AGENCE NATIONALE DE LA RECHERCHE ANR


# A combination of formal specification and testing for verification in Model-Based Development

**Philippe Baufreton** 1, Emmanuel Chailloux 2, Jean-Louis Dufour 1, Gregoire Henry 3, Pascal Manoury 3, Etienne Prun 4, Florian Thibord 3, Philippe Wang 2
1 Sagem Defense Sécurité - 2 LIP6/UPMC - 3 PPS/Paris Diderot - 4 ClearSY

**Theorem-Proving in Certification Workshop** Subtopic 5.3



CLEARSY SYSTEM ENGINEERING  LIP6  PPS  SAFRAN Sagem

# MOTIVATIONS

- **Scope: Smooth and progressive integration of (more) FM in industrial model-based design (MBD)**
- **Objective: Formal verification of the integration of reused components in a MBD approach**
  - Verification of the compatibility of the reused subsystem within a new environment.
- **Industrial FM approaches examples (extract)**
  - “Non-Functional” Properties
    - Not seen as an alternative to testing but used in many industrial companies (will not be discussed here).
  - Robustness
    - Analysis could be enhanced by automation through static analysis (Dassault Aviation @ SafeComp 2013)
  - “Functional” Properties
    - Correctness of code / formalized LLRs (Airbus unit proof)

# COMPOSITIONAL CERTIFICATION

## ➔ Conventional approach

- MBD for development, testing, review and analysis for verification

## ➔ Proposed approach

- Component based design in MBD through legacy components reuse
- Formal contract-based development
- Verification of system properties (invariants) at integration level

## ➔ Combination of formal methods and tests for verification

- Reconcile top-down and bottom-up (reuse) approaches
- Components are Simulink® / SCADE® legacy models
- Ensure that the composed development matches the functional system properties
- Use B as a verification method
- Cooperative use of static and dynamic methods

## ➔ Let's develop such an approach and try it in industrial settings

- 04/2011-08/2014





CLEARSY SYSTEM ENGINEERING

LIP6

SAFRAN Sagem

# WHY B?

## ➔ Main drivers

- Formal method with formal verification
- Multiple data abstraction levels
- Powerful composition mechanisms

## ➔ What is necessary to delevop?

- B is not known by software engineers in aeronautics
  - a Scade2Bgateway is required (still under development)
- An extension of the B formal language is considered by adding a TEST clause.
  - This clause is used to describe the test, initial values and expected values, with substitution. By proving refinement proof obligations (PO) for these substitutions with the values, initial and expected, it is possible to be sure that these tests will be correct.

# HOW TO INTEGRATE A FORMAL SPECIFICATION ACTIVITY IN A MBD ?

## ➔ Formal specification of the models

- In DO-178, data-flow models are formal LLRs; 'formal‘ because they have [or should have] a formal semantics,
- We need formal HLRs or 'contracts' to build B models.
- For example, for the operator y = sqrt(x),
  - the LLR (and the B implementation) describes 6 iterations of the Newton-Raphson algorithm, whereas
  - the HLR (and the B specification) will consist in the pre-condition x >= 0 and the post-condition f  abs(y*y - x) < 1e-12.
- The location of the contract could be in a separate file (which can already be a well-formed B machine)

# FROM SCADE TO B (PRINCIPLE)

- Scade 6 offers the simplest paradigm for components: the function.
- The preconditions of the functional components are requirements about the inputs whereas postconditions are requirements about their outputs.
- Those requirements can be stated in the Scade syntax as assertions:
  - an assume assertion for preconditions; a guarantee assertion for postconditions.
- Typing information is added of the inputs and outputs. The body of a Scade function is defined as a set of equations x = e where variables range over dataflow of values.

The challenge is to statically check that given a triple formed with a precondition, a function and a post-condition, then applying the function to inputs that satisfies the precondition produce an output that fulfill the post-condition. The aim of translating Scade functions to the formalism of the B method is to provide means to achieve the above challenge.

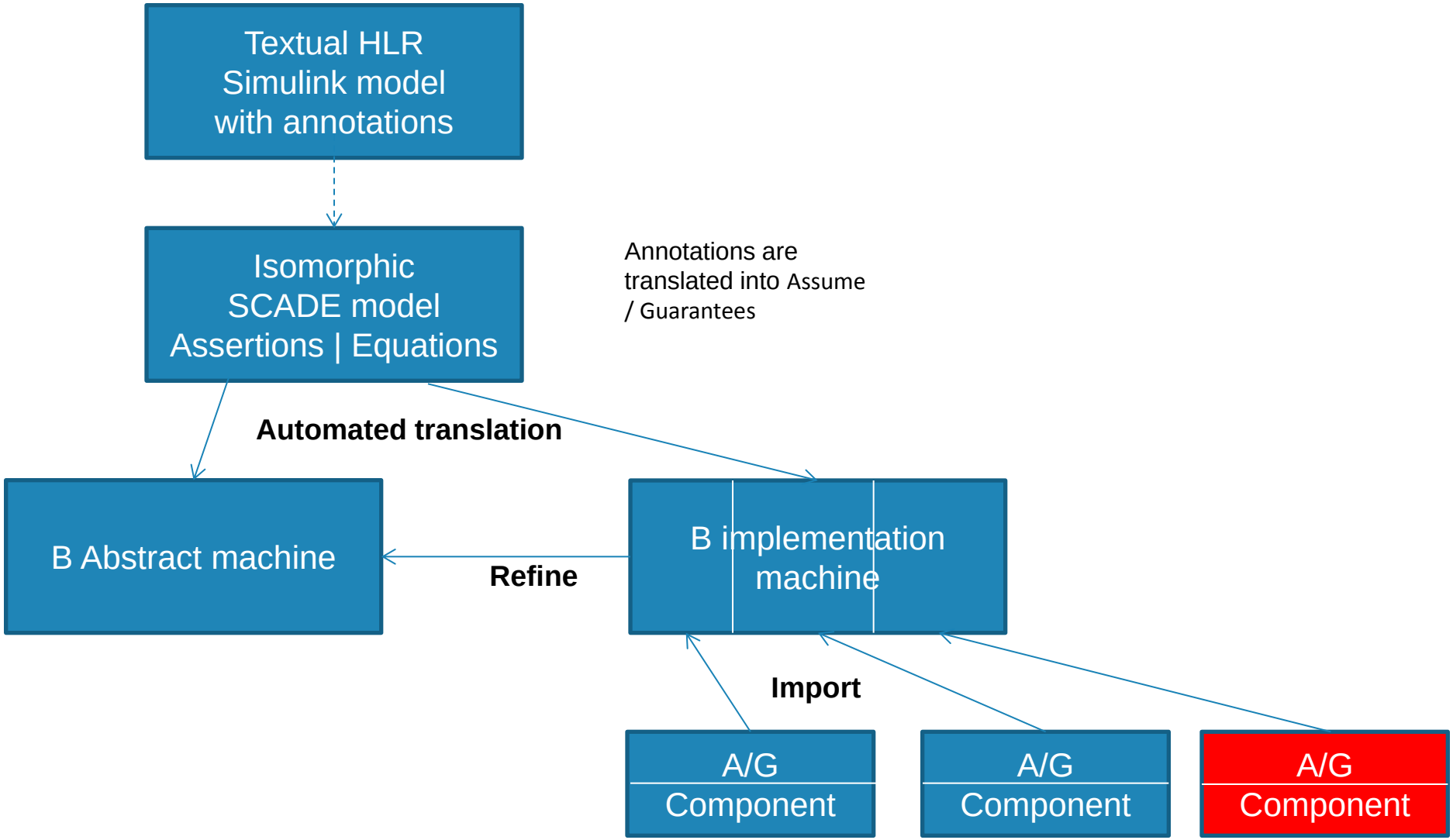
# B TRADITIONAL DEVELOPMENT METHOD

- In the B method, a software component is given by mean of a machine.
- A B machine may declare operations with inputs and outputs and an operation may affect the internal state of its machine.
- An operation may be defined as a high-level logical requirement using all the abstraction of predicate calculus and set theory.
- The B method paradigm is the use of refinement to derive step by step a final machine called implementation from the abstract setting of a B machine.
- The discipline of refinement guarantees that the requirements of the abstract machine are fulfilled by the implementation. Technically, at each refinement step a set of formulas called proof obligations are produced. The correctness of the implementation with regards to its abstract specification is guaranteed when all proof obligations are actually formally proved as theorems of the underlying theory of the B logical framework.

# OUR USE OF THE B METHOD FOR VERIFICATION

- **It is necessary to modify the components' reuse strategy and the composition proof in an MBD process**
  - The B method is no longer a development but a verification method
  - The source code is generated from Scade/Simulink (and no longer from the B0)
  - The B model is generated from the synchronous model in order to perform the composition proof.
  - An extension of the B formal language is considered by adding a TEST clause.
    - This clause is used to describe the test, initial values and expected values, with substitution.
  - By proving refinement PO for these substitutions with these values, initial and expected, it is possible to be sure that these tests will be correct.

# VERIFICATION PROCESS OUTLINE



# FROM SCADE TO B (DETAILS)

- A node, defined in Scade by mean of a set of equations, pre and post requirements and typing information is translated into two machines containing one operation corresponding to the node arity (same inputs, same outputs).
  - The first machine is an abstract one that simply require that the operation must satisfy that given inputs that satisfy the precondition, it will produce outputs that satisfy the post-condition (including typing).
  - The second machine is an implementation which rewrites the set of equations as a sequence of single assignments (simple substitutions in the B language). The assignments are ordered following the causality relation involved by the static analysis of the Scade equations (only correct Scade programs for which static analysis of causality succeeded are considered) .

# FORMAL VERIFICATION

➔ Two kinds are required.

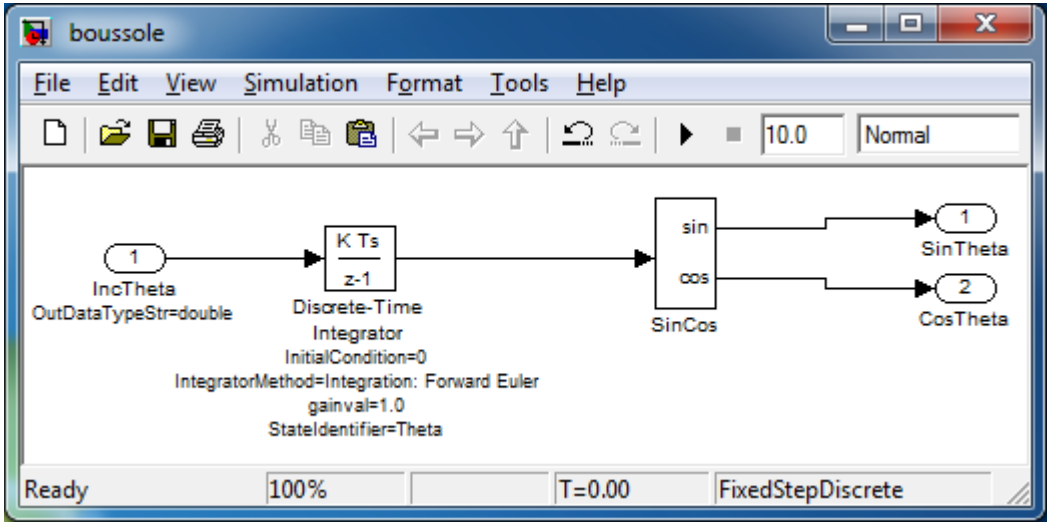
- Checking the internal coherence of a machine (or an implementation), eventual invariant properties.
  - verify that the initialization establishes the invariant and the operations do not violate it.
- Checking the validity of the refinement relation between a machine and its claimed implementation.
  - verify that the outputs of the operation computed by the implementation are the one specified by the abstract operation of the refined machine.

# Gyroscopic compass System example

## ➔ Gyroscopic compass System:

- In : IncTheta = angle increment (value from a gyroscope)
- Out : CosTheta, SinTheta (in order to dipslay an arrow on a wind rose)
- State : Theta = cap (North direction)
- Requirement : initialize Theta = 0, then update by IncTheta integration and give SinCos

- Design:
  - 1) Compute Theta by integration of IncTheta (init value equal to 0)
  - 2) Start function SinCos



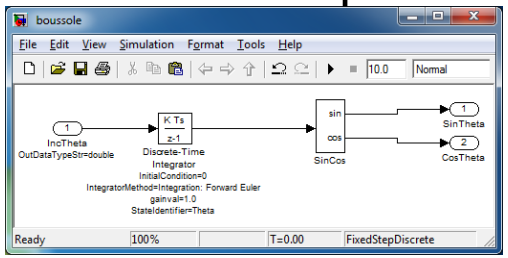
# The integration of reused components: …. error !!!

- **Integrator :**
  - In : Inc
  - Out : Integr
  - Requirement: Integr += Inc, and 0 at initialization step

- **SinCos :**
  - In : X
  - Out : SinX et CosX
  - Requirement: pre = X ∈ [- π .. π] , post = precision to $10^{-n}$

compass.mch (=type) ← HLR compass

compass.imp ←




Integrator.mch (= type)

SinCos.mch (pre-cond)

Integrator is under-specified

**X ∈ [- π .. π] ?**
**NO !!!!**

# How to solve the integration error?

## ➔ Solution 1: (impact on the code)

- add a ‘2π modulo’ at the integrator output➔ ‘integrator_modulo2π’ …
- … for which the post-condition changes from ‘integr : float64b’ to ‘integr : [-π.. π] …
- … in order to prove the SinCos precondition and to reuse the SinCos component as it is …
- … the new component ‘integrator_modulo2π’ has an ‘hybrid’ specification based on requirements, test-cases and a post-condition

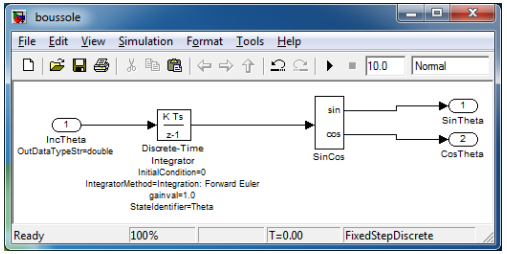
# New Integrator component

- **Integrator :**
  - In : Inc
  - Out : Integr
  - Requirement: Integr += Inc, and 0 at initialization step

- **SinCos :**
  - In : X
  - Out : SinX et CosX
  - Requirement: pre = X ∈ [- π .. π] , post = precision to $10^{-n}$

compass.mch (=type) ← HLR compass

compass.imp ←




integrator_modulo2π.mch
(= type)

SinCos.mch
(pre-cond)

New integrator type

X ∈ [- π .. π] ?
YES !!!!

# How to solve the integration error?

## ➔ Solution 2: (impact on the design)

- add a ‘2π modulo’ at the design level as a glue code → ‘integrator code is unchanged’ …SinCos is unchanged …

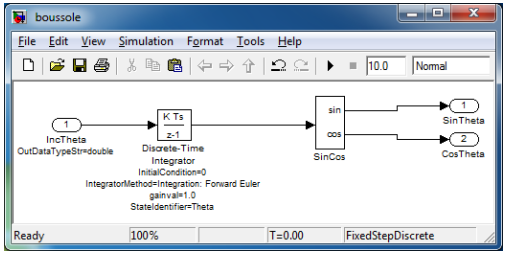
# Design change

- **Integrator :**
  - In : Inc
  - Out : Integr
  - Requirement: Integr += Inc, and 0 at initialization step

- **SinCos :**
  - In : X
  - Out : SinX et CosX
  - Requirement: pre = X ∈ [- π .. π] , post = precision to $10^{-n}$

compass.mch (=typage) ← HLR compass

compass.imp ←




2π modulo

Integrator.mch
(= type)

SinCos.mch
(pre-cond)

Integrator is unchanged

**X ∈ [- π .. π] ?**
**YES !!!!**

# FORMAL METHODS AND TESTS COVERAGE

➔ **Structural coverage defined so as to provide evidence on:**

- Completeness of specification
- Completeness of test cases
- Absence of unintended functions

➔ **An alternative method must:**

- Demonstrate achievement of these objectives

➔ **Source code level verification**

- When performing a verification activity at source level instead of binary level, an analysis of the object code generated by the compiler is performed, including the effects of the compiler options on the object code, to ensure that the compiler preserves in the object code the property proved on the upper software artifacts.

# COVERAGE ANALYSIS WHEN USING FM

➔**If only formal methods are used to verify requirements for a software component, then coverage consists of confirming 5 things:**

- **Requirements-based coverage**
  - at high-level requirements and low-level requirements
- **Complete coverage of each requirement**
- **Completeness of the set of requirements**
- **Detection of unintended dataflow relationships**
- **Detection of dead and deactivated code**

➔**If formal methods are used in combination with test for a component, then coverage must comply with DO-178C coverage objectives**

# HOW DO WE ENSURE COVERAGE?

➔**The HLRs are expressed formally (B model of the system) so the verification of the components' integration is done formally**

➔**The correctness of the components' composition with regards to BS (.mch) is ensured by the verification of the proof obligations with TP**

➔**Detection of Unintended Dataflow Relationships (**missing requirements)

# CONCLUSION

## ➔ Advantages of the approach

- Stronger confidence in component-based software design
- Better for some areas not well covered so far (components' integration)
- Alleviates activities performed so far
- Benefits for iterations during the project development and maintenance

## ➔ Not the solution to all problems

## ➔ Complementarity / Substitution

- Progressive introduction of FM in industrial practice
- Systematic use (goal) thus limited to invariant verification (TP could be complemented by model-checking for temporal properties)
- Limited by confidence built on long history
- Metrics adapted to the "classical" approach

## ➔ Towards "assurance-case" based approach





CLEARSY SYSTEM ENGINEERING

LIP6

SAFRAN Sagem

# PERSPECTIVES ON FM & TEST

## ➔ Enhancement of contract-based development and reuse

- More functional properties, at unit and integration level
- Cooperative use of static and dynamic verification
- DO-178 tool qualification

## ➔ FM .vs. Test ?

- FM & Test
- A cost-driven continuum of behavioural coverage: human < dynamic < "static"