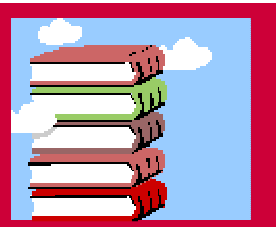
Formal Methods Integration in Software Engineering

Isabelle Perseil, Laurent Pautet

Workshop UML&FM'2009 / ICFEM 2009
Rio de Janeiro, Brazil, December, the 8th, 2009

Contents

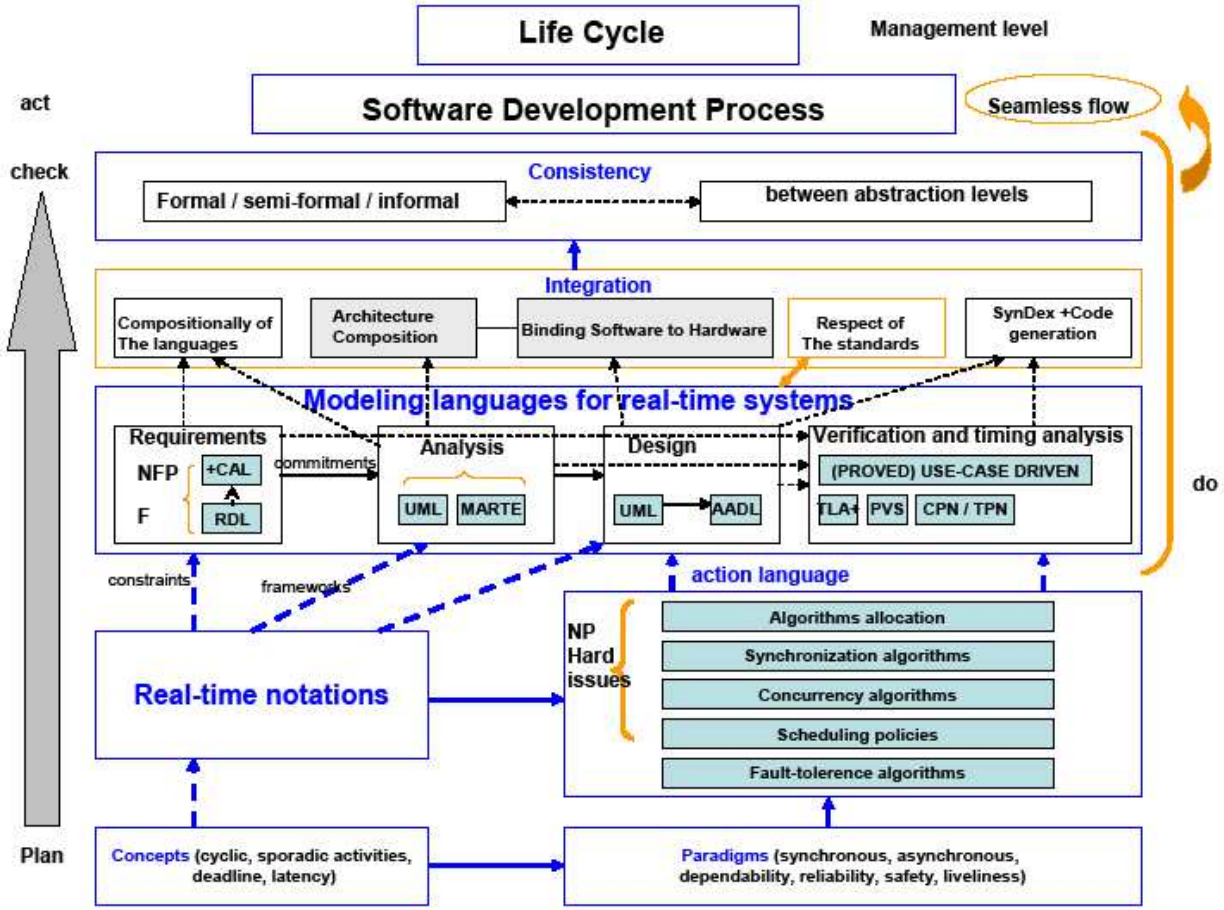


- **Objectives of the C-Method (a new software engineering method)**
- **Issues:** from non formal to semi-formal, from semi-formal to formal, from formal to semi-formal…
- **General approach**
  - Define the abstraction levels
  - Introduce intermediate languages to allow the management of the overall process (seamless process, understandable)
  - Make a round-trip between formal and semi-formal notations
- **Conclusions**

Objectives of the C-Method

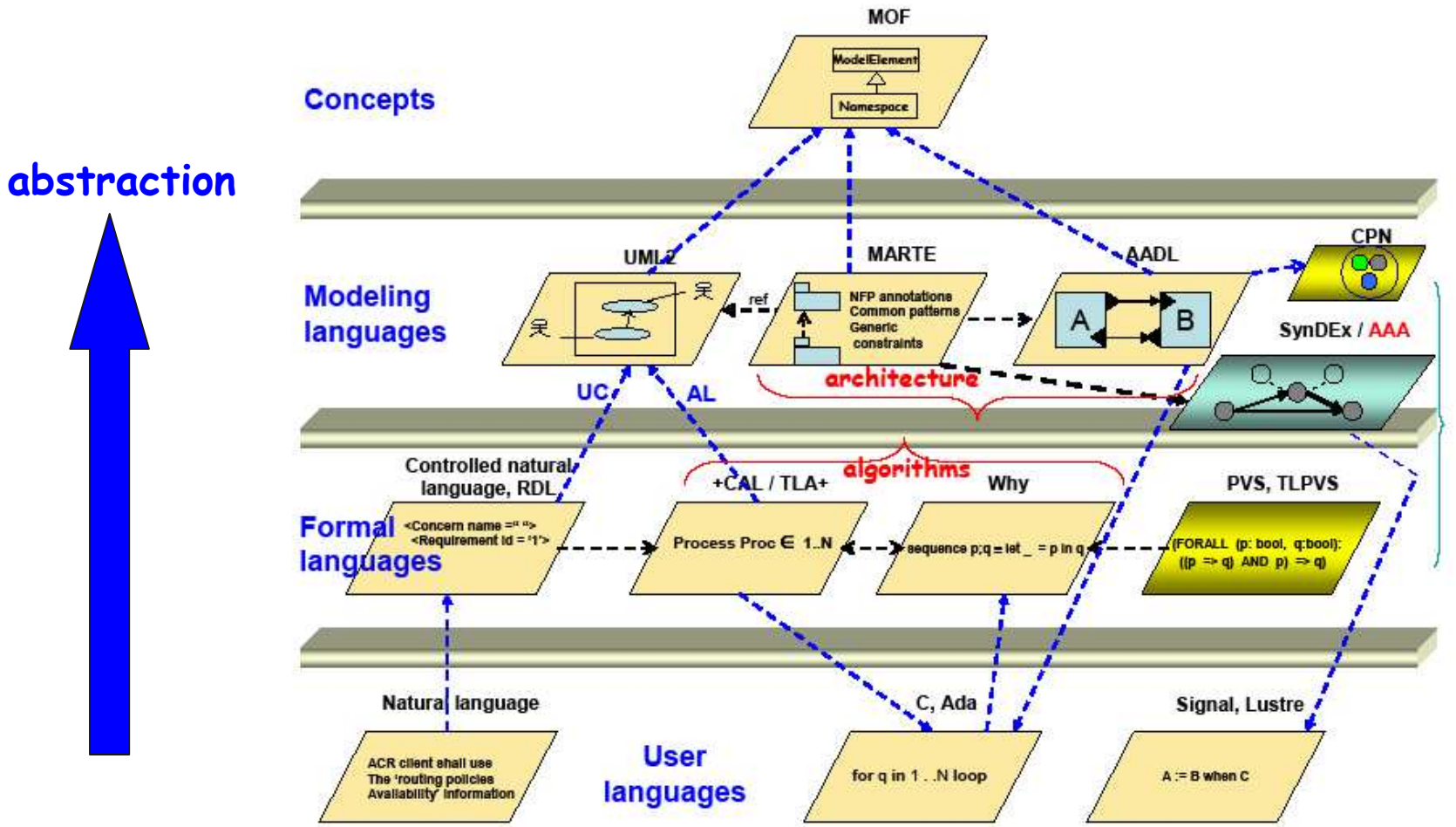
- 1. A well-defined software development life-cycle with a seamless flow
- 2. A software development process adapted for DRES
- 3. A set of (standards) modeling notations (for each purpose, requirements capture, architecture design, etc..)
- 4. A compositionality of these different notations to ensure they fit together
- 5. The availability of real-time notations as to describe concurrency, synchronization, etc
- 6. An early binding of software components to hardware components
- 7. A possible decomposition of the software architecture that is amenable to processor allocation, schedulability and timing analysis
- 8. **The integration of non-functional requirements**
- 9. The integration of scheduling paradigms within the design process
- 10. Ease of use of the method, CASE tool support

Starting with the elements of a metamethod



The elements identification ensure the exhaustiveness of all the necessary steps

Issues: dealing with heterogeneous languages and their abstraction levels

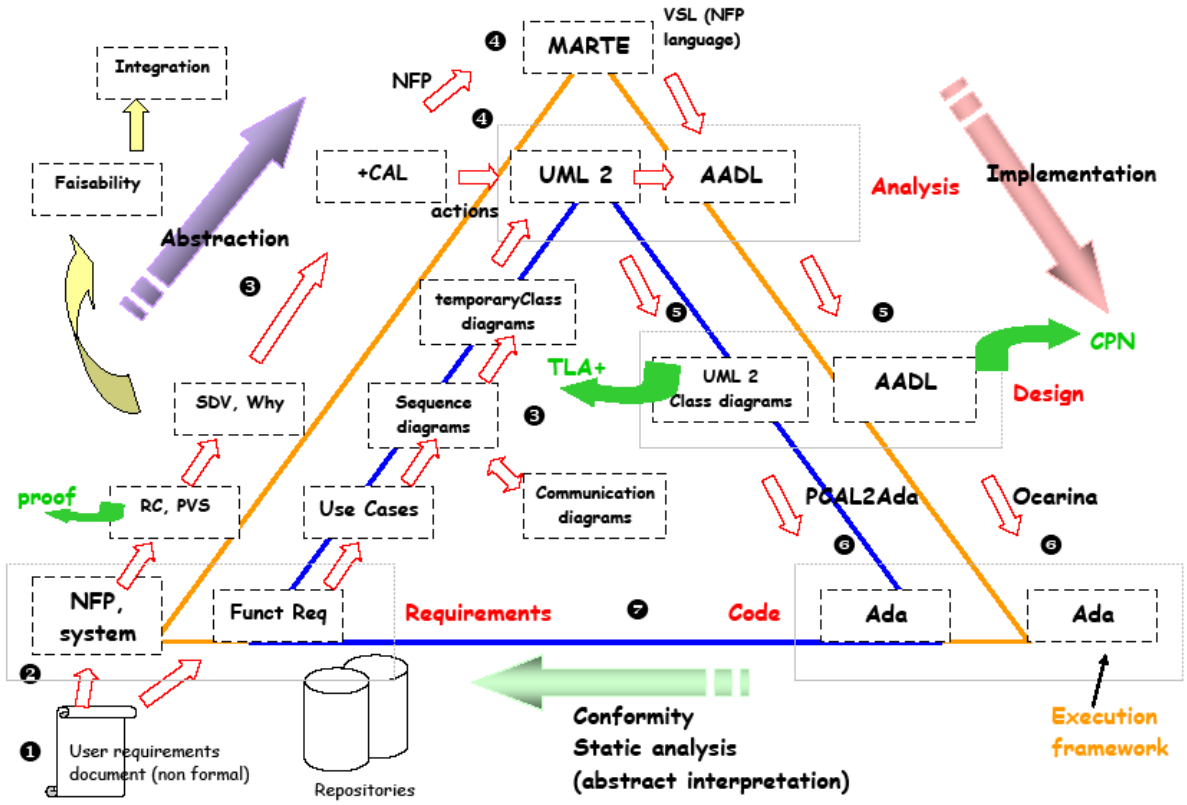


C-Method and its lifecycle guided by the abstraction levels

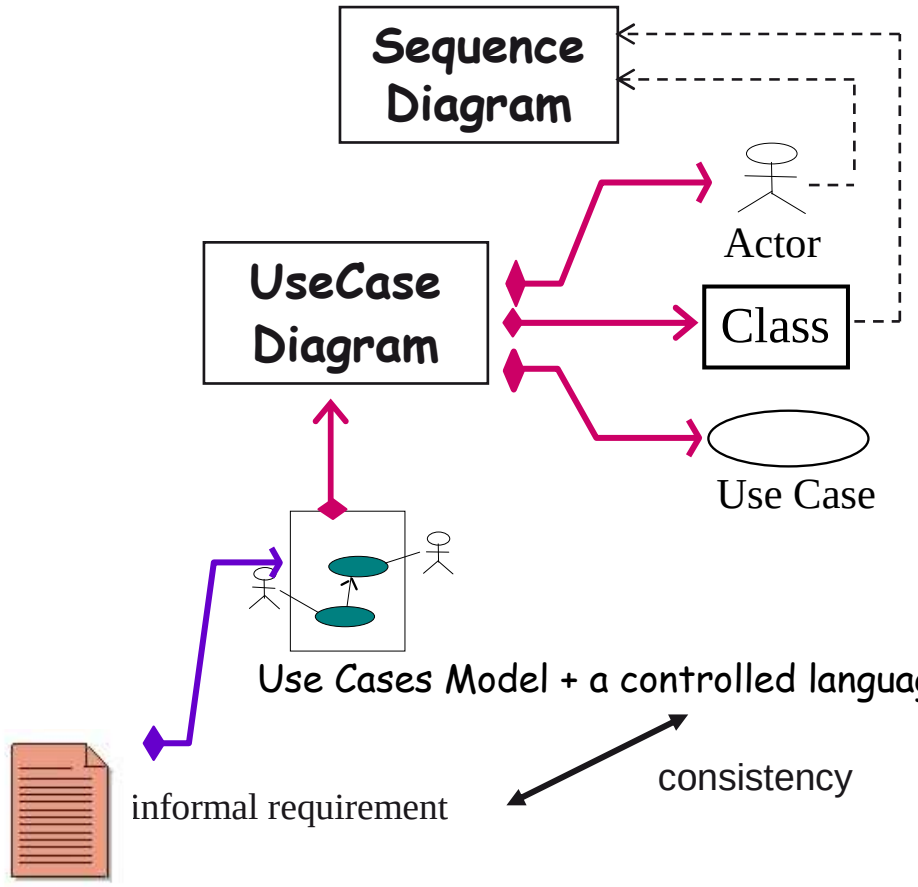
Non Functional

Functional

Proofs / Verification



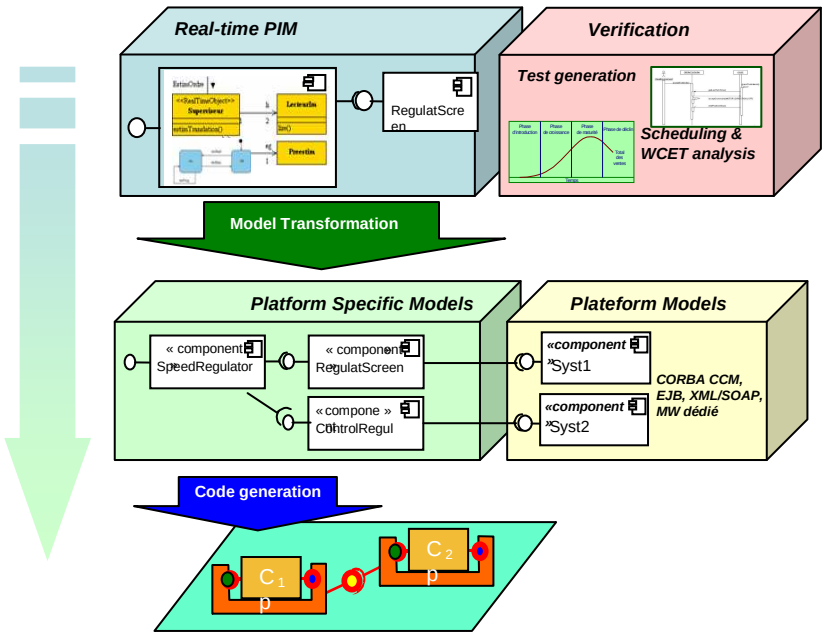
Detailing the abstraction phase (on the functional part)



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

Realizing the Implementation phase

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

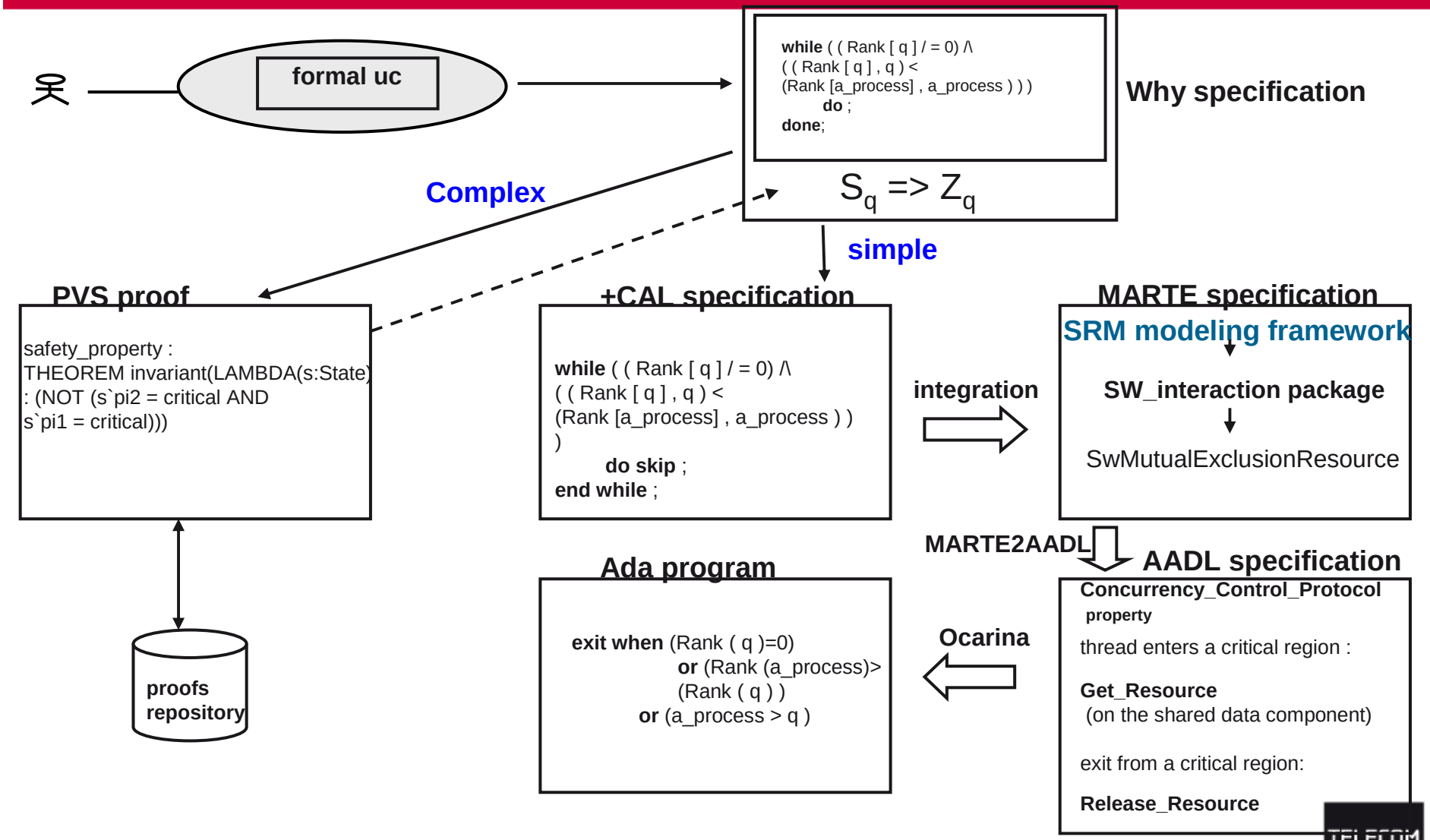


Concrete techniques:

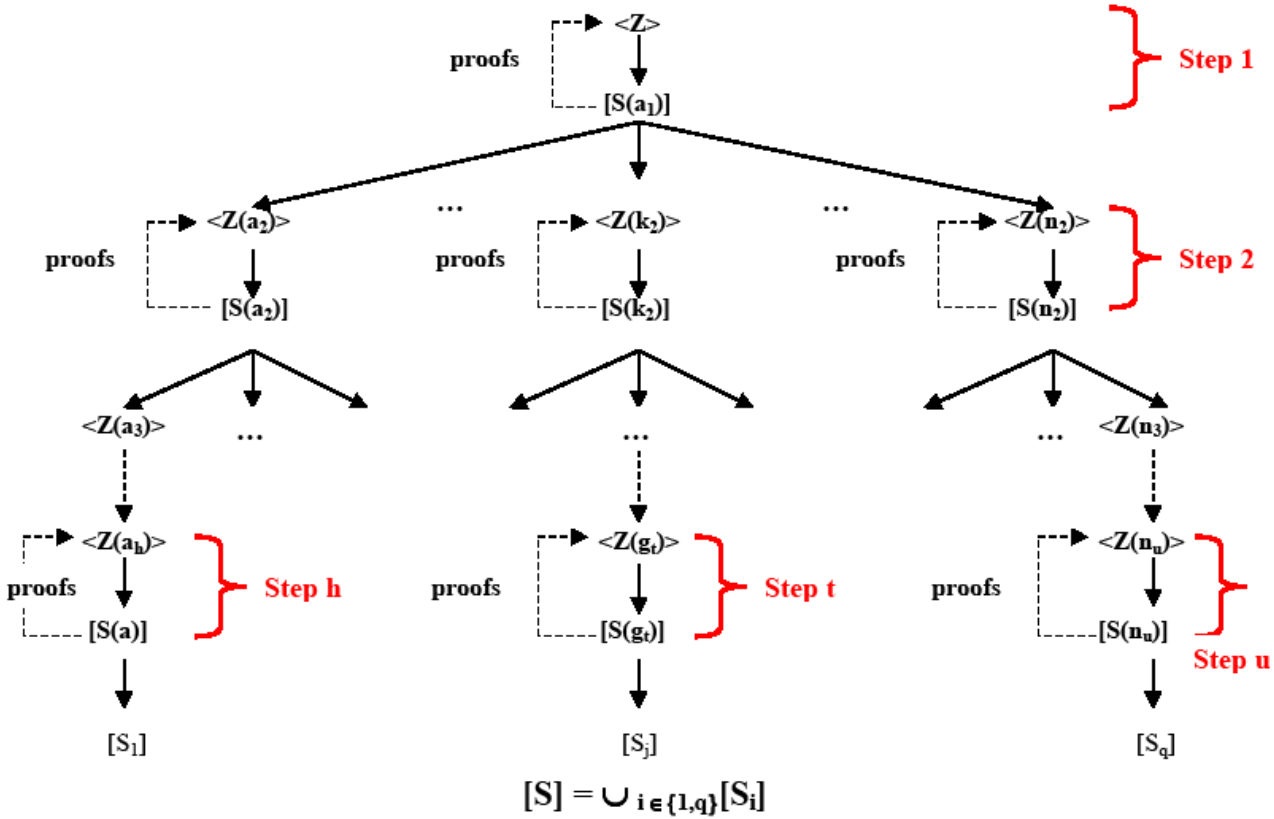
- **From MARTE to AADL**
  - Mapping MARTE → AADL
  - ATL Transformations
- *ATL : coding the transformations rules inside modules*

- **Subset of xUML (fUML) + Action semantics (concrete syntax)**
- **+CAL algorithm language**
- *ANTLR Ada code generation techniques*

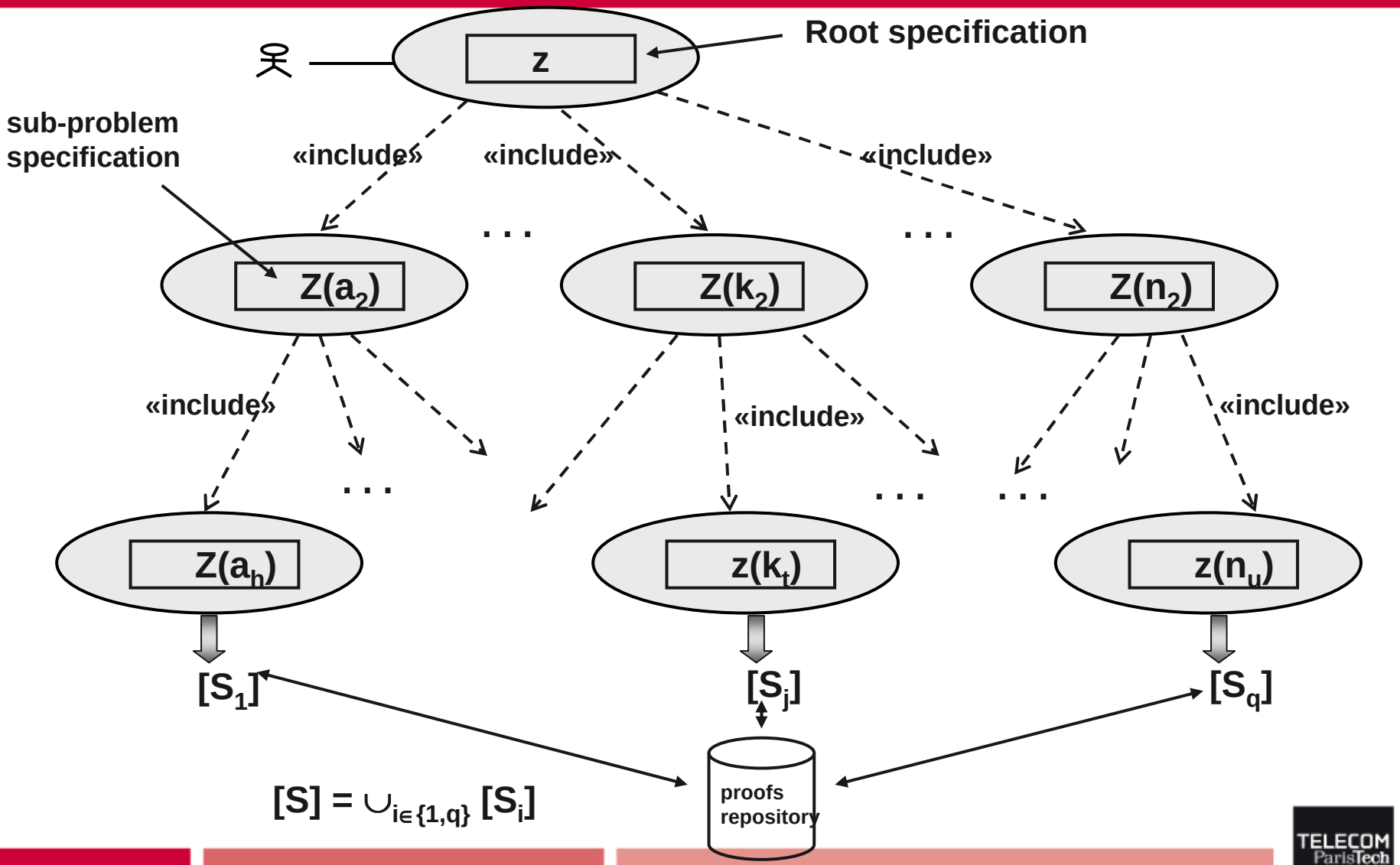
Enforcing the formal methods integration: a formal use-case driven method



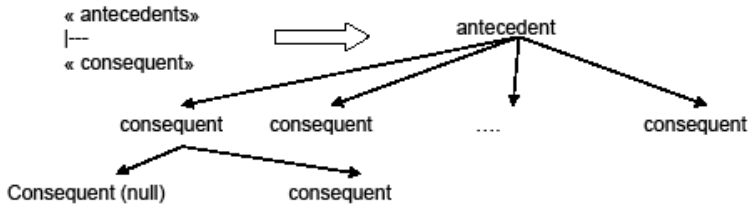
PBSE approach: the proof tree



Proof-based use cases: a sub-objectives technique



Proof-based use cases: a sub-objectives technique



$$Z \mid\!- S$$

$$Z_1 \mid\!- S_1$$

$$Z_{11} \mid\!- S_{11} \qquad Z_{12} \mid\!- S_{12} \qquad Z_{13} \mid\!- S_{13}$$

Approach: the sequent logic eases the sub-proofs representation

Conclusions and future works

- **The C-Method is based upon the use of three standards:**
  - +CAL/TAL+ for formal specification
  - MARTE at the Analysis level
  - AADL at the design level
- **Model transformation and code generation → seamless process**
  - Formal methods are part of the transformation
  - Understandable by the average engineer → reuse
- **Other integration techniques: Hybridization (as used for modeling discrete and continuous time with the Chi language)**