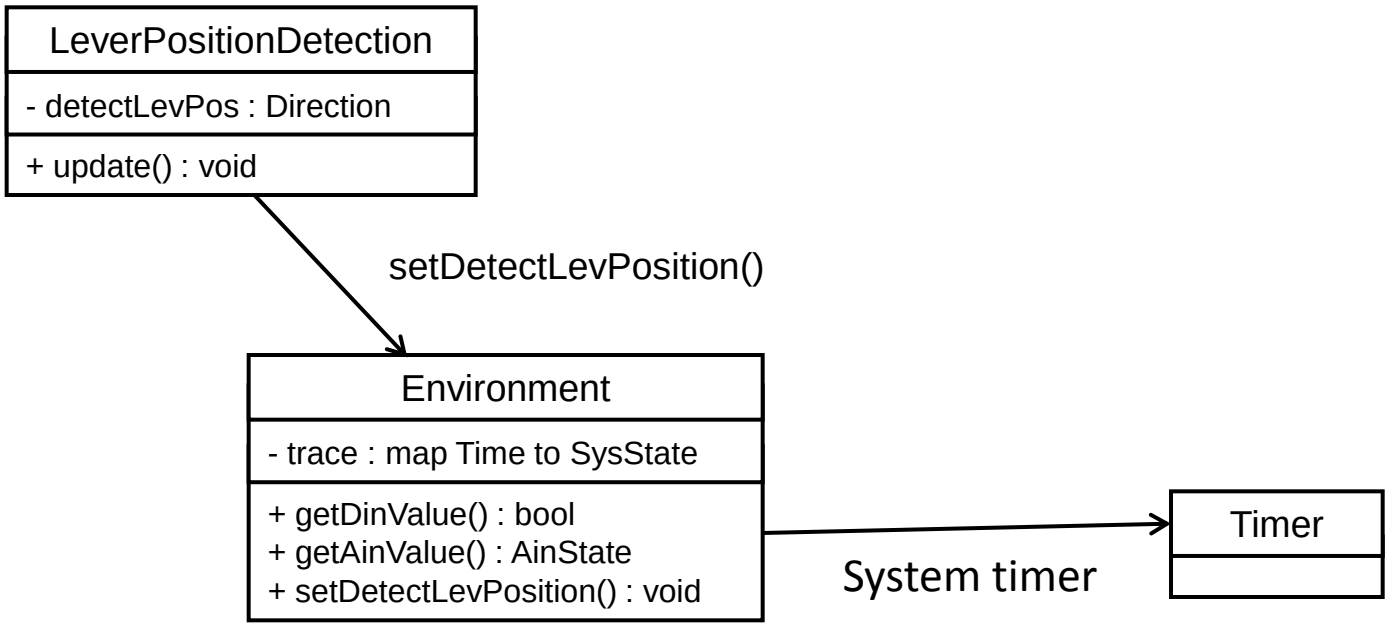
- Described as postconditions in Environment

R1: If any fault occurs in the system, the detected position of the direction lever must be consistent with the actual lever position or recognised as neutral (N).

R2: If any fault occurs in the system, the detected position of the direction lever must not change to F or R without lever manipulation by the operator of the vehicle.

Safety requirements

- Safety requirements are evaluated when the detected lever position is set to Environment



Safety requirements

```
public setDetectLevPosition : Direction ==> ()
setDetectLevPosition(dir) ==
    trace(sysTime.getTime()).detectLevPos := dir
pre
    sysTime.getTime() in set dom trace
post
    IfLeverIsFThenNotR()    and
    IfLeverIsRThenNotF()    and
    IfLeverIsNThenN()       and
    NotMoveWithoutOperation();
```

Safety requirements

```
private IfLeverIsFThenNotR: () ==> bool
IfLeverIsFThenNotR() ==
    let  curTime = sysTime.getTime()
    in
       return
       (((curTime >= Manager`SafetyCheckTime) and
         (forall  t  in set
               {curTime – Manager`SafetyCheckTime,..., curTime} &
                trace(t).levPos =  <DIR_F>))
       => trace(curTime).detectLevPos <>  <DIR_R>)
post  RESULT;
```

=> Safety requirement R1

Safety requirements

```
private NotMoveWithoutOperation: () ==> bool
NotMoveWithoutOperation() ==
    let  curTime = sysTime.getTime()
    in
       return
       (((curTime >= Manager`SafetyCheckTime) and
         (forall  t  in set
               {curTime – Manager`SafetyCheckTime,..., curTime - 1} &
                (trace(t).levPos = trace(curTime).levPos and
                 trace(t).detectLevPos =  <DIR_N>)))
       => trace(curTime).detectLevPos =  <DIR_N>)
post  RESULT;
```

=> Safety requirement R2

Outline

- Background and motivation
- Case study
  - Informal description of control specifications and safety requirements
  - Formal modelling using VDM++
  - Validation and safety analysis
- Conclusions

Validation

- Check if
  - the model behaves as expected
  - the safety requirements are satisfied
- The model is executed with various time series of input data (test scenarios)
- The results are compared with expected values
- Testing framework "VDMUnit" is used

An example of test cases

```
class SystemTest1 is subclass of TestCase, Environment
types
private TestData :: inData : SysState
                    expectVal : [Direction];
values
private testData: map Time to TestData =
{
    0 |-> mk_TestData(mk_SysState(false, true, false,
                                  <DIR_N>, <DIR_N>, nil), <DIR_N>),
    1 |-> mk_TestData(mk_SysState(false, false, false,
                                  <MID_FN>, <MID_FN_>, nil), <DIR_N>),
    2 |-> mk_TestData(mk_SysState(true, false, false,
                                  <DIR_F>, <DIR_F>, nil), <DIR_F>),
...
};
```

```
operations
public  runTest : () ==> ()
runTest() ==
(
    let  testInData = {t |-> testData(t).inData | t in set dom testData}
    in (
        dcl  mgr : Manager := new Manager(testInData);
        for  t = 0 to (card dom testData - 1)
        do (
            mgr.update();
            assertTrue("t=" ^ VDMUtil`val2seq_of_char[nat](t) ^ ", failed.",
                        mgr.env.getTrace()(t).detectLevPos =
                        testData(t).expectVal)
        )
    )
);
end  SystemTest1
```

```
new TestMain().executeSystemTest()
```

```
Start test - Direction Lever Test
Start test - System test
All 647 tests passed.
End test   - Direction Lever Test
*** All Tests Passed. ***
```

Success

```
Start test - Direction Lever Test
Start test - System test
System test, Test1, t=3, failed.
System test, Test5, t=23, failed.
2 of 647 tests failed.
End test   - Direction Lever Test
*** ERROR! ***
```

Failure

Results of validation

- Testing with 14 scenarios has been executed
- Test scenarios:
  - Normal lever manipulation (without faults)
  - Digital input F open-circuits, and then recovers
  - …
- Confirmed the model behaved as expected for all input data elaborated

Results of validation

- Test coverage (generated by Overture)
  - DirectionLeverDigital`update: 98.6%
  - Fault`doFaultNormal: 89.4%
  - The others: 100.0%
- The untested statements can never be executed under the current specifications
- Virtually whole model was tested

Safety requirements violated

R2: If any fault occurs in the system, the detected position of the direction lever must not change to F or R without lever manipulation by the operator of the vehicle.

- Safety requirements violation has been discovered for certain input data series

```
Start test - Direction Lever Test
Start test - System test
Error 4072: Postcondition failure: post_NotMoveWithoutOperation in
'Environment'
```

Safety requirements violated

- However, the case could never happen in reality
- Caused by a coincidence of several rare accidents
  1. Direction lever is in the middle of the positions F and N
  2. No digital input signals are “on”
  3. Analogue input signal indicates the position F
  4. Digital input N periodically short-circuits to power with a short period (less than the fault detecting time)
  5. Then, the short-circuit recovers

  => Detected lever position changes from N to F without lever manipulation

Outline

- Background and motivation
- Case study
  - Informal description of control specifications and safety requirements
  - Formal modelling using VDM++
  - Validation and safety analysis
- Conclusions

Conclusions

- A part of control systems of construction equipment has been formally modelled using VDM++
- A modelling pattern: a fault framework has been introduced
- The model has been tested using VDMUnit
- Violation of a safety requirement has been found
- This demonstrates availability of formal modelling to a practical control system

Future work

- Apply the approach to a larger scale system
- Improve testing environment
- Challenge formal verification of the model using a verification tool, e.g. UPPAAL
  - check if there exists another case which violates the safety requirements

Thank you for listening.