ROADMAP

TELECOM ParisTech

Foundations of a New Software Engineering Method for Real-time Systems

Abstract: The design of a fault-tolerant distributed, real-time,embedded system with safety-critical concerns requires the **use of formal languages**. Here, we present the <u>foundations of a new software engineering method for real-time systems</u> that enables the integration of semi-formal and formal notations. This new software engineering method is mostly based upon the ”COntinuuM” co-modeling methodology that we have used to integrate architecture models of real-time systems and a model-driven development process that both includes **model transformation** and **code generation** strategies

Kitakyushu-City, 13/11/2008

Main issues, state-of-the-art

-The Analysis and Design of DRES systems are performed without any standardized software design method

-Actual standards as DO-178B, ARP4754 and legacy guidelines

-MDA not suitable for embedded systems-, a UML Profile for MARTE, xUML, a UML Profile for AADL, lots of model-checking techniques…

-Proof Based System Engineering (PBSE), proof techniques PVS/Why TLA+/TLC/+CAL → cannot be applied on the whole lifecycle

-Methods in use : SART (non iterative) and HRT-HOOD (only one type of diagram) / MeMVaTEx (in progress, no formal notations) : AAA + ACCORD/UML

The major objective is to obtain a continuous integration of different languages that define a system at different abstraction levels, from end to end of the software lifecycle with an automation between the different phases.

each phase has its own notations

MDA for embedded systems ? : Physical environment has an impact on the system behavior → software et runtime frameworks do not have to be separated

DRES need a co-modeling methodology

- What are DRES , where are they? (transport, entertainment, at home..)
  - **Distributed→** Possible multi-tasks or/and multi-processors
  - **Real-time →** Outputs delivered on time
  - **Embedded →** Part of a larger product without direct user/system interfaces
  - **Systems →** Consists of both hardware and software parts

- DRES are more and more complex systems because of :
  - **Of their software part**
  - **They have to embed more and more functionalities**
  - **They are hybrid systems (continuous and discrete time, hw & sw)**

- Architecture Models of DRES in the design process
  - **From requirements to tests… the role of these models in DRE systems (need for formal languages, proof and model-checking methods)**
  - **Architecture styles are a Key notion in software engineering**

Distributed Real-time and Embedded Systems (DRES) modeling (1)

■ Real-time constraints

- In general, it is not so easy to take into account temporal constraints with Object-oriented modeling techniques
  - In particular, deadlines, WCET and tasks synchronization.

- One way to solve this issue is to attach them to messages
  - In order to manage tasks scheduling, we schedule messages processing
    - With objects deadlines
    - Priority inheritance
    - Dependency management

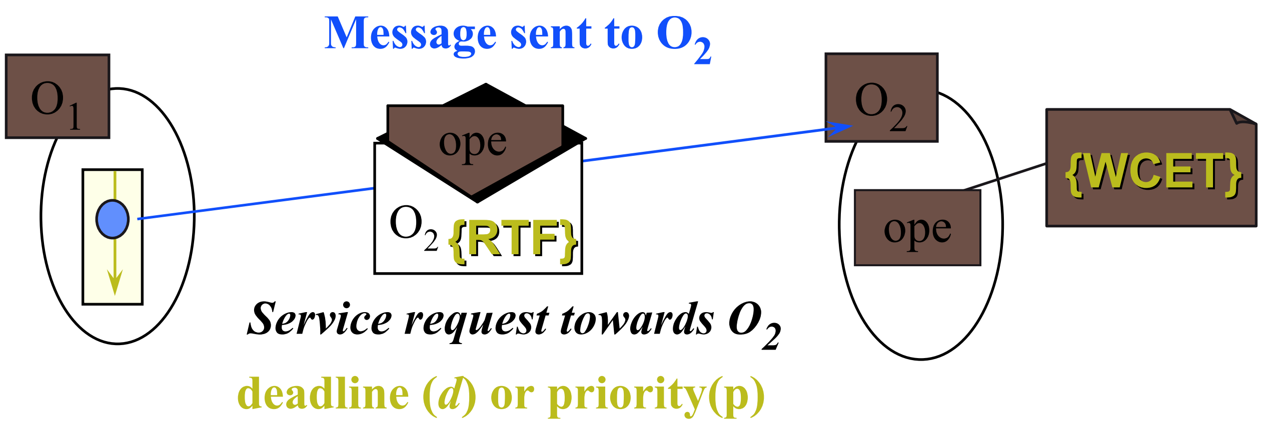
- There is a strong need for modeling real-time objects and its mechanisms → MARTE profile

Distributed Real-time and Embedded Systems (DRES) modeling (1)

Temporal characteristics are attached to functions : deadline, offset, period, WCET

In MARTE , we can model Date or Duration with **TimedValueSpecification**
→Peraldi-Frati and Sorel have used clocks constraints and complex duration expressions with multiple timeBases (in order to perform scheduling analysis)

Duration < MIN (**30 ticks on crkClk** , **MaxSamples * T on MyIdealClock**)



Distributed Real-time and Embedded Systems (DRES) modeling (2)

- **Concurrent, reactive and distributed systems can be modeled, refined and checked with temporal logic**
  - Among several temporal logics LTL (linear), TLA or CTL(more powerful than LTL)…with different time diagrams (linear or tree diagrams), different time granularity, we have chosen TLA+ for its high expressivity and assertional verification
  - TLA+ is very different : it is a temporal logic of actions : it has action and temporal operators
  - TLA+ has 2 levels of syntaxes : action formulas (which represent states) and the transitions system
- In TLA+ a system is defined as a set of traces

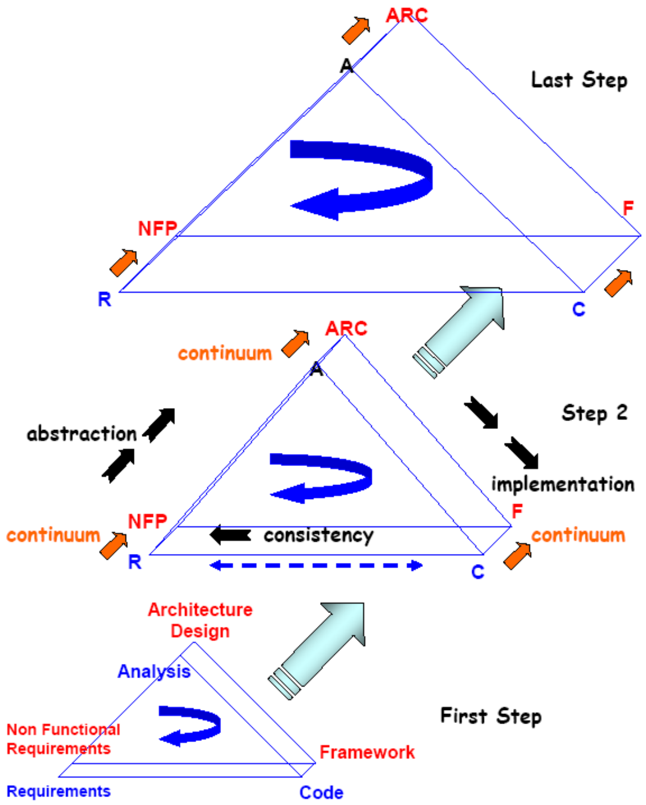
OO Methods

- Booch and OMT **(Object Modeling Technique), functional model difficult to use → UML**
- ROOM **(Real-time OO Modeling) → UML-RT→ UML 2.1 → MARTE**
- HRT- HOOD **→ HRT-UML (user base limited to ESA), much oriented towards Ada, but lack of inheritance, polymorphism concepts**
- KOBRA **→ Product lines and Komponent-based, aligned with industry standards, MDA, recursive, based on UML, suited for hierarchical systems (no way to describe actions happening among subsystems or components themselves)**
- Fusion **(comes from OMT) suffers from the same problem as OMT , based on a strict waterfall sequence of activities**

Not from the scratch : three generations of design methods for real-time systems

- **(1)** Structured design methods for the development of real-time software: **SA, SADT, JSD, CORE, DARTS, CODARTS** (Ada-based design approaches for real time), **MASCOT, OCTOPUS, HOOD**
- (2) **SART**
  - **cannot be used for prototyping and still does not have an iterative lifecycle**
- (2) **HRT-HOOD**
  - **object-based structured design method**
  - Has only one main diagrammatic type for modeling
- (3) *AAA + ACCORD/UML (MeMVaTex)*

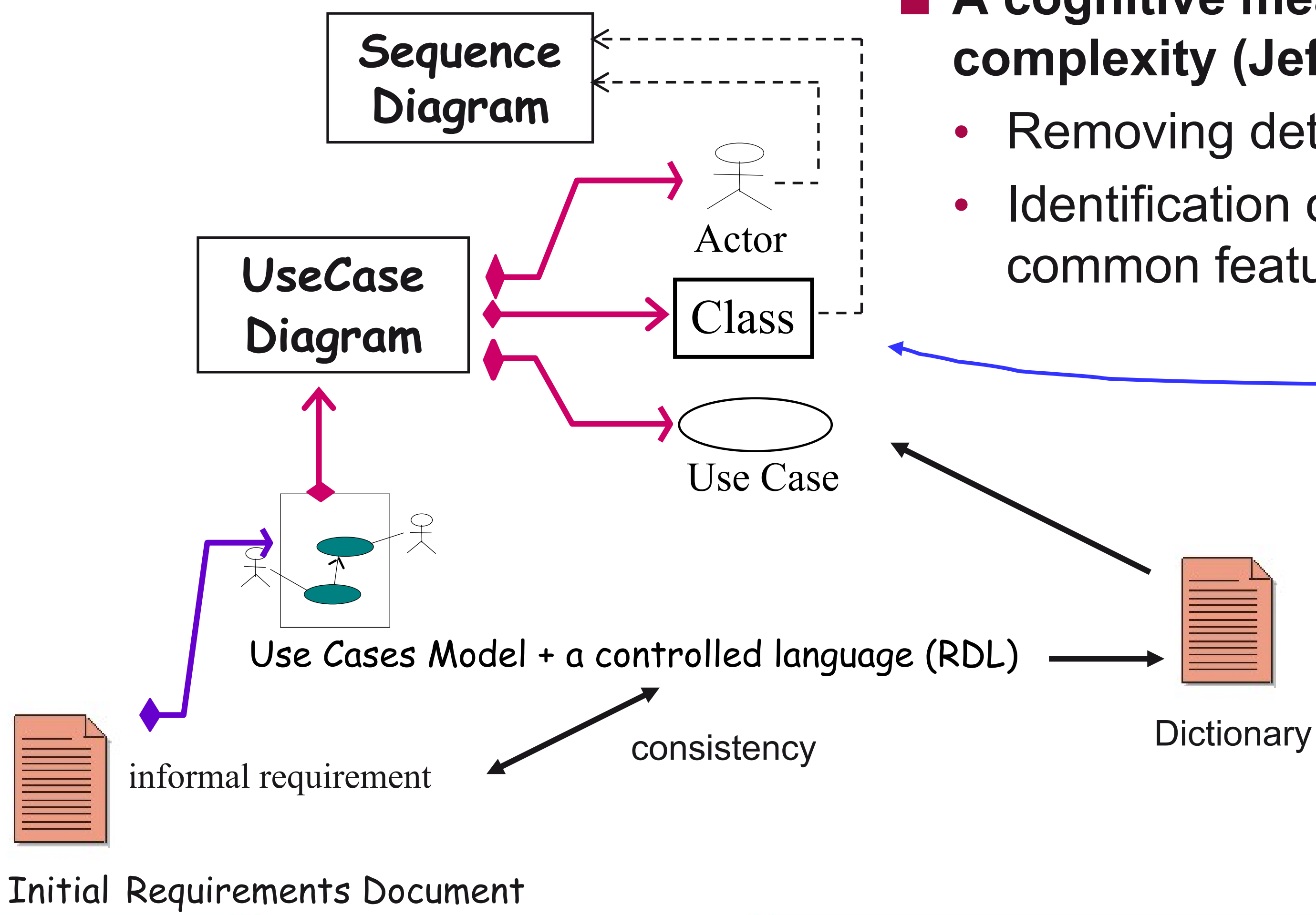
Approach (full Model-driven) : The keys of the method, and a seamless development life cycle



A well defined lifecycle

- An iterative lifecycle, prototyping/ feedback based
- The development process ensure continuity between phases
- Consistent notations set , semi-formal and formal notations
- Notations composition
- Proof and model-checking techniques
- Design of sporadic and cyclic activities
- Notations for real-time (clocks)
- Binding (MARTE allocation) of software components onto hardware components (AADL)
- Decomposition of architecture models (packages)
- Modes
- Integration and proof of NF requirements
- Integration of scheduling policies within the design process
- Ease of use
- Tools including WCET and schedulability analysis

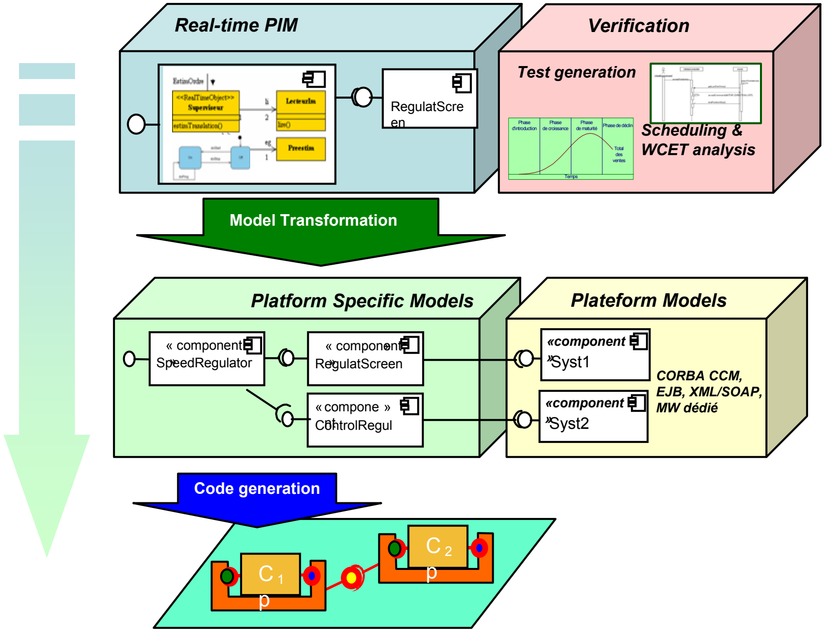
The Abstraction process (on ascending phase)



- **A cognitive means to deal with complexity (Jeff Kramer)**
  - Removing detail
  - Identification of **generalizations** or common features

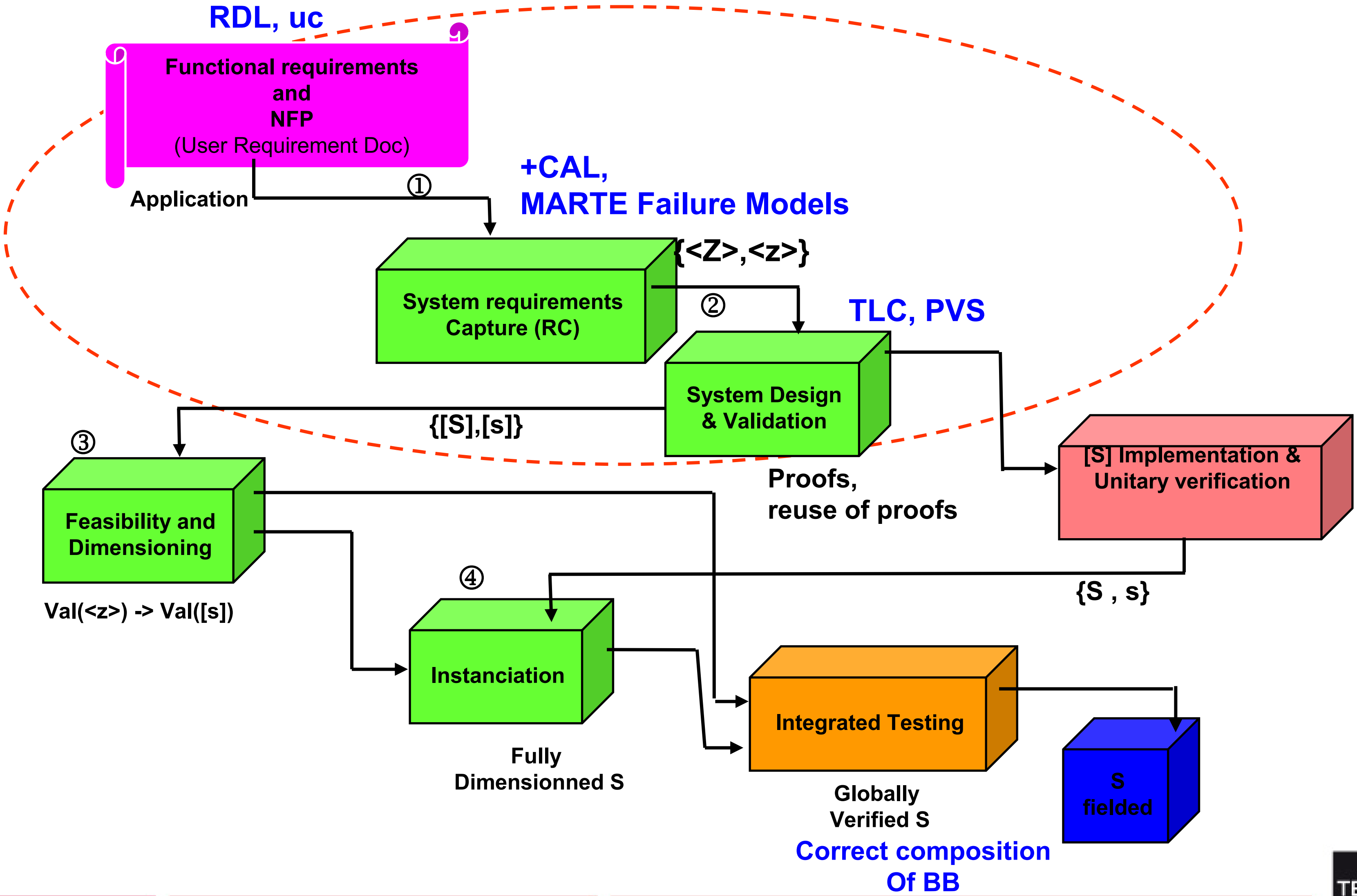
The Instanciation phases

- MDD approach (ACCORD/UML) on descending phase of lifecycle



- **From MARTE to AADL**
  - Mapping MARTE → AADL
  - ATL Transformations
- **ATL : coding the transformations rules inside modules**
- **Subset of xUML (fUML) + Action semantics (concrete syntax)**
- **+CAL algorithm language extensions**
- **ANTLR Ada code generation techniques**

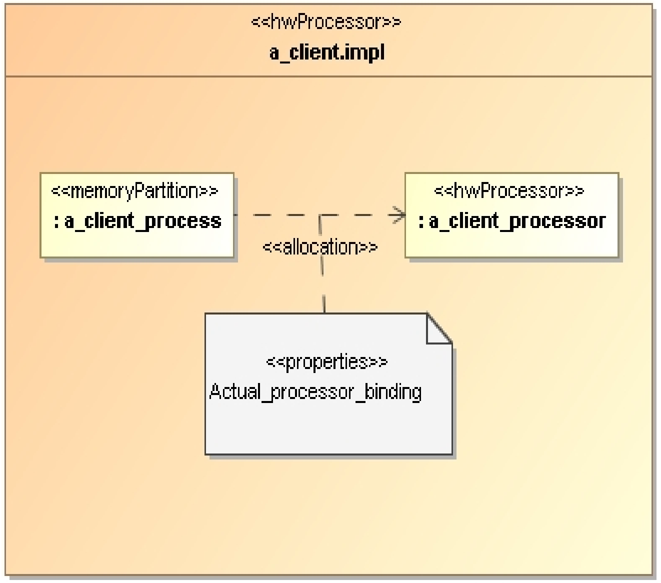
Requirements Capture phase in PBSE



TELECOM ParisTech

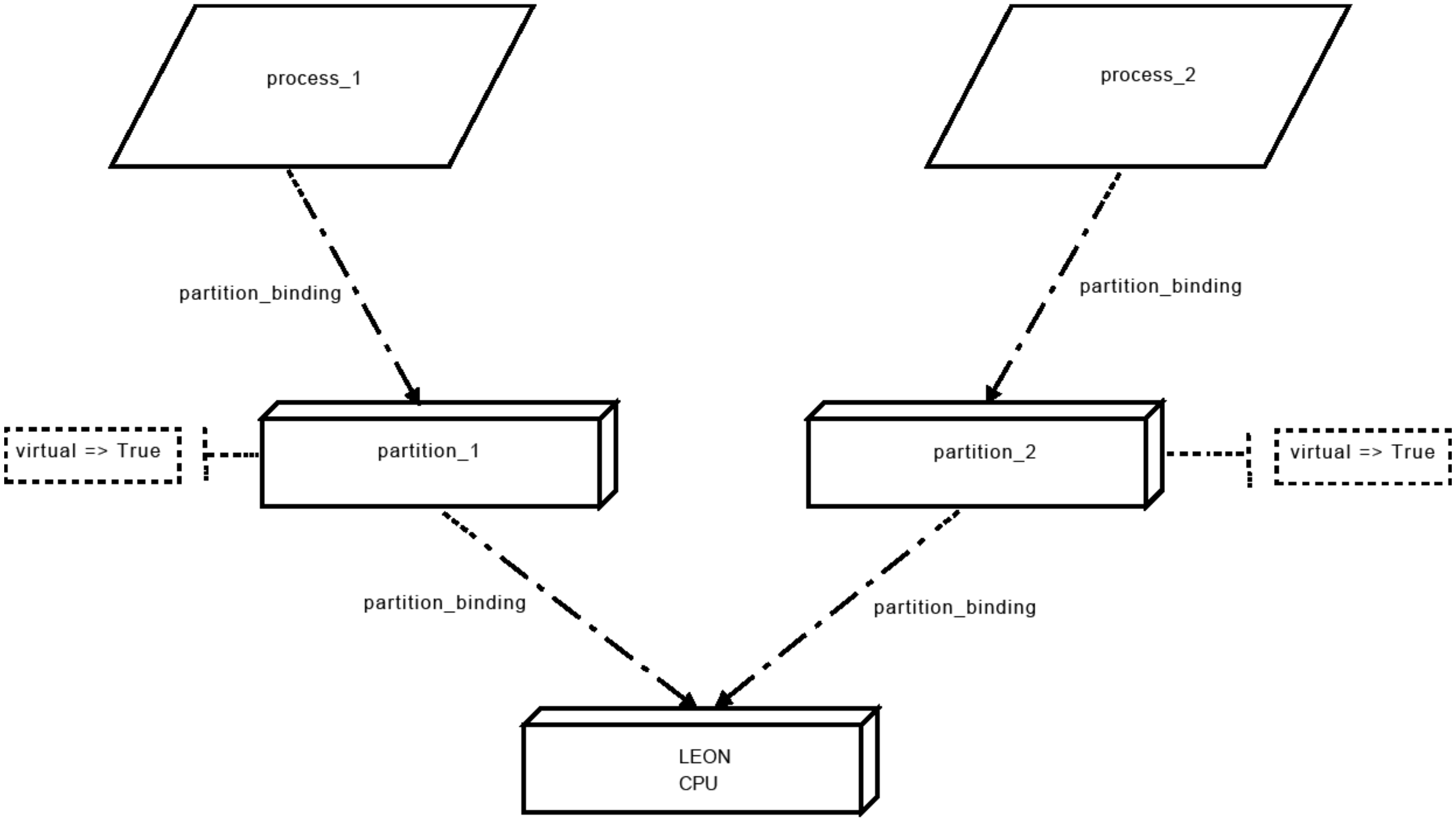
Using MARTE for a complete system specification

- **The binding is an <<allocation>> stereotype (SysML wording)**
  - The comment associated to this dependency contains the binding properties
  - Both components are bound by the <<allocation>> stereotyped UML dependency



Ex of Madeleine Faugère, Thales

Using AADL


process_1
process_2
partition_binding
partition_binding
virtual => True
partition_1
partition_2
virtual => True
partition_binding
partition_binding
LEON
CPU

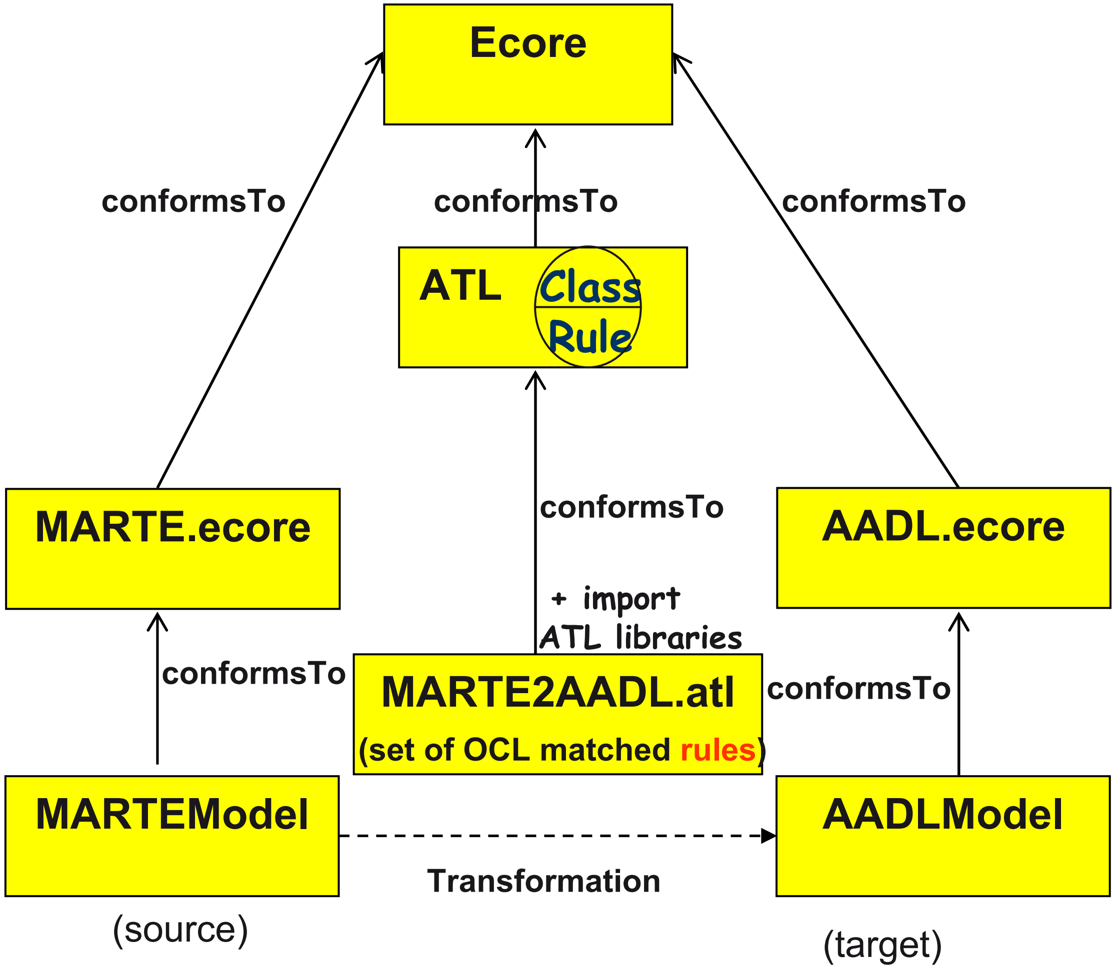
MARTE subset to AADL

… Corresponds to …

- <<swSchedulableRessource>> → Thread
- → Thread group
- → Shared Data…
- → Subprogram…
- → Processor
- → Memory
- → Device…
- → System
- → Port…
- → Mode
- …
- → System Binding
- …

rules established from
source model to target :
Stereotype MARTE to AADL element
Ex:
<<swSchedulableRessource>> to
ThreadClassifier

OCL :
do {thismodule.dsl(t, Thread);
}



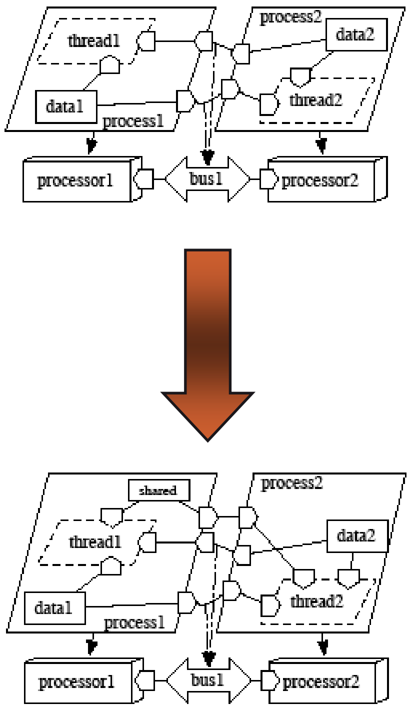
TELECOM ParisTech

Algorithms / Architecture , what's the right level of abstraction?

- An iterative approach : algorithms optimization into design models (AADL)→ Analysis models refinement

- To avoid the dependence of a limited set of proved properties, we did not choose to build **a proved algorithms library**
  - -The analysis of a given algorithm **through a set of parameters** coming from the initial requirements
  - -A **fixed number of parameters** / **a suitable algorithm**
    - To prove
    - To guaranty that it is consistent with global architectural non functional properties

- **A gradual formal expressivity** : +CAL / TLA+
  - \- Proofs at the Requirements level
  - \- Proofs at the analysis level
  - \- Proofs at the design level
  - \- Generation of implementation language

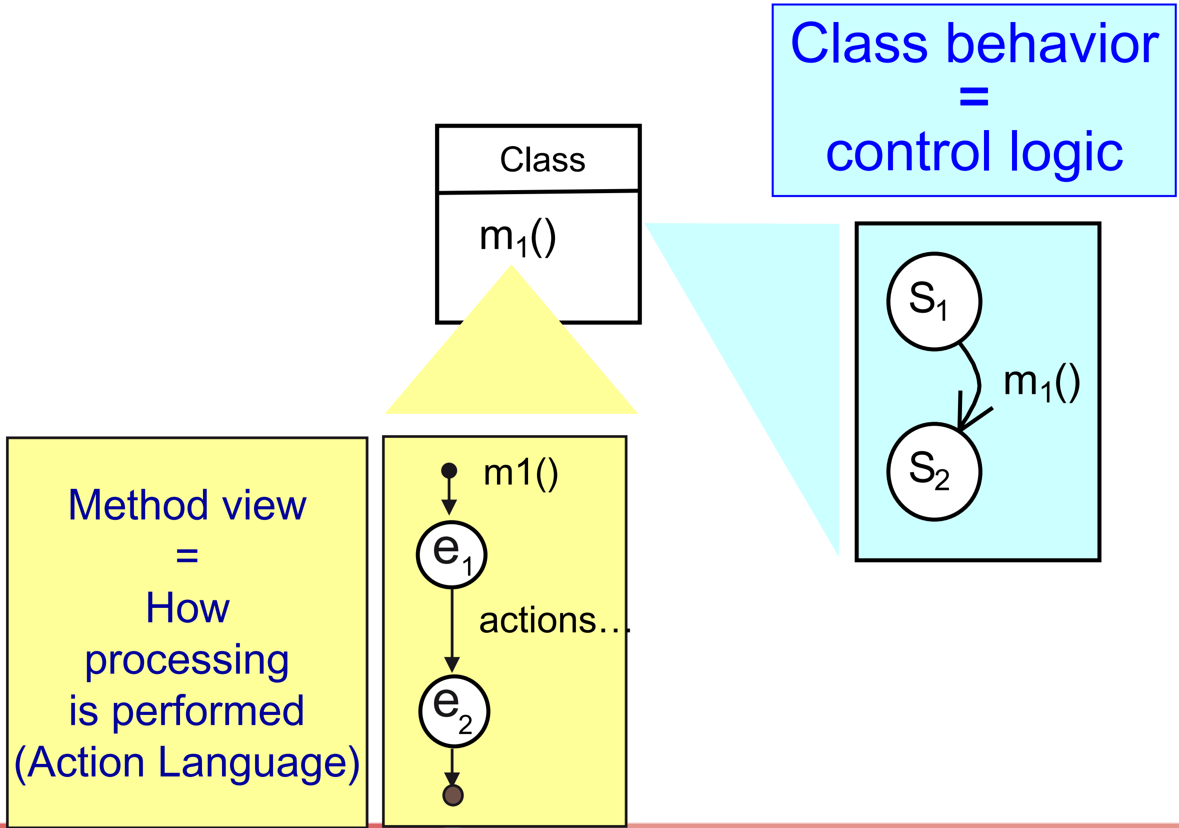


The lock policy of the shared data is centralized at the level of one process (process1)

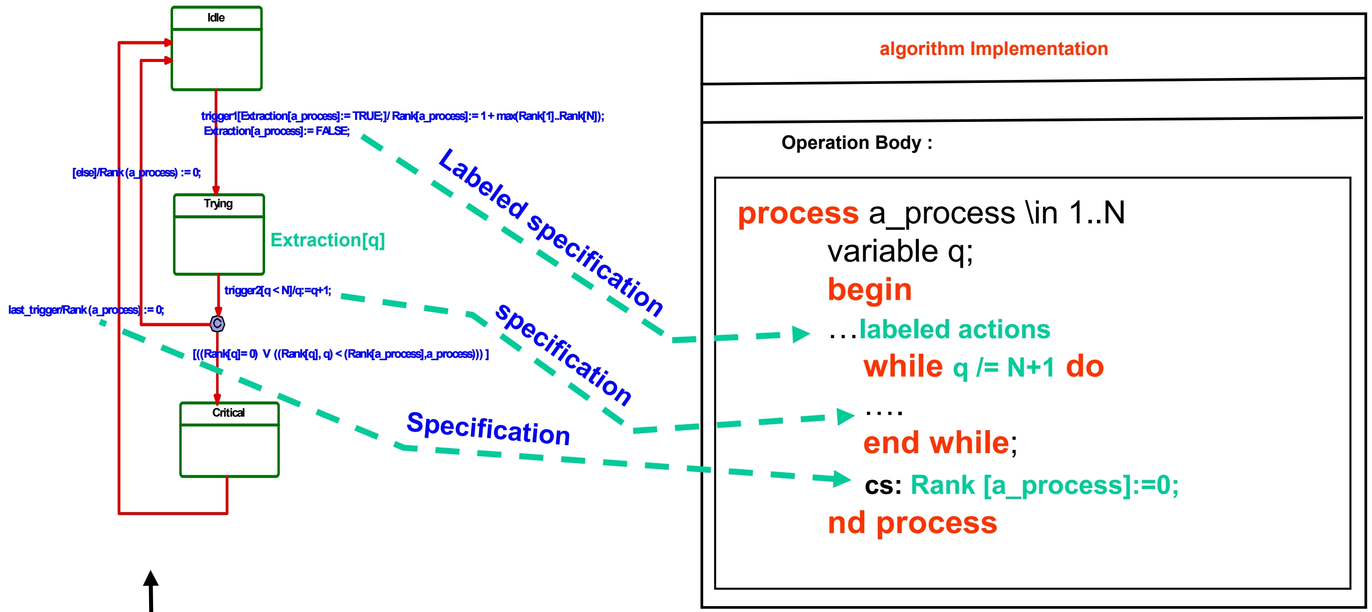
separation of concern to improve behaviour modelling **in ACCORD/UML (CEA)**

2 granularity levels

Control logic & Algorithms aspects



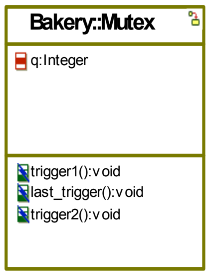
+CAL as an action language : executable and checked models



Lamport Bakery

← sequence of steps (labels)

State machine in +CAL



(bench on Rhapsody in Ada)

Ada generation : procedures (PCALToAda translator)

Invariance checking in TLA+ (with TLC)

Certification of the produced code

TELECOM ParisTech

Conclusions and Future works

- **There is no single notation that would cover the whole software development lifecycle**
  - UML does not have a suitable control flow diagram,
  - AADL does not allow requirements capture, behavior formalisms are separated in an annex, annexes are a poor extension mechanism
  - no formal language can be directly used with the end-user
- **How** to manage a matrix of modeling notations, in a methodical way that allows consistency, traceability, and a continuum between each development phase ?