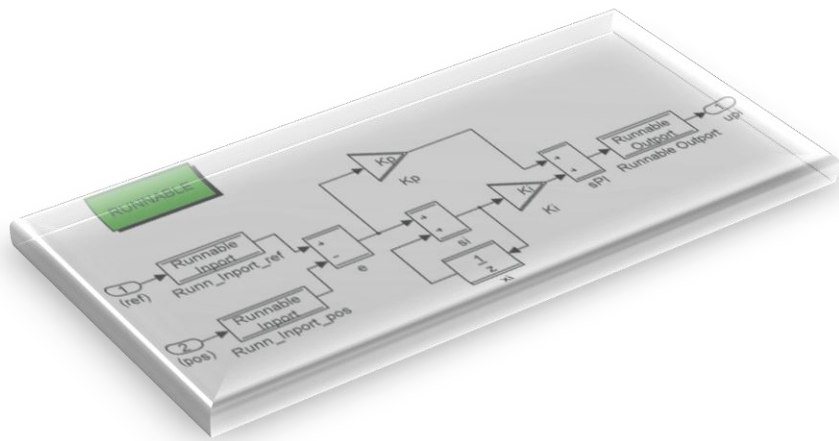
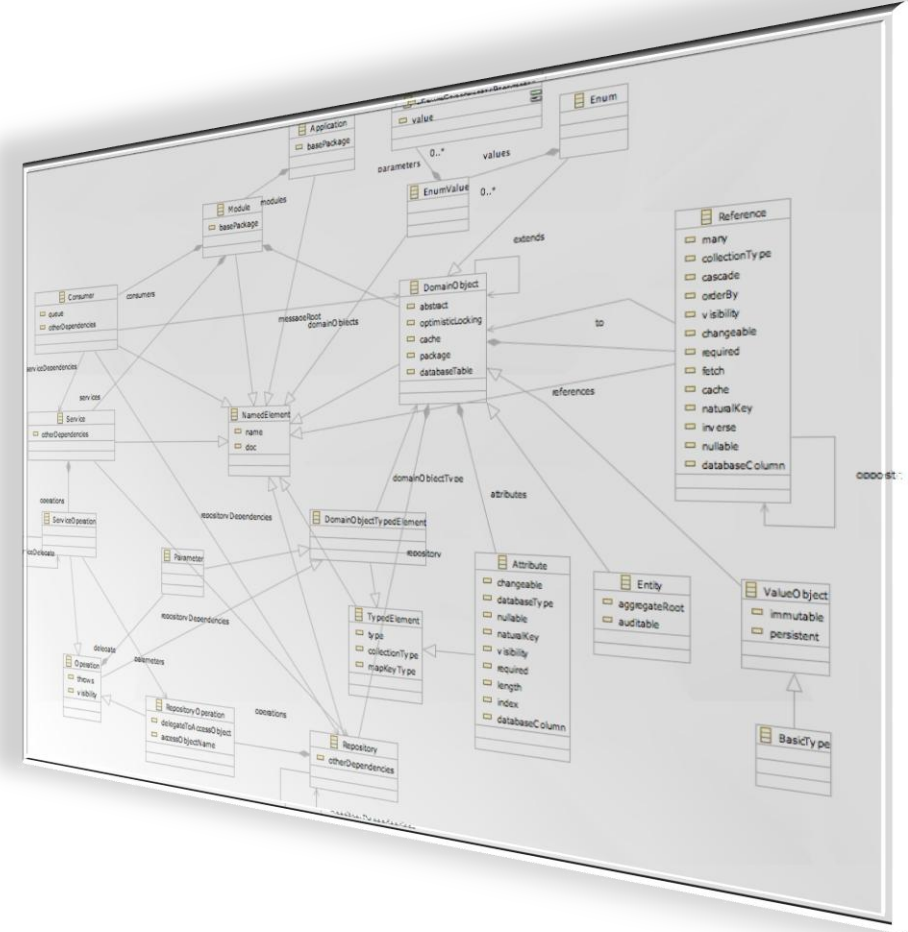


# Code Generation

Thierry Lecomte

[thierry.lecomte@clearsy.com](mailto:thierry.lecomte@clearsy.com)



# Introduction

---

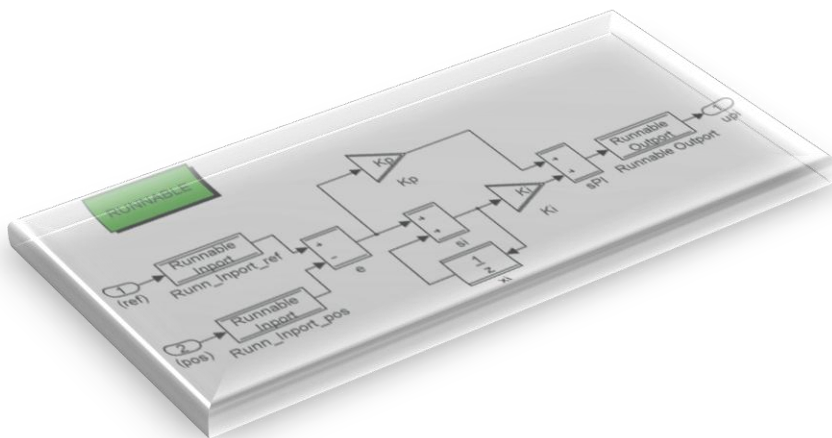
- B model is not end-product
- Hardly readable/understandable even by its creator
- No processor so far able to natively execute B models

# Introduction

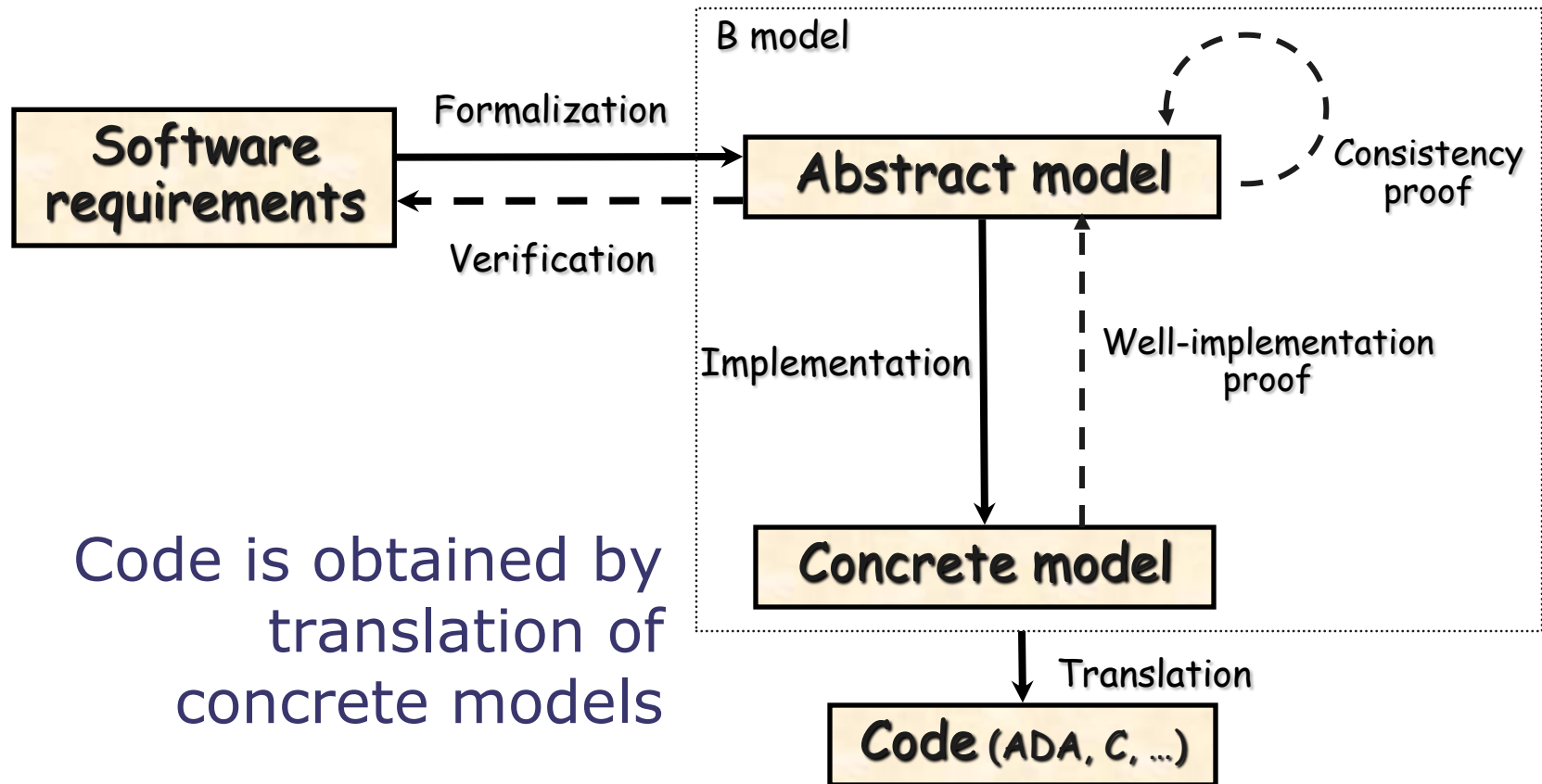
---

- Hence some transformations are required:
  - Animation
  - (Automatic) Refinement
  - “Code” Generation
- This presentation focuses on the last item

# ***Code Generation from B models***



# Code delivery

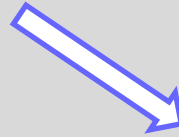


# Notion of concrete models

- Definition of a subset of B called B0
  - Implementable substitutions (no non-determinism, no || )
  - Implementable types
- Translators with no « intelligence »: 1 to 1 translation
  - C, Ada, HIA
- Translation schemas for cyclic software (data acquisition, computation, output)
- System level model to prove other execution mode (interruption, distributed)

# Example: translation in C

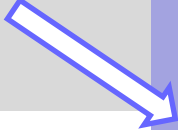
```
compute_initial_level =  
VAR m1, m2 IN  
  m1, m2 <-- measure_level;  
  estimated_level <-- minimum(m1, m2);  
  IF estimated_level <= WARNING_CAPACITY  
  THEN  
    status := LOW_LEVEL  
  ELSE  
    status := NOMINAL  
  END  
END
```



```
void fuel0__compute_initial_level(void) {  
  {  
    int fuel0__m1;  
    int fuel0__m2;  
  
    measure__measure_level (&fuel0__m1, &fuel0__m2);  
    utils__minimum (fuel0__m1, fuel0__m2,  
      &fuel0__estimated_level);  
    if (fuel0__estimated_level <= ctx__WARNING_CAPACITY)  
    {  
      fuel0__status = ctx__LOW_LEVEL;  
    }  
    else {  
      fuel0__status = ctx__NOMINAL;  
    }  
  }  
}
```

# Example: translation in High Integrity Ada

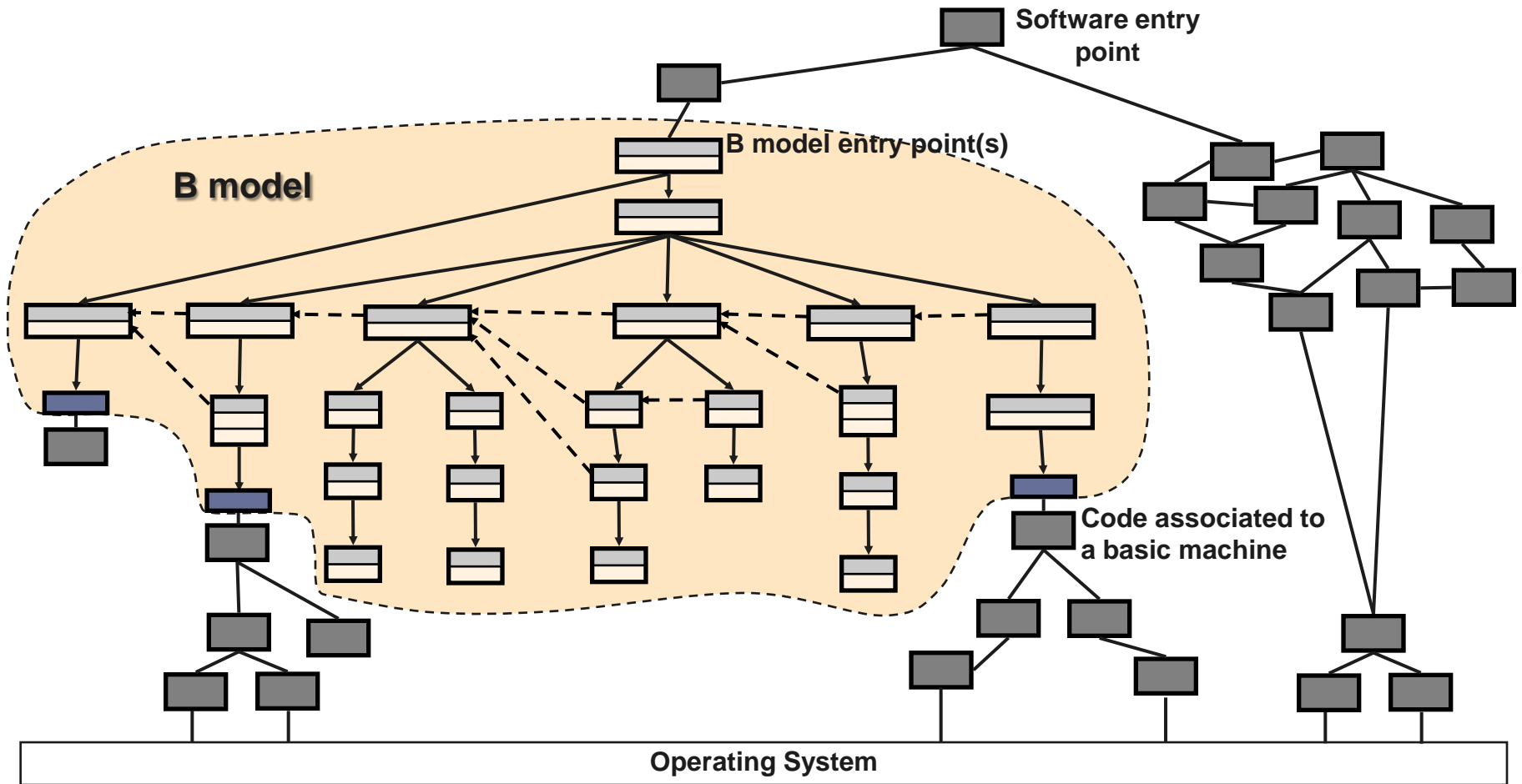
```
compute_initial_level =  
VAR m1, m2 IN  
  m1, m2 <-- measure_level;  
  estimated_level <-- minimum(m1, m2);  
  IF estimated_level <= WARNING_CAPACITY  
  THEN  
    status := LOW_LEVEL  
  ELSE  
    status := NOMINAL  
  END  
END
```



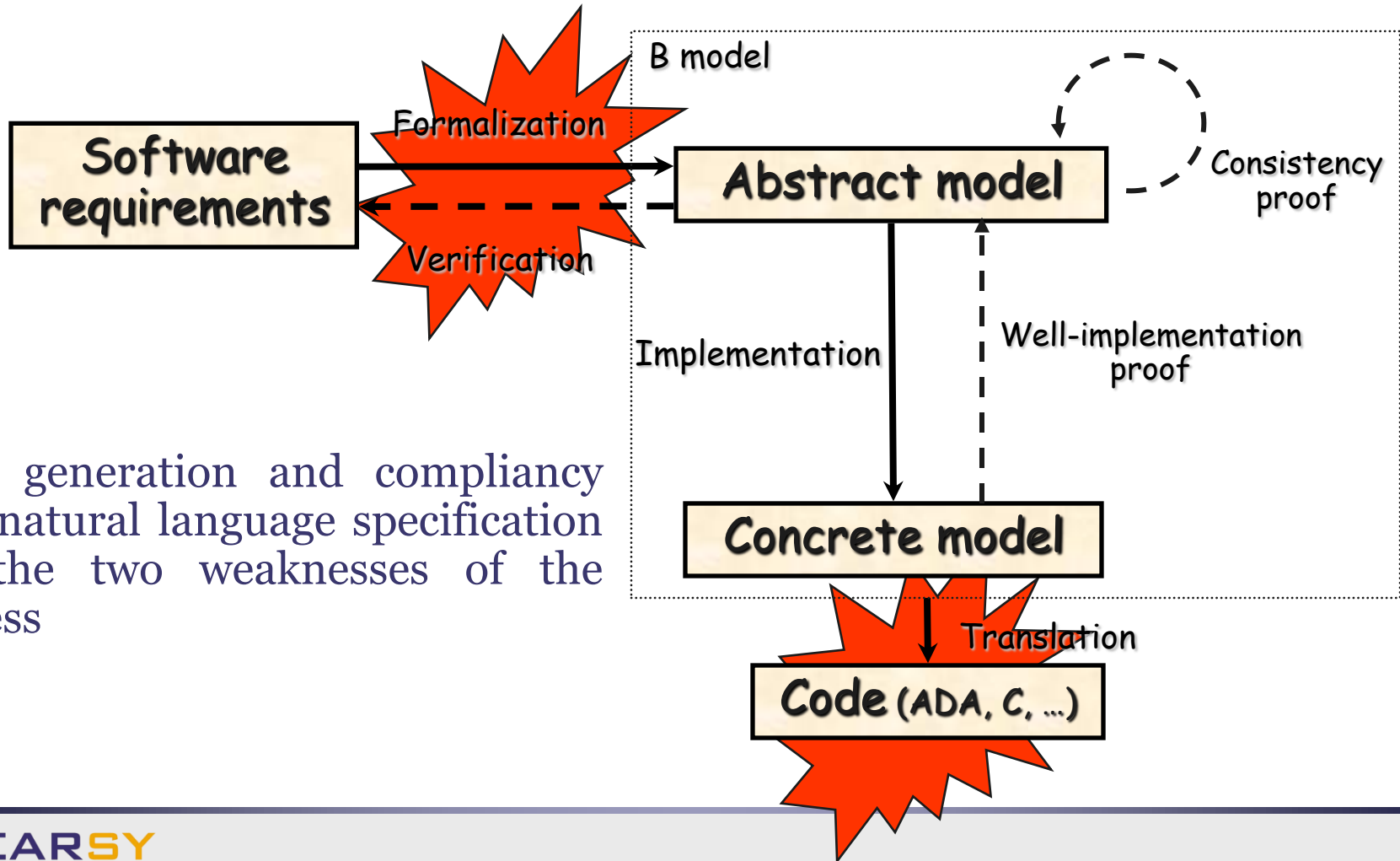
```
procedure #fuel0#compute_initial_level is  
  #fuel0#compute_initial_level#1#m1 : INTEGER ;  
  #fuel0#compute_initial_level#1#m2 : INTEGER ;  
begin  
  
  #MACHINE#fuel0#measure.#measure#measure_level(#fuel0#  
compute_initial_level#1#m1, #fuel0#compute_initial_level#1#m2) ;  
  
  #MACHINE#fuel0#utils.#utils#minimum(#fuel0#compute_initi  
al_level#1#m1, #fuel0#compute_initial_level#1#m2,  
#fuel0#estimated_level) ;  
  if (#fuel0#estimated_level <= ctx.#ctx#WARNING_CAPACITY)  
  then  
    #fuel0#status := ctx.#ctx#LOW_LEVEL ;  
  else  
    #fuel0#status := ctx.#ctx#NOMINAL ;  
  end if ;  
end #fuel0#compute_initial_level ;
```



# Integration with other software



# Weaknesses at interfaces



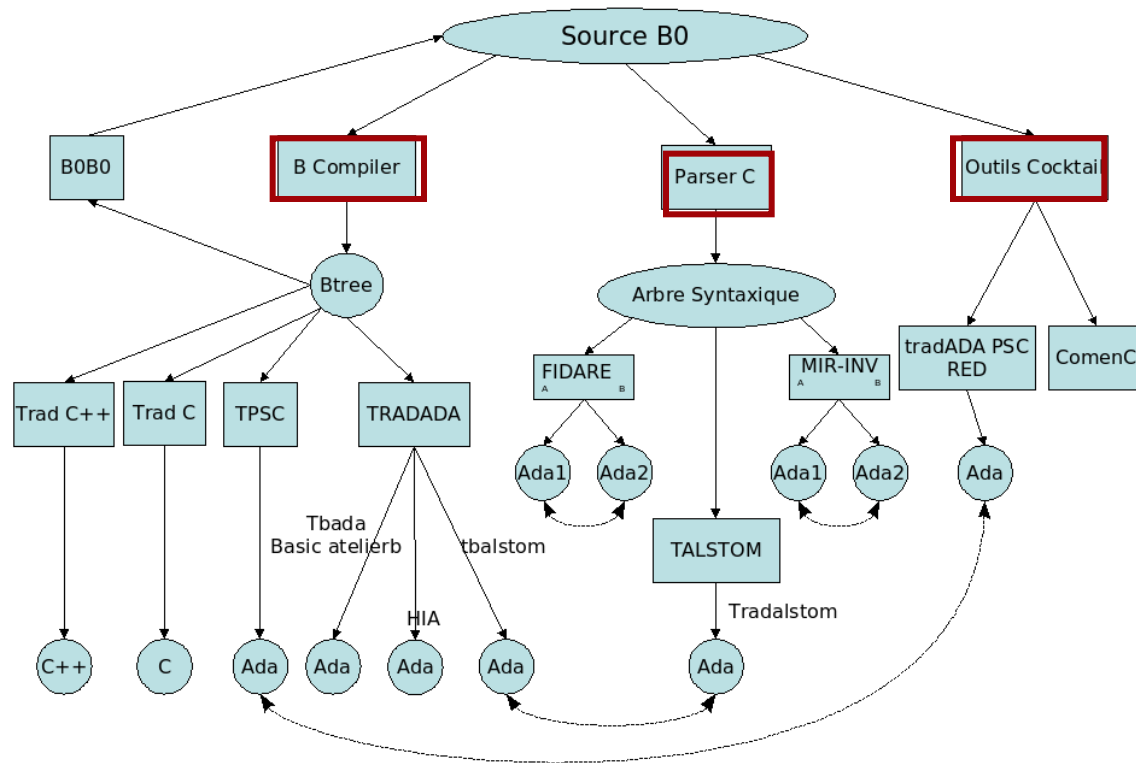
Code generation and compliancy with natural language specification are the two weaknesses of the process

# Safety critical applications

- Require guaranty to reach SIL3/SIL4 level
  - SIL = Safety Integrity Level
  - SIL3 =  $10^{-7}$  failure / h
  - SIL4 =  $10^{-9}$  failure / h
- Redundancy and diversification to avoid common mode failures
  - Two computers running two different instances
$$x := y + z \quad || \quad x := y + 1 + z - 1$$
bitwise little and big endian data
- Specific hardware and encoding: secure coded processor

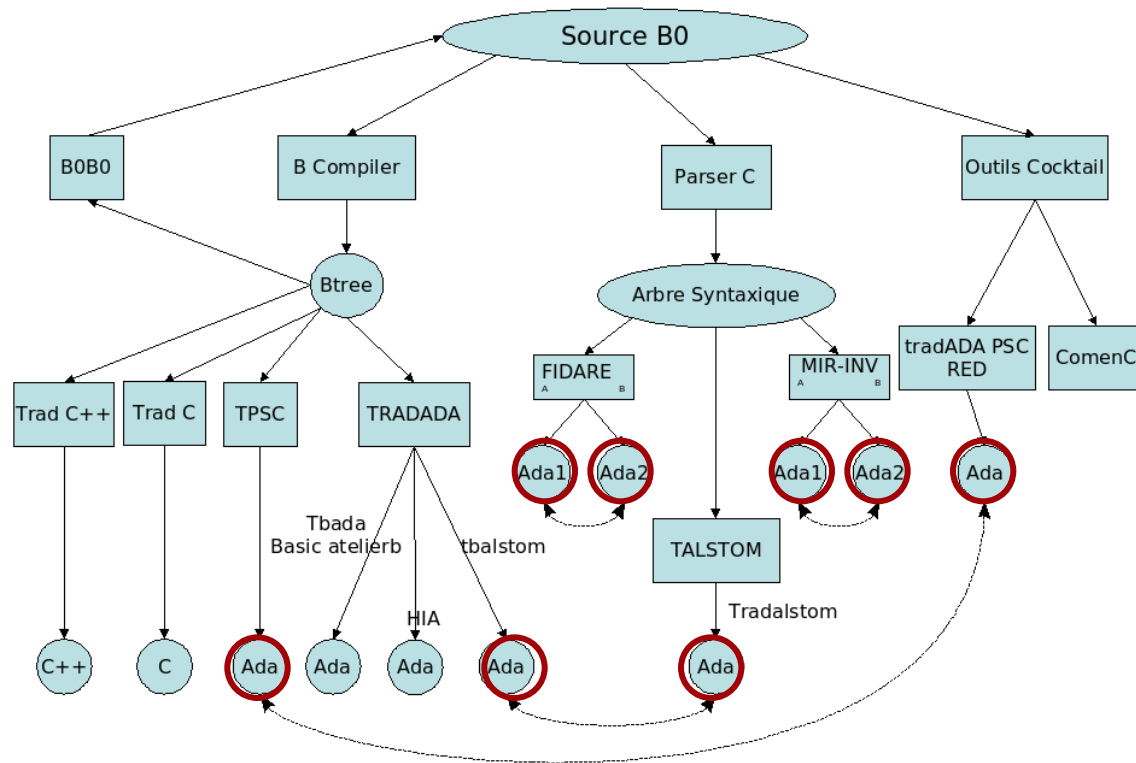
# Code generation

- Based on different tools to avoid common mode failure
  - Type-checker, B-parser, B-compiler



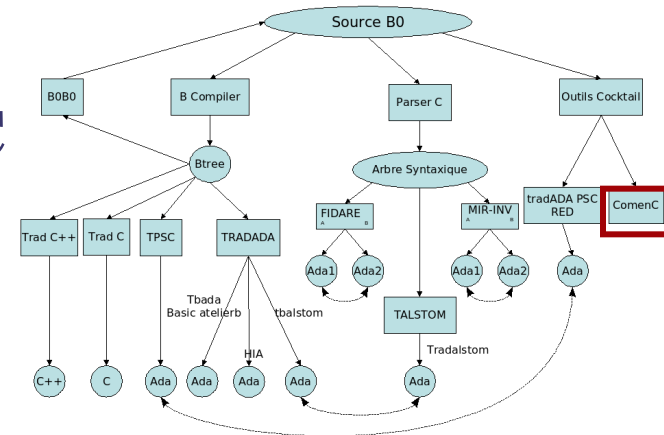
# Code generation

- Translators to be used in pair

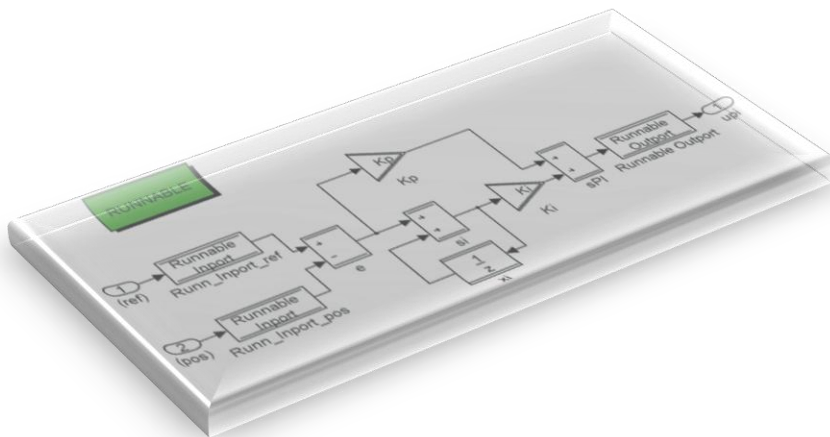


# C Code generation

- Safety critical standards recommend:
  - (1) A limited use of pointers
  - (2) No recursion
  - (3) No dynamic memory allocation
- With instantiated machines, point (1) was not reachable
- Development of a translator based on cocktail compiler compiler: ComenC
- C code more readable
- But discontinued support

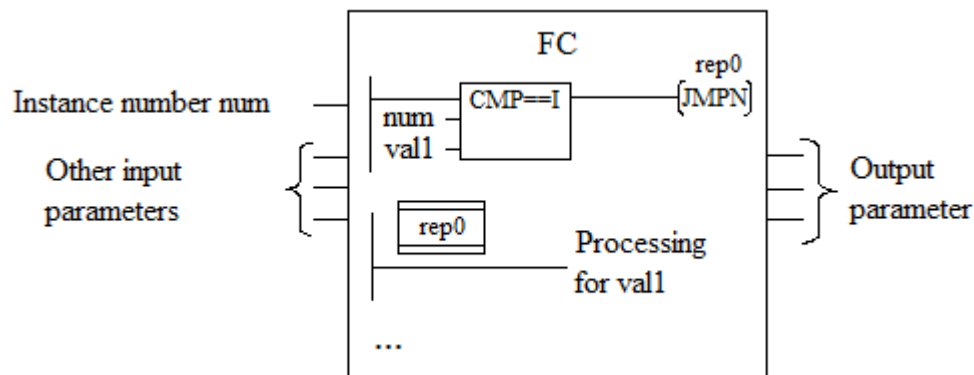


# ***Code Generation from Event-B models***



# Ladder Code Generation

- Transformation of an event-B model into a ladder code in order to feed a PLC



- For S7 Simatic (Siemens) [not sticking to IEC 61131-3 standard]

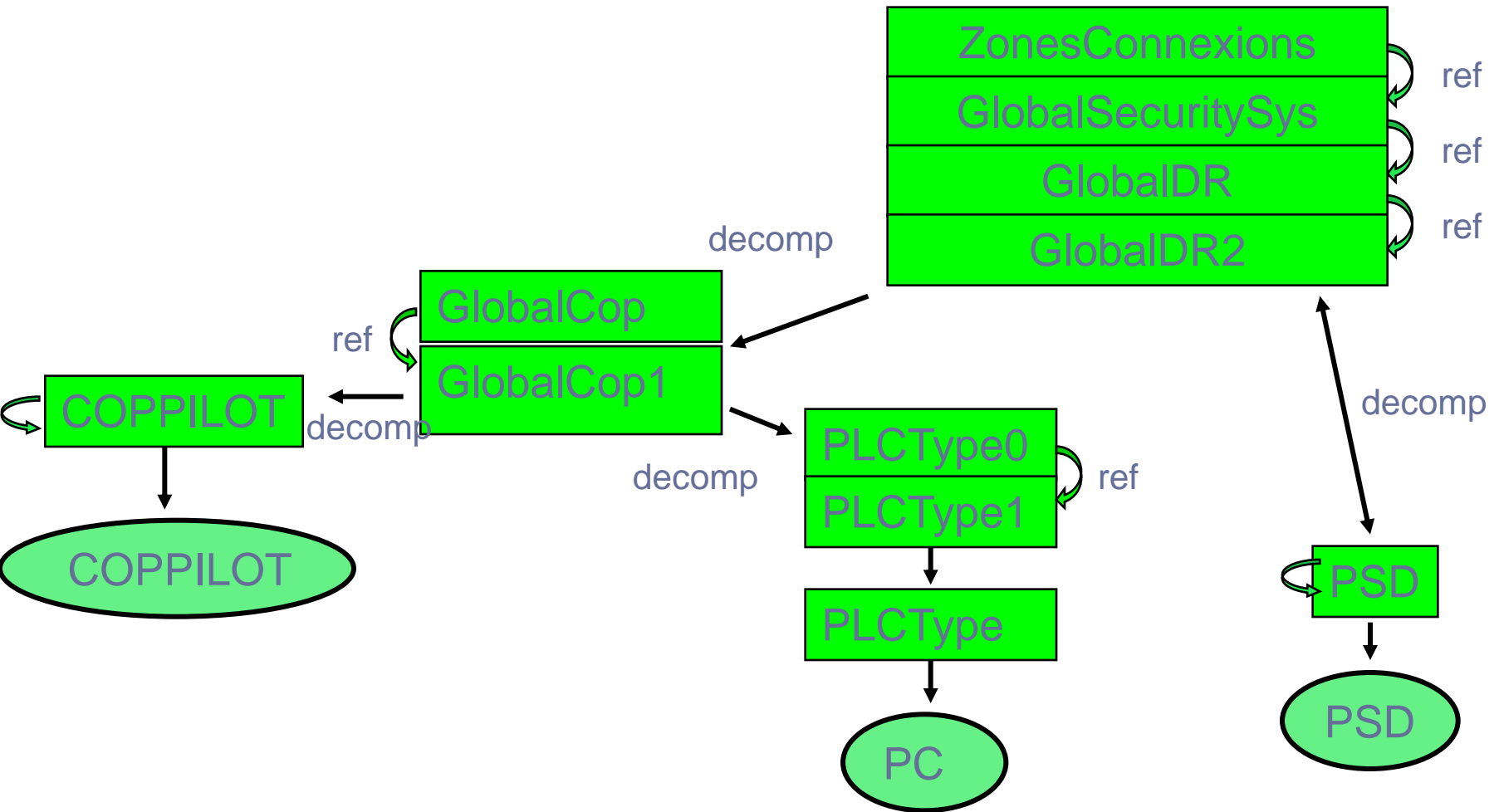


# Modelling phase 1 (PSD C&C System)



- define the properties expressing system safety
- demonstrate that any train + PSD system verifying some properties is safe
- open train doors iff train is at the standstill and doors in front of PSD
  - open PSD iff train at the standstill is present or in case of evacuation
  - a train should not move if at least one PSD is not closed

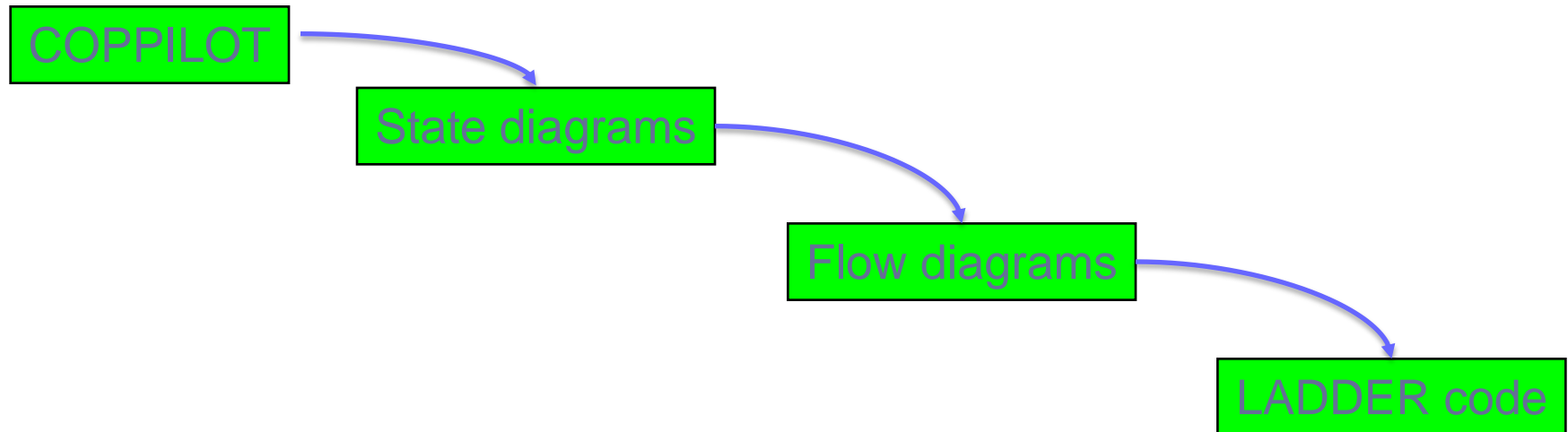
# Modelling phase 2 (PSD C&C System)



# Modelling phase 3 (PSD C&C System)

```
1.  ----[ ]-----|--[ ]--|-----{ }--  
      X              |   Y   |           S  
                    |       |  
                    |--[ ]--|  
                        Z
```

The above realises the function:  $S = X \text{ AND } (Y \text{ OR } Z)$

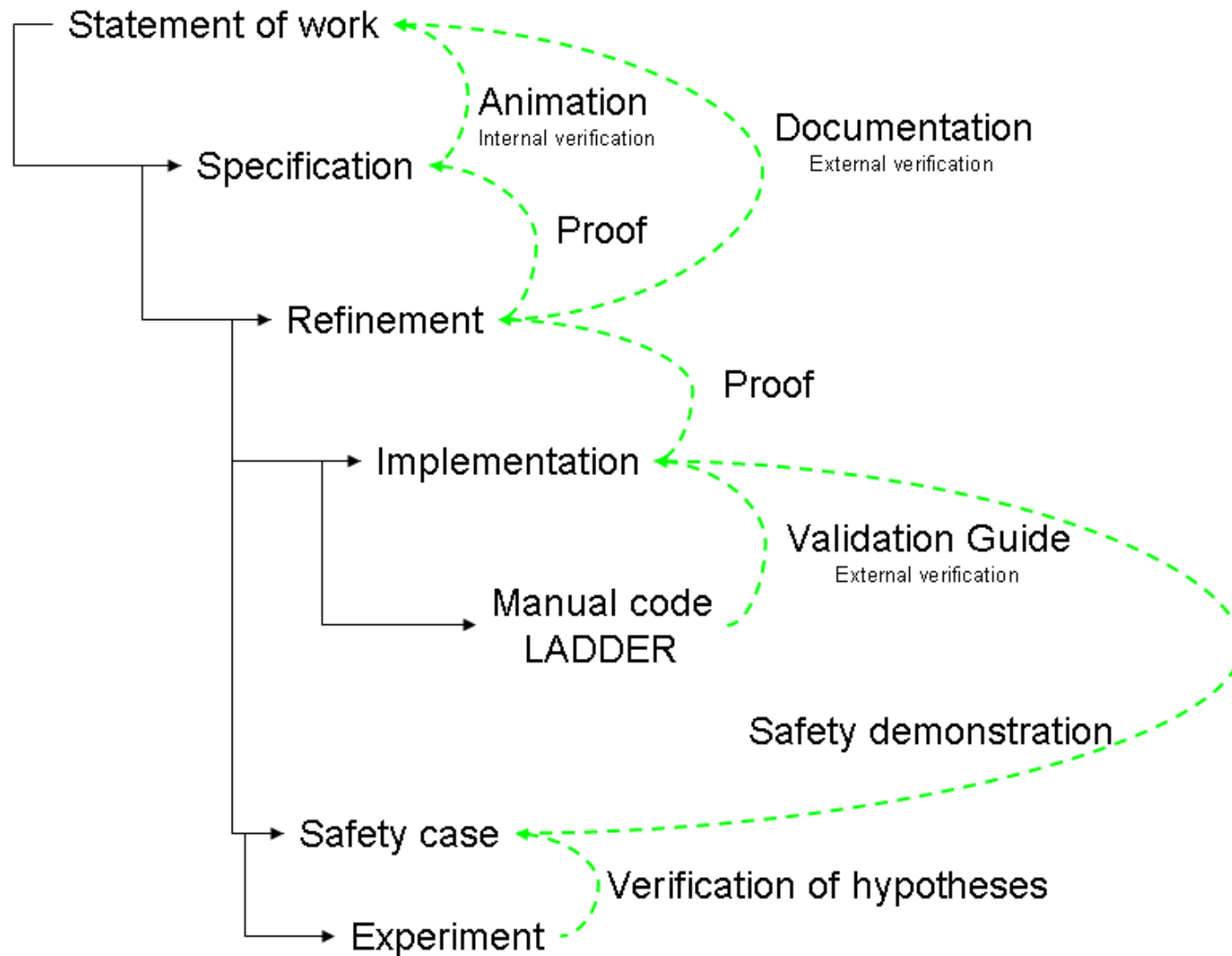


```
s1,s2 <-- evol =
VAR t1,t2 IN
    s1 <-- i1.fetat;
    i1.fevol;
    i2.fevol;
    t1 <-- i1.fetat;
    t2 <-- i2.fetat;
    IF t1=rouge & t2=rouge THEN
        IF s1=rouge THEN
            i1.fvert
        ELSE
            i2.fvert
        END
    END;

    s1<--i1.fetat;
    s2<--i2.fetat
END
```

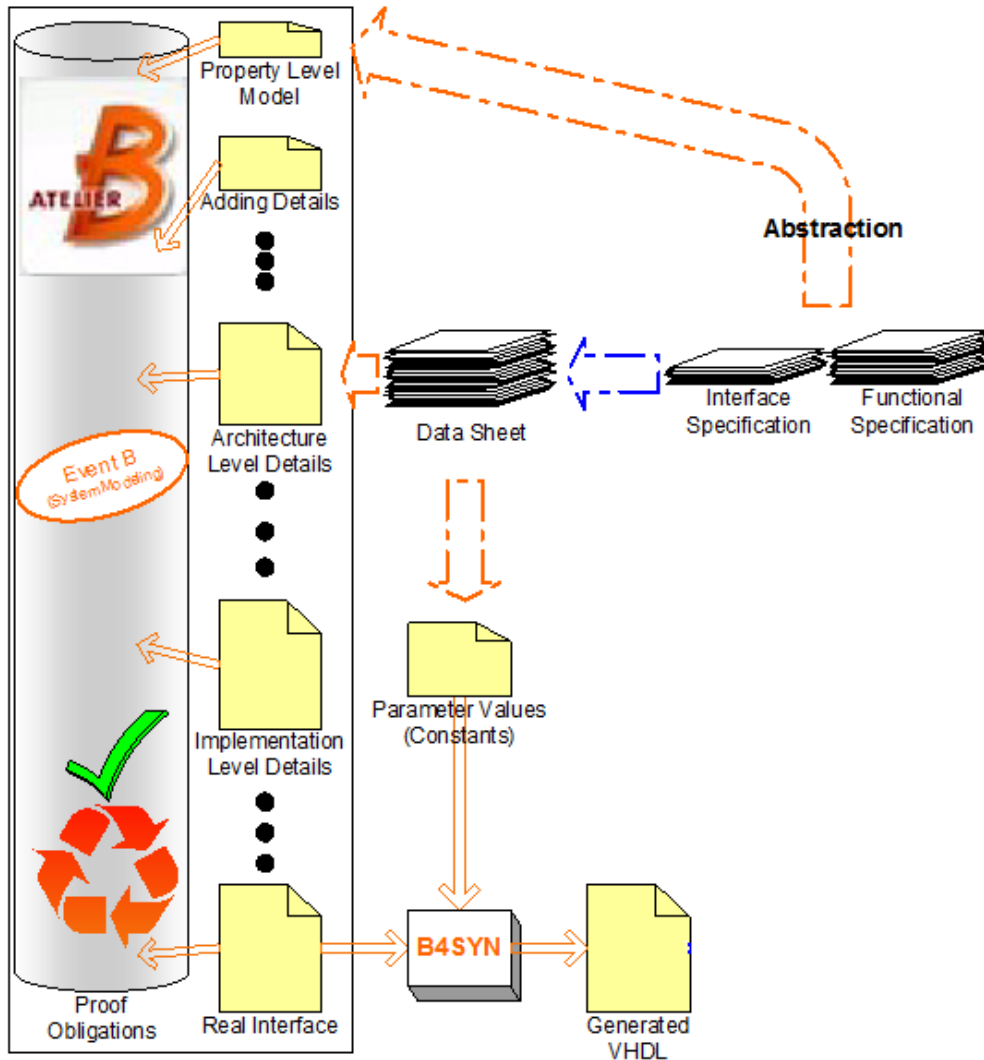


# SIL3-compliant process

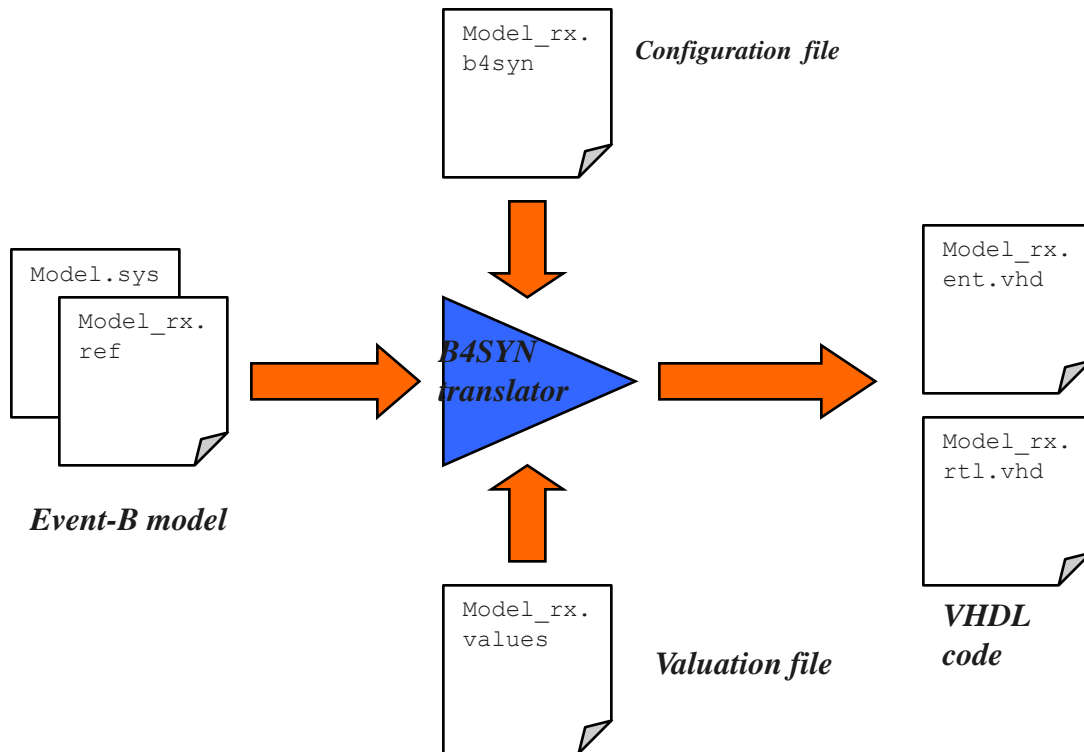


# VHDL Code Generation (B4SYN)

- Several translation schemas existing
- Not a 1-to-1 translation schema



# B4Syn Translation schema



# Feedback

- *Translator used with success on a microcircuit*
- Adequate generated VHDL models:
  - Size (5k gates)
  - Workload (even if different profiles)
  - Able to be tested with product testbenches
- *Translator probably lacking of generality*
  - *Connection to BlueSpec to make profit of a VHDL code generator*



# Generating Ada code from Event B model

- *Application of aggregation rules to transform a set of events into an algorithm*

```
SELECT P ∧ Q THEN R END  
[]  
SELECT P ∧ not Q THEN S END  
~>  
SELECT P THEN  
  IF Q THEN R ELSE S END  
END
```

*Condition:*

$P \wedge Q \Rightarrow [R] \text{ not } P$

$P \wedge Q \Rightarrow [S] \text{ not } P$

# Generating Ada code from Event B model

```
SELECT P THEN R END  
[]  
SELECT Q THEN S END  
~>  
SELECT P THEN R;S END
```

*Condition:*  
 $P \Rightarrow [R] Q$

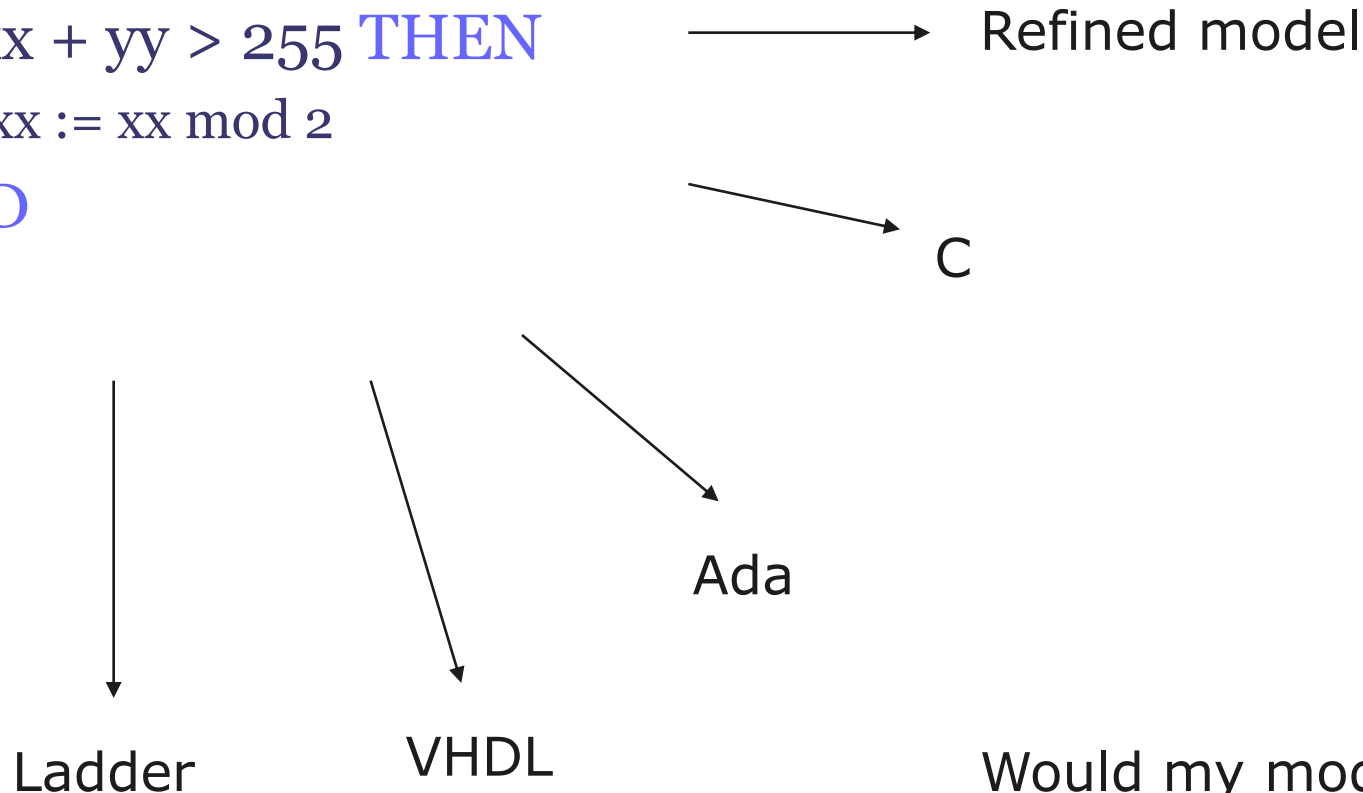
# Generating Ada code from Event B model

- *Obtained algorithm is not checkable with B*
- *Applied on part of the Ariane 5 flight software*
- *To obtain finally 80 lines of Ada, comparable to the handwritten ones*
- *Around 20 000 events would be required to replicate the branching structure of an Automatic Train Pilot*



# Semantics of B models

```
IF xx + yy > 255 THEN  
  xx := xx mod 2  
END
```



Would my modelling be the same for a different target formalism ?

# Conclusion

---

- Path to cyclic software well explored
- Different approaches for event based models
  - Ongoing researches:
    - “*Code Generation from Event-B - Using an Intermediate Specification Notation*” - Andy Edmunds
    - “Automatic Generation of C from Event-B”, Steve Wright
  - Main challenge: scalability





# Coffee break

## Useful links

|                        |  |
|------------------------|--|
| <b>Deploy project:</b> | <a href="http://www.deploy-project.eu">http://www.deploy-project.eu</a>  |
| <b>Rodin platform:</b> | <a href="http://www.event-b.org">http://www.event-b.org</a>  |
| <b>Atelier B:</b>      | <a href="http://www.atelierb.eu/index-en.php">http://www.atelierb.eu/index-en.php</a><br><a href="http://www.tools.clearsy.com/index.php5">http://www.tools.clearsy.com/index.php5</a> |
| <b>B method:</b>       | <a href="http://www.bmethod.com">http://www.bmethod.com</a>  |
| <b>Slides</b>          | <a href="http://bmethod.com/php/conference-grace-2010-en.php">http://bmethod.com/php/conference-grace-2010-en.php</a><br>available for download on March 18                            |

**GRACE International Symposium  
on Advanced Software Engineering 2010**

15-17 MARCH 2010 - TOKYO, JAPAN