FRENCH NATIONAL INSTITUTE FOR
TRANSPORT SAFETY AND RESEARCH

A Formal Framework for UML Modeling with Timed Constraints: Application to Railway Control Systems

Rafael Marcano
Samuel Colin
Georges Mariano

Specification and Validation of UML models for Real Time and Embedded Systems – SVERTS'2004

Context

UML & Railway Applications ...

UML/OCL as formal notation

- *Standard notation for systems modeling*
- *But lack of formal semantic*

Our context : railway control systems

Is actually UML/OCL formal enough to specify reliable software ?

Context

problem ...

...UML/OCL still informally defined

...insecure basis for software development

How to specify precisely ?

How to prove rigorously ? ...

Using the B Formal Method

- *Formal methods are used to :*
  - specify precisely the components of a system
  - prove rigorously the desired structural/behavioral properties (analysis and verification)

Plan
…

1. Objective
2. The proposed process
3. From requirements to UML
4. From UML models to B specifications
   - *mapping UML diagrams into B formal specifications*
   - *translating time features into extended B*
5. Analysis and verification
6. Conclusion and discussion

I. Objective …

Problem

How to enable analysis and verification on UML/OCL models with time properties ?

Solution

- *By combining the complementary strengths of UML/OCL and the B formal method*
- *By extending B with a real time semantic*
- *By providing automated proof tools for the B specification language*

Advantages of the B formal method

- *precise and rigorous description of a system*
- *detection of inconsistencies and mistakes on the UML model*
- *proof of structural and behavioral properties*
- *ensuring consistency between diagrams using only one formalism*
- *supporting refinement*

But it does not support timed characteristics

II. The Proposed Process …

1. Requirements elicitation

Create a valid UML description of the required system

2. Formal specification

Derive a B formal specification from UML diagrams

3. Analysis and verification

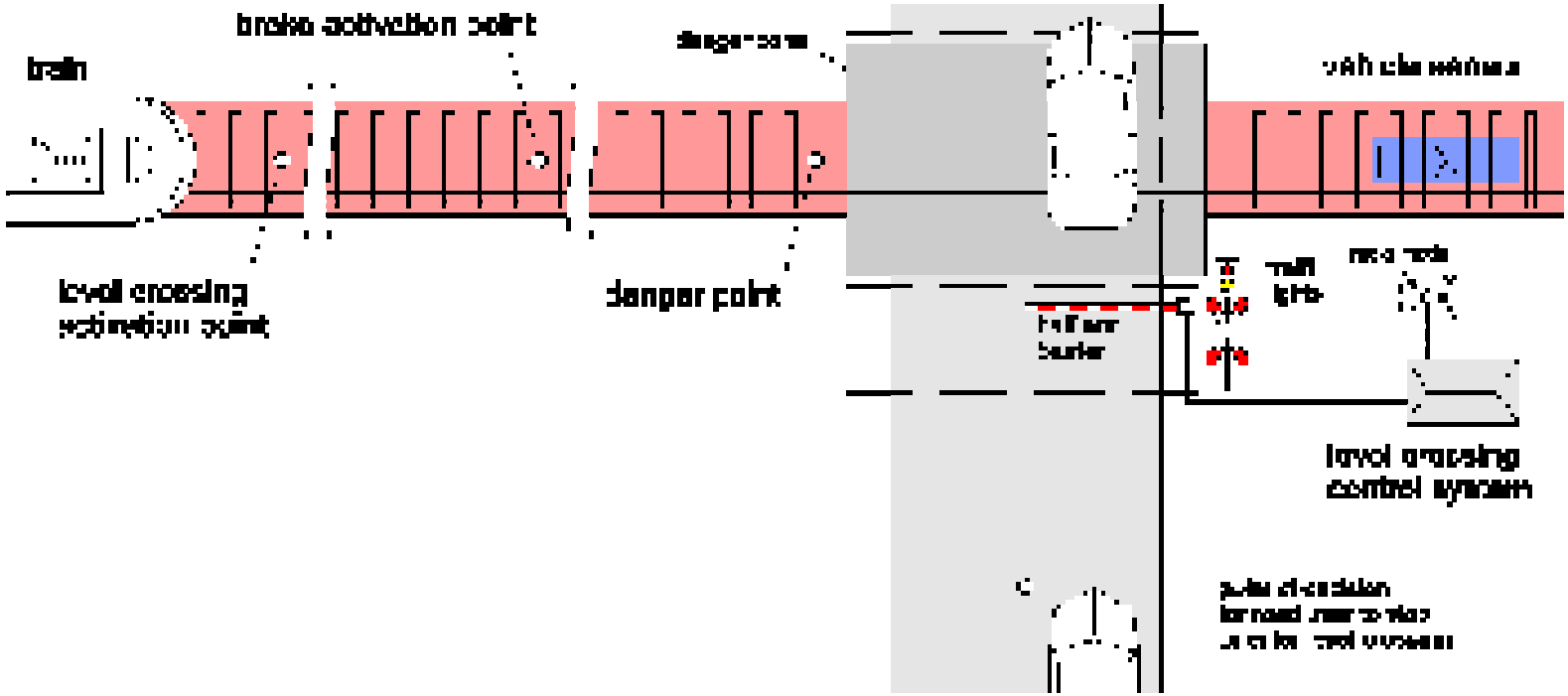
Generate proof obligations and prove them

III. From Requirements to UML

...

A railway level crossing system [JS'00]

TECHNISCHE UNIVERSITÄT BRAUNSCHWEIG

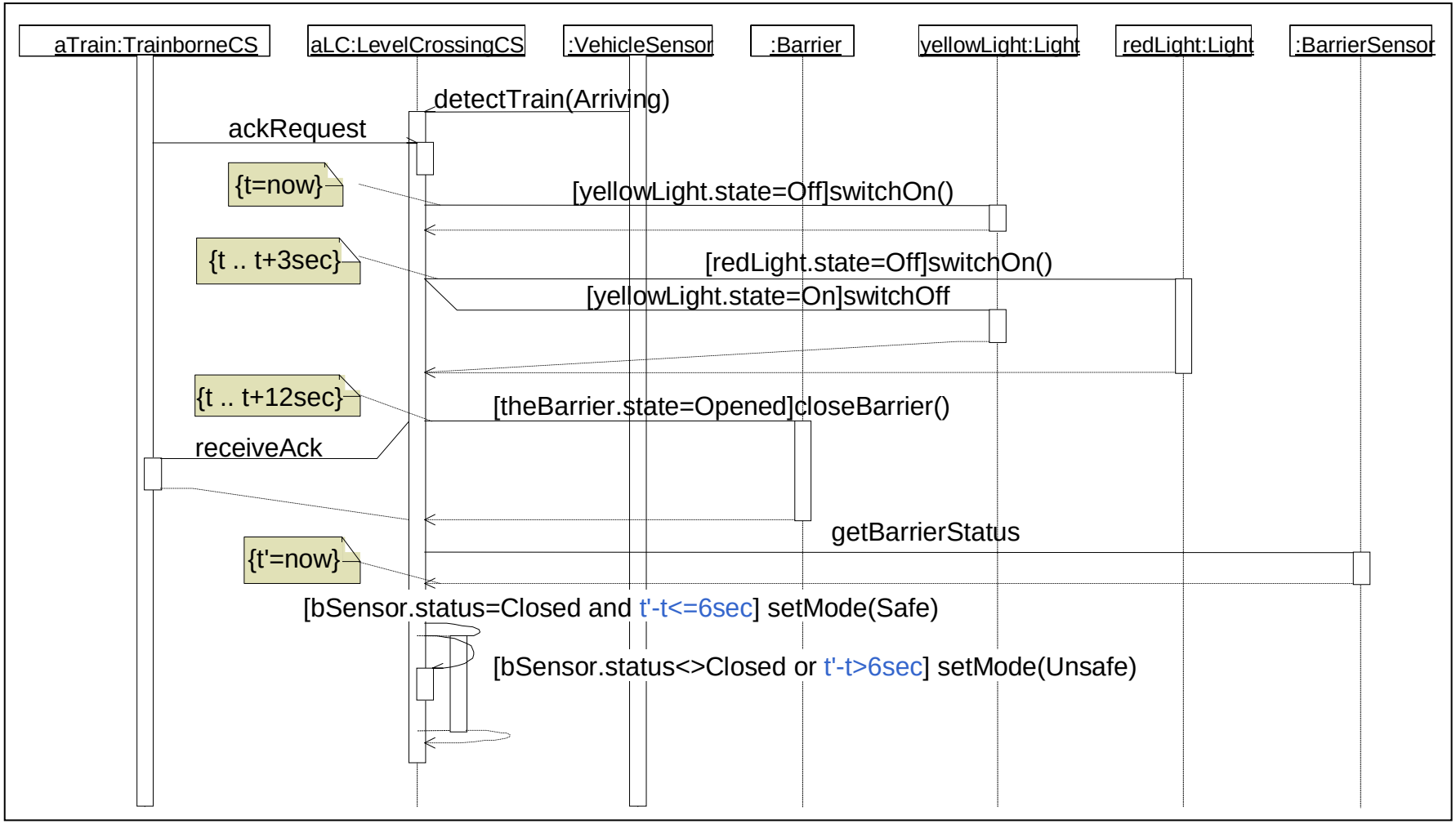


The level crossing: geometry and technical equipment

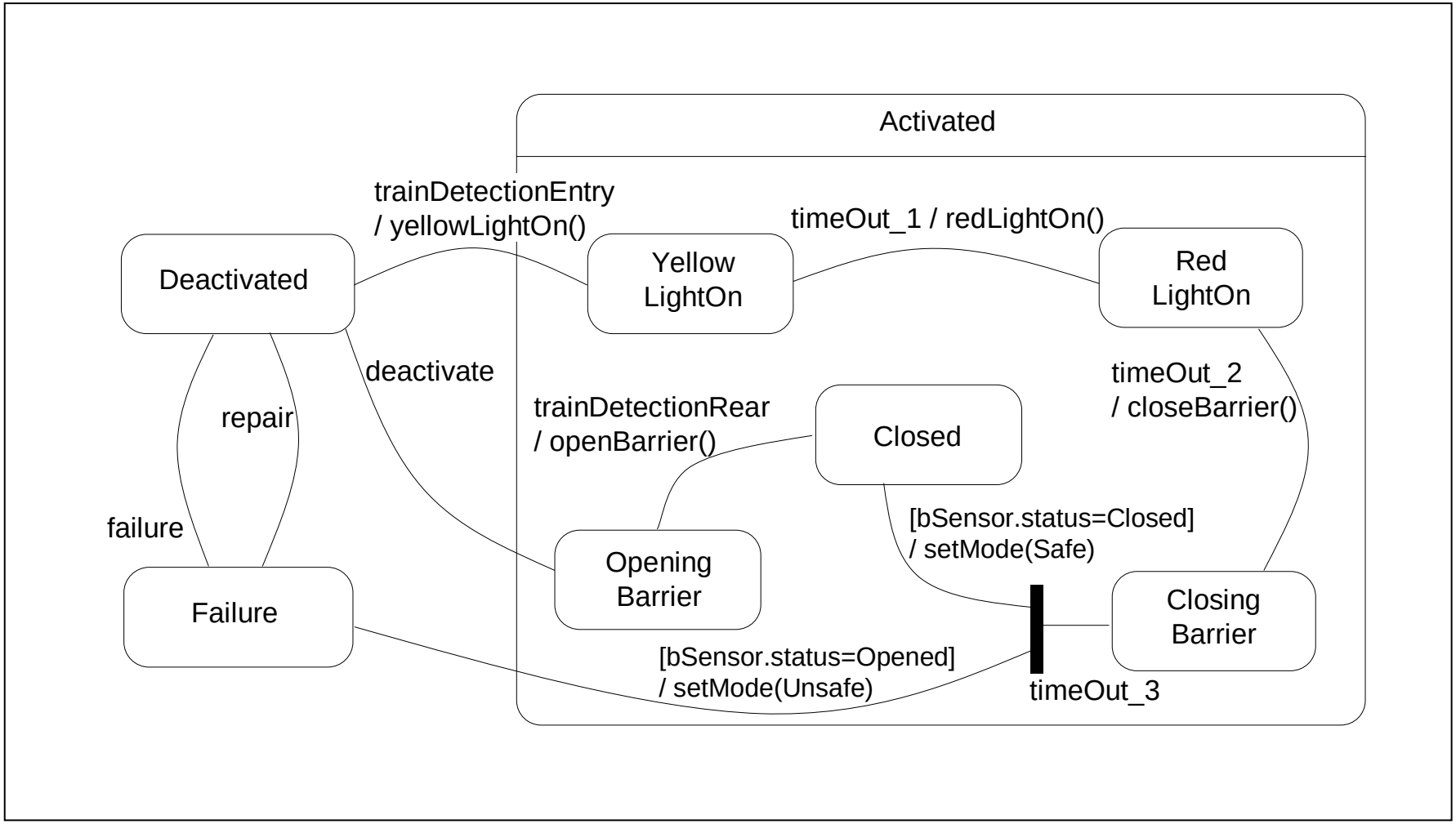
Requirements elicitation

sequence diagrams ...

Scenario of an approaching train



State diagram of the LCC system



Requirements elicitation

main safety properties ...

OCL invariants and pre/post-conditions

```
context LCC_System
inv 1:
   self.theBarrier.state=Closed implies self.redLight.state=On and
   self.theBarrier.state=Closing implies self.yellowLight.state=On and
   self.yellowLight.state=Off and self.redLight.state=Off implies self.theBarrier.state=Open
inv 2:
   self.state=Activated and self.bSensor.state=Open implies self.mode=Unsafe
inv 4:
   self.bSensor.state=Open and self.theBarrier.state=Closed implies self.mode=Unsafe

context LCC_System :: openBarrier
pre:
   self.theBarrier.state=Closed and self.mode=Safe
post:
   self.theBarrier.state=Open
```

IV. From UML to B

… mapping UML diagrams into B formal specifications

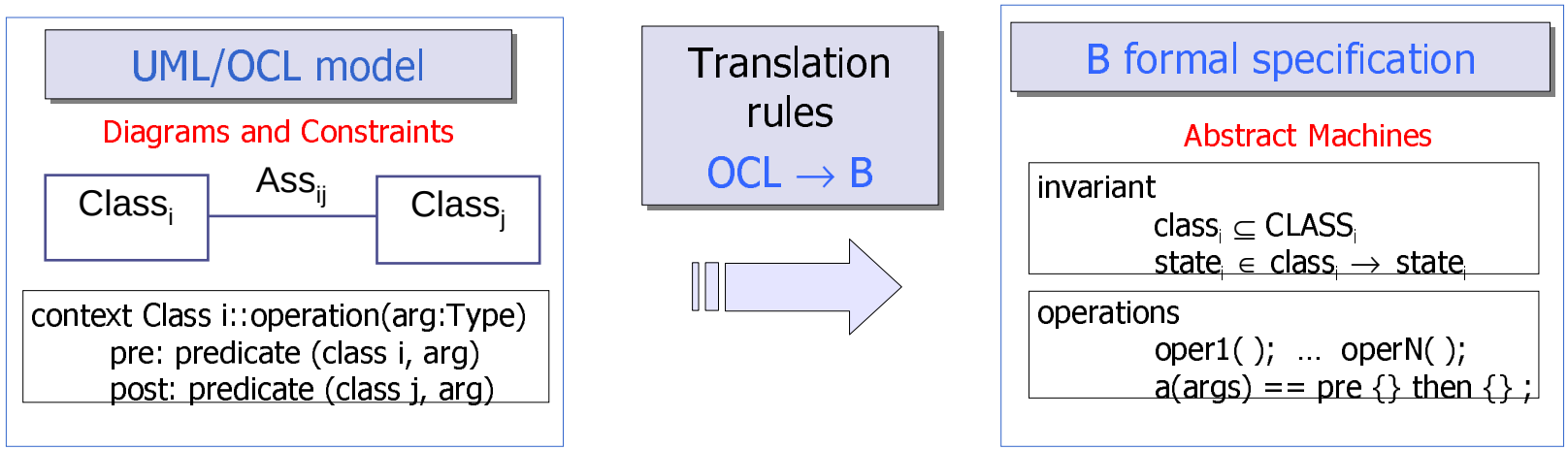
Formal semantics for UML/OCL

Why B ?

UML : collections of objects ↔ B : sets

How ?

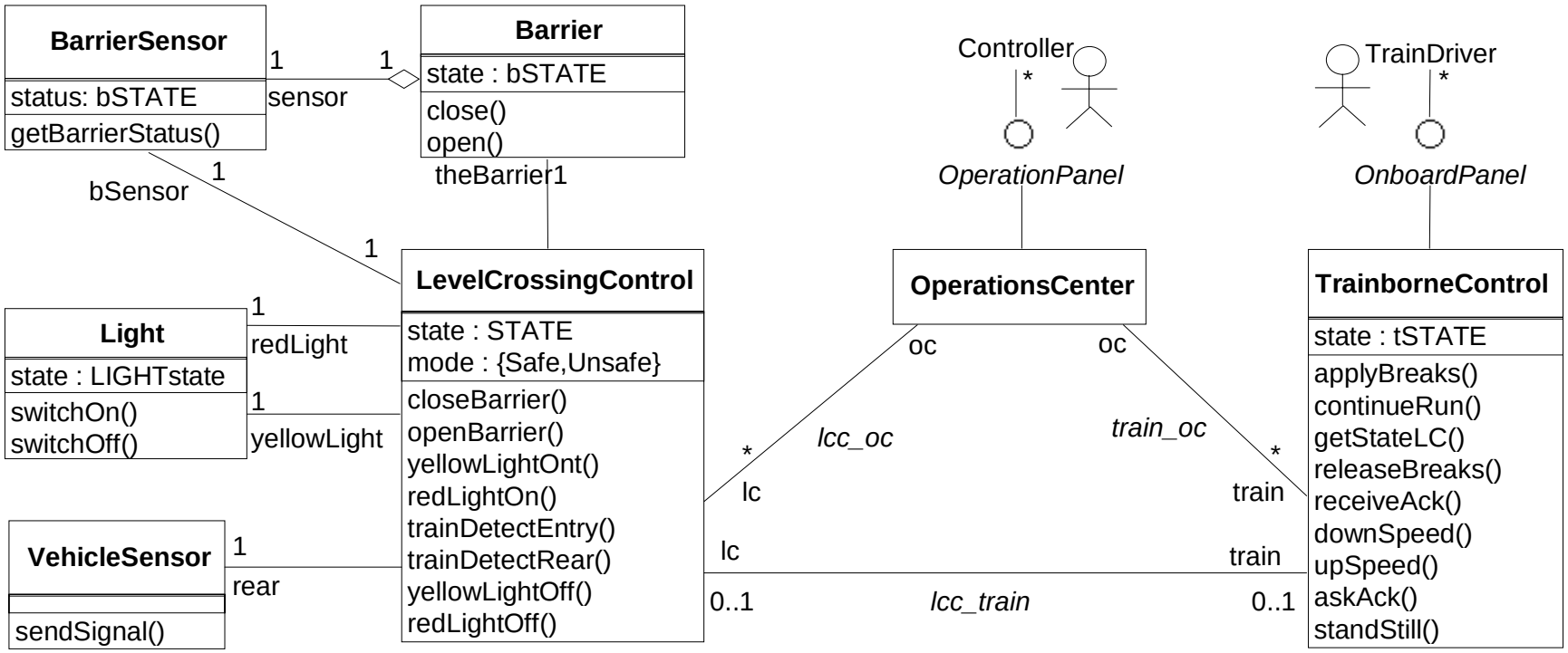
Transformation rules of UML/OCL into B



UML→B

formalization of class diagrams ...

UML:



Translation of a class:

MACHINE *Barrier*

USES *BarrierSensor*

SETS *BARRIER*

VARIABLES *barrier*, *bState*

INVARIANT

$barrier \subseteq BARRIER \wedge$

$bState \in barrier \rightarrow bSTATE$

INITIALISATION

$barrier, bState := \varnothing, \varnothing$

OPERATIONS

openBarrier(*obj*) =

PRE $obj \in barrier \wedge bState(obj) = Closed$

THEN

bState(*obj*):=*Opened*

END;

closeBarrier(*obj*) =

PRE $obj \in barrier \wedge bState(obj) = Opened$

THEN

bState(*obj*):=*Closed*

END;

Part of the B specification – Machine LCC_System:

MACHINE *LCC_System*

SETS *LCC*;

STATE={*Deactivated*,*ShowingYlight*,*ShowingRlight*,*ClosingB*,*OpeningB*,*ClosedB*,*Failure*};

MODE={*Safe*,*Unsafe*}

INCLUDES *Barrier*, *BarrierSensor*, *Red.Light*, *TrainborneCS*, *Yellow.Light*

VARIABLES

lcc, *lcc_barrier*, *lcc_sensor*, *state*, *redLight*, *yellowLight*, *lcc_train*, *mode*

INVARIANT

$lcc \subseteq LCC \wedge$

$lcc\_barrier \in lcc \rightarrow barrier \wedge$

$lcc\_sensor \in lcc \rightarrow bSensor \wedge$

$state \in lcc \rightarrow STATE \wedge$

$redLight \in lcc \rightarrow Red.light \wedge$

$yellowLight \in lcc \rightarrow Yellow.light \wedge$

$lcc\_train \in lcc \;\; train \wedge$

$mode \in lcc \rightarrow MODE \wedge \ldots$

Machine LCC_System – operations:

```
OPERATIONS
timeOut_1_showRlight(obj) =
    PRE     obj∈lcc ∧
            state(obj)=ShowingYlight ∧
            bStatus(lcc_sensor(obj))=Opened ∧
            bState(lcc_barrier(obj))=Opened ∧
            Red.lState(redLight(obj))=Off ∧
            Yellow.lState(yellowLight(obj))=On
    THEN
            state(obj):=ShowingRlight ||
            Yellow.switchOff(yellowLight(obj)) ||
            Red.switchOn(redLight(obj))
    END;

    timeOut_2_closeBarrier(obj)=
    PRE     obj∈lcc ∧
            state(obj)=ShowingRlight ∧
            Red.lState(redLight(obj))=On ∧
            Yellow.lState(yellowLight(obj))=Off
    THEN
            state(obj):=ClosingB ||
            closeBarrier(lcc_barrier(obj))
    END;

timeOut_3_setMode(obj) =
PRE         obj∈lcc ∧
            state(obj)=ClosingB ∧
            bState(lcc_barrier(obj))=Closed ∧
            Red.lState(redLight(obj))=On ∧
            Yellow.lState(yellowLight(obj))=Off
THEN
    SELECT bStatus(lcc_sensor(obj))=Closed THEN
            state(obj):= ClosedB ||
            mode(obj):=Safe
    WHEN bStatus(lcc_sensor(obj))=Opened THEN
            state(obj):= Failure
        ELSE skip
    END
END;
…
```

Machine LCC_System – invariant:

$$\forall obj.(obj \in lcc \wedge bState(lcc\_barrier(obj))=Closed$$
$$\Rightarrow Red.lState(redLight(obj))=On) \wedge$$
$$\forall obj.(obj \in lcc \wedge bState(lcc\_barrier(obj))=Closing$$
$$\Rightarrow Red.lState(yellowLight(obj))=On) \wedge$$
$$\forall obj.(obj \in lcc \wedge Yellow.lState(yellowLight(obj))=Off \wedge Red.lState(redLight(obj))=Off$$
$$\Rightarrow bState(lcc\_barrier(obj))=Opened) \wedge$$

$$\forall obj.(obj \in lcc \wedge state(obj) \in Activated \wedge bStatus(lcc\_sensor(obj))=Opened$$
$$\Rightarrow mode(obj)=Unsafe) \wedge$$

$$\forall obj.(obj \in lcc \wedge bStatus(lcc\_sensor(obj))=Opened \wedge bState(lcc\_barrier(obj))=Closed$$
$$\Rightarrow mode(obj)=Unsafe)$$

IV. From UML to B

translating times features into extended B ...

Classical B

- *Specifying a clock abstract machine*
- *Defining abstract variables holding the times*

Time extended B with timed substitutions

- *Semantic extension of B substitutions :*

  $\rightarrow$ *Derivative of Duration Calculus (WDC*)*

Time extended B

operations including time constraints ...

Machine LCC_System – classical B with time variables:

```
closeBarrier(obj) =
PRE        obj ∈ barrier ∧
           bState(obj)=Opened
THEN
           bState(obj):=Closed
||         ANY newtime WHERE
                newtime ∈ N ∧
                newtime ≥ time ∧
                newtime ≤ time+ClosingDelay
           THEN
                setTime(newtime)
           END
END;
```

```
timeOut_2_closeBarrier(obj)=
PRE        obj∈lcc ∧
           state(obj)=ShowingRlight ∧
           Red.IState(redLight(obj))=On ∧
           Yellow.IState(yellowLight(obj))=Off ∧
           time – trainDetectionEntry_Time ≤ 12
THEN
           state(obj):=ClosingB ||
           closeBarrier(lcc_barrier(obj)) ||
           closeBarrierTime:=time
END;
```

```
timeOut_3_setMode(obj) =
PRE        obj∈lcc ∧
           state(obj)=ClosingB ∧
           bState(lcc_barrier(obj))=Closed ∧
           Red.IState(redLight(obj))=On ∧
           Yellow.IState(yellowLight(obj))=Off
THEN
     SELECT bStatus(lcc_sensor(obj))=Closed ∧
            time – closeBarrierTime ≤ 6
     THEN
            state(obj):= ClosedB ||
            mode(obj):=Safe
     WHEN bStatus(lcc_sensor(obj))=Opened ∧
            time – closeBarrierTime > 6
     THEN
            state(obj):= Failure
          ELSE skip
     END
END;
```

Time extended B

timed substitutions ...

Classic B substitutions

GSL	$[GSL]P$	$\mathrm{dur}([GSL]P)$
skip	P	$[\mid\ \mid]$
$x := E$	$P\,[E/x]$	$[\mid\ \mid]$
$g \mid S$	$g \wedge [S]P$	
$g \Rightarrow S$	$g \Rightarrow [S]P$	
$S\,;\,T$	$[S]\,([T]P)$	
$S\,[]\,T$	$[S]P \wedge [T]P$	$\mathrm{dur}(\,[S]P\,) \wedge \mathrm{dur}(\,[T]P\,)$
delay d		$(\,l = d\,) \wedge [\mid P \mid]$

Extended Hoare Triplets

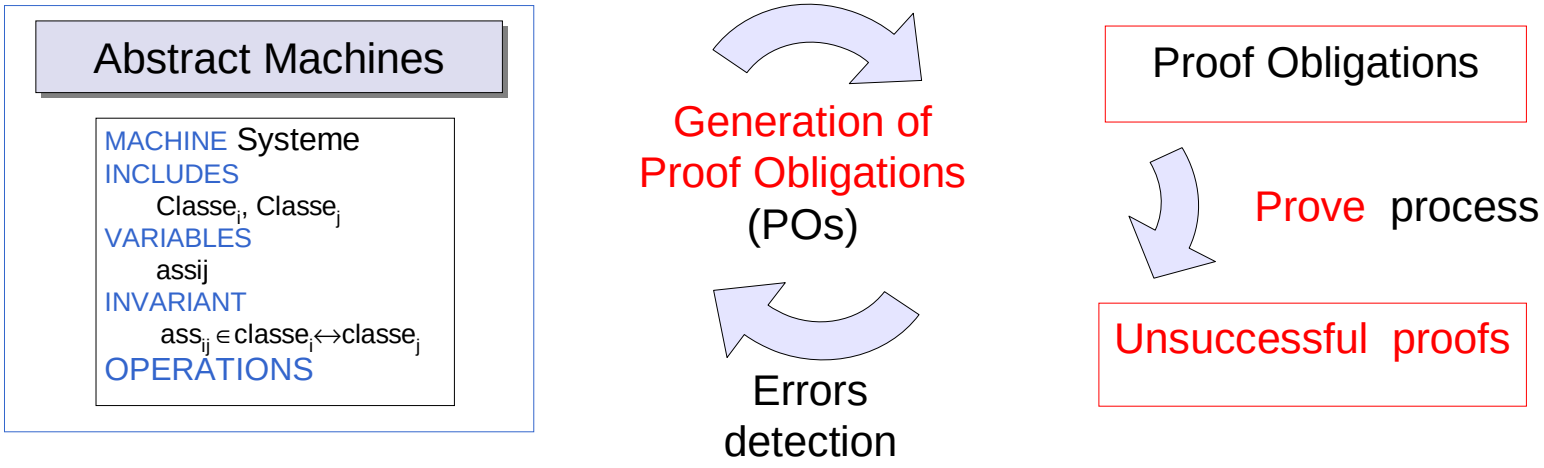
$$\frac{\{[S]I\}\ [S, dur([S], I)]\ \{I\} \qquad I \wedge C \Rightarrow [S]I \qquad I \wedge \neg C \Rightarrow P}{\{I\}\ \left[\begin{array}{ll} \text{WHILE } C & \\ \text{DO } S & , dur([S], I)^* \\ \text{INVARIANT } I & \end{array}\right]\ \{P\}}$$

Timed substitutions

```
x ← Example1 =
    TIMING
        PRE
            x ≥ 1
        THEN
            delay 1; x:=x-1;
        END
    POST
        x ≥ 0
    REQUIRES
        □ ( [| x ≥ 0 |] )
    END
```

V. Analysis and verification …

Generating Proof Obligations (POs)



The prove process facilitate errors detection

- *Syntax and type errors*
- *Incompleteness of pre-conditions*
- *Inconsistent post-conditions*

VI. Conclusions and discussion …

Classical B with time variables

- *Does not allow complex timed properties*
  (liveness and fairness)
- *Enough to railway control systems*
- *Easiest implementation*

Extended B with duration calculus

- *Timing diagrams needed*
- *Allows to express complex timing properties*

Event B ? → CTL semantics

Conclusions and discussion …

Advantages of using B

- Analysis, verification and simulation :
  - ➡ coherence/consistency of UML/OCL
- Use of existing proof tools
- Refinement
- Successfully used in railway applications