Formal specification, refinement and proof with B and Event-B

Frédéric Gervais

frederic.gervais@u-pec.fr

November 6, 2025

B/Event-B Method

What is it?

A formal method

The whole engineering lifecycle

- abstract specification
- refinement steps
- implementation to be translated into code

It is formal

- formal semantics
- proof obligations for invariant preservation and refinement
- supporting tools: Atelier B, ProB, RODIN

Jean-Raymond Abrial (1938-2025)

Assigning programs to meanings...

- creator of one of the first network database management systems (SOCRATE 1968-1974 Grenoble)
- author of a paper on data and object oriented models: *Data semantics* (1974)
- contributor to the ADA language (1978-1979)
- father of the Z language (1974-1980 EDF, then PRG Oxford Tony Hoare)
- creator of the B method and Event-B (1990s)
- most recent projects: RODIN, DEPLOY, ADVANCE (2000s, 2010s)

Main characteristics

- static part language: set theory and first order predicates
- dynamic part language: generalized substitutions (before-after predicates)
- refinement: from abstract to concrete specifications
- proof system based on Hoare logic
- B is supported by industry

Railway systems

- SNCF KVB - speed control by beacons (Alstom): 60K lines B, 10K proofs, 22K lines ADA
- RATP Meteor (line 14 metro) - automated train control (Siemens): 107K lines B, 29K proofs, 87K lines ADA
- Roissy VAL: driver system control (Siemens): 183K lines B, 43K proofs, 158K lines ADA
- other projects: line 1 Paris metro, New York metro, etc.

Other application domains

- Cars
- Space
- Finance
- Nuclear
- Management

MACHINE *Member*
SETS *MEMBER*
VARIABLES *members*
INVARIANT $members \subseteq MEMBER$
INITIALISATION $members := \emptyset$
OPERATIONS

    `AddMember`$(m) \triangleq$
        pre $m \in MEMBER - members$
        then $members := members \cup \{m\}$
        end

    ...

END

MACHINE *Member*
SETS *MEMBER*
VARIABLES *members*
INVARIANT $members \subseteq MEMBER$
INITIALISATION $members := \emptyset$
OPERATIONS

`AddMember()` $\triangleq$
pre $members \neq MEMBER$
then
any m `/* we do not know how` m `is chosen */`
where $m \in MEMBER - members$
then $members := members \cup \{m\}$
end
end

...

About this example

- document online with more details: `https://fredericgervais.com/research/gt-verif-lacl-6-november-2025/`
- non deterministic choice (**any**) is useful for abstract specification
- refinement will specify how to build or generate m
- proof activity helps us to identify some preconditions to preserve the invariant properties

More about the static language

- set theory and first order predicates
- notion of relation: $r \in S \leftrightarrow T = \mathbb{P}(S \times T)$
- maplet: $s \mapsto t$, also denoted by (s, t)
- very useful to describe links between elements

Somes relational operators

$r^{-1} = \{y \mapsto x \mid x \mapsto y \in r\}$	$r \in S \leftrightarrow T$
$p; q = \{x \mapsto z \mid \exists\, z \cdot x \mapsto z \in p \wedge z \mapsto y \in q\}$	$p \in S \leftrightarrow T \wedge q \in T \leftrightarrow U$
$\mathrm{dom}(r) = \{x \mid x \in S \wedge \exists\, y \in T \cdot x \mapsto y \in r\}$	$r \in S \leftrightarrow T$
$\mathrm{ran}(r) = \{y \mid y \in T \wedge \exists\, x \in S \cdot x \mapsto y \in r\}$	$r \in S \leftrightarrow T$
$r[U] = \{y \mid y \in T \wedge \exists\, x \in U \cdot x \mapsto y \in r\}$	$r \in S \leftrightarrow T \wedge U \subseteq S$
$U \lhd r = \{x \mapsto y \mid x \mapsto y \in r \wedge x \in U\}$	$r \in S \leftrightarrow T \wedge U \subseteq S$
$r \rhd V = \{x \mapsto y \mid x \mapsto y \in r \wedge y \in V\}$	$r \in S \leftrightarrow T \wedge V \subseteq T$
$U \ntriangleleft r = \{x \mapsto y \mid x \mapsto y \in r \wedge x \notin U\}$	$r \in S \leftrightarrow T \wedge U \subseteq S$
$r \ntriangleright V = \{x \mapsto y \mid x \mapsto y \in r \wedge y \notin V\}$	$r \in S \leftrightarrow T \wedge V \subseteq T$
$f \mathbin{\lhd\!\!\!+} g = (\mathrm{dom}(g) \ntriangleleft f) \cup g$	$f \in S \leftrightarrow T \wedge g \in S \leftrightarrow T$

Functions as relations

- functions as relations with constraints
- partial function: $f \in S \mathrel{+\!\!\!\rightarrow} T$
- constraint: each element has at most one image in the relation

$$S \mathrel{+\!\!\!\rightarrow} T = \{f \mid f \in S \leftrightarrow T \land \forall\, x \in S, \forall\,(y,z) \in T \times T \cdot \\ (x \mapsto y \in f \land x \mapsto z \in f) \Rightarrow y = z\}$$

Functions

Let $f \in S \nrightarrow T$

$$
\begin{array}{ll}
\text{Total functions:} & S \rightarrow T = \{f \mid f \in S \nrightarrow T \wedge \mathrm{dom}(f) = S\} \\
\text{Surjective functions:} & S \twoheadrightarrow T = \{f \mid f \in S \nrightarrow T \wedge \mathrm{ran}(f) = T\} \\
\text{Injective functions:} & S \rightarrowtail T = \{f \mid f \in S \nrightarrow T \wedge f^{-1} \in T \nrightarrow S\} \\
\text{Bijective functions:} & S \twoheadrightarrow T = \{f \mid f \in S \rightarrowtail T \wedge f \in S \twoheadrightarrow T\}
\end{array}
$$

MACHINE *Member*
SETS *MEMBER*
VARIABLES *members*, *memberId*
INVARIANT $members \subseteq MEMBER \wedge memberId \in members \rightarrowtail NAT$
...

MACHINE $Book$
SETS $BOOK, BOOKID$
VARIABLES $books, bookId$
INVARIANT $books \subseteq BOOK \wedge bookId \in books \rightarrowtail BOOKID$
$\quad \ldots$

MACHINE $Loan$
USES $Book, Member$
VARIABLES $loans$
INVARIANT $loans \in books \rightarrowtail members$

$\ldots$

MACHINE *Loan*
USES *Book*, *Member*
VARIABLES *loans*
INVARIANT $loans \in books \nrightarrow members$

...

OPERATIONS

$\mathtt{Lend}(m, b) \triangleq$

pre ???

then $loans :=$???

end

...

More about the dynamic language

- generalized substitution language (GSL)
- basic substitution: $x := E$
- initialization and a set of operations (pre and postcondition)
- semantics: weakest precondition

Generalized substitution language

No modification:	*skip*
Simple assignment:	$x := E$
Parallel composition:	$S \mid\mid S'$
Preconditioned substitution:	**pre** P **then** S **end**
Guarded substitution:	**select** P **then** S **end**
Bounded choice:	**choice** S_1 **or** S_2 $\cdots$ **or** S_n **end**
Guarded choice:	**select** P_1 **then** S_1
	when P_2 **then** S_2
	when P_3 **then** S_3
	...
	end
Alternative:	**if** P **then** S_1 **else** S_2 **end**
Non bounded choice:	**any** x **where** P **then** S **end**

Weakest precondition (wp)

- notation: $[S]P$
- substitution: link between before and after conditions
- weakest precondition: largest condition such that, after executing S, predicate P is satisfied
- example: $[x := x + 1](x \leq 5)$

Weakest precondition (wp)

- notation: $[S]P$
- substitution: link between before and after conditions
- weakest precondition: largest condition such that, after executing S, predicate P is satisfied
- example: $[x := x + 1](x \leq 5)$ answer: $x \leq 4$

wp for the main substitutions

$$
\begin{array}{ll}
[skip]P & \Leftrightarrow P \\
[x := E]P & \Leftrightarrow P^{[E/x]} \\
[x := E || y := F]P & \Leftrightarrow P^{[E,F/x,y]} \\
[\textbf{pre } Q \textbf{ then } S \textbf{ end}]P & \Leftrightarrow (Q \wedge [S]P) \\
[\textbf{select } Q \textbf{ then } S \textbf{ end}]P & \Leftrightarrow (Q \Rightarrow [S]P) \\
[\textbf{choice } S_1 \textbf{ or } S_2 \textbf{ end}]P & \Leftrightarrow ([S_1]P \wedge [S_2]P) \\
[\textbf{if } Q \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ end}]P & \Leftrightarrow (Q \Rightarrow [S_1]P \wedge \neg Q \Rightarrow [S_2]P) \\
[\textbf{any } x \textbf{ where } Q \textbf{ then } S \textbf{ end}]P & \Leftrightarrow \forall\, x \cdot (Q \Rightarrow [S]P)
\end{array}
$$

MACHINE *Loan*
USES *Book*, *Member*
VARIABLES *loans*
INVARIANT $loans \in books \leftrightarrow members$

$\quad\ldots$

OPERATIONS

$\quad\mathtt{Lend}(m, b) \triangleq$

$\quad\quad$ **pre** $m \in members \wedge b \in books$

$\quad\quad$ **then** $loans := loans \cup \{b \mapsto m\}$

$\quad\quad$ **end**

$\quad\ldots$

Machine consistency

- main principle: the behavior must preserve the invariant properties
- initialization: $[Init]INV$
- for each operation: $Pre \wedge INV \Rightarrow [S]INV$

MACHINE $Name(param)$
SETS T
VARIABLES V
INVARIANT INV
INITIALISATION $Init$
OPERATIONS
    `Op(...)` $\triangleq$
        pre Pre
        then S
        end

Proof obligation for Lend

PO template: $Pre \wedge INV \Rightarrow [S]INV$

application: $m \in members \wedge b \in books \wedge loans \in books \nrightarrow members$
$\Rightarrow [loans := loans \cup \{b \mapsto m\}](loans \in books \nrightarrow members)$

Proof obligation for Lend

PO to discharge: $m \in members \land b \in books \land loans \in books \rightarrow\!\!\!\!\!\!+\; members$
$\Rightarrow (loans \cup \{b \mapsto m\} \in books \rightarrow\!\!\!\!\!\!+\; members)$

Question: how to ensure that b has only one image in *loans*?

MACHINE *Loan*
USES *Book*, *Member*
VARIABLES *loans*
INVARIANT $loans \in books \leftrightarrow members$

...

OPERATIONS

$\mathtt{Lend}(m, b) \triangleq$

pre $m \in members \wedge b \in books$
then $loans := loans \lhd\!\!\!+ \{b \mapsto m\}$
end

...

MACHINE *Loan*
USES *Book*, *Member*
VARIABLES *loans*
INVARIANT $loans \in books \nrightarrow members$

$\quad \ldots$

OPERATIONS

$\quad \mathtt{Lend}(m, b) \triangleq$

$\qquad$ **pre** $m \in members \land b \in books \land b \notin \mathrm{dom}(loans)$

$\qquad$ **then** $loans := loans \cup \{b \mapsto m\}$

$\qquad$ **end**

$\quad \ldots$

Main differences

- evolution of the B language to specify complex systems
- closed systems: system and interactions with its environment as a whole
- event-based descriptions (no parameter, non bounded choice, guarded substitutions)
- refinement extended: ability to add new events
- supporting tools: Rodin platform

MACHINE *Example_System*
VARIABLES x
INVARIANT $x \in 0..2$
EVENTS

```
Initialisation ≜
    begin x := 0
    end
change0to1 ≜
    when x = 0
    then x := 1
    end
change1to2 ≜
    when x = 1
    then x :∈ {1, 2}
    end
```

Additional proof obligations in Event-B

- invariant preservation (INV): as in B
- feasibility (FIS): if an event is enabled, its action is possible
- well definedness (WD): formula are not evaluated outside their domain and ill-defined expressions are avoided
- deadlock freeness (DLF): there is always at least one event which is enabled after the execution of other events
- event convergence (VAR): new events introduced by refinement cannot take control forever

Different kinds of refinement together

- main idea: to go from "what" to "how"
- abstract specification: what the system does and must satisfy
- refinement: how the system computes the results
- hint: use as many refinement steps as required!
- data refinement: from an abstract state space to a concrete one linked by a gluing invariant
- operation/event refinement: rewriting of substitutions
- superposition refinement: ability to add concrete variables and new events
- proof obligations defined for each kind of refinement

About modularity

Different approaches

- very limited for both approaches
- modularity in B: 3 different levels of access to sets, variables and operations (SEES, USES, INCLUDES)
- organization of Event-B models: contexts for data types and static properties and machines for the description of the behaviour
- different strategy in Event-B: superposition refinement for incremental specifications
- parachute paradigm from Jean-Raymond Abrial

```
CONTEXT
  Library_Ctx
SETS
  BOOKS
  MEMBERS
END
```

Library parachuting

Library machine

Library_Ctx | Abstract_Library | Library_Ref1

```
VARIABLES
  loans
INVARIANTS
  inv1  :  loans ⊆ BOOKS × MEMBERS
EVENTS
  INITIALISATION  ≙
  STATUS
   ordinary
  BEGIN
   act1  :  loans ≔ ∅
  END

  Lend  ≙
  STATUS
   ordinary
  ANY
   b
   m
  WHERE
   grd1  :  b↦m ∈ (BOOKS × MEMBERS) \ loans
  THEN
   act1  :  loans ≔ loans ∪ {b↦m}
  END
```

```
MACHINE
  Library_Ref1
REFINES
  Abstract_Library
SEES
  Library_Ctx
VARIABLES
  books
  members
  loans
INVARIANTS
  inv1   :   books ⊆ BOOKS
  inv2   :   members ⊆ MEMBERS
  inv3   :   loans ∈ books ⇸ members
EVENTS
  INITIALISATION  ≙
  STATUS
   ordinary
  BEGIN
   act1   :   books ≔ ∅
   act2   :   members ≔ ∅
   act3   :   loans ≔ ∅
  END
```

```
Lend  ≙
STATUS
 ordinary
REFINES
 Lend
ANY
 b
 m
WHERE
 grd1  :  m ∈ members
 grd2  :  b ∈ books
 grd3  :  b ∉ dom(loans)
THEN
 act1  :  loans ≔ loans ∪ {b ↦ m}
END
```

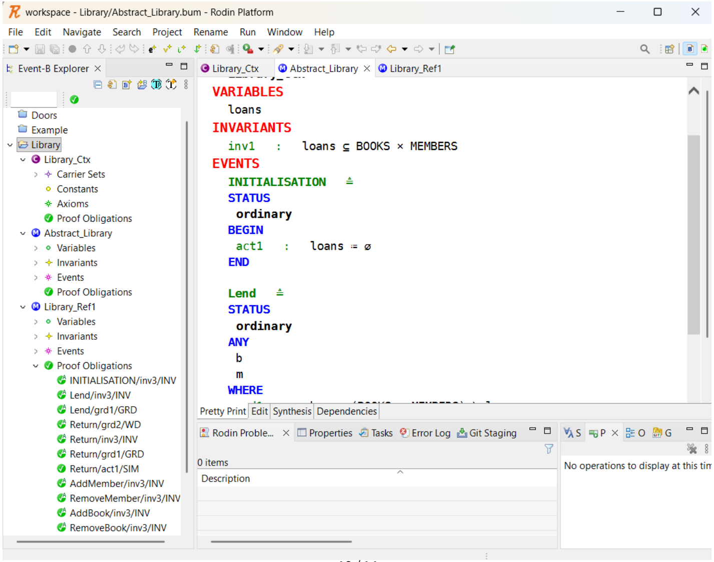
- Library
  - Library_Ctx
    - Carrier Sets
    - Constants
    - Axioms
    - Proof Obligations
  - Abstract_Library
    - Variables
    - Invariants
    - Events
    - Proof Obligations
  - Library_Ref1
    - Variables
    - Invariants
    - Events
    - Proof Obligations
      - INITIALISATION/inv3/INV
      - Lend/inv3/INV
      - Lend/grd1/GRD
      - Return/grd2/WD
      - Return/inv3/INV
      - Return/grd1/GRD
      - Return/act1/SIM
      - AddMember/inv3/INV
      - RemoveMember/inv3/INV
      - AddBook/inv3/INV
      - RemoveBook/inv3/INV

Refinement *à la* Event-B

- Rodin project online with more details: `https://fredericgervais.com/research/gt-verif-lacl-6-november-2025/`
- not well adapted for this particular case study
- more appropriate when considering the system as a whole
- interesting case studies: `https://www.event-b.org`

Supporting tools for Event-B

- open source environment for modeling, refinement and proof activities in Event-B
- based on the IDE Eclipse
- many plugins for specific purposes like editing, animation, proof, etc.
- one interesting tool for animation and verification: ProB
- for more details: `https://www.event-b.org/install.html`



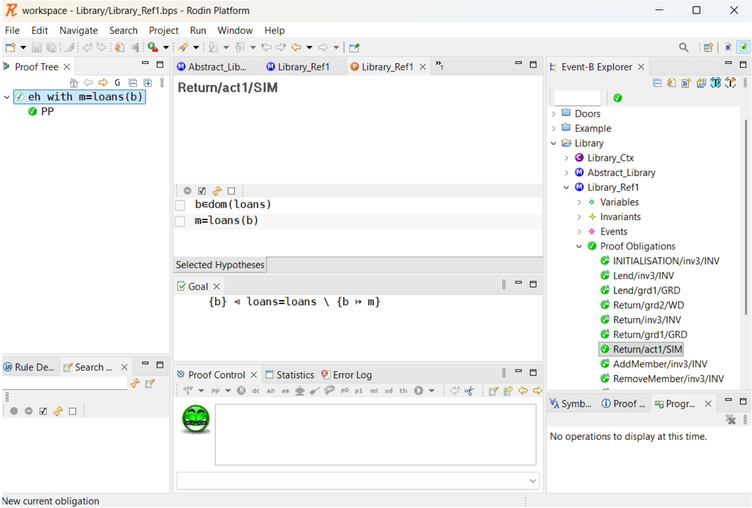
workspace - Library/Abstract_Library.bum - Rodin Platform
File Edit Navigate Search Project Rename Run Window Help
Event-B Explorer
Doors
Example
Library
Library_Ctx
Carrier Sets
Constants
Axioms
Proof Obligations
Abstract_Library
Variables
Invariants
Events
Proof Obligations
Library_Ref1
Variables
Invariants
Events
Proof Obligations
INITIALISATION/inv3/INV
Lend/inv3/INV
Lend/grd1/GRD
Return/grd2/WD
Return/inv3/INV
Return/grd1/GRD
Return/act1/SIM
AddMember/inv3/INV
RemoveMember/inv3/INV
AddBook/inv3/INV
RemoveBook/inv3/INV
Library_Ctx
Abstract_Library
Library_Ref1
VARIABLES
loans
INVARIANTS
inv1 : loans ⊆ BOOKS × MEMBERS
EVENTS
INITIALISATION ≙
STATUS
ordinary
BEGIN
act1 : loans ≔ ∅
END
Lend ≙
STATUS
ordinary
ANY
b
m
WHERE
Pretty Print
Edit
Synthesis
Dependencies
Rodin Proble...
Properties
Tasks
Error Log
Git Staging
0 items
Description
No operations to display at this tim

Proof activity

- proving perspective: explorer with proof obligations to discharge
- main automatic provers:
  - **PP** (predicate prover)
  - **ML** (mono-lemma)
  - **NewPP** (predicate prover from Rodin)
- some interactive tactics:
  - **ah** (add hypothesis)
  - **ct** (proof by contradiction)
  - **dc** (do cases)
  - **eh**/**he** (equality hypothesis)
  - **ae** (abstract expression)

Rodin platform

Proving perspective



Team on system modeling with formal methods

- pragmatic approach of formal methods
- languages and tools used: B, Event-B, Rodin, Theory plugin
- several research projects: FORMOSE, DISCONT, EBRP, TAPAS

References

- J.R. Abrial: *The B-Book*. Cambridge University Press, 1996
- J.R. Abrial: *Modeling in Event-B*. Cambridge University Press, 2010
- Atelier B: `https://www.atelierb.eu`
- Rodin: `https://www.event-b.org`
- ProB : `https://prob.hhu.de`