Self-adaptive multi-fidelity railway simulation

LMF Non-permanent seminar

Inès Yeterian

November 2nd 2026

Sommaire

Industrial context

Software architecture

Self-adaptation for multi-fidelity simulation

Experimental validation

Conclusion

Modeling and Simulation

My PhD work focuses on railway simulation.
We can distinguish simulation models by the model of execution, eg the way time is modeled:

- Time = succession of timestamps (discrete event paradigm),
- Time = continuous function,
- Time = succession of variable time increments,
- etc.

Modeling and Simulation

My PhD work focuses on railway simulation.
We can distinguish simulation models by the model of execution, eg the way time is modeled:

- Time = succession of timestamps (discrete event paradigm),
- Time = continuous function,
- Time = succession of variable time increments,
- etc.

The execution can be handled sequentially or concurrently:

Modeling and Simulation

My PhD work focuses on railway simulation.
We can distinguish simulation models by the model of execution, eg the way time is modeled:

- Time = succession of timestamps (discrete event paradigm),
- Time = continuous function,
- Time = succession of variable time increments,
- etc.

The execution can be handled sequentially or concurrently:
Choosing the most appropriate simulation model depends on the modeling context !

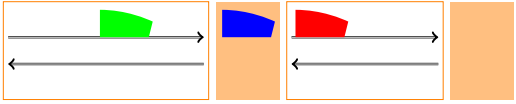
OLISTIC discrete event simulator [DB14]

SNCF developed a discrete event simulator called OLISTIC for Simulation and Optimization.
The use case for those types of simulations : high fidelity and **speedy computations.**

OLISTIC discrete event simulator [DB14]

SNCF developed a discrete event simulator called OLISTIC for Simulation and Optimization.
The use case for those types of simulations : high fidelity and **speedy computations.**
$\rightarrow$ Discrete Event Simulation.

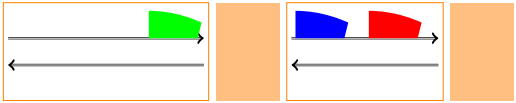
The OLISTIC model of railway operations



Train	OP	Timestamp (in)	Timestamp (out)
Red	GdN	26/06/22 9:25:00	26/06/22 9:28:00
Blue	GdN	26/06/22 9:30:00	26/06/22 9:33:00
Green	GdN	26/06/22 9:35:00	26/06/22 9:37:00
Red	St-Denis	26/06/22 9:38:00	26/06/22 9:42:00

$t = 26/06/22\ 9:30:00$

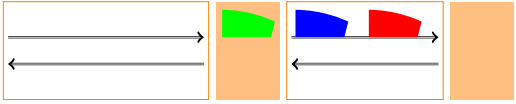
The OLISTIC model of railway operations



Train	OP	Timestamp (in)	Timestamp (out)
Red	GdN	26/06/22 9:25:00	26/06/22 9:28:00
Blue	GdN	26/06/22 9:30:00	26/06/22 9:33:00
Green	GdN	26/06/22 9:35:00	26/06/22 9:37:00
Red	St-Denis	26/06/22 9:38:00	26/06/22 9:42:00

$$t = 26/06/22\ 9:33:00$$

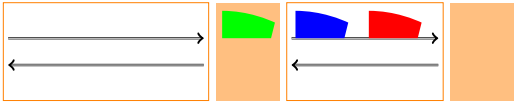
The OLISTIC model of railway operations



Train	OP	Timestamp (in)	Timestamp (out)
Red	GdN	26/06/22 9:25:00	26/06/22 9:28:00
Blue	GdN	26/06/22 9:30:00	26/06/22 9:33:00
Green	GdN	26/06/22 9:35:00	26/06/22 9:37:00
Red	St-Denis	26/06/22 9:41:00	26/06/22 9:44:00

$$t = 26/06/22\ 9:35:00$$

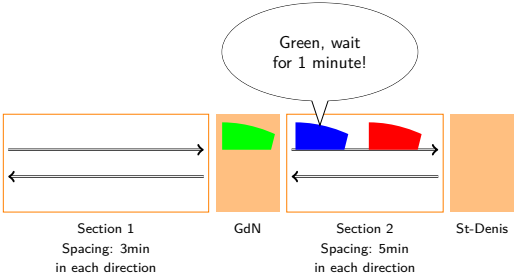
The OLISTIC model of railway operations



Train	OP	Timestamp (in)	Timestamp (out)
Red	GdN	26/06/22 9:25:00	26/06/22 9:28:00
Blue	GdN	26/06/22 9:30:00	26/06/22 9:33:00
Green	GdN	26/06/22 9:35:00	26/06/22 9:37:00
Red	St-Denis	26/06/22 9:38:00	26/06/22 9:42:00

$t = 26/06/22\ 9:37:00$

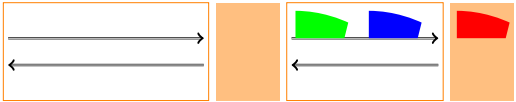
The OLISTIC model of railway operations



Train	OP	Timestamp (in)	Timestamp (out)
Red	GdN	26/06/22 9:25:00	26/06/22 9:28:00
Blue	GdN	26/06/22 9:30:00	26/06/22 9:33:00
Green	GdN	26/06/22 9:35:00	26/06/22 9:38:00
Red	St-Denis	26/06/22 9:38:00	26/06/22 9:42:00

$$t = 26/06/22\ 9:37:00$$

The OLISTIC model of railway operations



Train	OP	Timestamp (in)	Timestamp (out)
Red	GdN	26/06/22 9:25:00	26/06/22 9:28:00
Blue	GdN	26/06/22 9:30:00	26/06/22 9:33:00
Green	GdN	26/06/22 9:35:00	26/06/22 9:38:00
Red	St-Denis	26/06/22 9:38:00	26/06/22 9:42:00

$t = 26/06/22\ 9:38:00$

As any model, the OLISTIC model has limitations.

As any model, the OLISTIC model has limitations.

The idea behind the PhD

No simulation model can be singlehandedly work for all use cases in the railway industry.

Depending on the state of the simulation model, other models might be more appropriate, and the choice of the best model might change during execution.

As any model, the OLISTIC model has limitations.

The idea behind the PhD

No simulation model can be singlehandedly work for all use cases in the railway industry.

Depending on the state of the simulation model, other models might be more appropriate, and the choice of the best model might change during execution.

As any model, the OLISTIC model has limitations.

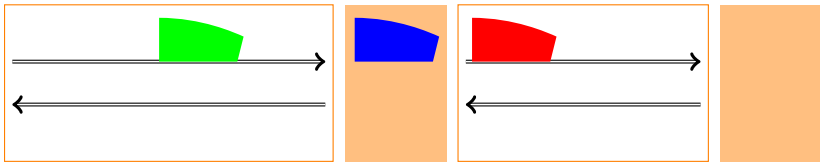
The idea behind the PhD

No simulation model can be singlehandedly work for all use cases in the railway industry.

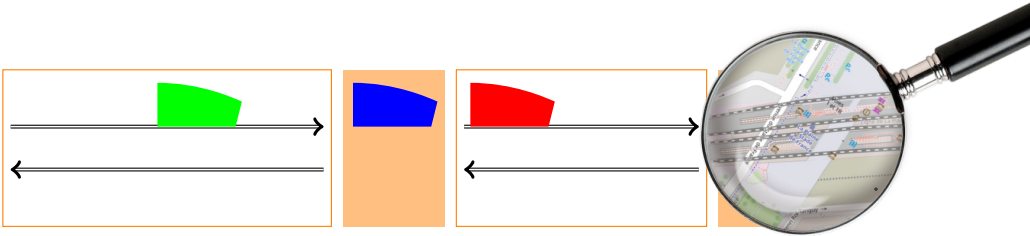
Depending on the state of the simulation model, other models might be more appropriate, and the choice of the best model might change during execution.

$\rightarrow$ We want to design a framework which can switch models during execution.

This would allow us to "zoom in" during simulation !



This would allow us to "zoom in" during simulation !



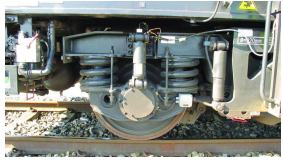
The railway, a complex ecosystem



(a) Trains at stations



(b) Signaling



(c) Dynamics

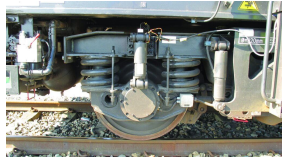
The railway, a complex ecosystem



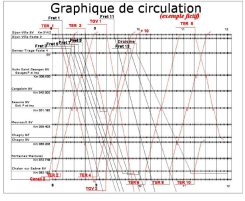
(a) Trains at stations



(b) Signaling



(c) Dynamics



(a) Scheduling

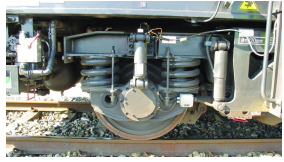
The railway, a complex ecosystem



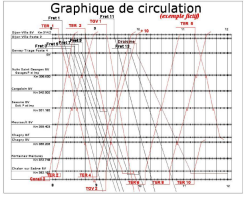
(a) Trains at stations



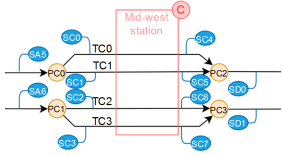
(b) Signaling



(c) Dynamics



(a) Scheduling



(b) Railway plans

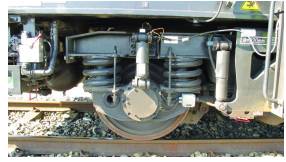
The railway, a complex ecosystem



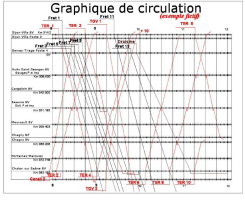
(a) Trains at stations



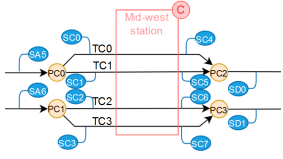
(b) Signaling



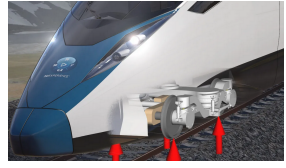
(c) Dynamics



(a) Scheduling



(b) Railway plans



(c) Dynamics simulations

The main problem

All those models are adapted to specific use cases. None of them are sufficient to describe the entirety of the system nor do they correspond to all the contexts in which simulations are used (complexity in time or complexity, data availability).

Main research questions

Going from there, how do we:

- design a global simulator of railway operations,
- which leverages the advantages of existing simulation models,
- without developing monolithic software?

Our approach

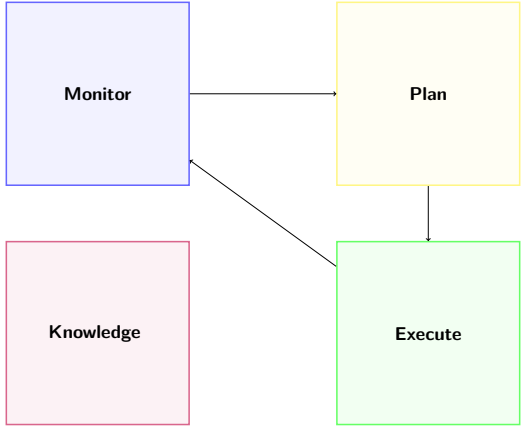
The design of a framework for the execution of existing simulations and tools, which can adapt to changing objectives by switching those tools during execution.
This orchestrator would determine which simulation models are the most adapted to the context, and would schedule the execution and stopping of new models.

Self-adaptive systems

The concept of self-adaptation comes from the field of software where the cost of stopping and recompilation is too big in case of a change.
Any change in the components used has to be done during execution, and by the software itself, which means needing an adequate software architecture.

The MAPE-K architecture [IBM06]

The MAPE-K loop is an example of a auto-adaptative architecture.



The MAPE-K architecture

Definition : Monitor, Analyze and Invariant

The **Monitor** makes a snapshot of the simulation model. It then evaluates each planned changed in the state of the simulation model. The evaluation is based on the validity of simulation **pre and post invariants**, we consider it to be the **Analyze** part of the MAPE-K loop. An **invariant** is a logical propriety that the state of the simulation model has to validate throughout its execution.

Invariant example: In OLISTIC, two trains can never enter a section one after the other if they are not spaced out enough.

The MAPE-K Architecture

Definition : Plan

The **Planner** chooses the plan which is most adapted to the invalidated invariant. The plan can be executed to resolve the invalidity.

The MAPE-K Architecture

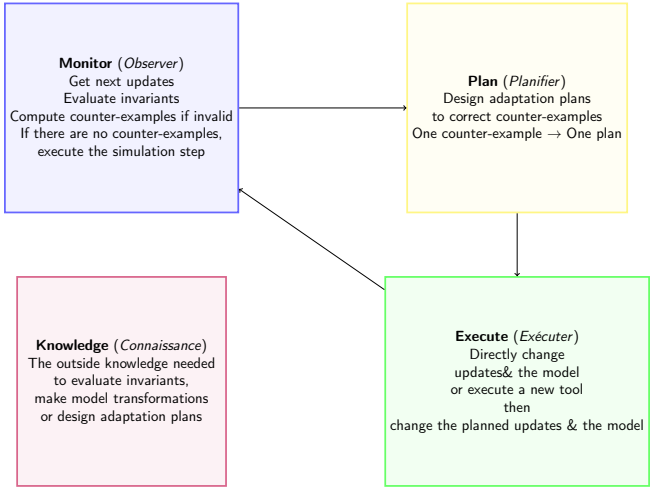
Definition : Plan

The **Planner** chooses the plan which is most adapted to the invalidated invariant. The plan can be executed to resolve the invalidity.

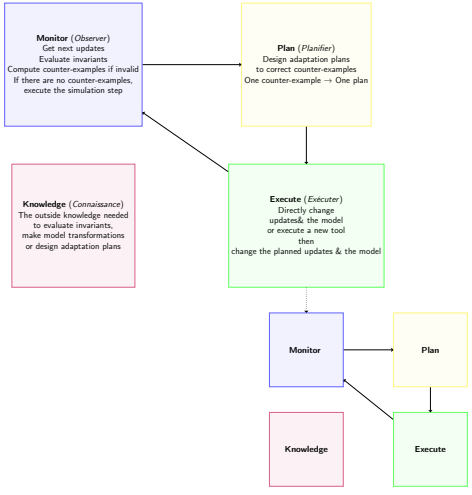
Definition : **Knowledge**

To enact the steps of the MAPE loop we might sometimes need access to **Knowledge** which is exterior to the models.

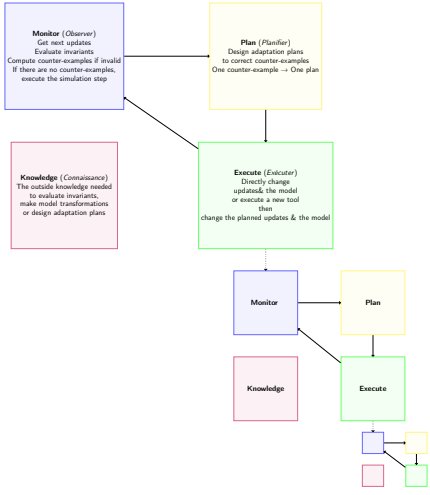
The MAPE-K Architecture



The MAPE-K Architecture



The MAPE-K Architecture



A meta-model for simulations

A Simulation has several methods:

- `init`: initializes the simulation model and the start time.
- `start`: adds the simulation to its orchestrator.
- `step`: computes one step of the simulation and updates its model accordingly.
- `get_current_timestamp`: depends on the Model of Execution
- `get_next_timestamp`: depends on the Model of Execution
- `preflight`: executes the MAPE-K loop using the pre-invariants to check the validity and fix it if necessary.
- `postflight`: executes the MAPE-K loop using the post-invariants to check the validity and fix it if necessary.
- `kill`: stops the simulation and marks it as not alive.
- `update`: updates the model with results provided by another simulation.

A meta-model of our orchestrator

The orchestrator has:

- a simulation tree, where the nodes are simulations, and where the hierarchy of the tree is the same as the parent-child hierarchy we presented earlier,
- a global simulation queue, where the simulations are ordered by the value of their next action timestamp,
- a local queue to order simulations that are equal according to the global queue order, to take the dependencies between simulations into account: a child (with a higher level of detail) is treated before its parent (lower level).

A meta-model of our orchestrator

The orchestrator has the following methods:

- the `step` method, which extracts the "smallest" simulations (according to the global simulation queue order), then extracts the smallest simulation of the local queue and treats it.
- the `reorder` method, which empties the local queue back into the global queue and reconstitutes the local queue. This is useful after an adaptation plan has changed the order of the simulations, by delaying an event for example, in the case of Discrete-Event Simulation.

Checking validity, in general and in the case of scale

The monitor invariants can be applied to multi-scale modeling. Among the invariants, some can express whether the simulation model is described at the adequate level of detail.

Resolving invalidity through adaptation plans

```python
def resolve_ce(self, simulation: Simulation, ce: Tuple[Invariant, Any]) -> None:
    plan = self.devise_plan(simulation, ce)
    (inv, ce) = ce
    # We see the result in our execution sandbox
    print(f"Suggesting resolution : {plan.option}")
    sandbox_outcome = plan.execute(plan.sliced_simu)
    match sandbox_outcome:
        case PlanOutcome.UPDATE:
            if inv.check(plan.sliced_simu) is None:
                print("Resolution makes invariant true in sandbox.")
                print("Enacting resolution :")
                plan.execute(simulation)
                print("Reordering simulations in orchestrator.")
                simulation.orchestrator.reorder()
            else:
                raise SystemError("Update plan failed")
        case PlanOutcome.KILL:
            print(f"Killing simulation {simulation}")
            plan.execute(simulation)
            simulation.orchestrator.remove_sim(simulation)
```

`SpawnChild` plan

`SpawnChild` plan = Plan executed when the violated invariant corresponds to an invalidity in the level of detail chosen

`SpawnChild` plan

`SpawnChild` plan = Plan executed when the violated invariant corresponds to an invalidity in the level of detail chosen

→ Adds child simulation to the orchestrator

→ Reorders the orchestrator to ensure child simulation step is executed before parent

→ Needs pre-defined model transformations based on mappings.

`Delegation` mechanism

Parent and child coexist in the orchestrator, we need to ensure their consistency
The `SpawnChild` plan executions adds itself to a `Delegation` pre-invariant to the simulation.
The associated `Delegation` plan depends on the simulation paradigm.

Puppeteer mechanism

✓Invariant which describes the concordance of a given model to its assigned scale : defined by the domain experts

Puppeteer mechanism

✓Invariant which describes the concordance of a given model to its assigned scale : defined by the domain experts
What about having an invariant for stopping a child simulation?

Puppeteer mechanism

✓Invariant which describes the concordance of a given model to its assigned scale : defined by the domain experts
What about having an invariant for stopping a child simulation? → We can add it automatically using the puppeteer pattern [MMP18].

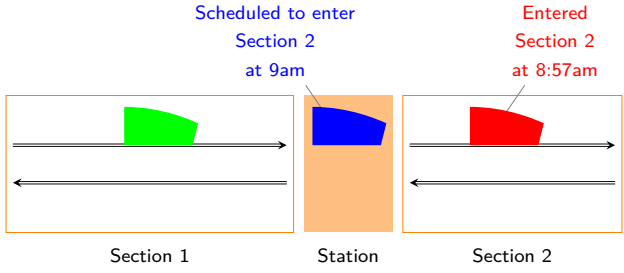
Puppeteer mechanism

Each new child is given two post invariants:

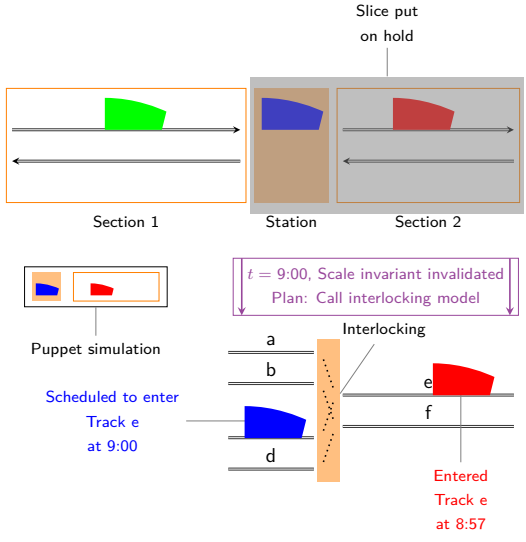
- one for checking if the puppet model has changed,
- one for checking if the puppet model state makes its scale invariant valid again.

If the puppet model state has evolved, then a plan execution updates the parent state with the new state of the puppet.
If this new puppet state makes the scale invariant valid, a plan execution discards the child simulation.

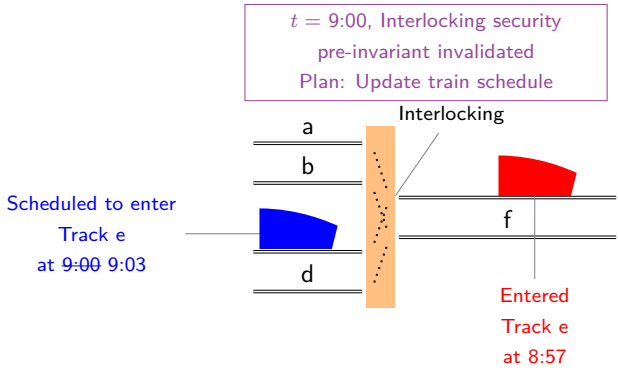
An example


Scheduled to enter
Section 2
at 9am
Entered
Section 2
at 8:57am
Section 1
Station
Section 2

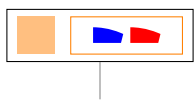
An example


Slice put
on hold
Section 1
Station
Section 2
Puppet simulation
$t = 9{:}00$, Scale invariant invalidated
Plan: Call interlocking model
Interlocking
a
b
e
f
d
Scheduled to enter
Track e
at 9:00
Entered
Track e
at 8:57

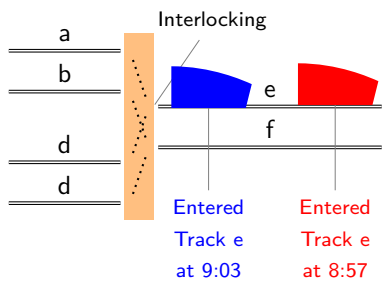
An example


t = 9:00, Interlocking security
pre-invariant invalidated
Plan: Update train schedule
Interlocking
a
b
d
f
Scheduled to enter
Track e
at ~~9:00~~ 9:03
Entered
Track e
at 8:57

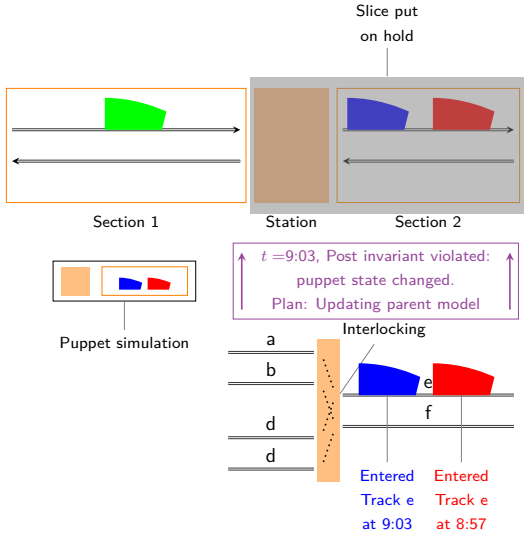
An example



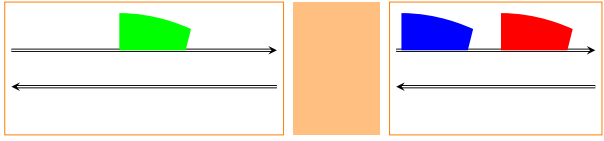
$t = 9{:}03$, No invalidated pre-invariant
Executing step in child simulation



An example


Slice put
on hold
Section 1
Station
Section 2
Puppet simulation
t =9:03, Post invariant violated:
puppet state changed.
Plan: Updating parent model
Interlocking
a
b
d
d
e
f
Entered
Track e
at 9:03
Entered
Track e
at 8:57

An example



t =9:03, Post invariant invalidated:
the puppet state is described at a valid scale.
Plan: Remove child simulation

Developping a discrete event simulator

Our simulator works with resources, and actors which acquire and release those resources. We use it in two models (for now) :

- In a first "Macro" model, resources are Operational Points and Sections, like in OLISTIC.
- In the second "Meso" model, resources are tracks. A twist : interlocking is abstracted in PSA (Pre Station Area) objects. (same as OptiSillons [Jou23])
  In both versions, the actors are trains.

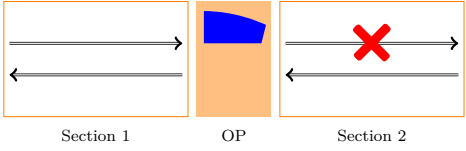
"Macro" simulation: input = railway schedule

Train	OP	Timestamp (in)	Timestamp (out)
Red	GdN	26/06/22 9:25:00	26/06/22 9:28:00
Blue	GdN	26/06/22 9:30:00	26/06/22 9:33:00
Green	GdN	26/06/22 9:35:00	26/06/22 9:38:00
Red	St-Denis	26/06/22 9:41:00	26/06/22 9:44:00

"Macro" simulation: output = "corrected" railway schedule

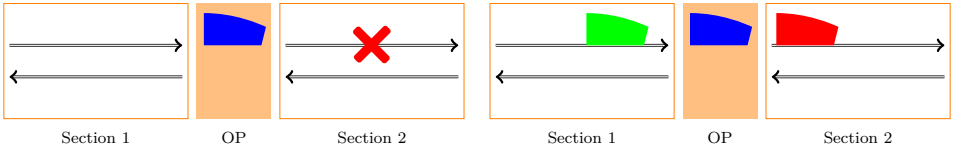
Train	OP	Timestamp (in)	Timestamp (out)
Red	GdN	26/06/22 9:25:00	26/06/22 9:28:00
Blue	GdN	26/06/22 9:30:00	26/06/22 9:33:00
Green	GdN	26/06/22 9:35:00	26/06/22 9:41:00
Red	St-Denis	26/06/22 9:41:00	26/06/22 9:44:00

Pre-invariants for the "macro" simulation



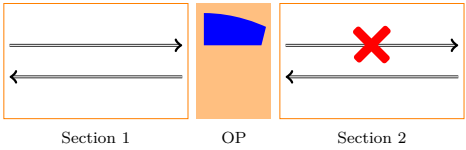
(a) Functional

Pre-invariants for the "macro" simulation

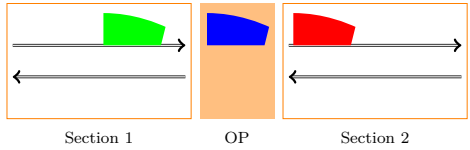


(a) Functional (b) Dense traffic

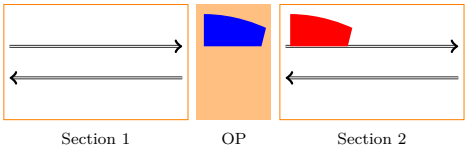
Pre-invariants for the "macro" simulation



(a) Functional

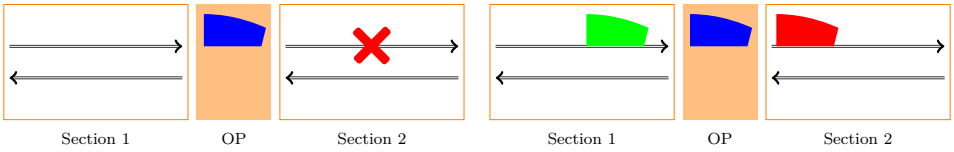


(b) Dense traffic



(c) Security

Pre-invariants for the "macro" simulation



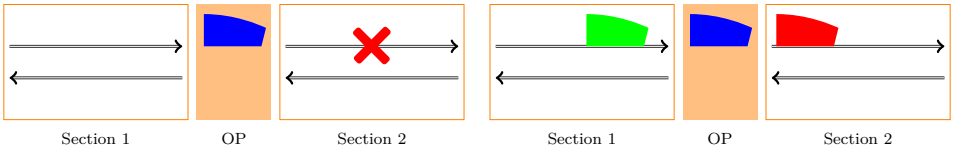
(a) Functional

(b) Dense traffic

(c) Security

(d) Track Change

Pre-invariants for the "macro" simulation



(a) Functional

(b) Dense traffic

(c) Security

(d) Track Change

Adaptation plans

Invalidated invariant	Strategy
Functional	Changing itineraries (delays or new route)
Dense traffic	Executing the “meso“ simulation for the concerned area and including its results
Security	Updating the train itinerary (delay)

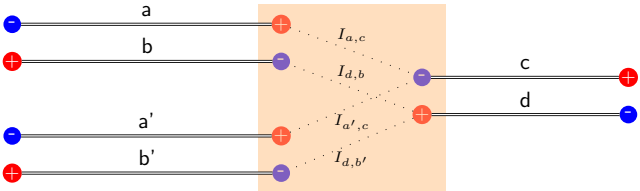
Adaptation plans

Invalidated invariant	Strategy
Functional	Changing itineraries (delays or new route)
Dense traffic	Executing the “meso“ simulation for the concerned area and including its results
Security	Updating the train itinerary (delay)

NB

Those adaptation plans hide modeling choices and work on data, which might lead to calling external tools!

"Mesoscopic" simulation: inputs = tracks, PSAs and itineraroes



	$I_{a,c}$	$I_{d,b}$	$I_{a',c}$	$I_{d,b'}$
$I_{a,c}$	3	0	5	0
$I_{d,b}$	0	3	5	5
$I_{a',c}$	5	5	3	0
$I_{d,b'}$	0	5	0	3

Compatibility matrix

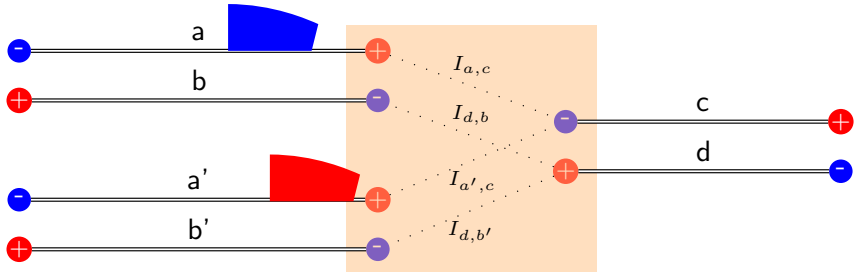
"Meso" simulation pre-invariants

- PSA Invariant
- Dense traffic pre-invariant (like the Macro one)

"Meso" Itineraries

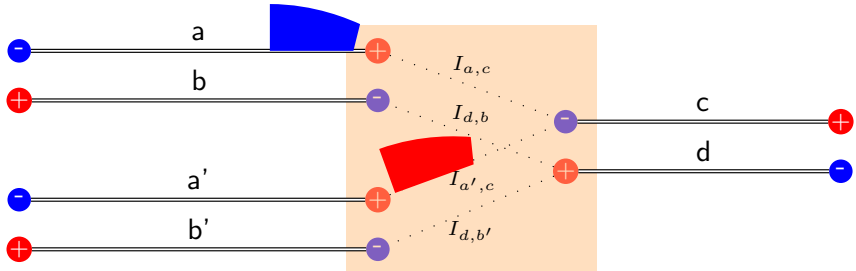
Train	Track	Direction	Timestamp (in)	Timestamp (out)
Red	a	Conventionnelle	26/06/22 9:20:00	26/06/22 9:28:00
Blue	a'	Conventionnelle	26/06/22 9:21:00	26/06/22 9:29:00
Red	c	Conventionnelle	26/06/22 9:32:00	26/06/22 9:38:00
Blue	c	Conventionnelle	26/06/22 9:33:00	26/06/22 9:39:00

"Meso" simulations



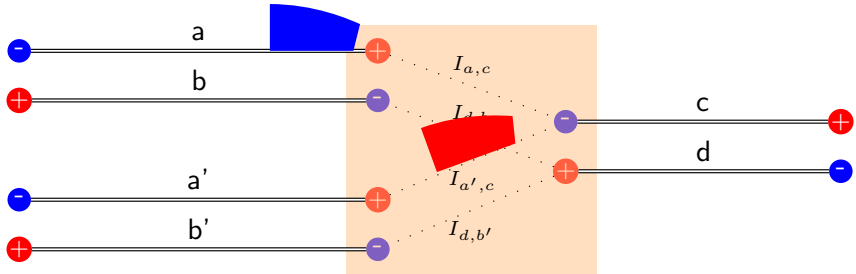
$t = 26/06/22\ 9:28:00$

"Meso" simulations



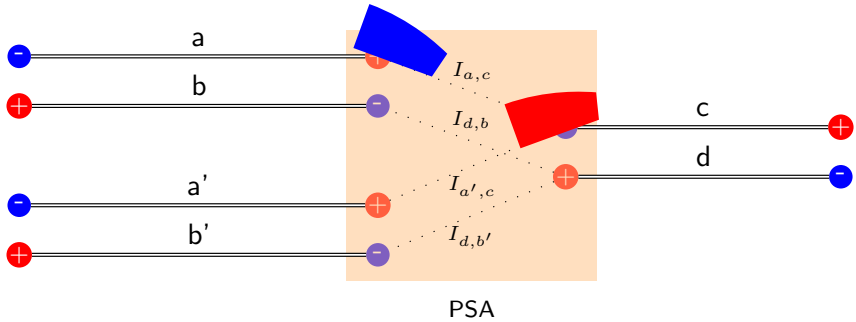
$t = 26/06/22\ 9:28:00$

"Meso" simulations



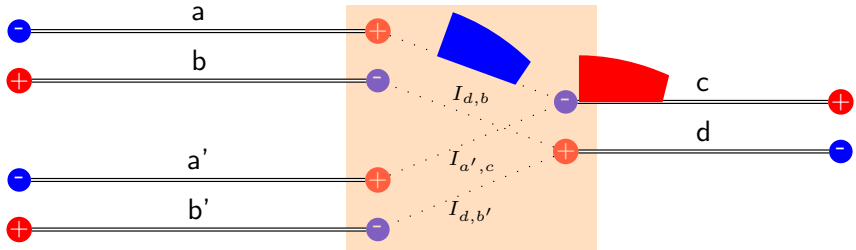
$t = 26/06/22\ 9:29:00$: the blue train cannot use the itinerary $I_{a,c}$

"Meso" simulations



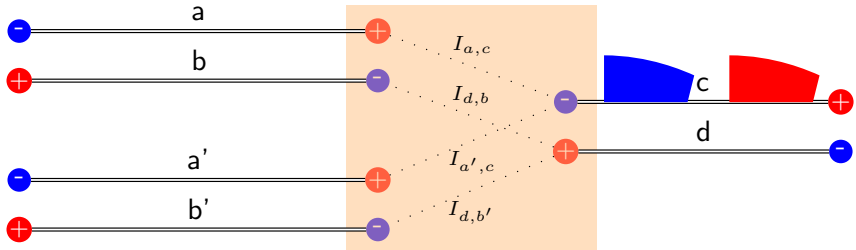
$t = 26/06/22\ 9:31:00$: the blue train is delayed by 2 minutes

"Meso" simulations



$t = 26/06/22\ 9:32:00$

"Meso" simulations



$t = 26/06/22\ 9:35:00$

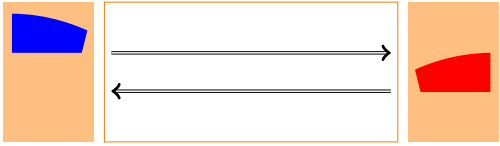
"Meso": output = "corrected" schedule

Train	Track	Direction	Timestamp (in)	Timestamp (out)
Red	a	Conventionnelle	26/06/22 9:20:00	26/06/22 9:28:00
Blue	a'	Conventionnelle	26/06/22 9:21:00	26/06/22 9:29:00
Red	c	Conventionnelle	26/06/22 9:32:00	26/06/22 9:38:00
Blue	c	Conventionnelle	26/06/22 9:33:00	26/06/22 9:39:00

"Meso": output = "corrected" schedule

Train	Track	Direction	Timestamp (in)	Timestamp (out)
Red	a	Conventionnelle	26/06/22 9:20:00	26/06/22 9:28:00
Blue	a'	Conventionnelle	26/06/22 9:21:00	26/06/22 9:31:00
Red	c	Conventionnelle	26/06/22 9:32:00	26/06/22 9:38:00
Blue	c	Conventionnelle	26/06/22 9:35:00	26/06/22 9:41:00

A real use case for changning scales: *Banalisation*



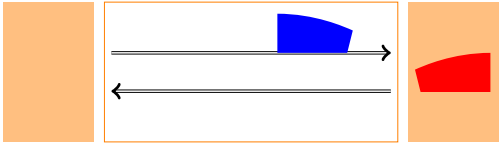
GdN | Section 2 | St-Denis

Capacité: 2 trains in each direction

$t = 26/06/22\ 9:30:00$

Train	OP	Timestamp (in)	Timestamp (out)
Blue	GdN	26/06/22 9:30:00	26/06/22 9:33:00
Red	St-Denis	26/06/22 9:36:00	26/06/22 9:38:00
Blue	St-Denis	26/06/22 9:40:00	26/06/22 9:42:00

A real use case for changning scales: *Banalisation*



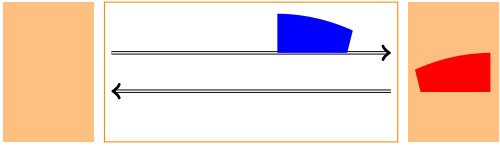
GdN | Section 2 | St-Denis

Capacité: 2 trains in each direction

$t = 26/06/22\ 9:33:00$

Train	OP	Timestamp (in)	Timestamp (out)
Blue	GdN	26/06/22 9:30:00	26/06/22 9:33:00
Red	St-Denis	26/06/22 9:28:00	26/06/22 9:38:00
Blue	St-Denis	26/06/22 9:40:00	26/06/22 9:42:00

A real use case for changning scales: *Banalisation*



GdN | Section 2 | St-Denis

Capacité: 2 trains in each direction

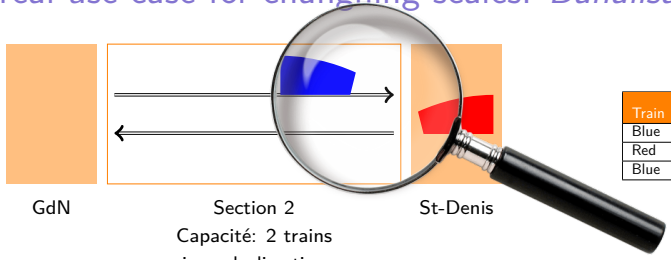
$t = 26/06/22\ 9:38:00$

Train	OP	Timestamp (in)	Timestamp (out)
Blue	GdN	26/06/22 9:30:00	26/06/22 9:33:00
Red	St-Denis	26/06/22 9:28:00	26/06/22 9:38:00
Blue	St-Denis	26/06/22 9:40:00	26/06/22 9:42:00

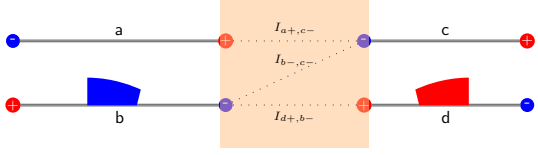
Outside knowledge

The blue train is going to change tracks and the red train is not far enough.

A real use case for changning scales: *Banalisation*



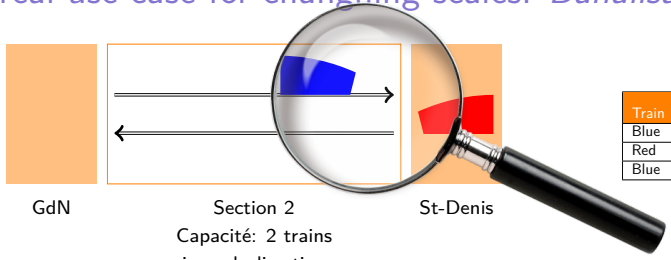
Train	OP	Timestamp (in)	Timestamp (out)
Blue	GdN	26/06/22 9:30:00	26/06/22 9:33:00
Red	St-Denis	26/06/22 9:28:00	26/06/22 9:38:00
Blue	St-Denis	26/06/22 9:40:00	26/06/22 9:42:00



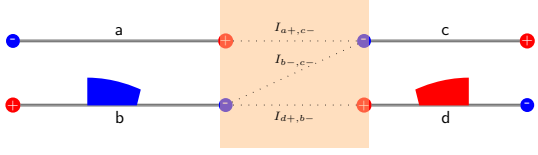
$t = 26/06/22\ 9:38:00$

Train	Track	Direction	Timestamp (in)	Timestamp (out)
Red	d	Conventionnelle	26/06/22 9:28:00	26/06/22 9:38:00
Blue	b	Non-Conventionnelle	26/06/22 9:33:00	26/06/22 9:39:00
Red	b	Conventionnelle	26/06/22 9:38:00	26/06/22 9:44:00
Blue	c	Conventionnelle	26/06/22 9:40:00	26/06/22 9:44:00

A real use case for changning scales: *Banalisation*



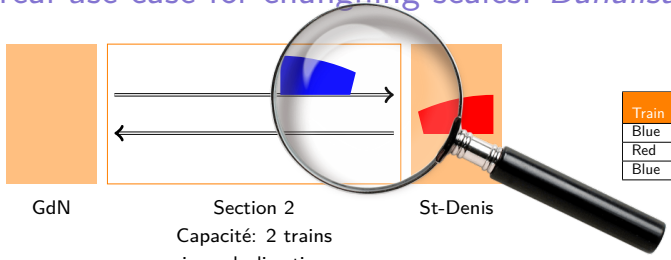
Train	OP	Timestamp (in)	Timestamp (out)
Blue	GdN	26/06/22 9:30:00	26/06/22 9:33:00
Red	St-Denis	26/06/22 9:28:00	26/06/22 9:38:00
Blue	St-Denis	26/06/22 9:40:00	26/06/22 9:42:00



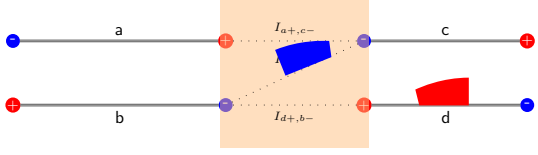
$t = 26/06/22\ 9:39:00$

Train	Track	Direction	Timestamp (in)	Timestamp (out)
Red	d	Conventionnelle	26/06/22 9:28:00	26/06/22 9:38:00
Blue	b	Non-Conventionnelle	26/06/22 9:33:00	26/06/22 9:39:00
Red	b	Conventionnelle	26/06/22 9:41:00	26/06/22 9:47:00
Blue	c	Conventionnelle	26/06/22 9:40:00	26/06/22 9:44:00

A real use case for changning scales: *Banalisation*



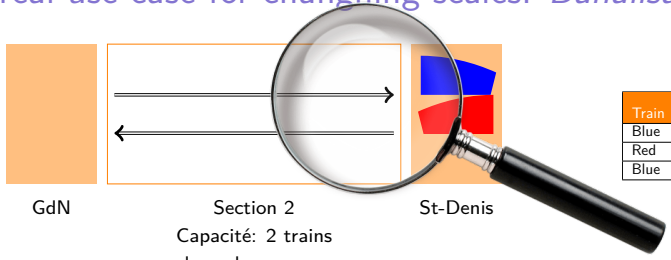
Train	OP	Timestamp (in)	Timestamp (out)
Blue	GdN	26/06/22 9:30:00	26/06/22 9:33:00
Red	St-Denis	26/06/22 9:28:00	26/06/22 9:41:00
Blue	St-Denis	26/06/22 9:40:00	26/06/22 9:42:00



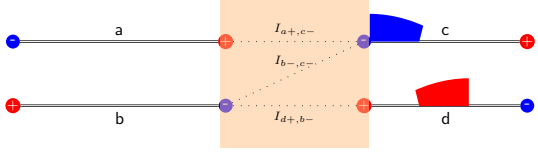
$t = 26/06/22\ 9:40:00$

Train	Track	Direction	Timestamp (in)	Timestamp (out)
Red	d	Conventionnelle	26/06/22 9:28:00	26/06/22 9:38:00
Blue	b	Non-Conventionnelle	26/06/22 9:33:00	26/06/22 9:39:00
Red	b	Conventionnelle	26/06/22 9:41:00	26/06/22 9:47:00
Blue	c	Conventionnelle	26/06/22 9:40:00	26/06/22 9:44:00

A real use case for changning scales: *Banalisation*



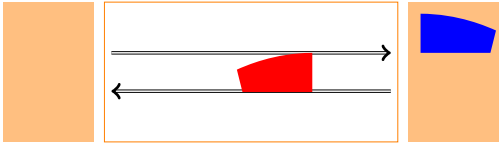
Train	OP	Timestamp (in)	Timestamp (out)
Blue	GdN	26/06/22 9:30:00	26/06/22 9:33:00
Red	St-Denis	26/06/22 9:28:00	26/06/22 9:41:00
Blue	St-Denis	26/06/22 9:40:00	26/06/22 9:42:00



Train	Track	Direction	Timestamp (in)	Timestamp (out)
Red	d	Conventionnelle	26/06/22 9:28:00	26/06/22 9:38:00
Blue	b	Non-Conventionnelle	26/06/22 9:33:00	26/06/22 9:39:00
Red	b	Conventionnelle	26/06/22 9:41:00	26/06/22 9:47:00
Blue	c	Conventionnelle	26/06/22 9:40:00	26/06/22 9:44:00

$t = 26/06/22\ 9:40:00$

A real use case for changning scales: *Banalisation*



GdN Section 2 St-Denis

Capacité: 2 trains
in each direction

$t = 26/06/22\ 9:41:00$

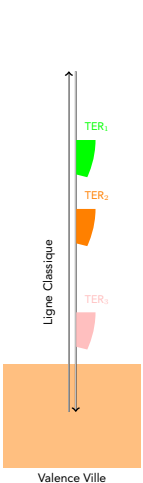
Train	OP	Timestamp (in)	Timestamp (out)
Blue	GdN	26/06/22 9:30:00	26/06/22 9:33:00
Red	St-Denis	26/06/22 9:28:00	26/06/22 9:41:00
Blue	St-Denis	26/06/22 9:40:00	26/06/22 9:42:00

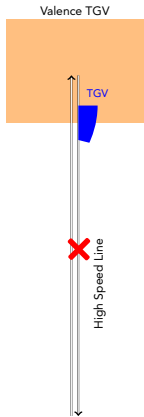
Model mapping and transformation : "Macro" to "Meso" to "Macro"

- Each track is associated to a set of sections.
- We know (through data external to the "Macro" model) of input and output tracks for trains.
- We know (through data external to the "Meso" model) the track lengths.

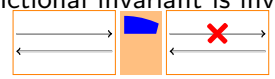
Those mappings help us compute "Meso" train itineraries from their "Macro" counterparts.

Case study

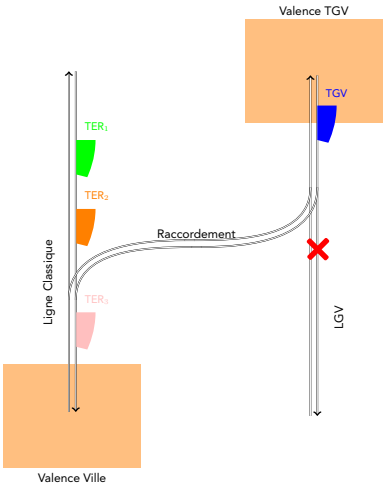




The functional invariant is invalidated

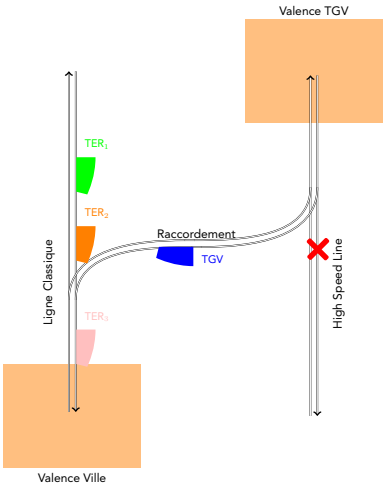


Case study



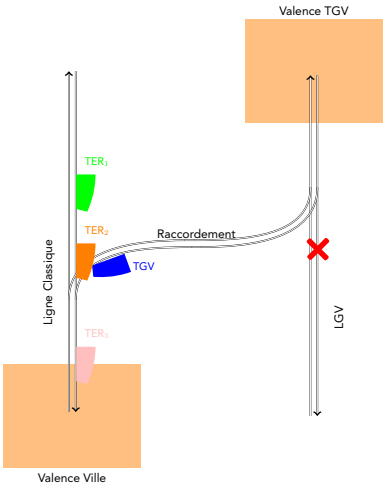
Knowledge (delays and itineraries)

Case study



Plan execution

Case study



The dense traffic invariant is invalidated

Perspectives

Academic perspectives:

- Creating a DSL to specify our simulation experiments,
- Make transformations from the DSL to executable code + other frameworks / tools, eg formal verification tools

Industrial perspectives:

- Make testing interfaces between models easier,
- Encourage people to formalize modeling choices, including in-house databases and models,
- Long term: possessing a generic and modular tool that operators can personnalize.

Références citées I

[DB14] J. Damay and A. Boillot. "An efficient macroscopic railway system simulator.". In: *Proceedings of the Conference of the International Federation of Operational Research Societies (IFORS)*. 2014.

[IBM06] IBM. "An architectural blueprint for autonomic computing". In: *IBM White paper* 31.2006 (2006), pp. 1–6.

[Jou23] G. Joubert. "Ordonnancement de sillons ferroviaires périodiques avec affectation de voies à l'échelle mésoscopique.". PhD thesis. Université de Technologie de Compiègne, 2023.

[MMP18] Philippe Mathieu, Gildas Morvan, and Sebastien Picault. "Multi-level agent-based simulations: Four design patterns". In: *Simulation Modelling Practice and Theory* 83 (2018), pp. 51–64.