Automatic Annotated Code Generation from B Formal Specifications

Dorian Petit, Georges Mariano, Vincent Poirriez and Jean-Louis Boulanger

dorian.petit@univ-valenciennes.fr .

Outlines

- B method
- Motivations
- Technical solutions
- Conclusions

B method

- Developped by J.R. Abrial
- Used (at origine) in the railway industry

Constituants:

- Set Theory
- Substitution Calculus
- Development by refinement steps
- Composition links (INCLUDES, IMPORTS, SEES, USES)

B method: code generation

- No specification of the code generators
- Black box tools
- Impossible to simply extends them to other languages

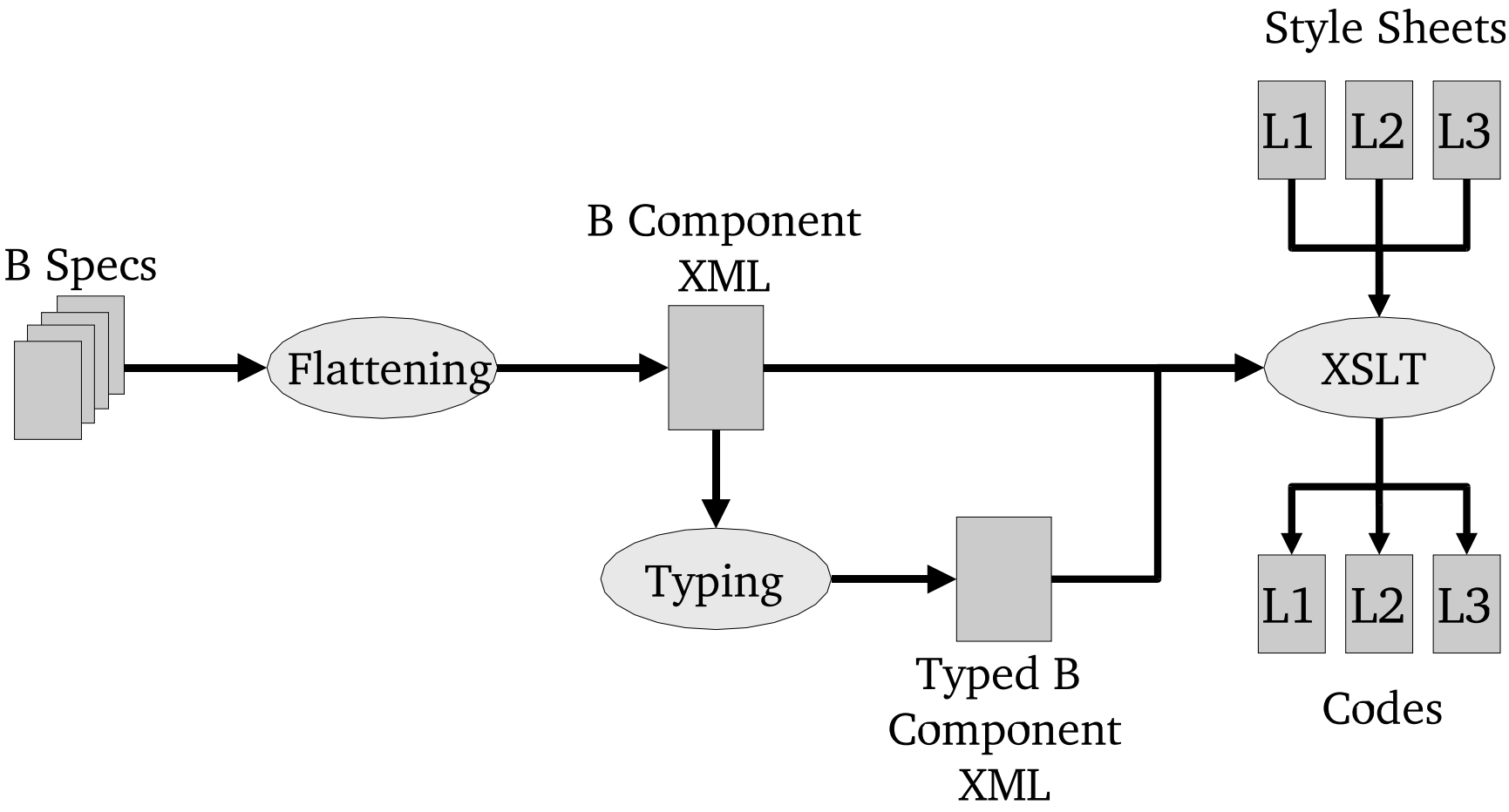
Motivations

- To understand the B code generation process
- To increase this process
- What about mixing the B method and design by contract ?

Flattening

- Algorithm defined by S. Behnia
- Principle: merging several formal texts into only one
- A set of components + some checks on this set can be transformed in a flatten component

Code generation process



Target languages

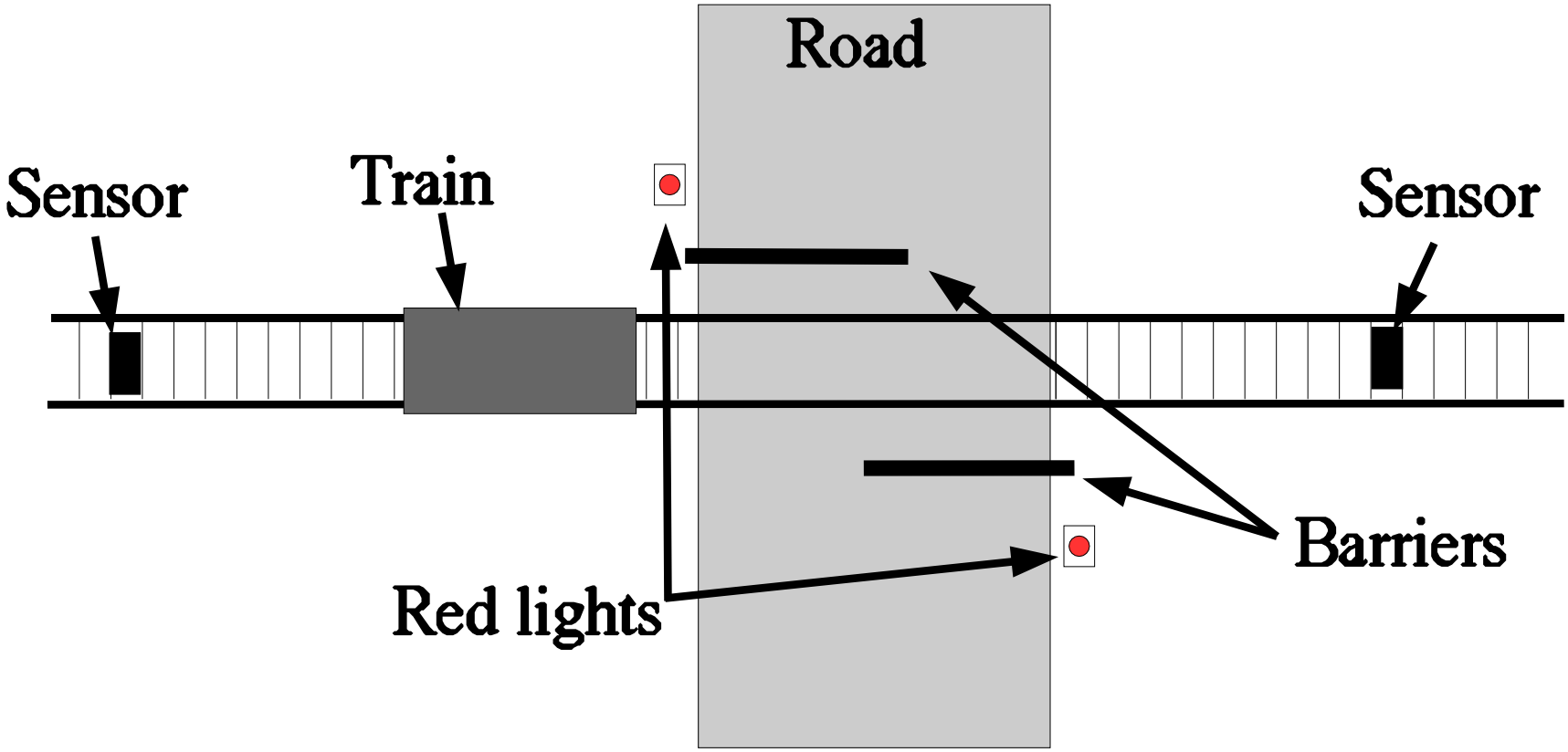
OCaml

- Object Oriented constructions in the language
- Assertions
- Implicit type checking

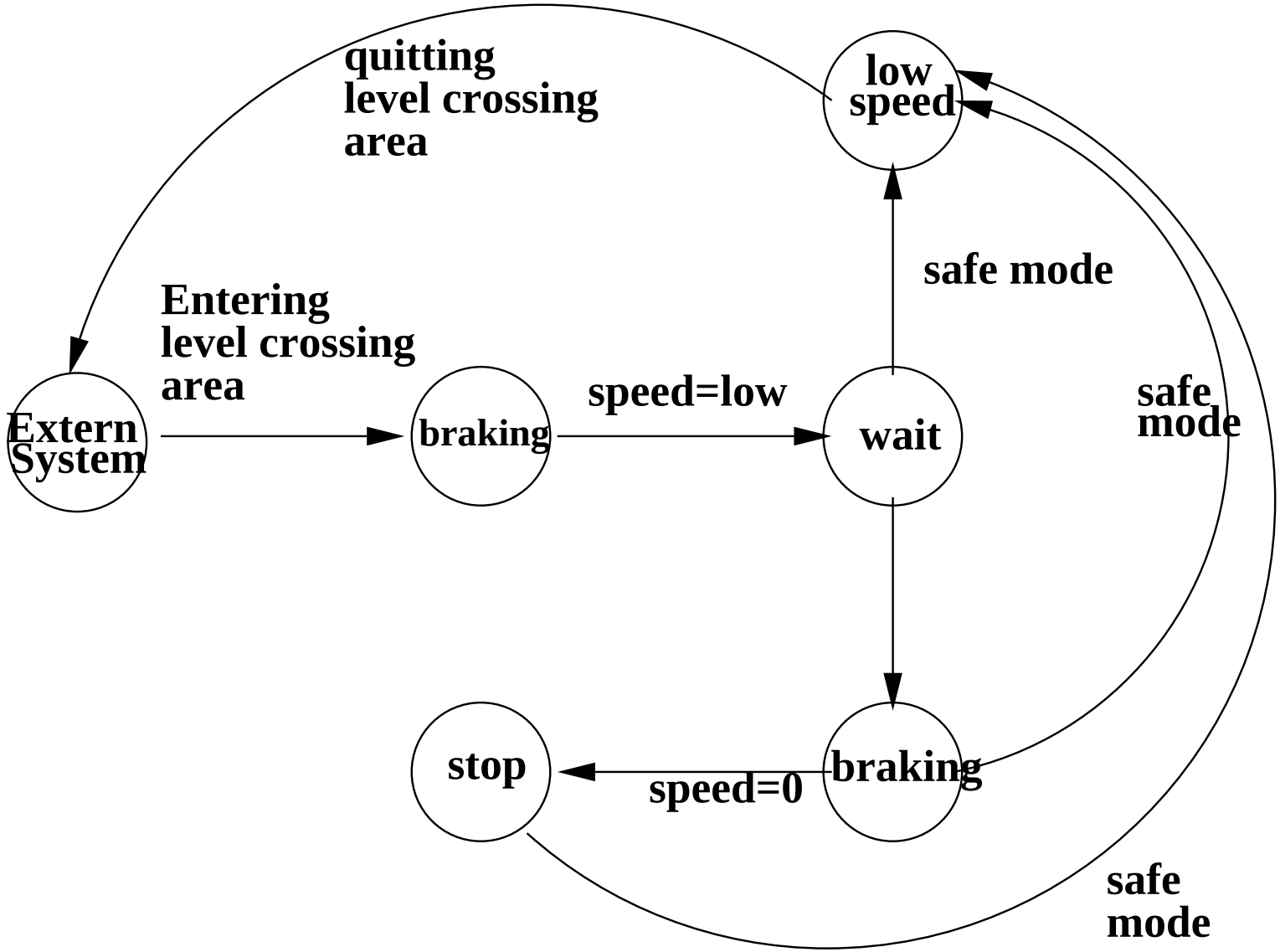
Ada

- More relevant for railway industry
- Possibilities to connect to other tools such as Spark Ada suite

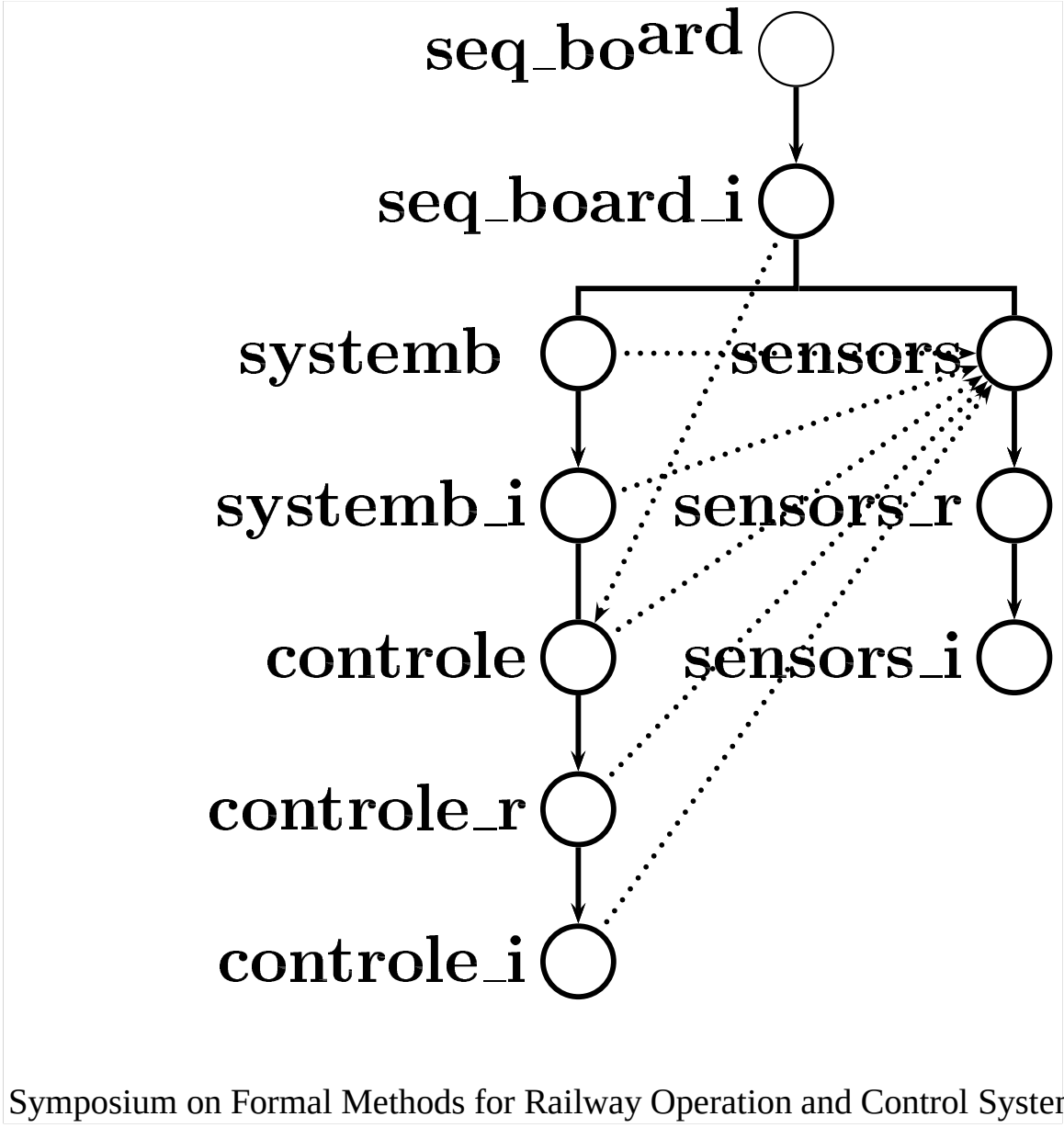
Example



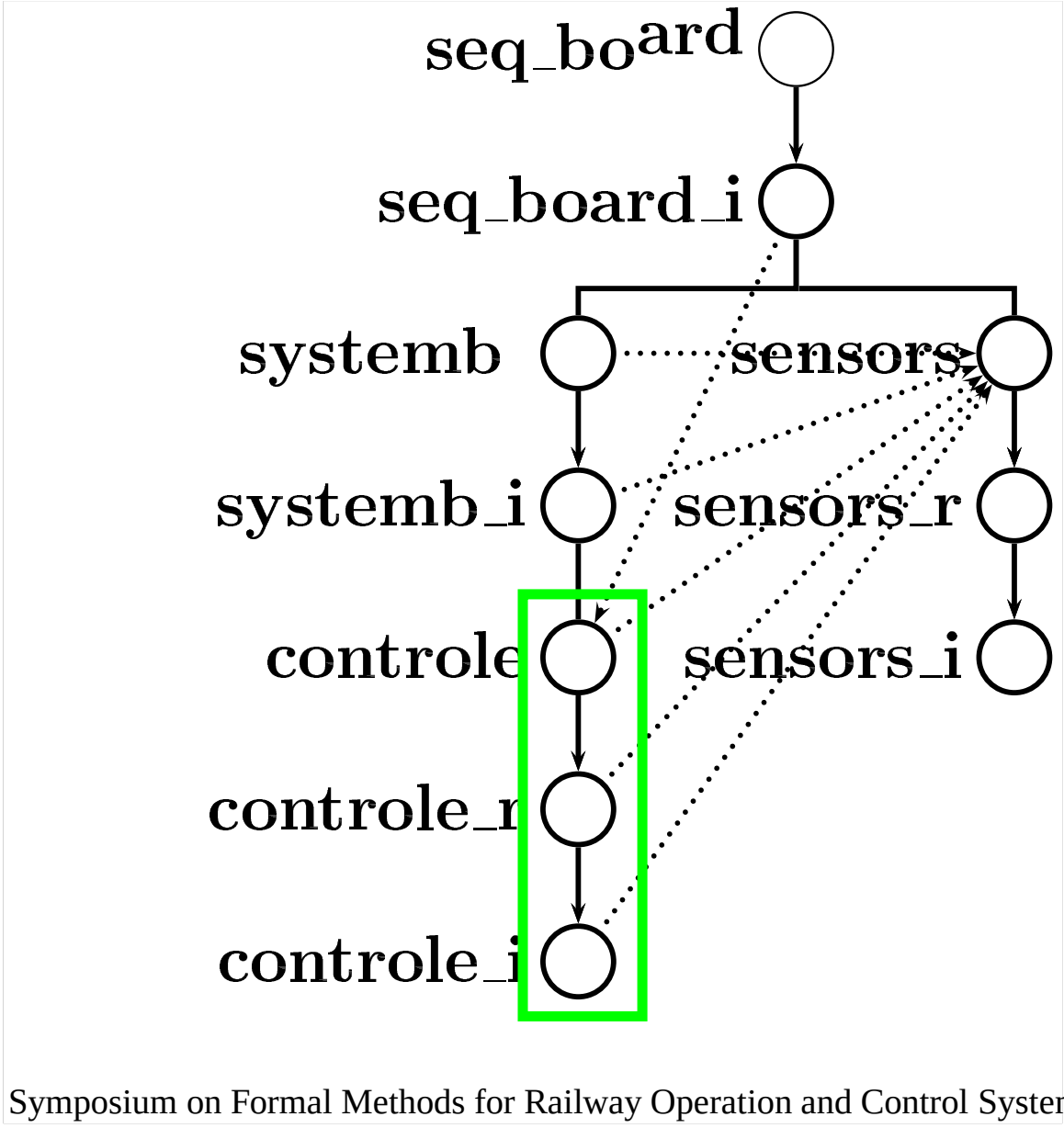
Example: automata



Example: B components



Example: B components



abstract specification

MACHINE controle

...

INVARIANT

md $\in$ MODE $\wedge$ ack $\in$ ACK $\wedge$ light $\in$ LIGHT $\wedge$ br $\in$ BARRIER

$\wedge$ train $\subseteq$ TRAINS

$\wedge$ ((br = top) $\Rightarrow$ light = off) $\wedge$ ((br $\neq$ top) $\Rightarrow$ light = on)

...

OPERATIONS

trans_BBa =

PRE (light = on) $\wedge$ md = safe $\wedge$ gr = rest

THEN br := movement_down

END ;

...

END

concrete specification

IMPLEMENTATION controle_i

REFINES controle_r

...

INVARIANT

md $\in$ MODE $\wedge$ zn $\in$ ZONE $\wedge$ br $\in$ BARRIER

$\wedge$ train_ $\in$ 1 .. max_train $\rightarrow$ TRAINS

$\wedge$ train = train_ [1..max_train] - num_train

$\wedge$ ((br = top) $\Rightarrow$ light = off) $\wedge$ ((br $\neq$ top) $\Rightarrow$ light = on)

OPERATIONS

trans_BBa =

BEGIN

br := mouvement_bas

END ;

...

END

code generated

method trans_BBa=

require (abstract_invariant_e
∧ refinement_invariant_e
∧ concrete_invariant);
require(light=On ∧ md=Safe);

br := Movement_down;

ensure (abstract_invariant_e
∧ refinement_invariant_e

∧ concrete_invariant)

code generated

method trans_BBa=

require (abstract_invariant_e
∧ refinement_invariant_e
∧ concrete_invariant);
require(light=On ∧ md=Safe);

br := Movement_down;

ensure (abstract_invariant_e
∧ refinement_invariant_e

∧ concrete_invariant)

code generated

method trans_BBa=

require (abstract_invariant_e
∧ refinement_invariant_e
∧ concrete_invariant);
require(light=On ∧ md=Safe);

br := Movement_down;

ensure (abstract_invariant_e
∧ refinement_invariant_e

∧ concrete_invariant)

code generated

method trans_BBa=

require (abstract_invariant_e
∧ refinement_invariant_e
∧ concrete_invariant);
require(light=On ∧ md=Safe);

br := Movement_down;

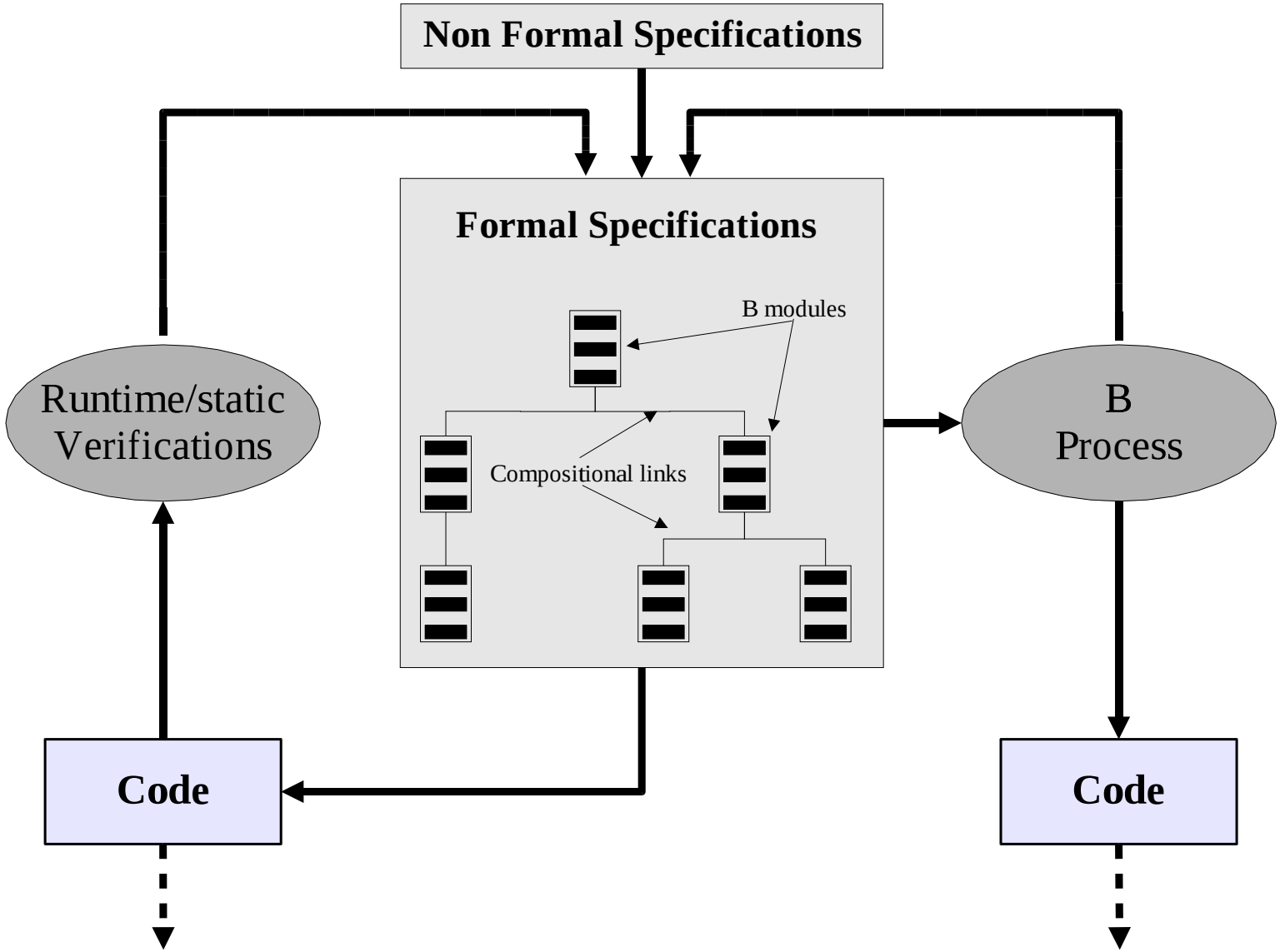
ensure (abstract_invariant_e
∧ refinement_invariant_e
∧ concrete_invariant)

code generated

```
class controle =
  object
    open Train, Communication, Capteurs
    type mode = Nonspecified | Safe | Notsafe ;
    type barrier = Top | Down
                 | Movement_top | Movement_down ;
    type feu = On | Off ;
    method  getMode = ...
    method  trans_DPr zz = ...
    method  trans_BBa = ...
    ...

  end;;
```

new B development process



Conclusions

Conclusions:

- Implementation of a code generator
- Actual target language : OCaml and Ada
- Easy to adapt to new language

Future Work:

- Working on the B modularity
- Connecting to verification tools
- Developping a case study using these tools