ON FAULT TOLERANCE REUSE DURING REFINEMENT

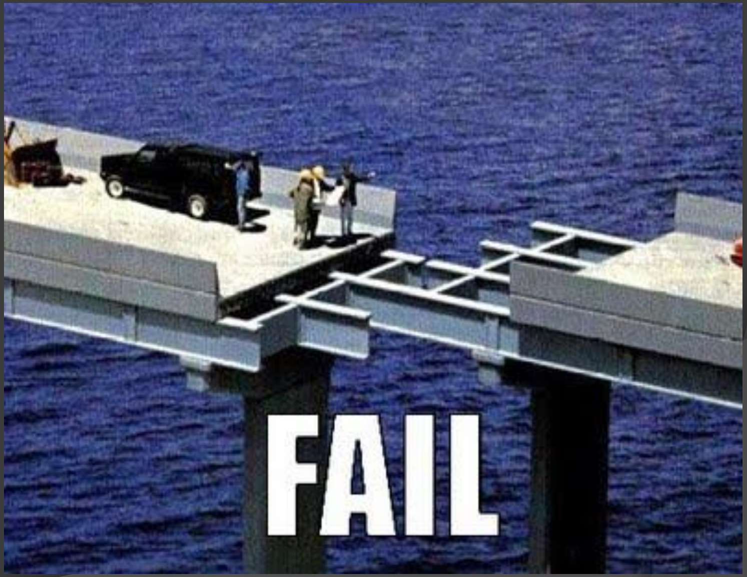
Ilya Lopatkin, Alexei Iliasov, Alexander Romanovsky
CSR, Newcastle University

Outline

- Motivations, refinement context
- Fault Tolerance view – main contribution
  - Concepts
  - Formal link with Event-B
- Model transformation patterns – ongoing research

Motivations

- Amount of FT-related requirements to critical systems
- Early modelling of FT

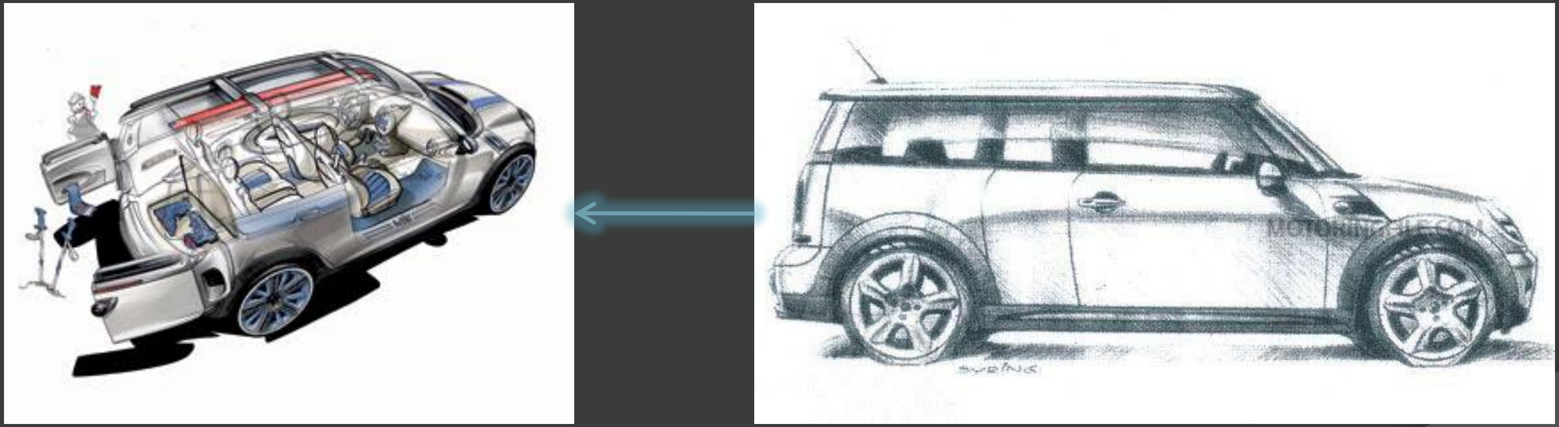


Motivations

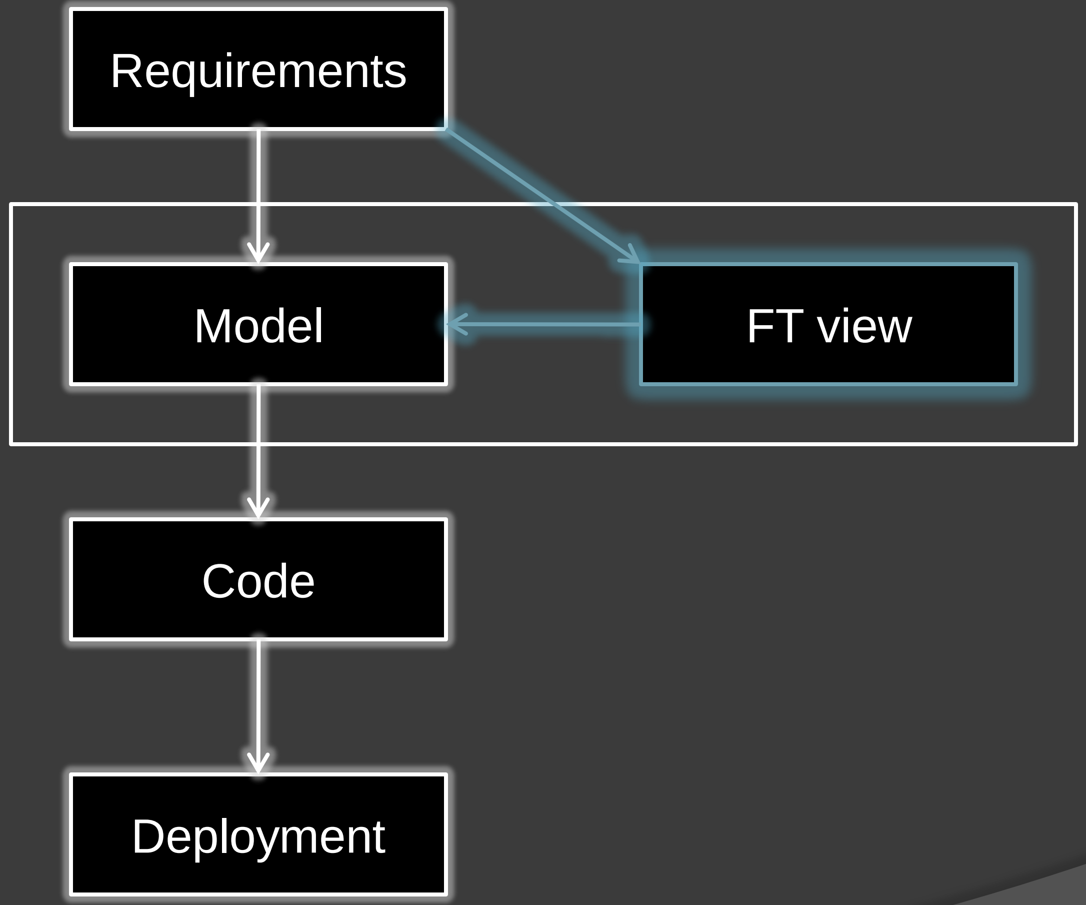
- Why model?
  - There are requirements and specification
    - Define context: what might go wrong
  - Trace
  - Certify
- Recurring artefacts
- Separation of concerns
- Explicitness

View

- Fix dimensions = narrow the focus
- Restrict changes

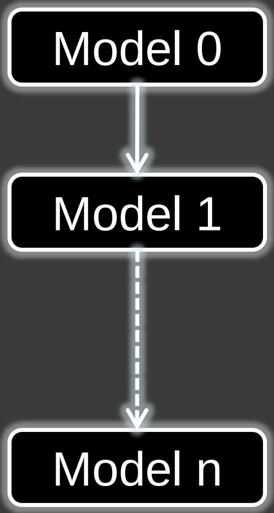


Where FT view stands

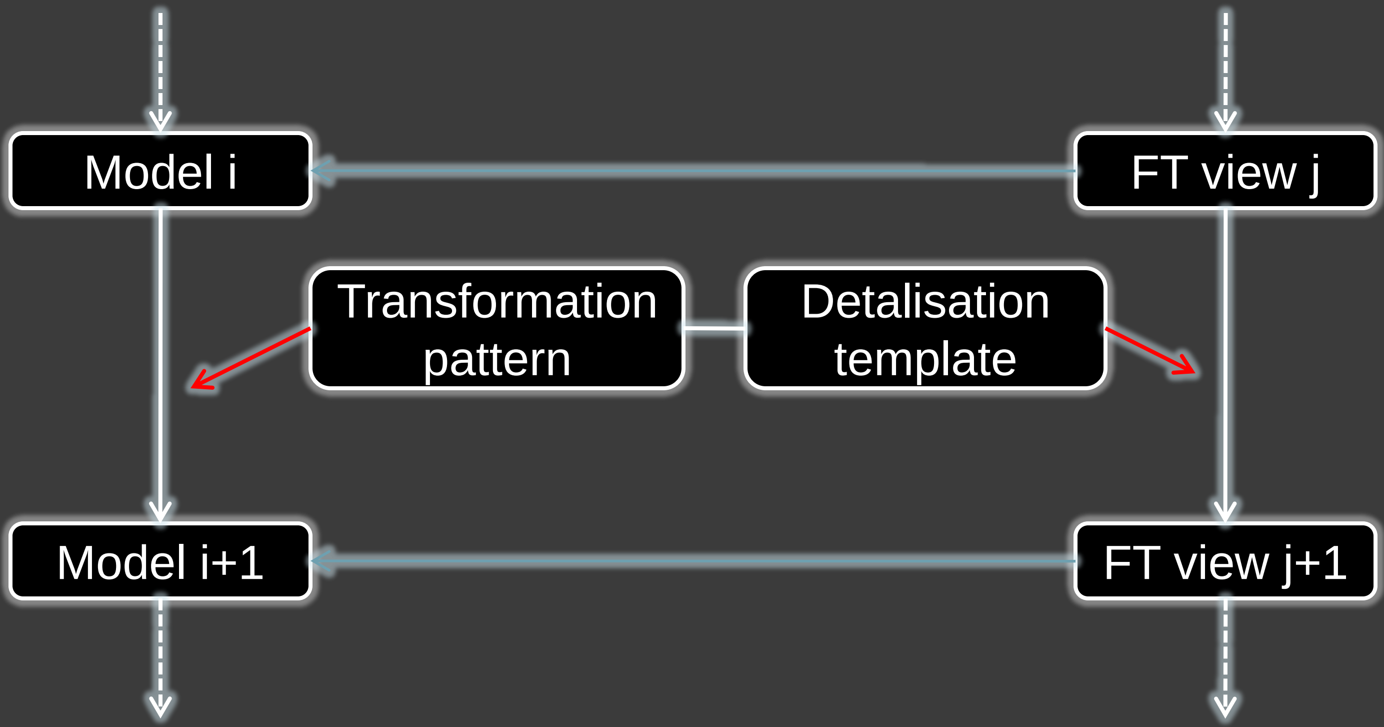


Refinement

- Very abstract at 0-level
- Add details
- Show consistency
- Finish when happy and/or tired

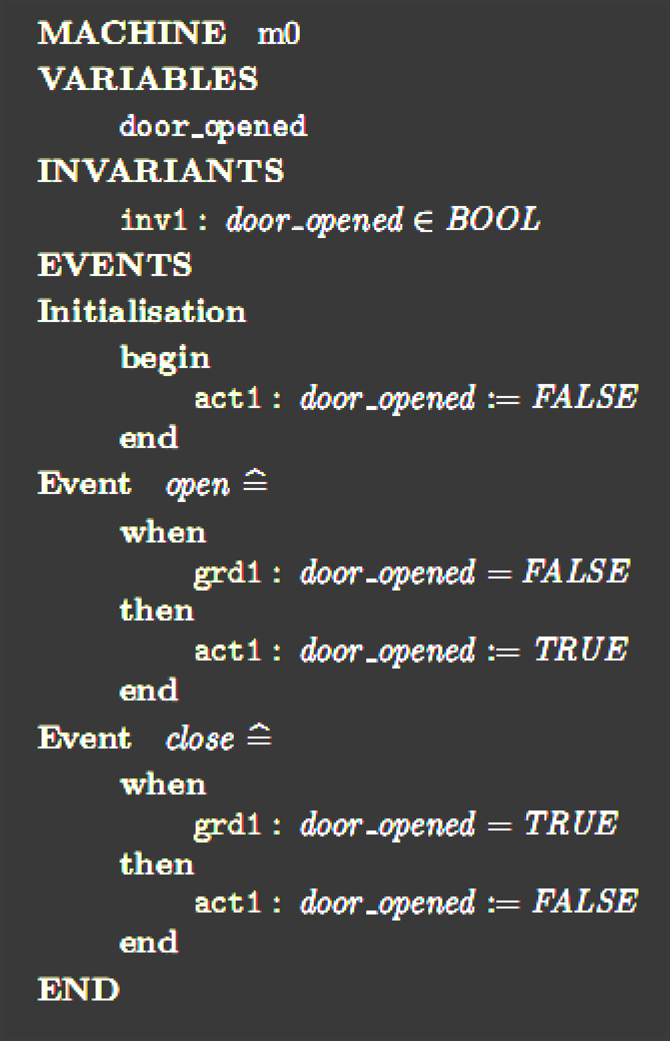


Patterns + FT view templates

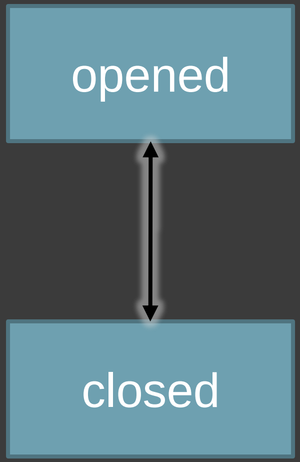


Event-B

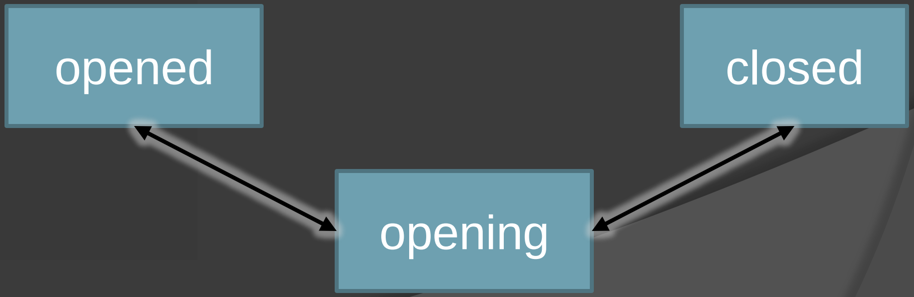
- State-based
- Model consists of
  - State v
  - Guarded events E
    $H(c, s, v) \rightarrow S(c, s, v, v')$
  - Invariant $I(c, s, v)$
- Model refinement
- Proofs

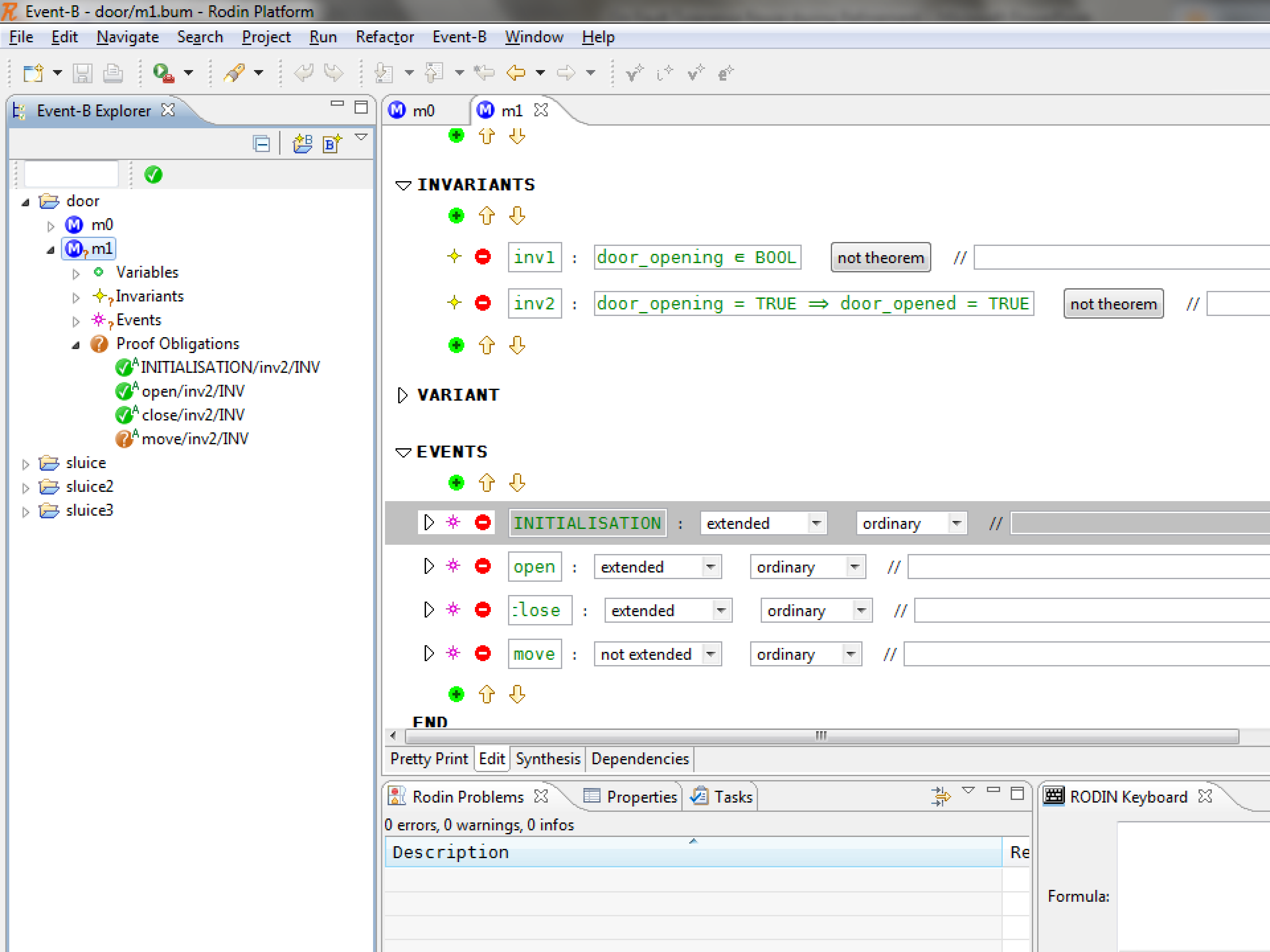


```
MACHINE   m0
VARIABLES
      door_opened
INVARIANTS
      inv1 :  door_opened ∈ BOOL
EVENTS
Initialisation
      begin
            act1 :  door_opened := FALSE
      end
Event   open ≙
      when
            grd1 :  door_opened = FALSE
      then
            act1 :  door_opened := TRUE
      end
Event   close ≙
      when
            grd1 :  door_opened = TRUE
      then
            act1 :  door_opened := FALSE
      end
END
```



```
MACHINE  m1
REFINES  m0
VARIABLES
    door_opening
    door_opened
INVARIANTS
    inv1 :  door_opening ∈ BOOL
    inv2 :  door_opening = TRUE ⇒ door_opened = TRUE
EVENTS
Initialisation
    extended
    begin
        act1 :  door_opened := FALSE
        act2 :  door_opening := FALSE
    end
Event  open ≙
extends  open
    when
        grd1 :  door_opened = FALSE
        grd2 :  door_opening = TRUE
    then
        act1 :  door_opened := TRUE
        act2 :  door_opening := FALSE
    end
Event  close ≙
extends  close
    when
        grd1 :  door_opened = TRUE
        grd2 :  door_opening = TRUE
    then
        act1 :  door_opened := FALSE
        act2 :  door_opening := FALSE
    end
Event  move ≙
    when
        grd1 :  door_opening = FALSE
    then
        act1 :  door_opening := TRUE
    end
END
```




Event-B - door/m1.bum - Rodin Platform

File Edit Navigate Search Project Run Refactor Event-B Window Help

Event-B Explorer

```
door
  m0
  m1
    Variables
    Invariants
    Events
    Proof Obligations
      INITIALISATION/inv2/INV
      open/inv2/INV
      close/inv2/INV
      move/inv2/INV
sluice
sluice2
sluice3
```

m0 m1

```
INVARIANTS
  inv1 : door_opening ∈ BOOL                         not theorem //
  inv2 : door_opening = TRUE ⇒ door_opened = TRUE   not theorem //

VARIANT

EVENTS
  INITIALISATION : extended      ordinary //
  open           : extended      ordinary //
  close          : extended      ordinary //
  move           : not extended  ordinary //

END
```

Pretty Print | Edit | Synthesis | Dependencies

Rodin Problems | Properties | Tasks

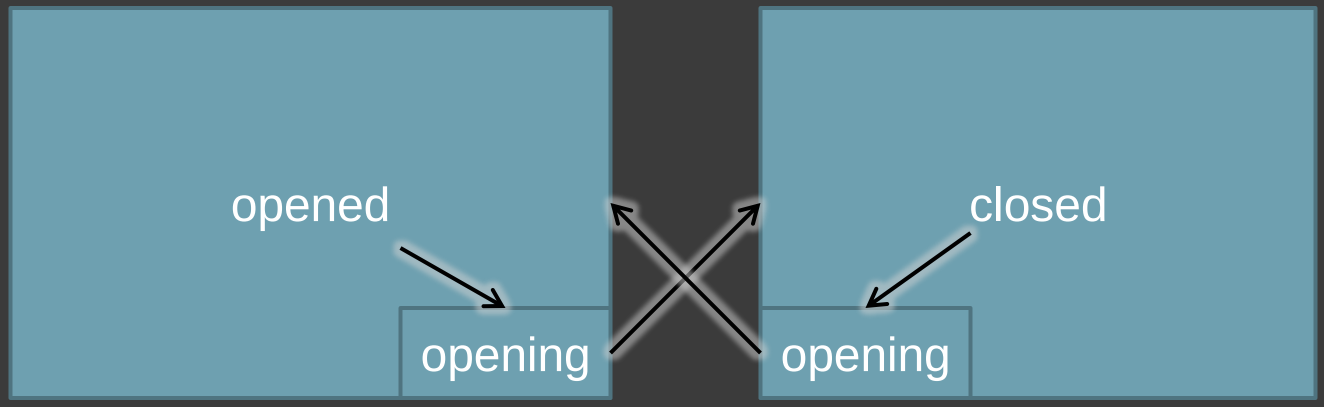
0 errors, 0 warnings, 0 infos

Description

RODIN Keyboard

Formula:

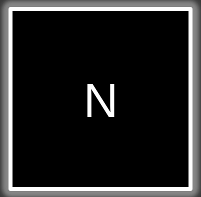
Refinement world



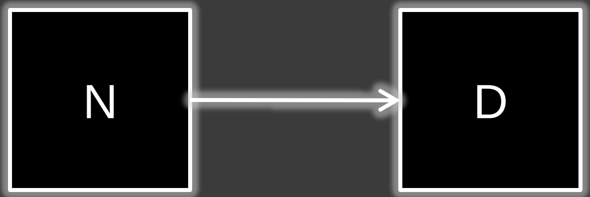
Where should we put a fault handler?

Abstract classes of FT systems

- Normal
  - All errors are recoverable



- Normal + Degraded
  - There are errors that cannot be masked



FT view concepts

- Activities
- Errors

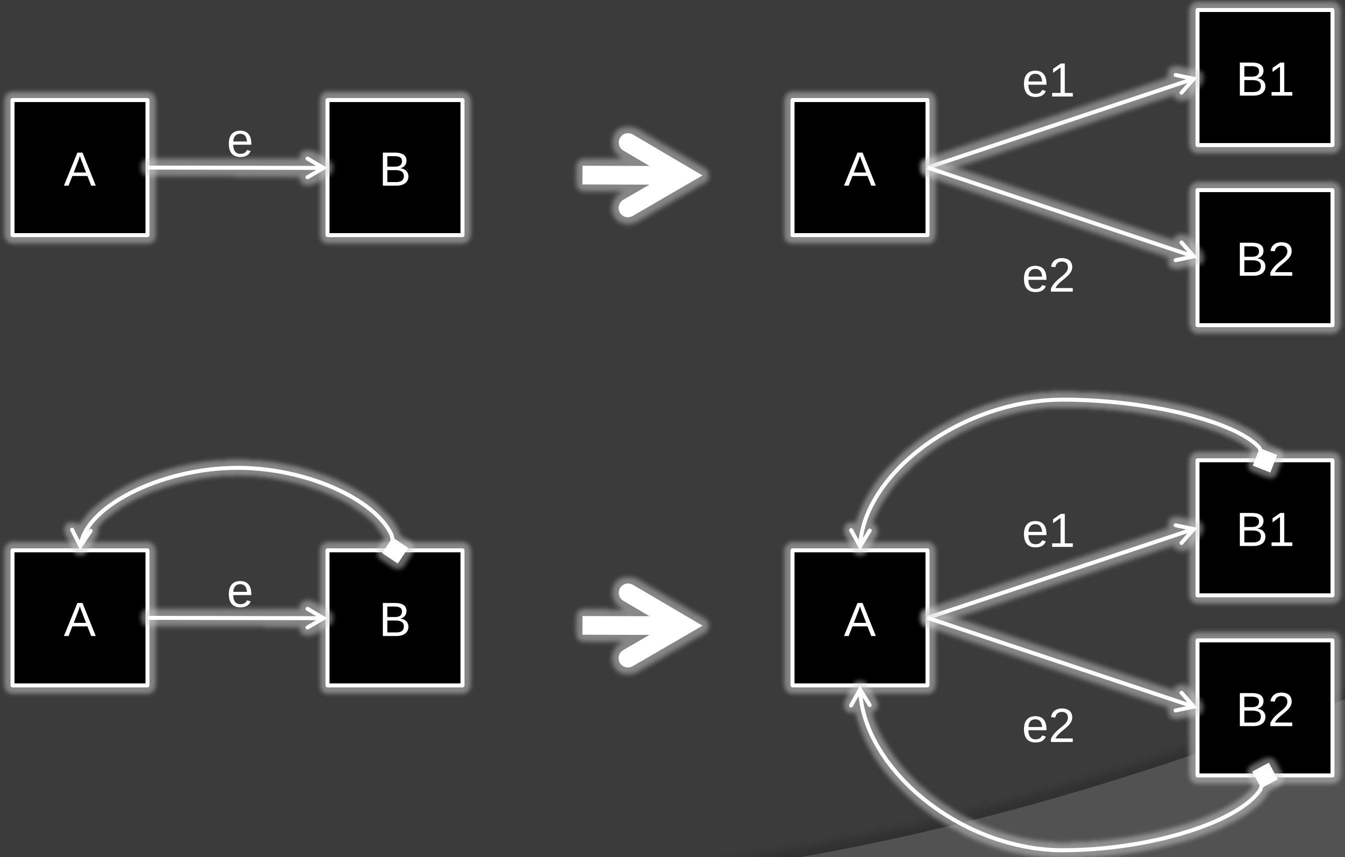
$$\begin{tikzcd}
\text{A} \arrow[r, "\text{e1}"] & \text{B} \arrow[r, "\text{e2}"] & \text{C}
\end{tikzcd}$$

$$\begin{tikzcd}
\text{Normal} \arrow[r] & \text{Degraded}
\end{tikzcd}$$

$$\begin{tikzcd}
\text{Normal} \arrow[r] & \text{Recovery} \arrow[l, bend right]
\end{tikzcd}$$

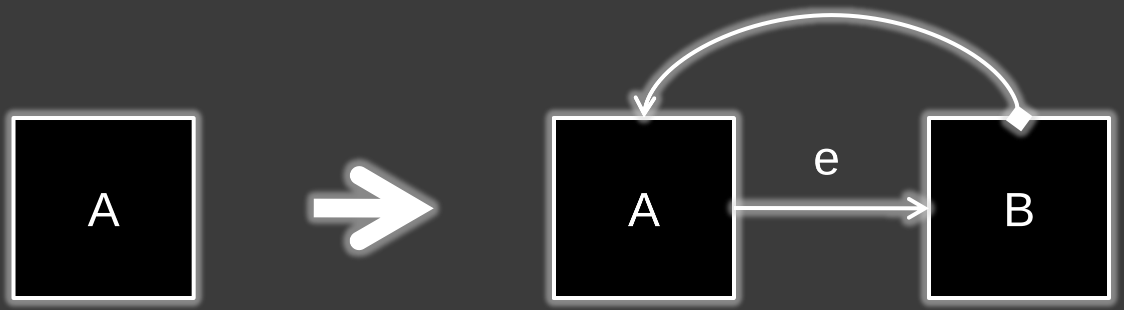
Detalisation templates

- Template 1: Detalisation of an error

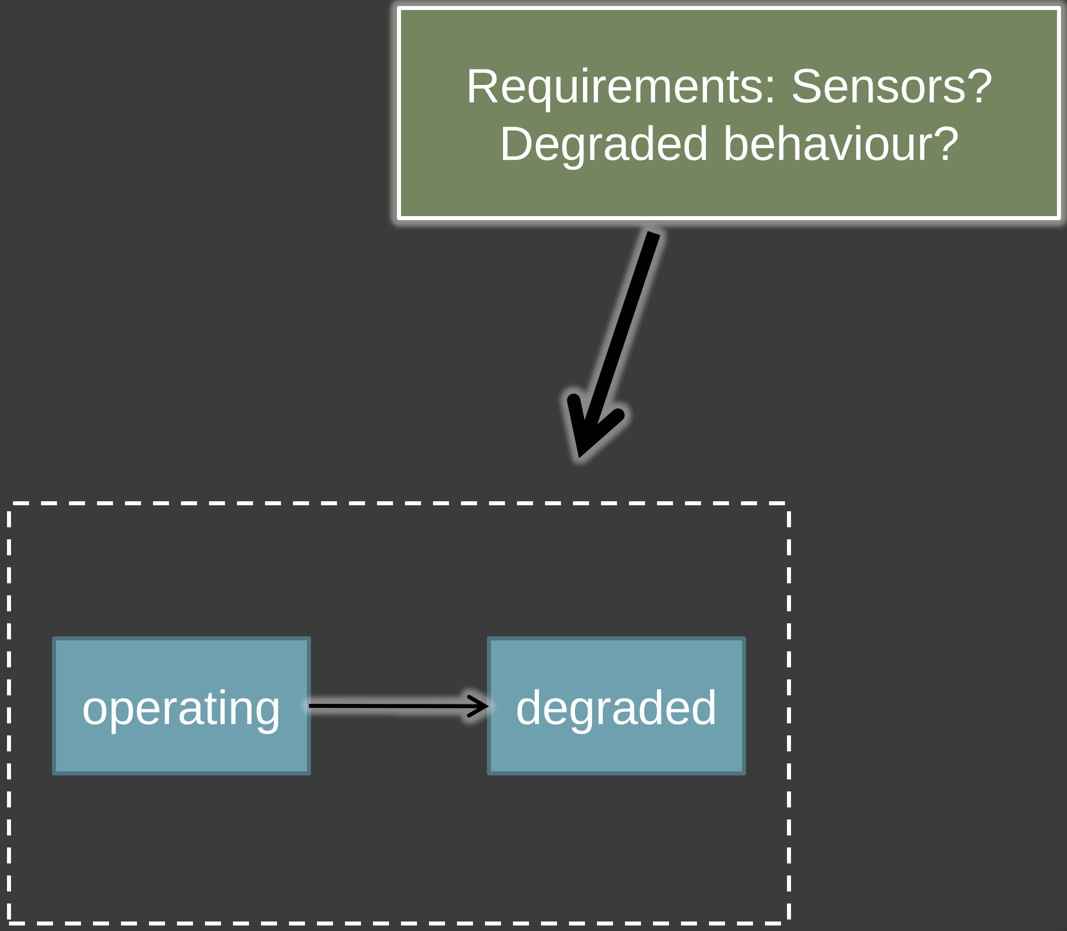


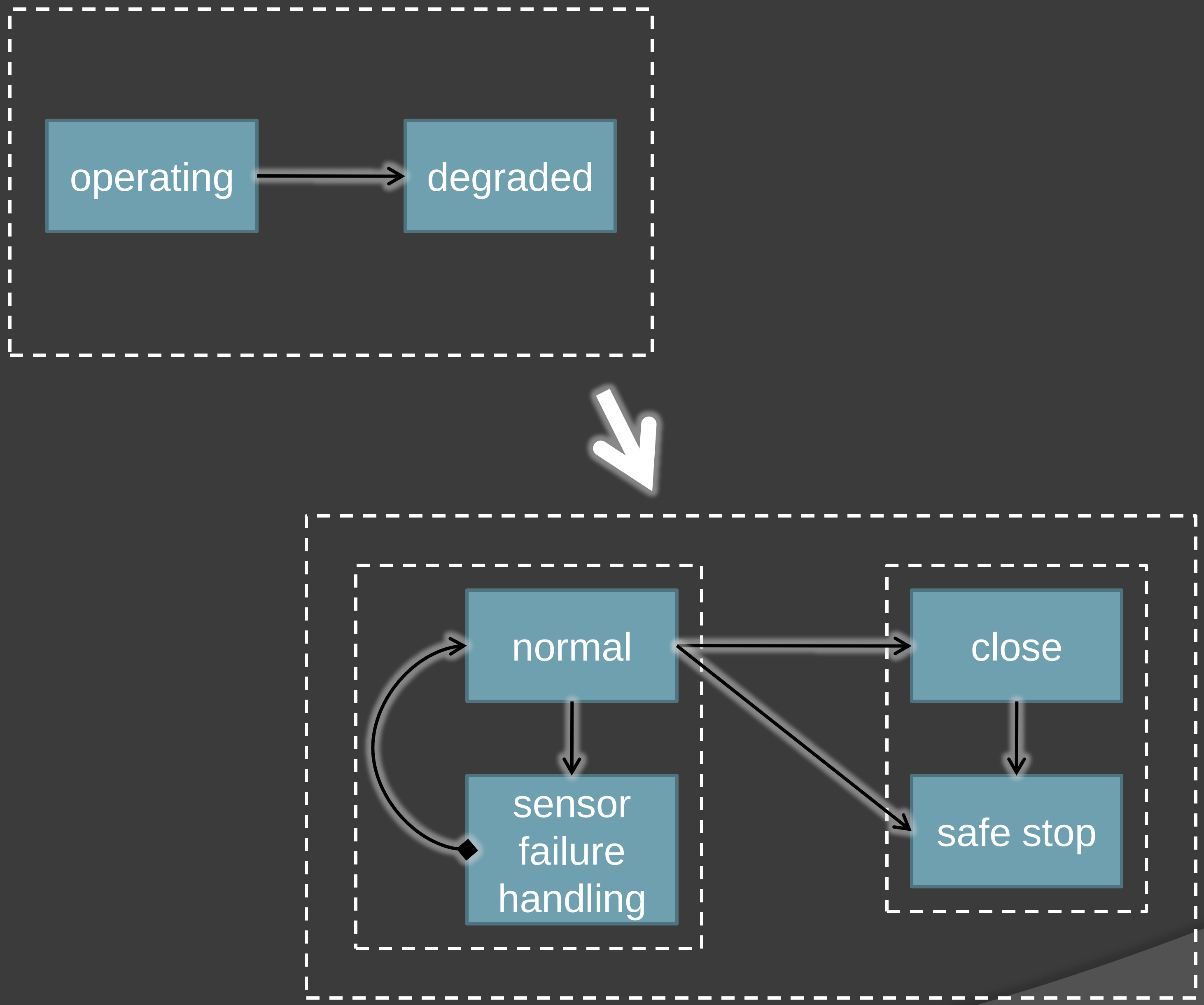
Detalisation templates

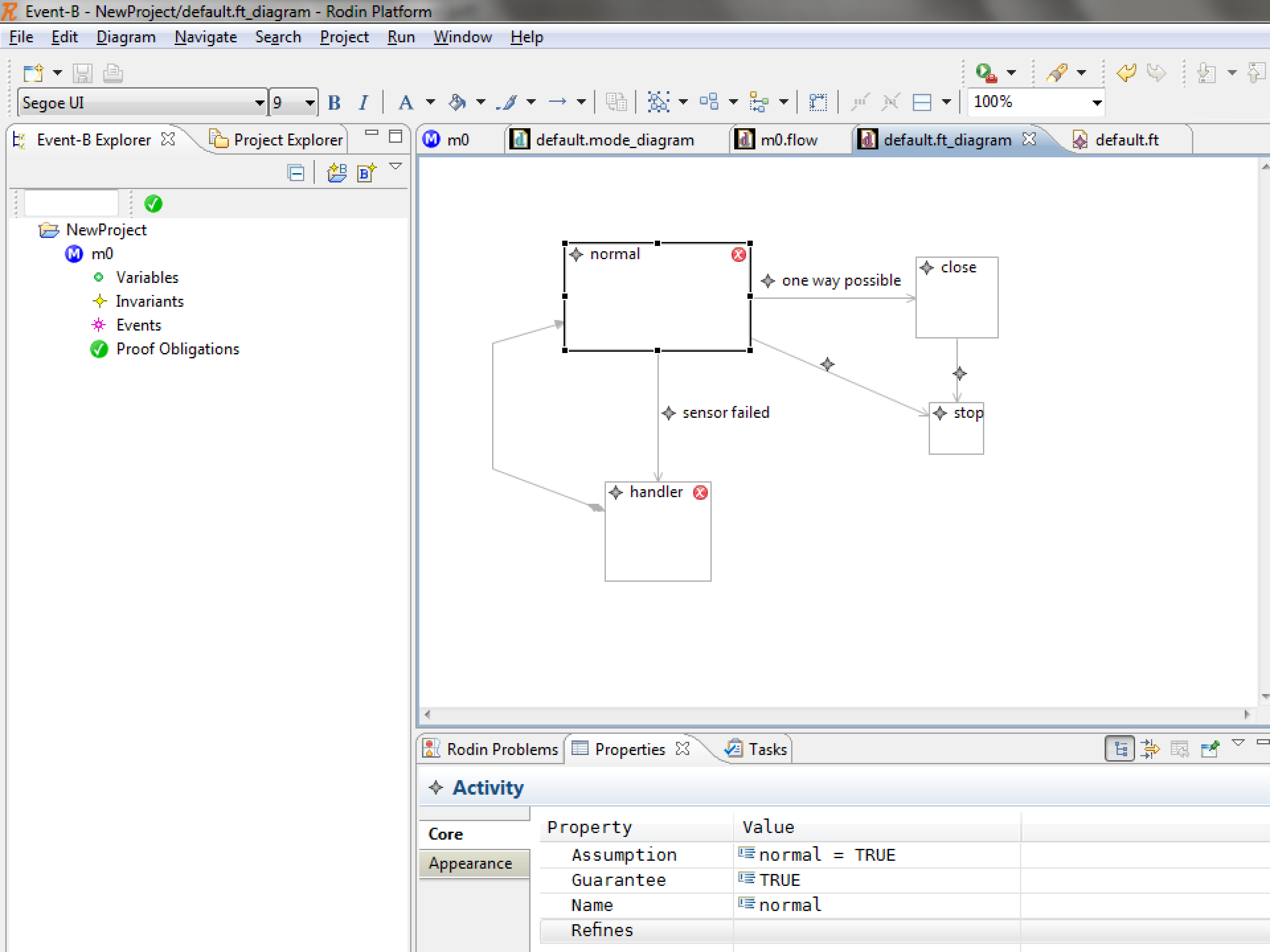
- Template 2: New error



Our door

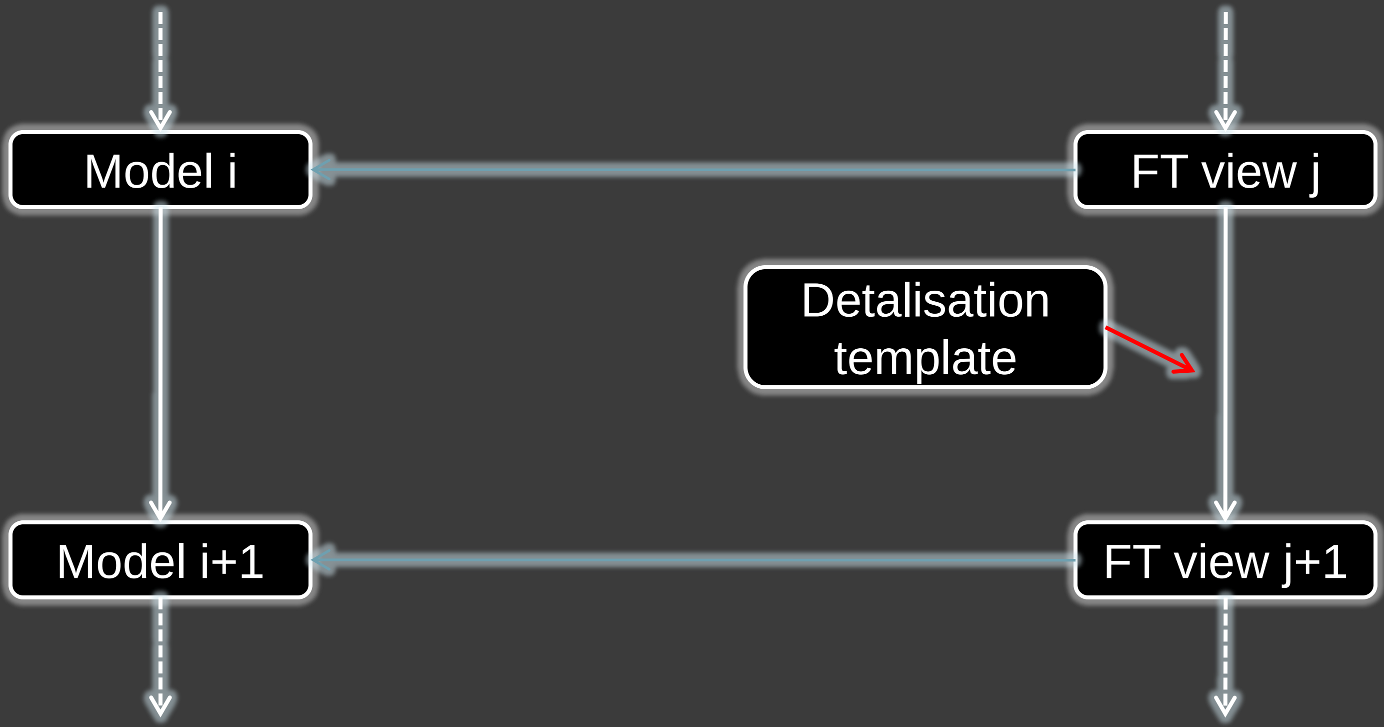

Requirements: Sensors? Degraded behaviour?
operating
degraded


operating
degraded
normal
sensor failure handling
close
safe stop



Activity

Property	Value
Assumption	normal = TRUE
Guarantee	TRUE
Name	normal
Refines	


Model i
FT view j
Detalisation template
Model i+1
FT view j+1

FT view formalisation

- Activities provide different functionalities under differing operating conditions
- Each activity is characterized by A/G
- $A(v)$ – assumption
- $G(v, v')$ – guarantee
- v – model variables

FT view formalisation

- Assumptions exhaust the invariant

$$I(v) \Rightarrow A_1 \vee A_2 \vee \cdots \vee A_n$$

- There exists a transition within activity

$$\exists v, v' \cdot I(v) \wedge A(v) \Rightarrow G(v, v')$$

- Activities do not overlap

$$I(v) \Leftarrow A_1(v) \oplus \cdots \oplus A_n(v)$$

FT view formalisation

- Detalisation conditions

$$A(v)/G(v,v') \sqsubseteq A'(u)/G'(u,u') \quad \text{iff} \begin{cases} J(v,u) \wedge A(v) \Rightarrow A'(u) \\ J(v,u) \wedge G'(u,u') \Rightarrow G(v,v') \end{cases}$$

$$A(v)/G(v,v') \sqsubseteq \begin{matrix} A_1(u)/G_1(u,u') \\ A_2(u)/G_2(u,u') \end{matrix}, \quad \text{iff} \begin{cases} J(v,u) \wedge A(v) \Rightarrow A_1(u) \vee A_2(u) \\ J(v,u) \wedge G_1(u,u') \vee G_2(u,u') \Rightarrow G(v,v') \end{cases}$$

FT view formalisation

$$A_1/G_1 \mapsto E_1$$
$$A_2/G_2 \mapsto E_2$$
$$\dots$$
$$A_n/G_n \mapsto E_n$$

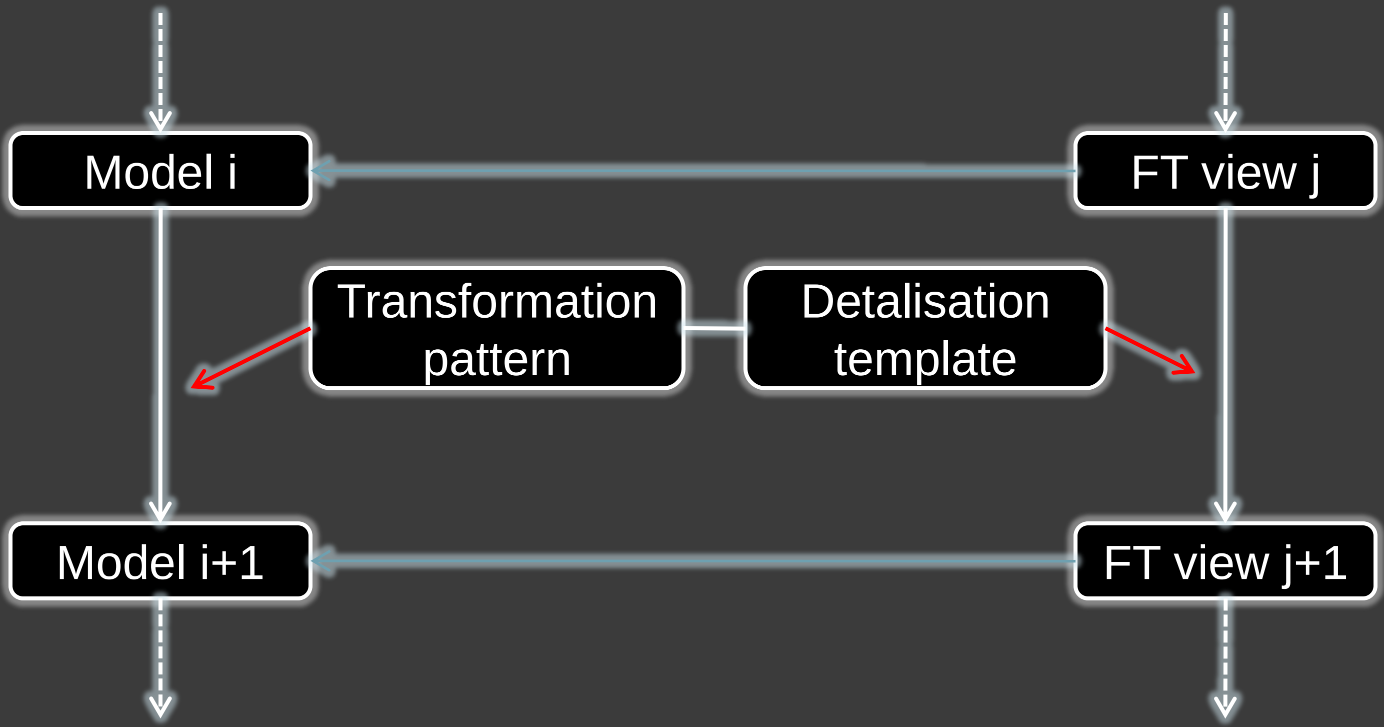
- Relate activities to events
- Events must satisfy the activity guarantee

$$I(v) \wedge A(v) \wedge H(v) \wedge R(v,v') \Rightarrow G(v,v')$$

- Partitioning of events into activities must agree with guards

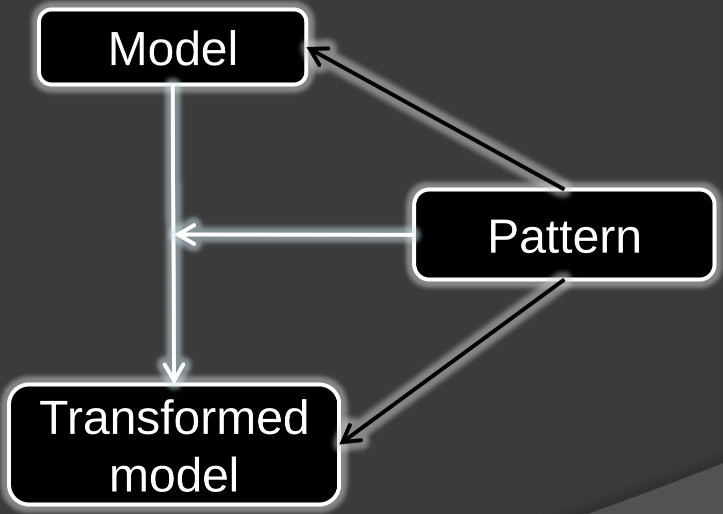
$$H(v) \Rightarrow A_1(v) \vee \dots \vee A_k(v)$$
$$A_{k+1}(v) \vee \dots \vee A_n(v) \Rightarrow \neg H(v)$$

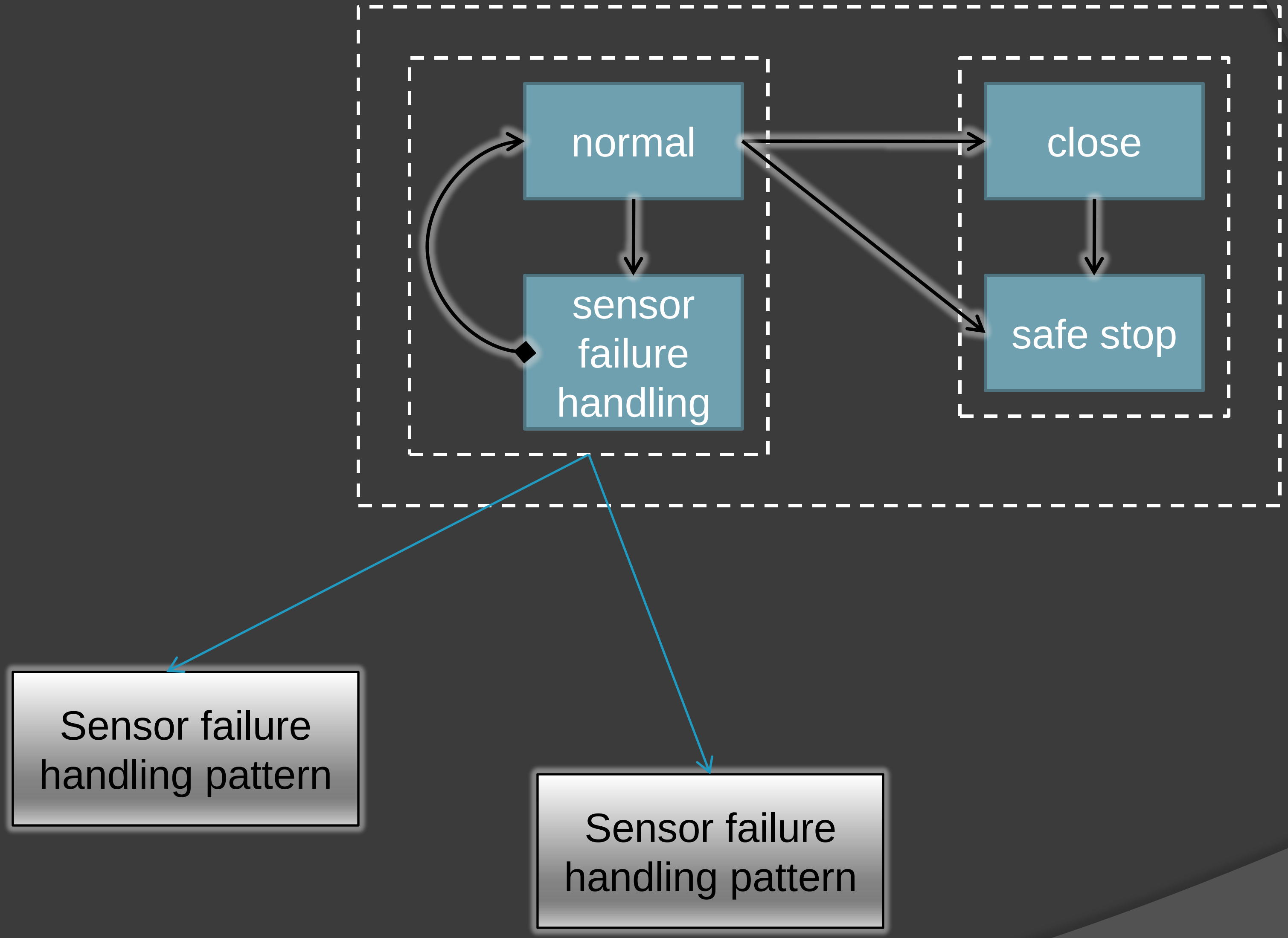
Patterns + FT view templates



Model transformations

- Model transformation - pattern
  - Applicability conditions
  - Effects
  - Proof

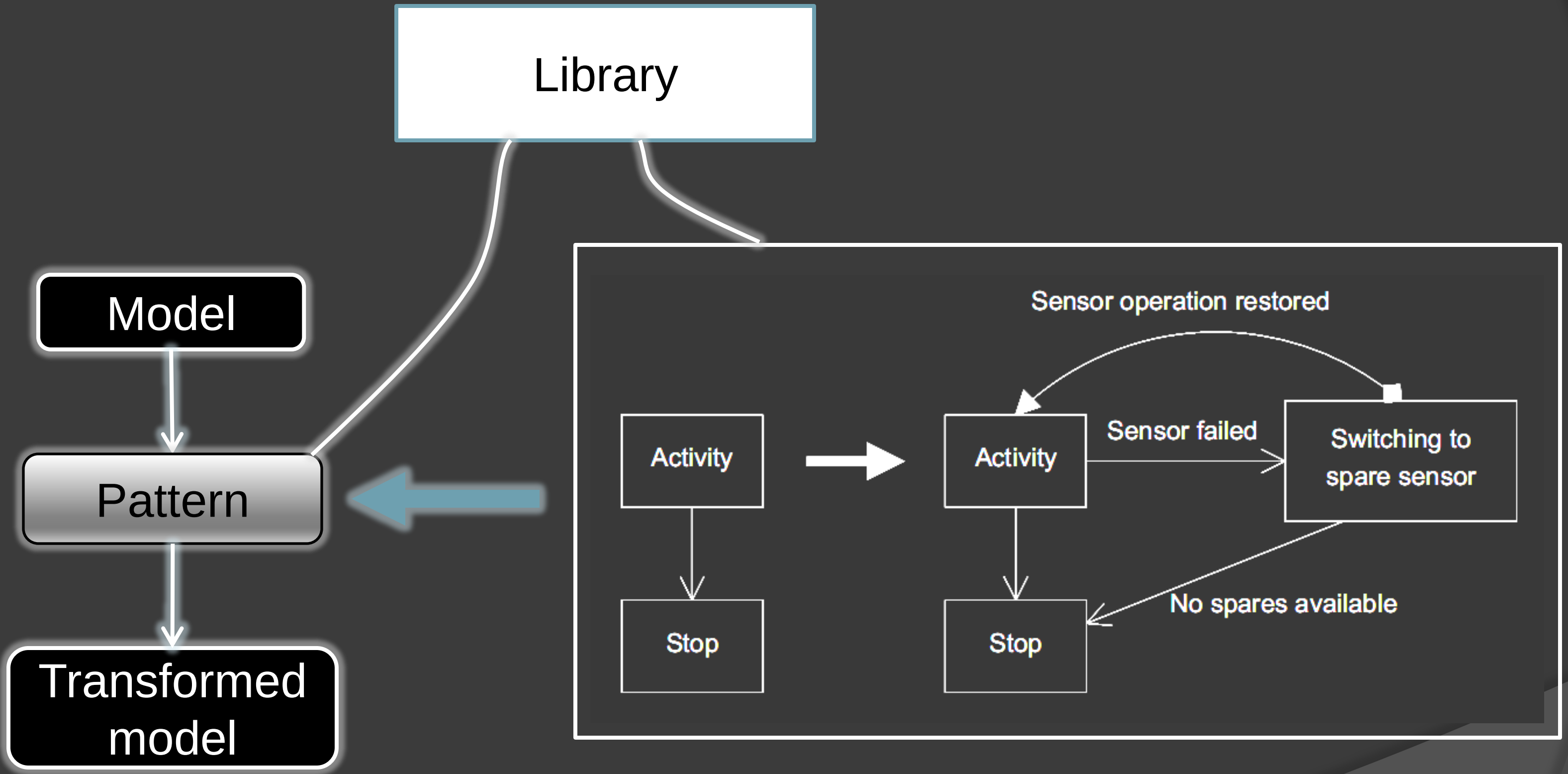



normal
sensor failure handling
close
safe stop
Sensor failure handling pattern
Sensor failure handling pattern

Library of FT patterns

- Patterns for fault tolerance
  - Specific to- or domain-independent
  - Reuse by applying to a model
  - Gradually introduced fault tolerance
  - Complementing existing models without FT
  - Finer-grained patterns: create replica, save state, voting, etc...
- Tool for such library

Patterns + FT view templates



Ongoing & Future work

- Tool for FT view
- Model transformation patterns
  - Tool for application
  - FT library
- Couple templates with patterns

Summary

- Approach to facilitating FM of FT
- FT view orthogonal to formalism
- Encourage use of architectural abstractions at early phases + refinement via FT templates
- Improve traceability
- Templates + patterns = discipline, expressive link with FM
- Libraries of reusable FT components