Using B as a High Level Programming Language in an Industrial Project: Roissy VAL

Wayside Control Unit – Safety Critical Software

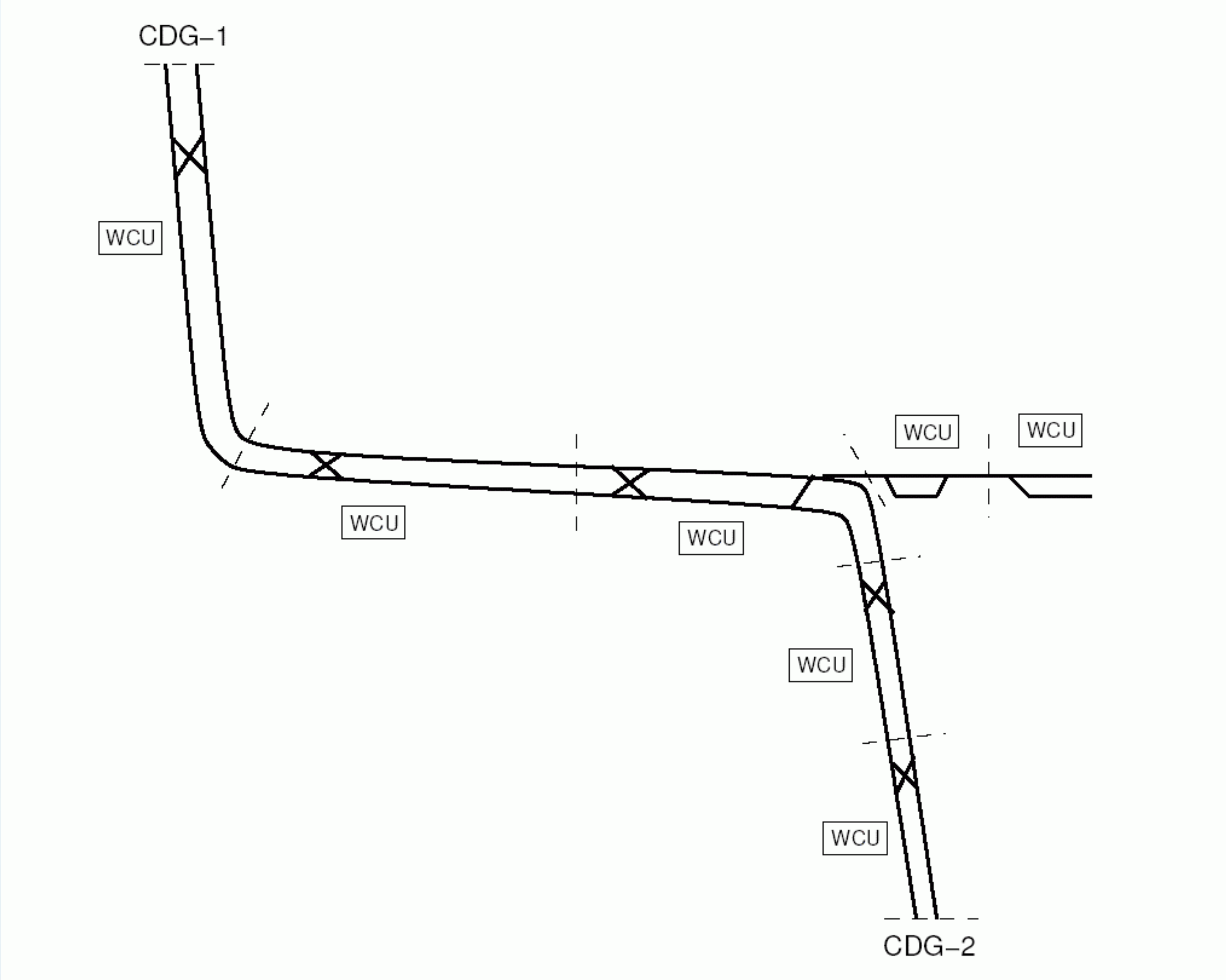
(WCU)

Introduction

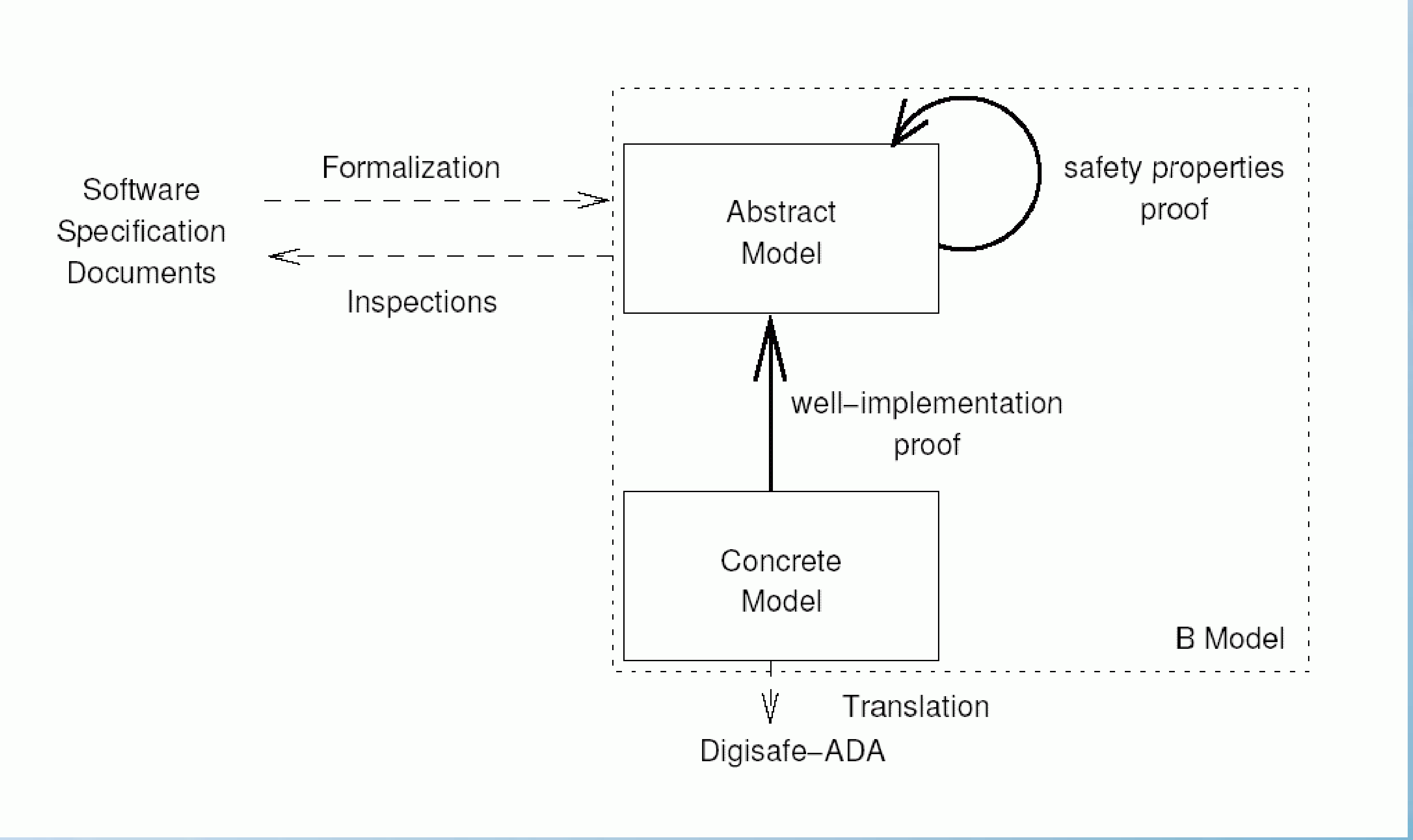
- **Historic**
  - ✓ Procedural B: producing a piece of software
  - ✓ Event Driven B: formalizing systems

- **An industrial process**
  - ✓ Building a software, correct by construction / no Unit Tests
  - ✓ A process created by Siemens for the Meteor project
  - ✓ The Size Factor
    - keeping the benefits of B **and** controlling cost on a large project

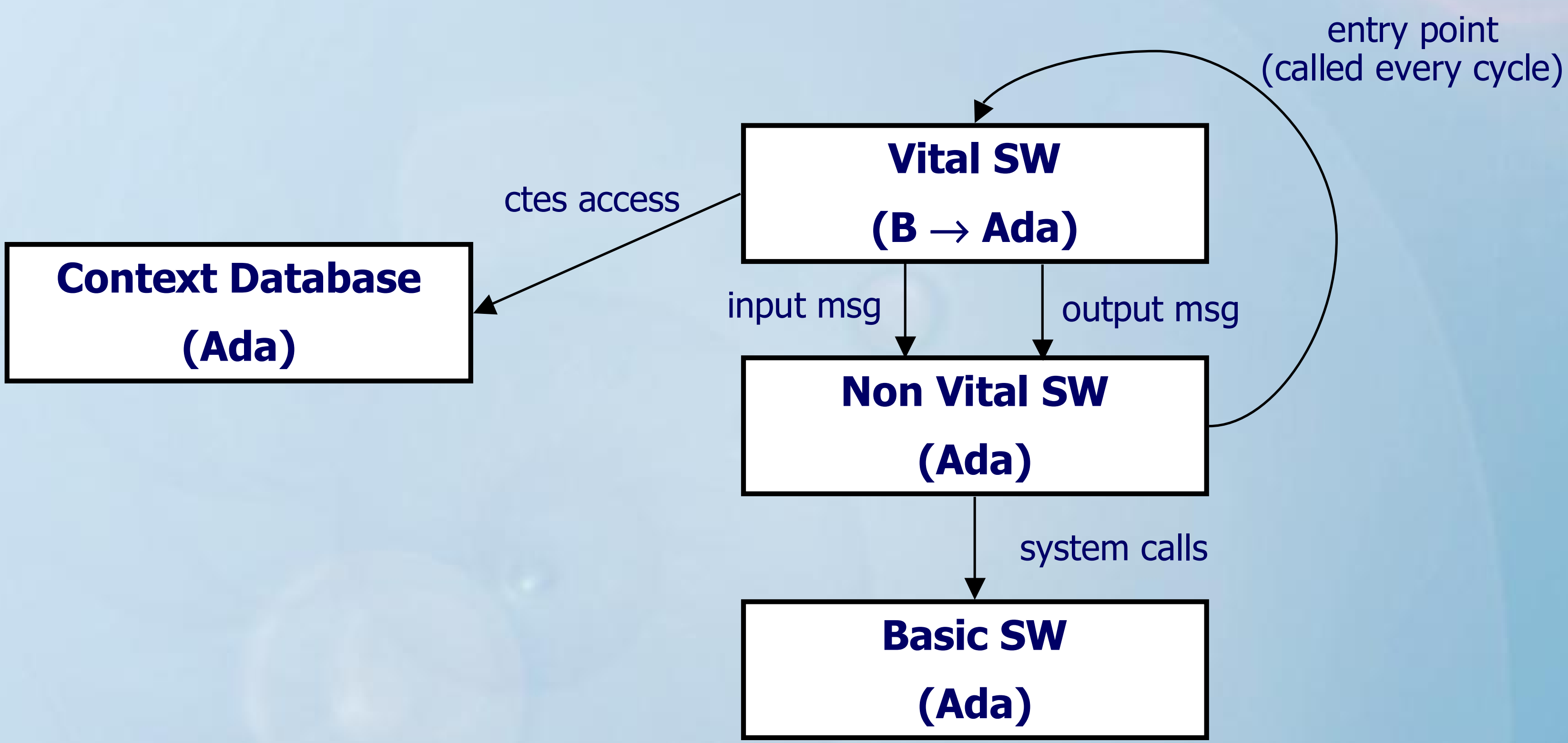
Roissy Airport VAL Shuttle



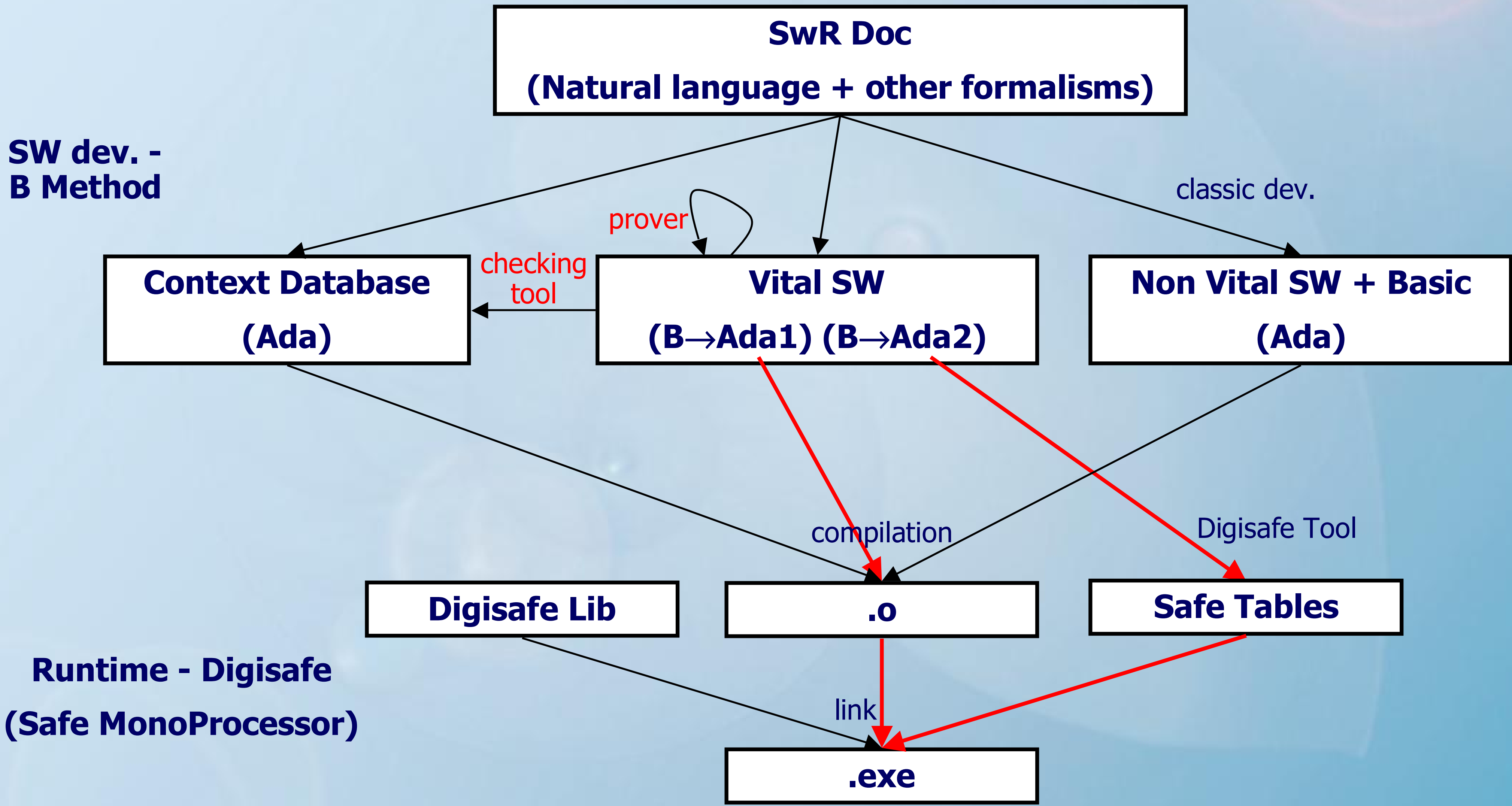
Abstract and Concrete Models



Software Architecture



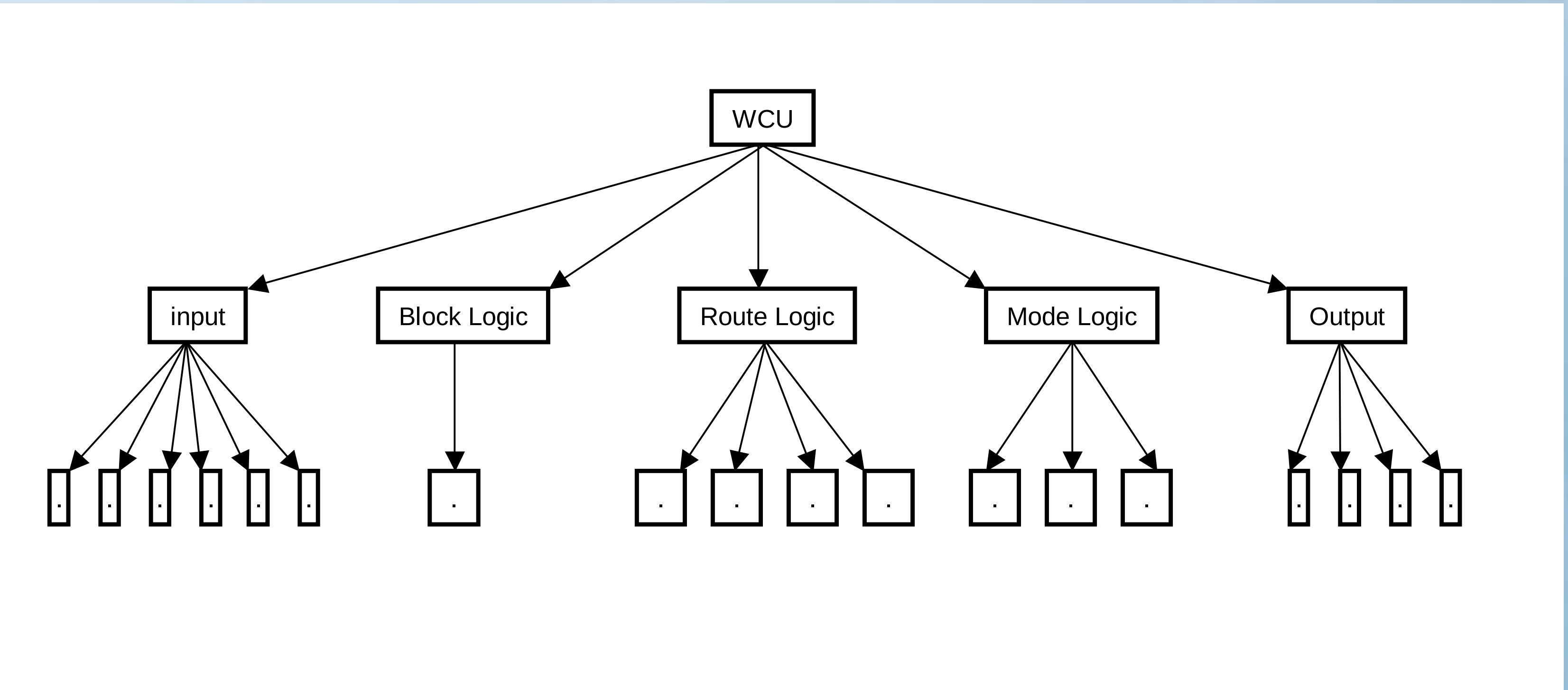
Software + Runtime Architecture



Software Specification Documents

- Detailed software specification documents
- The result of System Analysis
- Functional breakdown, data flow
- High-level data and (deterministic) treatments

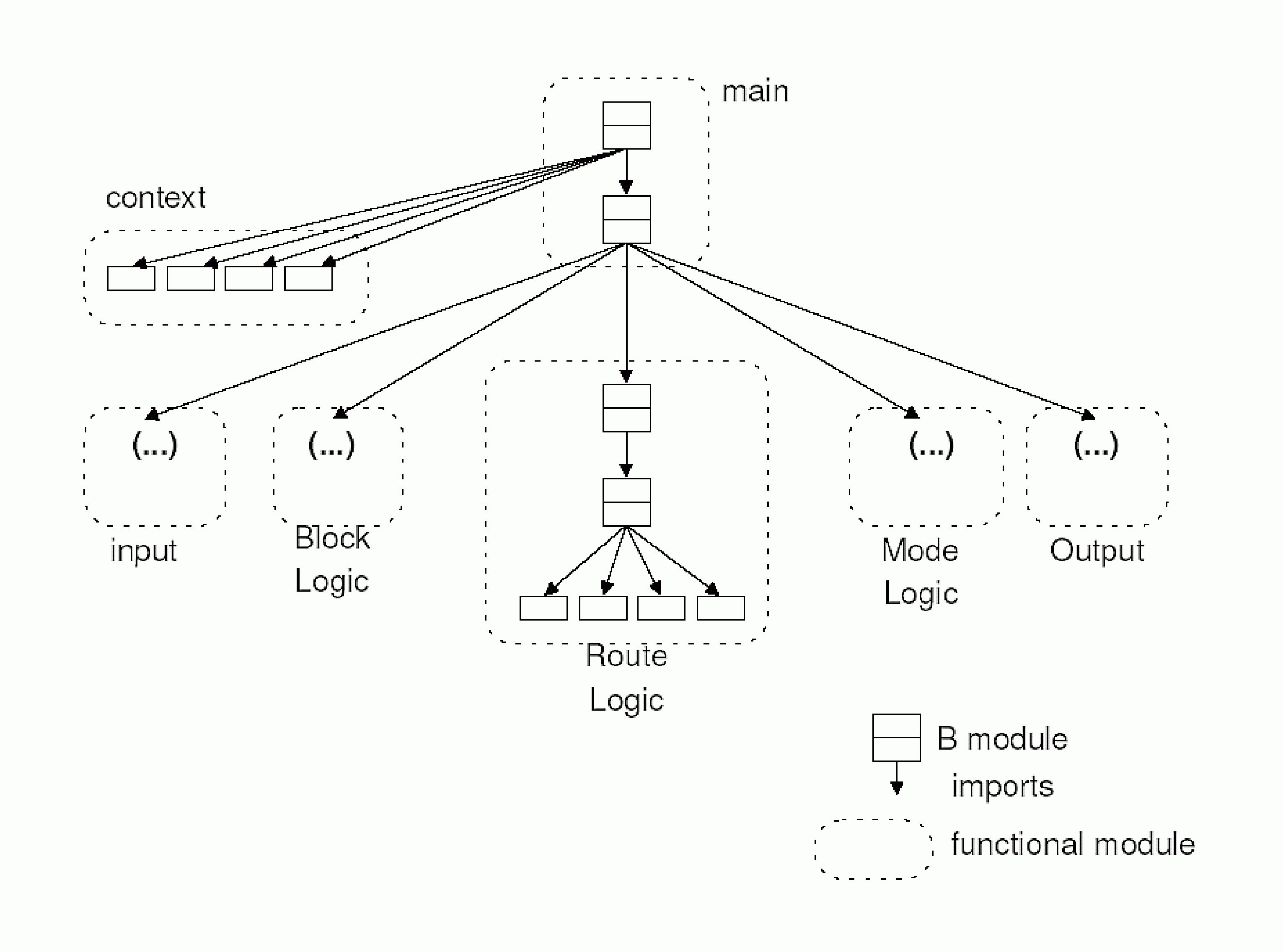
Software Spec Functional Breakdown



Abstract Model

- Its architecture follows the functional breakdown
- Context: to define types and constants (railway configuration database)
- Terminal and sequencing modules
- A high-level language (data and treatments)
- Safety critical properties strengthen the abstract model (up to the « run_cycle » entry operation)
- Code inspections check the model against its spec documents

Abstract Model



A High Level Programming Language: Types

- **Basic Types**
  - ✓ Fixed SETS
    t_block
  - ✓ Enumerated SETS
    t_block_type = { c_normal_block,
    c_switch_block,
    c_station_block}
  - ✓ BOOL
  - ✓ INT (for delays)

- **Generic data types**
  - ✓ t_a
  - ✓ P(t_a)
  - ✓ t_a 1 t_b
  - ✓ t_a 2 t_b
  - ✓ t_a 3 t_b
  - ✓ t_a 3 P(t_b)
  - ✓ t_a 3 (t_b 2 t_c)
  - ✓ seq(t_a)

A High Level Programming Language: Data

- ✓ Abstract Variables
  - occupied_blocks (t_block

- ✓ Abstract Constants
  - ctx_next_block_up : t_block 2 t_block

- ✓ Generic read operations
  - p_bool c read_occupied_blocks(p_block) =

```
PRE
      p_block : t_block
THEN
      p_bool := bool(p_block : occupied_blocks)
END
```

A High Level Programming Language

❑ Operations

- ✓ terminal operations:
  - occupied_blocks := occupied_blocks u otd u
    (ctx_b2bd_up u ctx_b2bd_down)$^{-1}$[obd]

- ✓ sequencing / “for each” loop operations

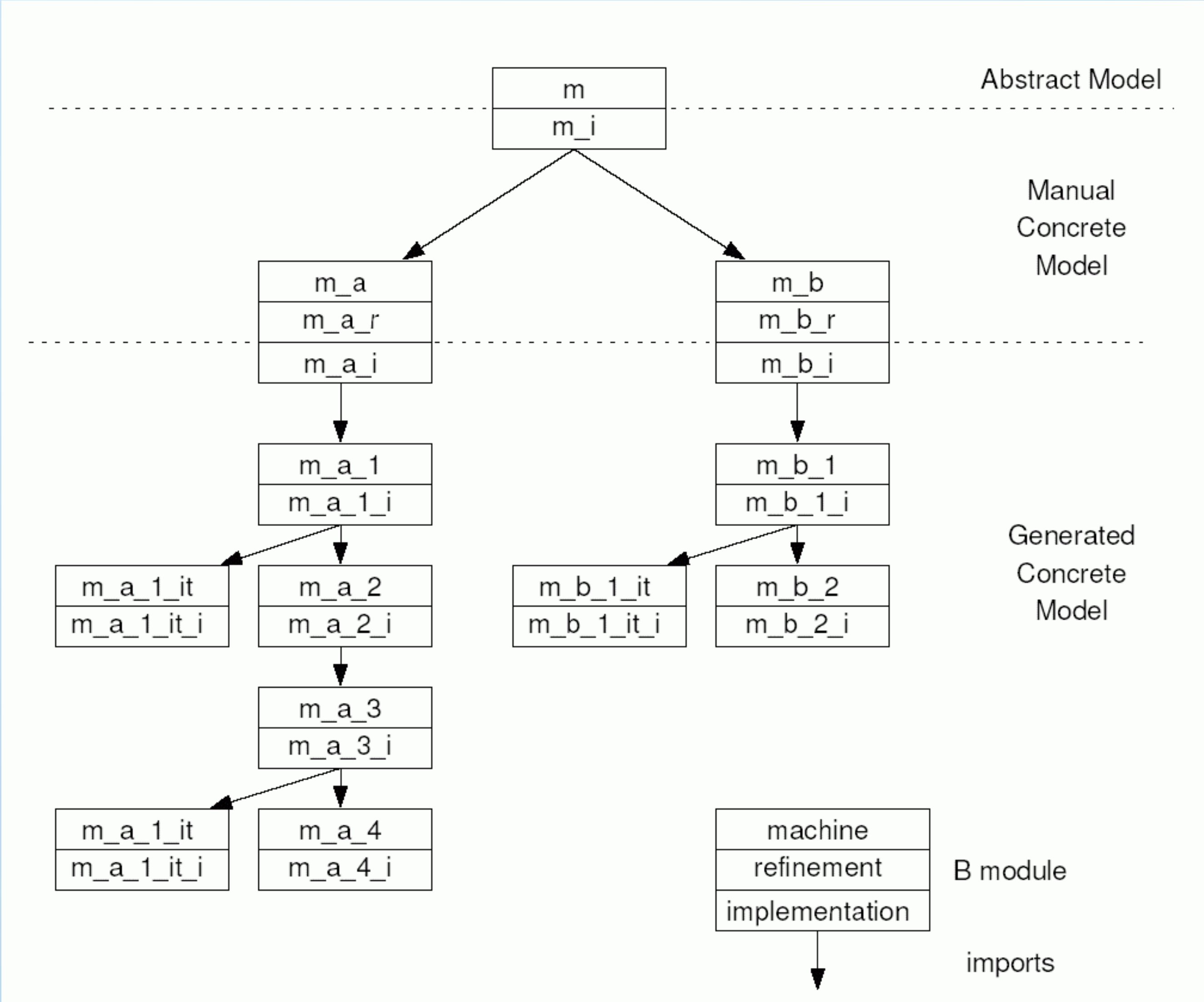
❑ Properties

- ✓ important safety critical properties
  - ! block . (block : t_block &
    ((ctx_b2bd_up u ctx_b2bd_down)[{block}] i obd d 0 o
    block : otd)
    y
    block : occupied_blocks)

Concrete Model: Principles

- **it should implement every terminal module of the Abstract Model (the new *complete* contract)**
- **the well-implementation is 100% covered by proof**
- **uses:**
  - ✓ manual refinement preparation, based on copy/paste
    - split by *importing* several modules
    - intermediate refinements
  - ✓ automatic refinement tools developed by Siemens
    - Edith B: first refinement step (to normalize)
    - Bertille: next refinement steps (to implement step by step)

Concrete Model



Automatic Refinement

- ✓ using systematic rules to implement the "high-level programming language" with B0

- ✓ data refinement rules (complete)
  - occupied_blocks_i : t_block 3 BOOL &
    occupied_blocks = occupied_blocks_i^{-1}[{TRUE}]

- ✓ substitutions refinement rules
  - completing rules manually
  - based on generic iterators modules that iterate on the elements of a constant set
  - 10% loss on generated code efficiency / high development cost reduction

Automatic Refinement

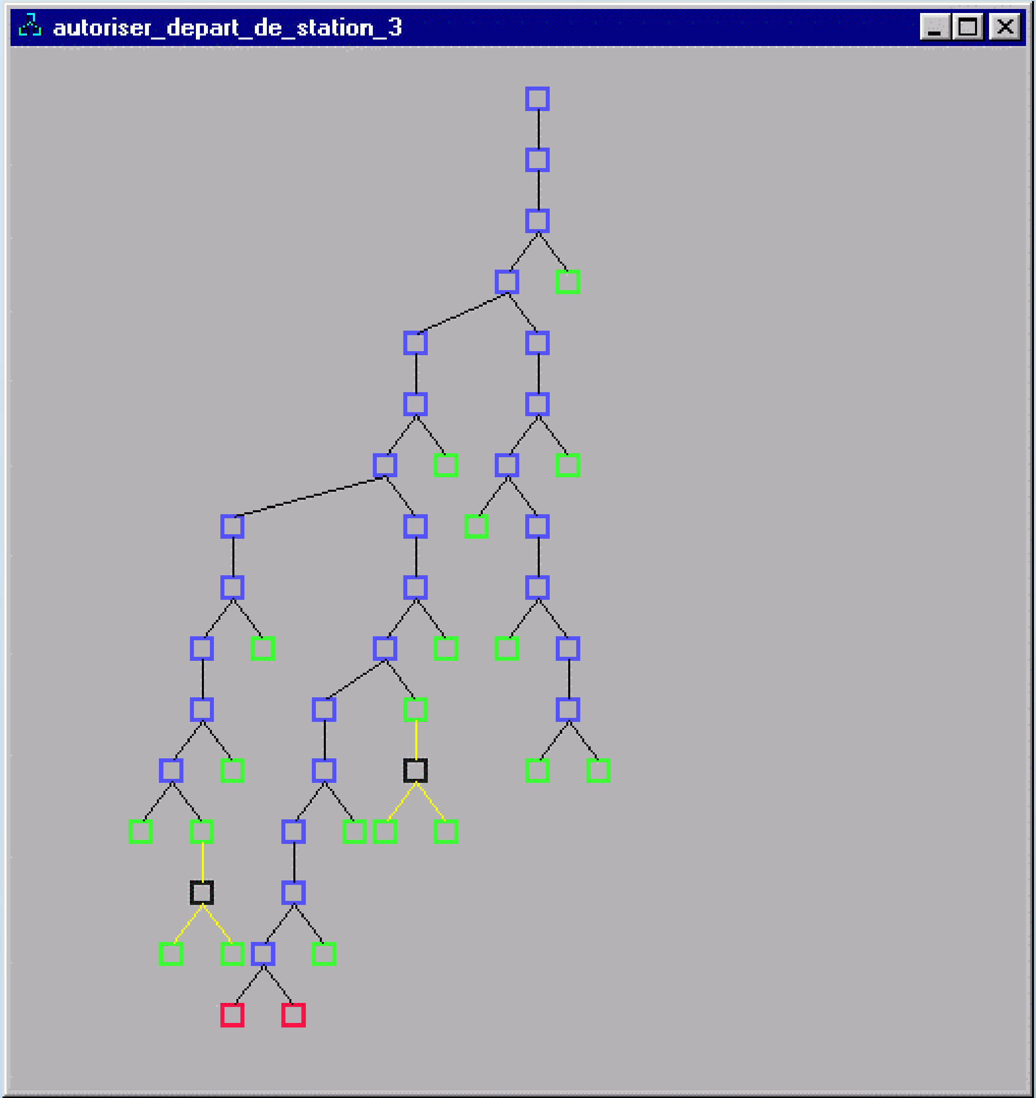
$$\begin{array}{l}\textbf{MACHINE}\ m\_a\_1\\ op\_1\ \ \widehat{=}\\ \quad a := bool(S \neq \varnothing)\\ \dots\end{array}$$

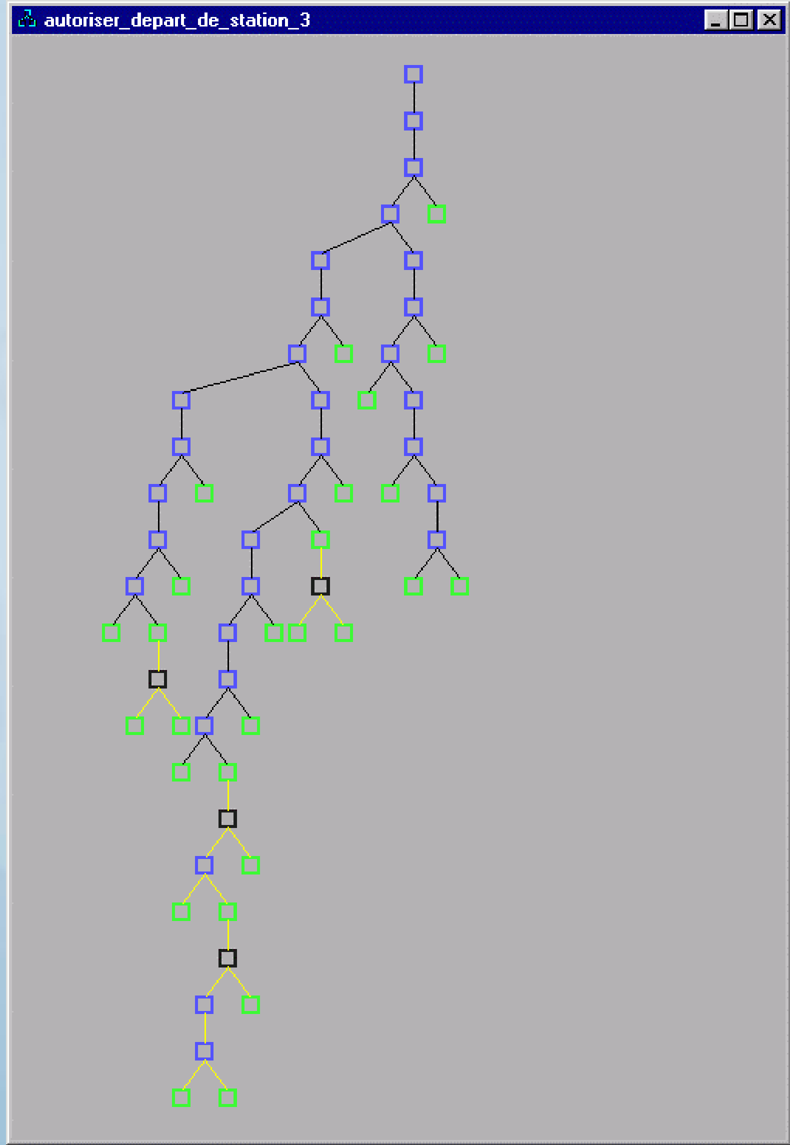
$$\begin{array}{l}\textbf{IMPLEMENTATION}\ m\_a\_1\_i\\ op\_1\ \ \widehat{=}\\ \quad \dots\\ \quad a := \mathrm{FALSE};\\ \quad \textbf{WHILE}\ continue = \mathrm{TRUE}\ \textbf{DO}\\ \quad\quad continue, x \leftarrow iterate\_t\_a;\\ \quad\quad y \leftarrow op\_1\_1(x);\\ \quad\quad \textbf{IF}\ y = \mathrm{TRUE}\ \textbf{THEN}\\ \quad\quad\quad a := \mathrm{TRUE}\\ \quad\quad \textbf{END}\\ \quad\quad \dots\end{array}$$

$$\begin{array}{l}\textbf{MACHINE}\ m\_a\_1\_it\\ p\_bool, p\_elt \leftarrow iterate\_t\_a\ \ \widehat{=}\\ \dots\end{array}$$

$$\begin{array}{l}\textbf{MACHINE}\ m\_a\_2\\ p\_y \leftarrow op\_1\_1(p\_x)\ \ \widehat{=}\\ \textbf{PRE}\\ \quad p\_x \in t\_a\\ \textbf{THEN}\\ \quad p\_y := bool(p\_x \in S)\\ \textbf{END}\\ \dots\end{array}$$

Automatic Refinement: Adding New Rules





Statistics: Sizes

Specification document sizes	**230 p.**	**50 p.**	**80 p.**	**100 p.**

B Model Size	**Lines**	**Rate**
Abstract Model without read operations and iterators (324 operations)	28,000	15%
Abstract Model: read operations and iterators	11,000	6%
Manual Concrete Model without read operations	28,000	15%
Automatic Concrete Model	118,000	64%
Total (532 modules, 1,093 components)	**184,000**	**100%**

Digisafe Ada Lines number	**158,000**
Number of Ada procedures	4,809

Statistics: Proof

Proof of Lemmas	Nb	Rate
Grand Total	**43,600**	**100%**
Prover force 0	38,800	89%
Prover force 1	1,400	3%
Generic demonstrations (61 demo) - based on the predicate prover	2,000 (1,300)	5% (65%)
Total of automatic demonstrations	**42,200**	**97%**
Interactive demonstrations (745 demo)	1,400	3%

Validation of Proof Rules	Nb	Rate
Total	**290**	**100%**
Validated by the predicate prover	243	84%
Validated semi-automatically	27	9%
Validated manually	20	7%

Statistics: Cost

Manpower Cost	Rate
Warmup	5%
Project management	8%
Abstract Model - questions/answers and doc analysis - proof - model inspections	55% (33%) (29%) (9%)
Concrete Model - proof	24% (46%)
Finalization	8%
Total	**100%**

CLEARSY SYSTEM ENGINEERING

Conclusion

❑ A process

- ✓ using B as a high-level programming language, with proof capabilities
- ✓ main features of the process:
  - 2 steps: abstract and concrete model
  - genericity: data types, read operations, iterators
  - automatic tools: refinement, …
- ✓ to build software, correct by construction
- ✓ suitable for procedural, logical software (no floating point)