



# *chrono*Event-B: A Rodin Plugin for Timed Constraints in Event-B

Amel Mammar[1] , Cabrel Feukeng Momo[1,2], **Marc Frappier[2]**, Paul Gibson[1]

[1] SAMOVAR, Institut Polytechnique de Paris, Télécom SudParis, France

first.last@telecom-sudparis.eu

[2]Université de Sherbrooke, 2500 Boulevard de l'Université, Sherbrooke(QC), Canada

first.last@usherbrooke.ca

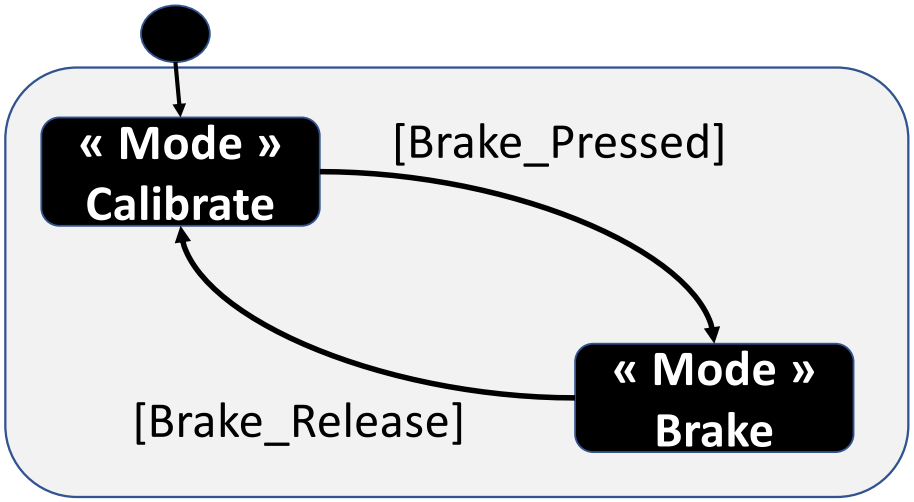
# 1 - Context

- Timing constraints are often considered too late in the design process
- Unsatisfactory for critical systems
  - temporal properties are essential for safety
  - must be integrated from the early modeling stages to prevent catastrophic failures
- Time is not a native concept in Event-B
- Managing timing constraints introduces a lot of details that are cumbersome to deal with
- This plugin hide some of the details

# 2 - Case Study

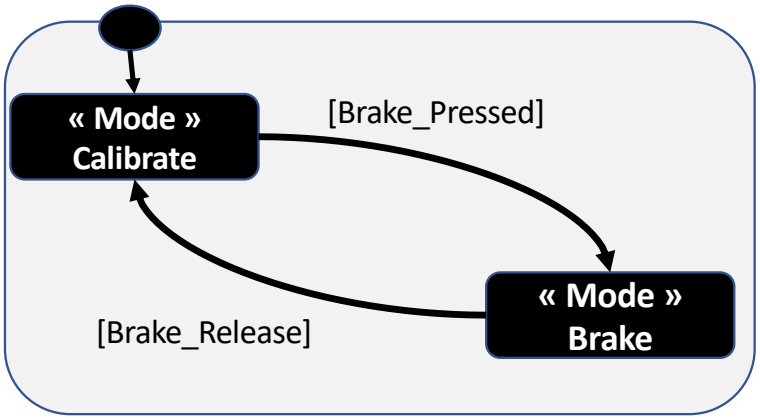
*Anti-lock Braking System : ABS*

The ABS is a control unit that ensures a car's stability while driving and in extreme braking situations. It operates in two modes: Calibrate mode, which is the default mode, and Brake mode.




- **Calibration Mode**
  - SenseSpeed
  - Calibrate
- **Braking Mode**
  - SenseBrake
  - BrakeControl
  - BrakeWheel

# 2 - Case Study *Anti-lock Braking System requirements*




## Calibrate Mode (Calibration)

**ABS1**: SenseSpeed is always followed by Calibrate ***Scheduling**

**ABS2**: SenseSpeed is periodic with a period of 100 ms ***Periodicity**

**ABS3**: Calibrate is performed within 10 ms following SenseSpeed ***Duration**

## Brake Mode (Braking)

**ABS4**: SenseBrake is always followed by BrakeControl ***Scheduling**

**ABS5**: BrakeControl is always followed by BrakeWheel ***Scheduling**

**ABS6**: SenseBrake is periodic with a period of 10 ms ***Periodicity**

**ABS7**: BrakeWheel is performed within 1 ms following SenseBrake ***Duration**

# 3 - State of Art

| | Contributions | Limitations |
|---|---|---|
| H. Peng, X. Zhang, G. Cao, Z. Liu, Y. Jing, and L. Rao. *A **time refinement framework based on iUML-B state machine. Scientific Programming*** | Modeling delays using temporal variables and deadlines. | Limited handling of simultaneous events. |
| M. R. Sarshogh and M. J. Butler. ***Specification and refinement of discrete timing properties in Event-B.*** | Addition of Deadline, Delay, and Expiry properties | Approach specific to trigger-response models |
| C. Fu and K. Zheng. ***Patterns for modeling task-level timing constraints with Event-B.*** | Task synchronization (priority, coincidence) | No tool support available |
| R. Banach, M. J. Butler, S. Qin, N. Verma, and H. Zhu. **Core hybrid event-b I : *Single hybrid Event-B machines*.** | Hybrid extension with continuous behavior | Lack of a functional tool |
| A. Iliasov, A. B. Romanovsky, L. Laibinis, E. Troubitsyna, and T. Latvala. ***Augmenting Event-B modeling with real-time verification***<br>*&*<br>J. Berthing, P. Boström, K. Sere, L. Tsiopoulos, and J. Vain. ***Refinement-based development of timed systems*** | Verification of temporal properties using timed automata (UPPAAL) | Multiplicity of formalisms and the need for advanced knowledge to effectively use them |

# 4 - Definitions

- A **Task** is a unit of work or an operation that needs to be executed. It represents a specific activity within a system or application.

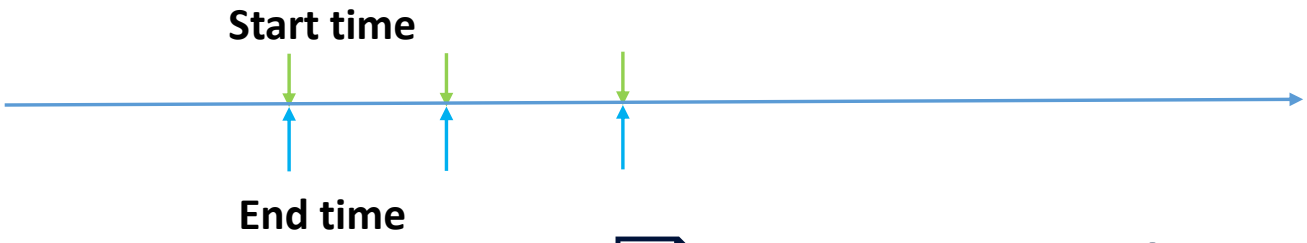
  eg : processing a request

- A **Scheduler** is the component responsible for planning and organizing task execution. It determines *when* and *in what order* tasks should run.

  eg : operating system scheduler

# 5 – Timed Properties : Timed constraints

## ❑ Atomic Task



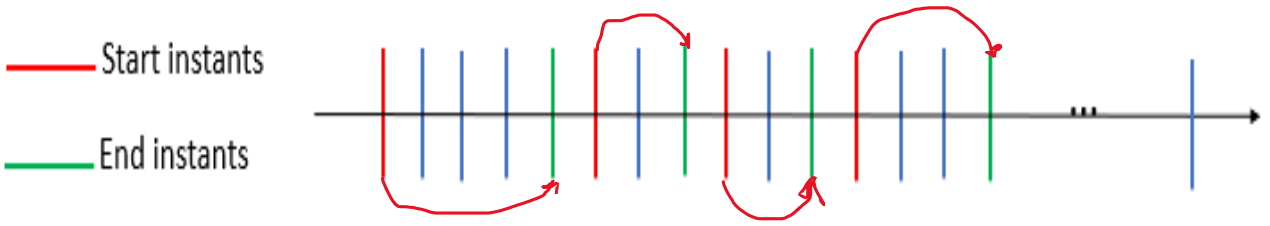

## ❑ Non-Atomic Task

# 5 – Modeling : TaskScheduler Type

The set of all non-decreasing functions $f$ from $1..n$ to $\mathbb{N}$ such that values at odd positions separated by two indices are different.

$$\{n, f.\ n \in \mathbb{N} \land f \in 1..n \to \mathbb{N} \land \big(\forall x, y.\ x \in 1..n \land y \in 1..n \land x \leq y \implies f(x) \leq f(y)\big) \land (\forall k \cdot k \in 1 \cdot\cdot (n-1) \div 2 \Rightarrow f(2k-1) \neq f(2k+1)) \mid f\}$$

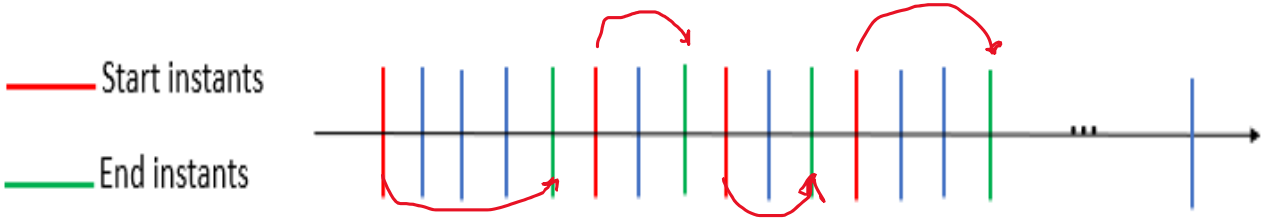
$$\{1 \mapsto 0, 2 \mapsto 4, 3 \mapsto 5, 4 \mapsto 7, 5 \mapsto 8, 6 \mapsto 10, 7 \mapsto 11, 8 \mapsto 14, \ldots\}$$



# 5 – Modeling : TaskScheduler Type

## Each task has its own task scheduler

$$\{1 \mapsto 0, 2 \mapsto 4, 3 \mapsto 5, 4 \mapsto 7, 5 \mapsto 8, 6 \mapsto 10, 7 \mapsto 11, 8 \mapsto 14, \ldots\}$$




## TaskScheduler operations

- TaskSchedulerBefLast
- TaskSchedulerAppend
- TaskSchedulerSize
- TaskSchedulerFirst
- TaskSchedulerLast

# 5 – Modeling : Progress Event and global clock

To mark the progress of our global clock **CK** in steps of time unit, we define the *ProgressEvent* event with the step parameter.

```
ProgressEvent:   not extended ordinary          ›
ANY
◦   step      ›
WHERE
◦   @progressGrd:   step ∈ ℕ1 not theorem ›
THEN
◦   @progressAct:   CK ≔ CK + step ›
END
```

# 5 – Modeling : Atomic Task

Let $\boldsymbol{SH_A}$ Be the taskScheduler of the task A

INV $\quad \boldsymbol{atomicTaskSchedule}(\boldsymbol{SH_A}) \triangleq$

$$(\exists k.\ k \in \mathbb{N} \wedge taskSchedulerSize(SH_A) = 2k)$$

$$\wedge$$

$$(\forall k.k \in 1 \ldots taskSchedulerSize(SH_A) \div 2 \Rightarrow SH_A(2k-1) = SH_A(2k))$$

**Event** Atomic_A $\widehat{=}$

**then**

act1: $SH_A$ := **taskSchedulerAppend(taskSchedulerAppend(**$SH_A$,$CK$**),CK)**

**end**

# 5 – Modeling : Task with Duration

Let $\boldsymbol{SH_A}$ Be the taskScheduler of the task A

INV $\quad \boldsymbol{taskSchedulerDuration(SH_A, d)} \triangleq$

$$(\forall k.\ k \in 1 \ldots taskSchedulerSize(SH_A) \div 2)$$

$$\Rightarrow$$

$$SH_A(2k) - SH_A(2k-1) = d$$

**Event** start_A $\widehat{=}$

**where**

grd1: $\exists k.(k \in \mathbb{N} \wedge \textbf{size}(SH_A)=2k)$

**then**

act1: $SH_A := \textbf{taskSchedulerAppend}(SH_A, CK)$

**end**

**Event** end_A $\widehat{=}$

**where**

grd1: $\exists k.k \in \mathbb{N} \wedge \textbf{taskSchedulerZise}(SH_A)=2k+1$

grd2: $CK$ - $\textbf{taskSchedulerLast}(SH_A) = d$

**then**

act1: $SH_A := \textbf{taskSchedulerAppend}(SH_A, CK)$

**end**

# 5 – Modeling : Task with period

The difference between the start time of an occurrence (index 2k+1) and the start time of the preceding occurrence (index 2k−1) is equal to p u.t.


INV $\boldsymbol{periodicalTaskScheduler(SH_A, p)} \triangleq$

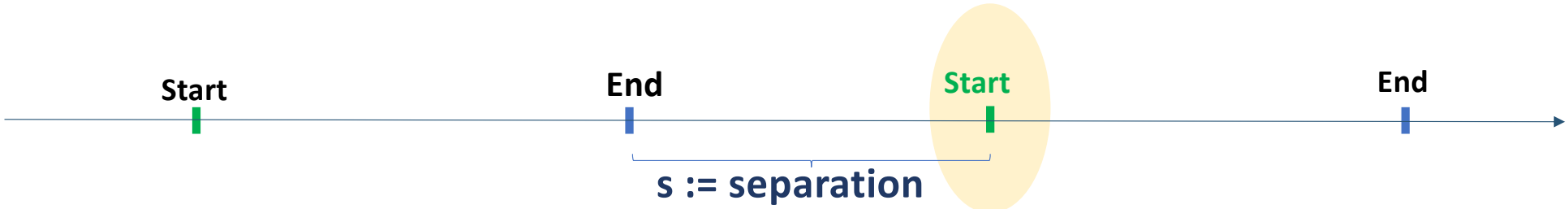
$$\forall k.k \in 1 \ldots (taskSchedulerSize(SH_A) - 1) \div 2$$
$$\Rightarrow$$
$$SH_A(2k+1) - SH_A(2k-1) = p$$

A guard @grd1 is added to the event **Atomic_A** for an atomic task (respectively **start_A** for a non-atomic task).

@grd1: $SH_A \neq \varnothing \Rightarrow CK$ - $\mathbf{taskSchedulerBefLast}(SH_A) = p$

# 5 – Modeling : Task with Separation

The difference between the start time of an occurrence (index 2k+1) of A and the end time of the preceding occurrence (index 2k) is equal to s u.t




INV $sepDuration(SH_A, s) \triangleq$

$$\forall k.k \in 1 \ldots (taskSchedulerSize(SH_A) - 1) \div 2$$
$$\Rightarrow$$
$$SH_A(2k+1) - SH_A(2k) = s$$

A guard @grd4 is added to the event **Atomic_A** for an atomic task (respectively **start_A** for a non-atomic task).

@grd4: $taskSchedulerSize(SH_A) \geq 2 \Rightarrow CK - \text{taskSchedulerLast}(SH_A) = s$

# 5 – Modeling : Precedence relationship

Given two tasks A and B, this dependency requires that every execution of an occurrence of task A be preceded by the execution of an occurrence of task B before the next occurrence of A.

INV

$$\boldsymbol{preceded}(\boldsymbol{SH_A}, \boldsymbol{SH_B}) \triangleq$$

$$\forall k.\, k \in 1..(taskSchedulerSize(SH_A) + 1) \div 2$$
$$\Rightarrow$$
$$\exists k'.\, k' \in 1..taskSchedulerSize(SH_B) \wedge$$
$$SH_B(2k') \leq SH_A(2k-1) \wedge$$
$$2k - 2 \in dom(SH_A) \Rightarrow SH_B(2k'-1) \geq SH_A(2k-2)$$

**PRECEDED_BY(A,B)**

- ✓ **BABAB…**
- ✓ **BBBABBBABBA**
- ✗ **A…**

# 5 – Modeling : Preceded relationship

The guard @grdpcd is added to the event Atomic_A if task A is atomic (respectively start_A if task A is non-atomic).

@grdpcd: $(SH_A = \varnothing \Rightarrow \exists\, k.\ 2k \in \mathbf{dom}(SH_B)) \wedge$

$(SH_A \neq \varnothing \Rightarrow \exists\, k.\ 2k+1 \in \mathbf{dom}(SH_B \rhd \mathbf{taskSchedulerLast}(\mathrm{SH}_A)..CK) \wedge 2k+2 \in$

$\mathbf{dom}(SH_B \rhd \mathbf{taskSchedulerLast}(\mathrm{SH}_A)..CK))$

**PRECEDED_BY(A,B)**

- ✓ **BABAB…**
- ✓ **BBBABBBABBA**
- ✗ **A…**

# 5 – Modeling : followed relationship

Given two tasks A and B, this constraint requires that the execution of an occurrence of task A be followed by the execution of an occurrence of task B, before the next occurrence of A.

INV $\boldsymbol{followed}(\boldsymbol{SH_A}, \boldsymbol{SH_B}) \triangleq \forall k.$

$$k \in 1..(taskSchedulerSize(SH_A) - 1) \div 2$$

$$\Rightarrow$$

$$\exists k'.\ (2k' - 1 \in dom(SH_B \rhd SH_A(2k)..SH_A(2k+1)) \wedge$$

$$2k' \in dom(SH_B \rhd SH_A(2k)..SH_A(2k+1))$$

**FOLLOWED_BY(A,B)**

- ✓ **ABABAB…**
- ✗ **BBABBBABBA**
- ✗ **BABAABAA…**
- ✗ **AA…**

**PRECEDED_BY(A,B)**

- ✓ **BABAB…**
- ✓ **BBBABBBABBA**
- ✗ **A…**

# 5 – Modeling : followed relationship

The guard @grdfwd is added to the event Atomic_A if A is an atomic task (respectively start_A if A is a non-atomic task).

$$\texttt{@grdfwd}: \quad \mathrm{SH}_A \neq \varnothing \Rightarrow \exists\, k.\ (2k+1 \in \mathbf{dom}(SH_B \rhd \mathbf{taskSchedulerLast}(\mathrm{SH}_A) \mathinner{..} CK) \wedge 2k+2 \in \mathbf{dom}(SH_B \rhd \mathbf{taskSchedulerLast}(\mathrm{SH}_A) \mathinner{..} CK))$$

**FOLLOWED_BY(A,B)**

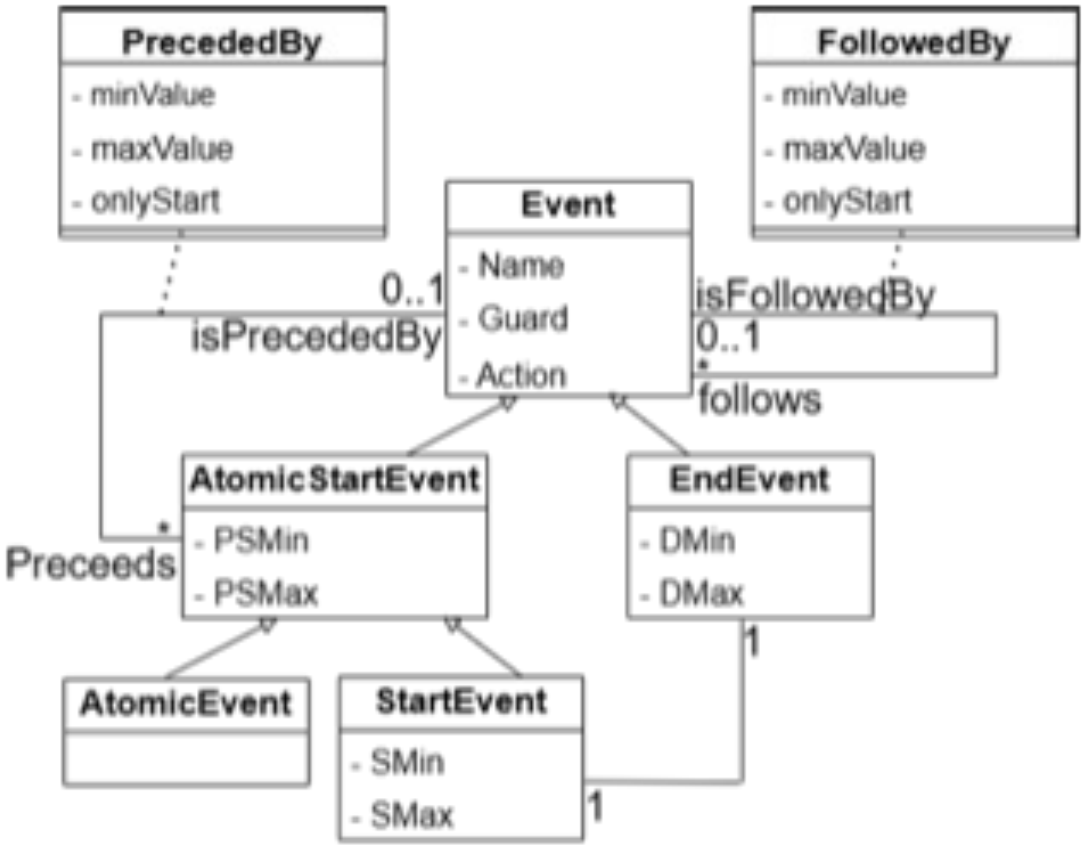
- ✓ **ABABAB…**
- ✓ **AA…**
- ✗ **BBABBBABBA**
- ✗ **BABAABAA…**

# 6- Plugin Features

- Creation of atomic and non-atomic tasks
- Creation of dependency constraints between tasks
  - PrecededBy (min, max)
  - FollowedBy (min,max)
- Adding temporal constraints to an existing task
  - Period and Separation
- Automatic generation of guards and invariants
- Proof rules provided in theory

Ref : https://github.com/AmelMammar/chronoEventB

# Event Metamodel



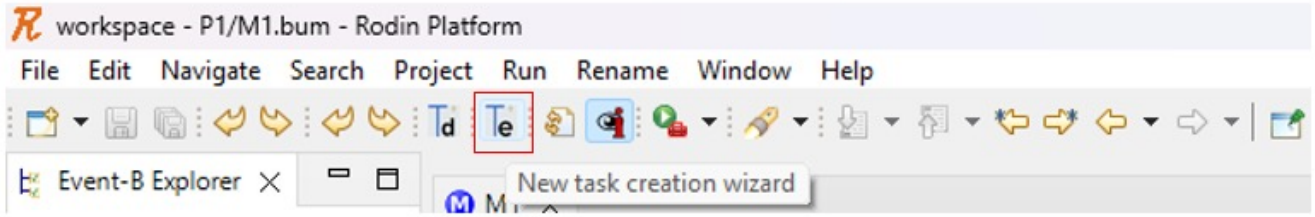
# Conclusion

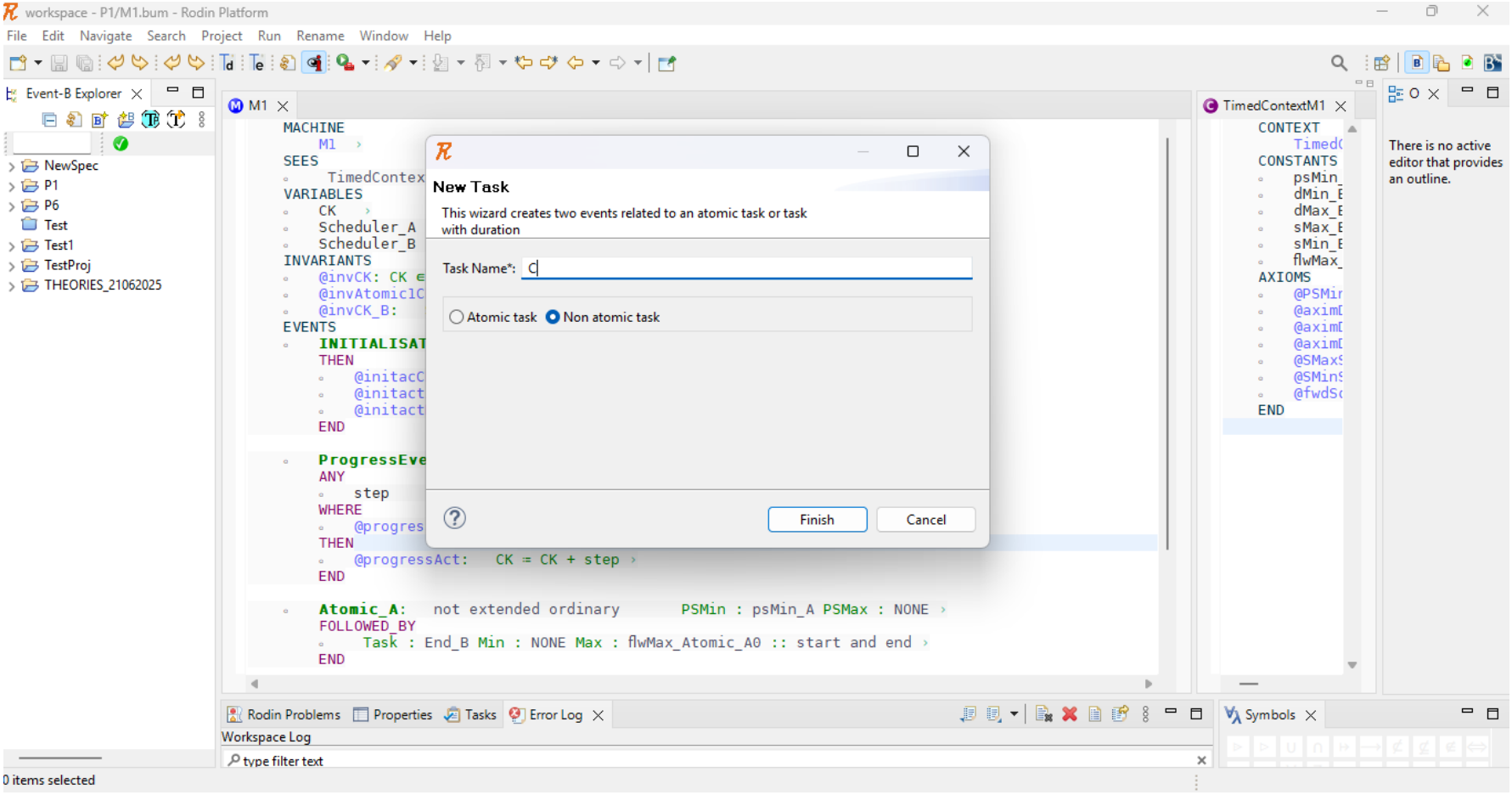
- A classification of temporal constraints for Event-B, providing a structured framework for their specification: a dual modeling of tasks
- A mathematical formalization of temporal dependencies, ensuring consistency between tasks.
- The implementation of a functional Rodin plugin that seamlessly integrates our theoretical advances into the development environment.

# Perspectives

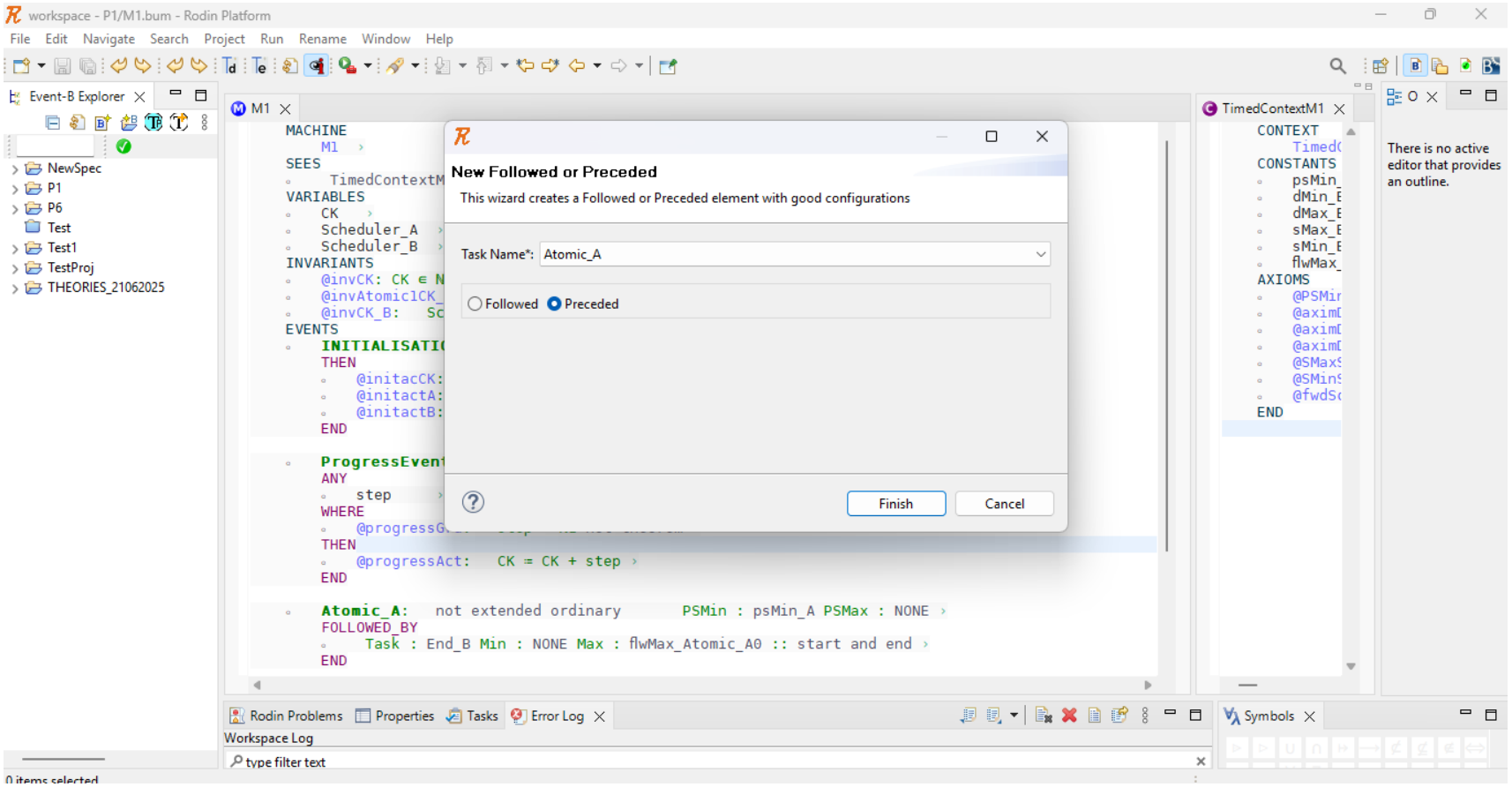
- Hierarchical task refinement
  - Develop a mechanism allowing an abstract task to be refined into a set of more precise subtasks,
  - while preserving temporal properties.
- Adaptive temporal dependencies
  - Introduce the ability to refine dependencies between tasks
  - Enable the specification of increasingly precise temporal constraints throughout the system refinement process.

# Creation of a new task




Creation of a new dependency



# Illustration

```
Atomic_A:   not extended ordinary       PSMin : psMin_A PSMax : NONE ›
FOLLOWED_BY
    Task : End_B Min : NONE Max : flwMax_Atomic_A0 :: start and end ›
    Task : Atomic_D Min : NONE Max : flwMax_Atomic_A1 :: only start ›
PRECEDDED_BY
    Task : Start_B Min : pcdMin_Atomic_A0 Max : pcdMax_Atomic_A0 :: start and end ›
    Task : Atomic_T Min : NONE Max : pcdMax_Atomic_A1 :: only start ›
END
```

```
Start_B:    not extended ordinary   PMin : NONE PMax : NONE SMin : sMin_B SMax : sMax_B   ›
FOLLOWED_BY
    Task : Atomic_A Min : NONE Max : NONE :: only start ›
    Task : Atomic_T Min : NONE Max : NONE :: start and end ›
END
```

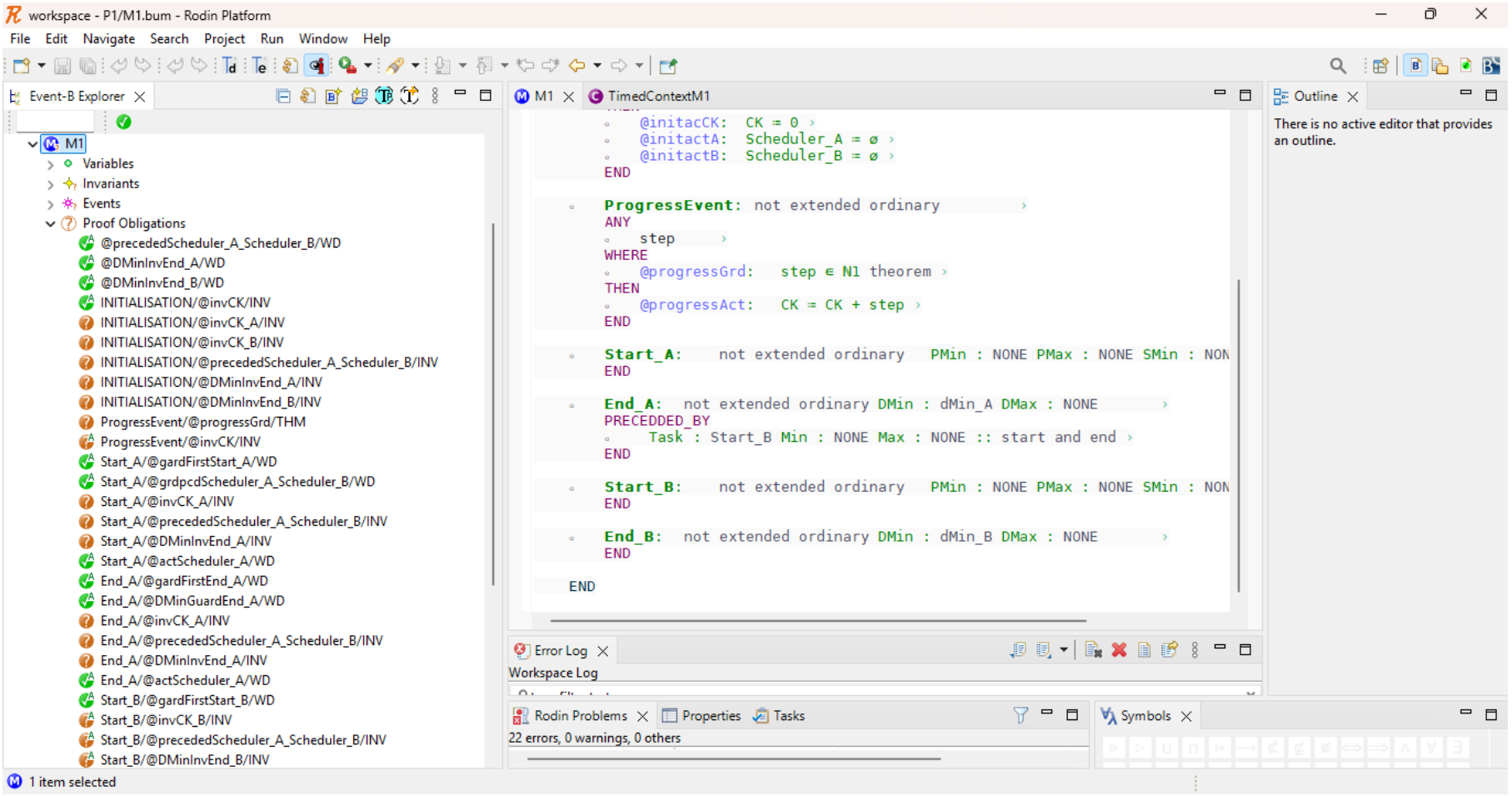
```
End_B:  not extended ordinary DMin : dMin_B DMax : dMax_B       ›
PRECEDDED_BY
    Task : Atomic_D Min : pcdMin_End_B0 Max : NONE :: only start ›
    Task : Atomic_T Min : NONE Max : pcdMax_End_B1 :: start and end ›
END
```

```
Start_B:    not extended ordinary   PMin : NONE PMax : NONE SMin : sMin_B SMax : sMax_B   ›
END
```

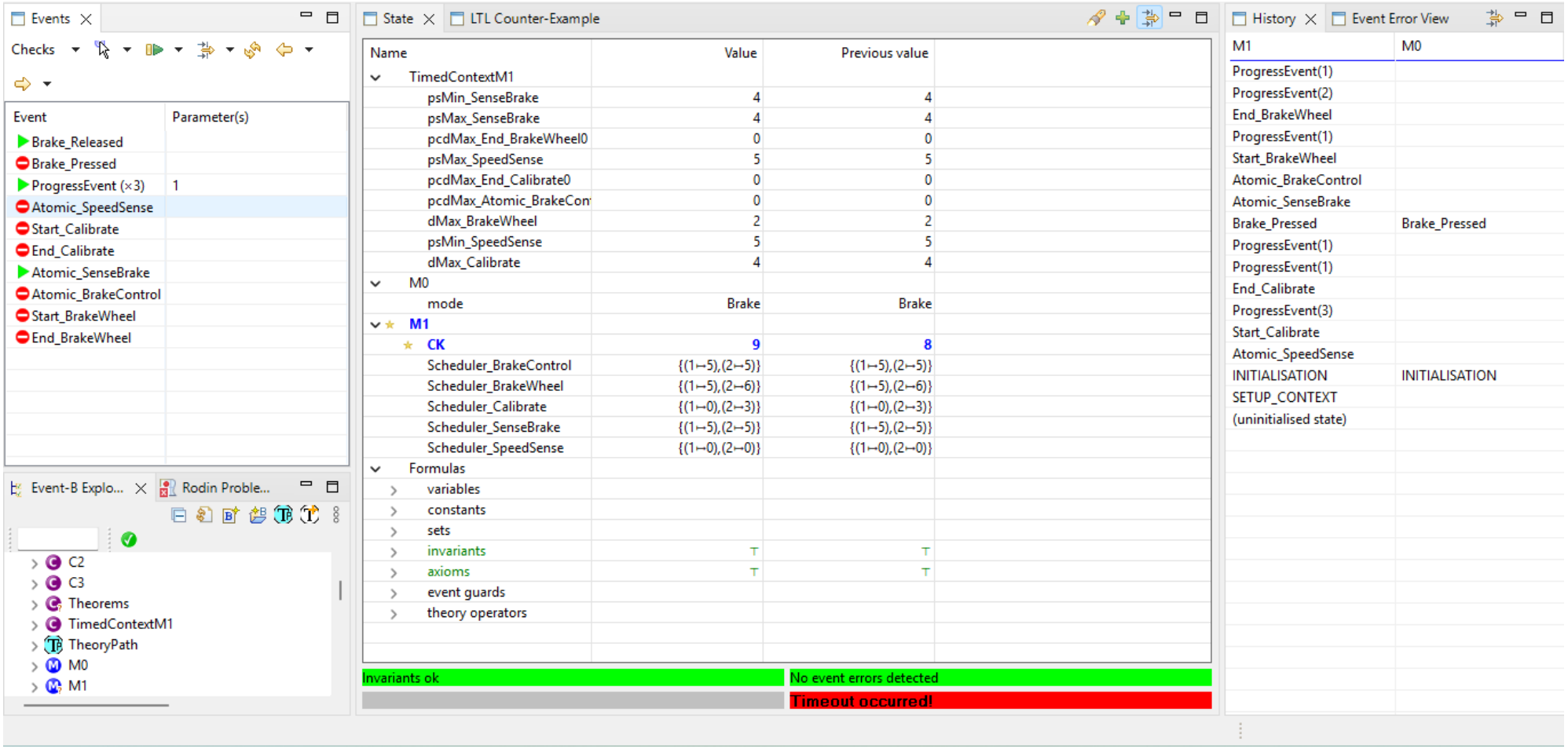
```
End_B:  not extended ordinary DMin : dMin_B DMax : dMax_B       ›
END
```

```
Atomic_D:   not extended ordinary       PSMin : NONE PSMax : NONE ›
END
```

# Proof obligations



# ProB Model Checking



# ABS requirements

| N° Contrainte | Modélisation en *Event-B* |
|---|---|
| *ABS2* | *periodicalMinMaxTaskScheduler*(SenseSpeed, $10^5$, $10^5$) |
| *ABS6* | *periodicalMinMaxTaskScheduler*(SenseBrake, $10^4$, $10^4$) |
| *ABS3* | *taskSchedulerMaxDuration*(Calibrate, $10^4$) |
| *ABS7* | *taskSchedulerMaxDuration*(BrakeWheel, $10^3$) |
| *ABS1* | *followed*(SenseSpeed, Calibrate) |
| *ABS4* | *followed*(SenseBrake, BrakeControl) |
| *ABS5* | *followed*(BrakeControl, BrakeWheel) |
| *ABS3* | *precededMax*(Calibrate, SenseSpeed, 0) |
| *ABS7 et ABS4* | *precededMax*(BrakeControl, SenseBrake, 0) |
| *ABS7 et ABS5* | *precededMax*(BrakeWheel, BrakeControl, 0) |
| *ABS7* | *preceded*(BrakeWheel, SenseBrake) |

*Thank you*



# *chrono*Event-B: A Rodin Plugin for Timed Constraints in Event-B

Amel Mammar[1] , Cabrel Feukeng Momo[1,2], **Marc Frappier[2]**, Paul Gibson[1]

[1] SAMOVAR, Institut Polytechnique de Paris, Télécom SudParis, France

first.last@telecom-sudparis.eu

[2]Université de Sherbrooke, 2500 Boulevard de l'Université, Sherbrooke(QC), Canada

first.last@usherbrooke.ca