ROADMAP

An Emerging Need for a New Software Engineering Method

Isabelle Perseil, Laurent Pautet

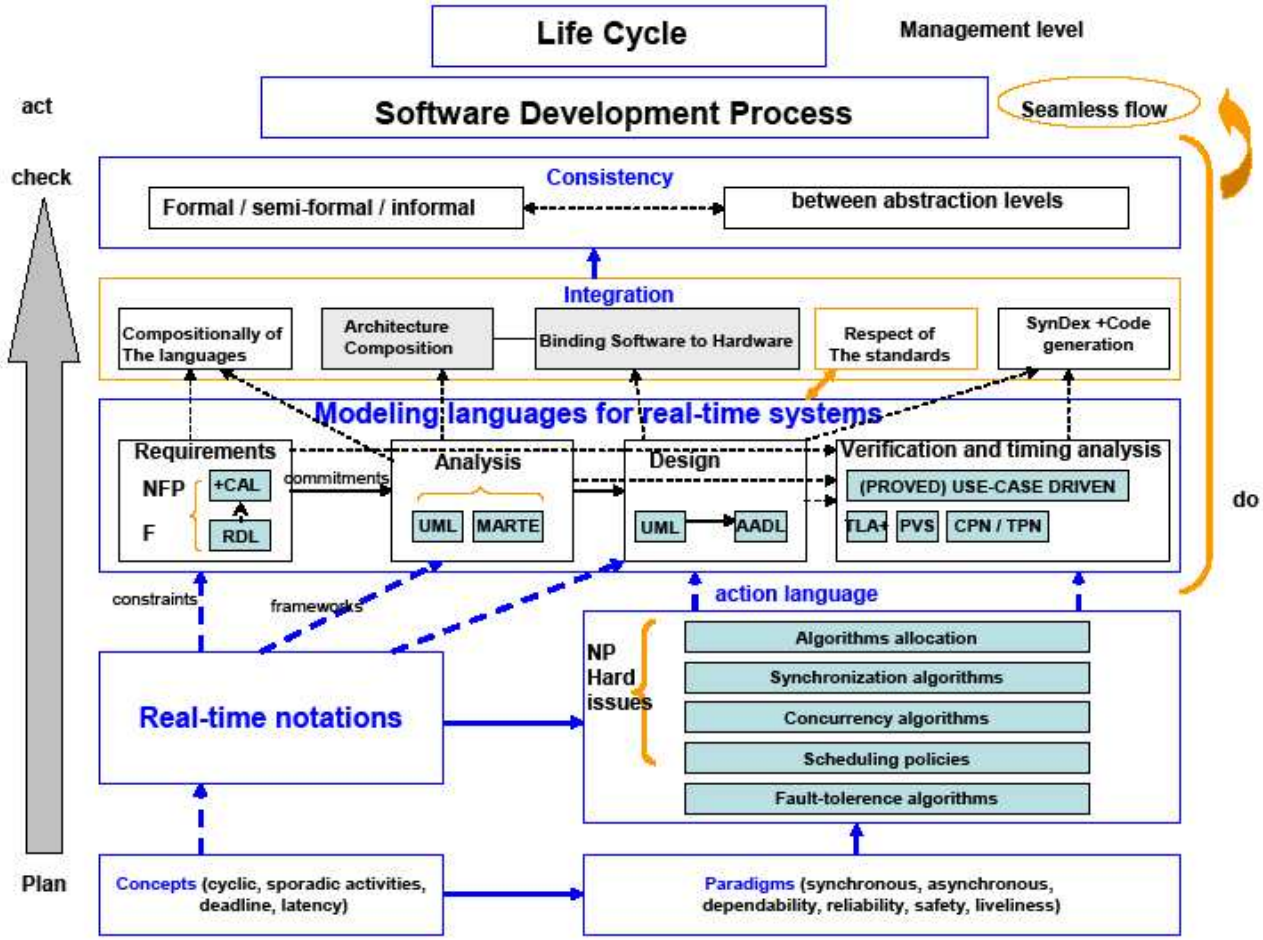
Potsdam, June 2nd, 2009

Abstract

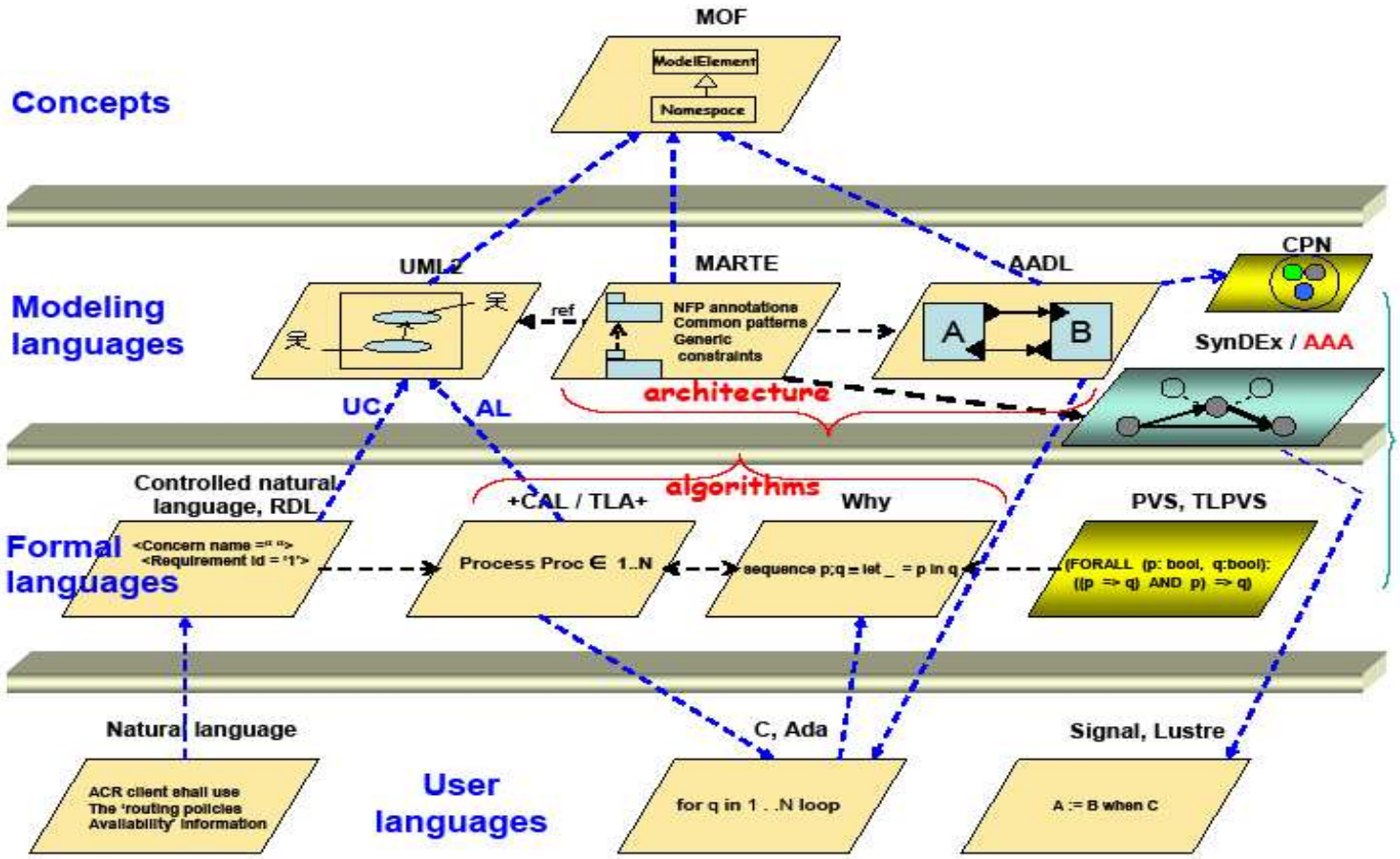
The strong convergence of recent modeling languages, development processes and methodologies for developing real-time systems underlines a set of requirements for a more methodical approach.

This paper presents the issues related to the lack of method in the field of software engineering for real-time systems (in particular, avionic systems). We will mainly describe what converge in the new methodologies that are quite adopted but not implemented as methods. The Unified Process is analyzed and revisited in order to support the new types of requirements that we have identified to require the integration of formal methods, a proof-based system engineering approach in the first steps, and a refocusing on the model-driven development.

The Big picture - Elements



The Big picture – abstraction levels



-1- A Software Development Process

- **A software development process can be applied with many methodologies or many methods (or without any)**

- **The concept of software development process is closely related to the concept of task scheduling, and it is essentially a matter of temporal order**
  - The underlying field is **project management**.
  - How do we manage avionic embedded systems projects?
    - Respect of the standards (DO178B and ED-12-B)
    - Modularity
    - Check lists

-2- A Methodology

- **From a methodology, several methods can be created, and this methodology must refer to a software development process**
- **The concept of methodology is linked to a particular (and global) approach used to perform the main activities of software engineering,**
  - requirement engineering,
  - analysis and design,
  - proofs and verifications,
  - code generation.
- **This approach does not have to be complete and does not necessary provides any guidelines**
  - The underlying field is mathematical logic.
- **At the level of a methodology, we favor a logic that allows to ease the definition of the problem and its solutions**
  - but the way all the issues are going to be solved is not detailed.
  - This is **an overall logic**, with its own structures, rules and theories, most often justified through practical study cases.

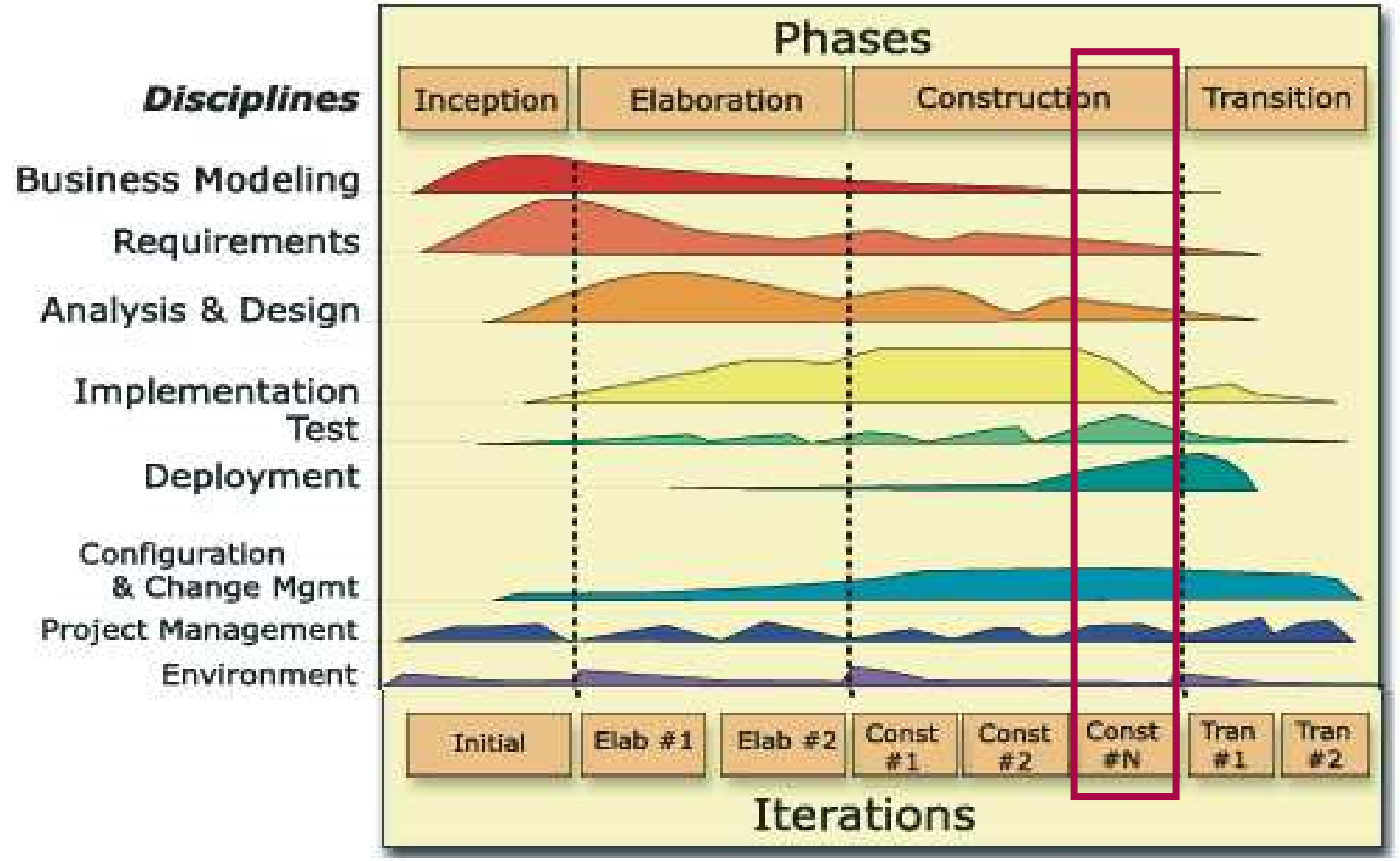
-3- A Method

- **A method is built from a particular methodology and is adopting only one software development process**
- **The concept of method is closely related to the concept of strategy.**
- **The method provides means for executing the development process tasks in an optimal way**
  - The method refines the methodology with a set of improvements.
  - The underlying field is **Operations Research**.
- **This concept goes beyond the development process concept**
  - there is not a unique way of performing an activity, whatever is the issue
- **At the level of a method, any type of issue has to be considered, with the optimal way of solving it**
  - there is a logic that allows us to solve any issue with more efficiency
  - the method shows in details ***how to apply this logic***

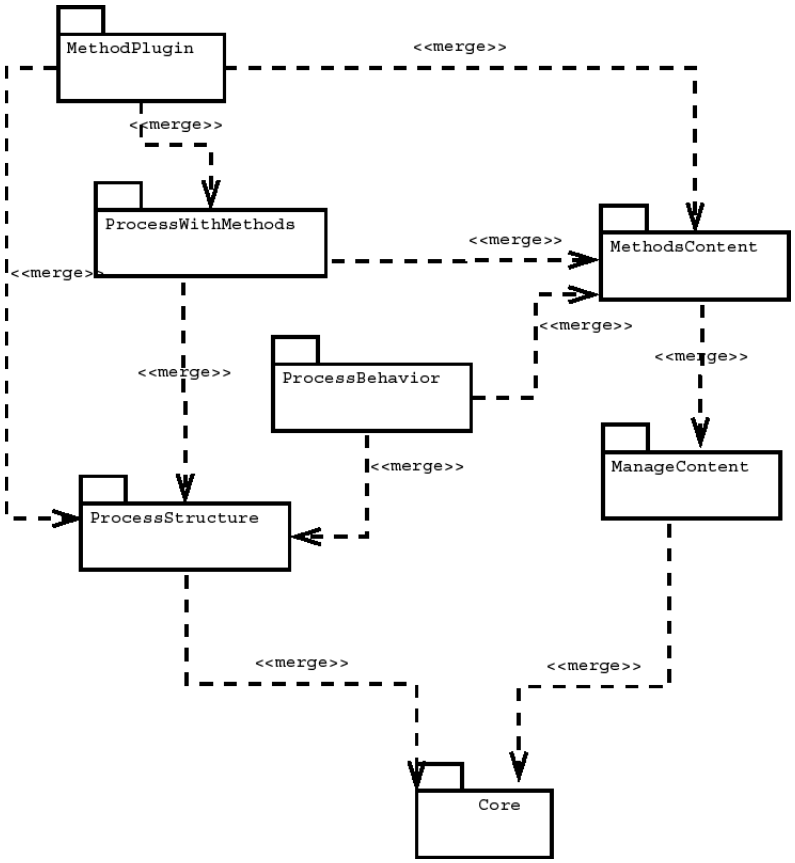
-4- Best Practices

- Develop Iteratively
- Manage Requirements
- Use Component Architectures
- Model Visually (UML)
- Continuously Verify Quality
- Manage Change
- Manage languages heterogeneity, integration
- organization in three views, functional, structural and dynamical

-5- From RUP to UP



-6- From UP to SPEM



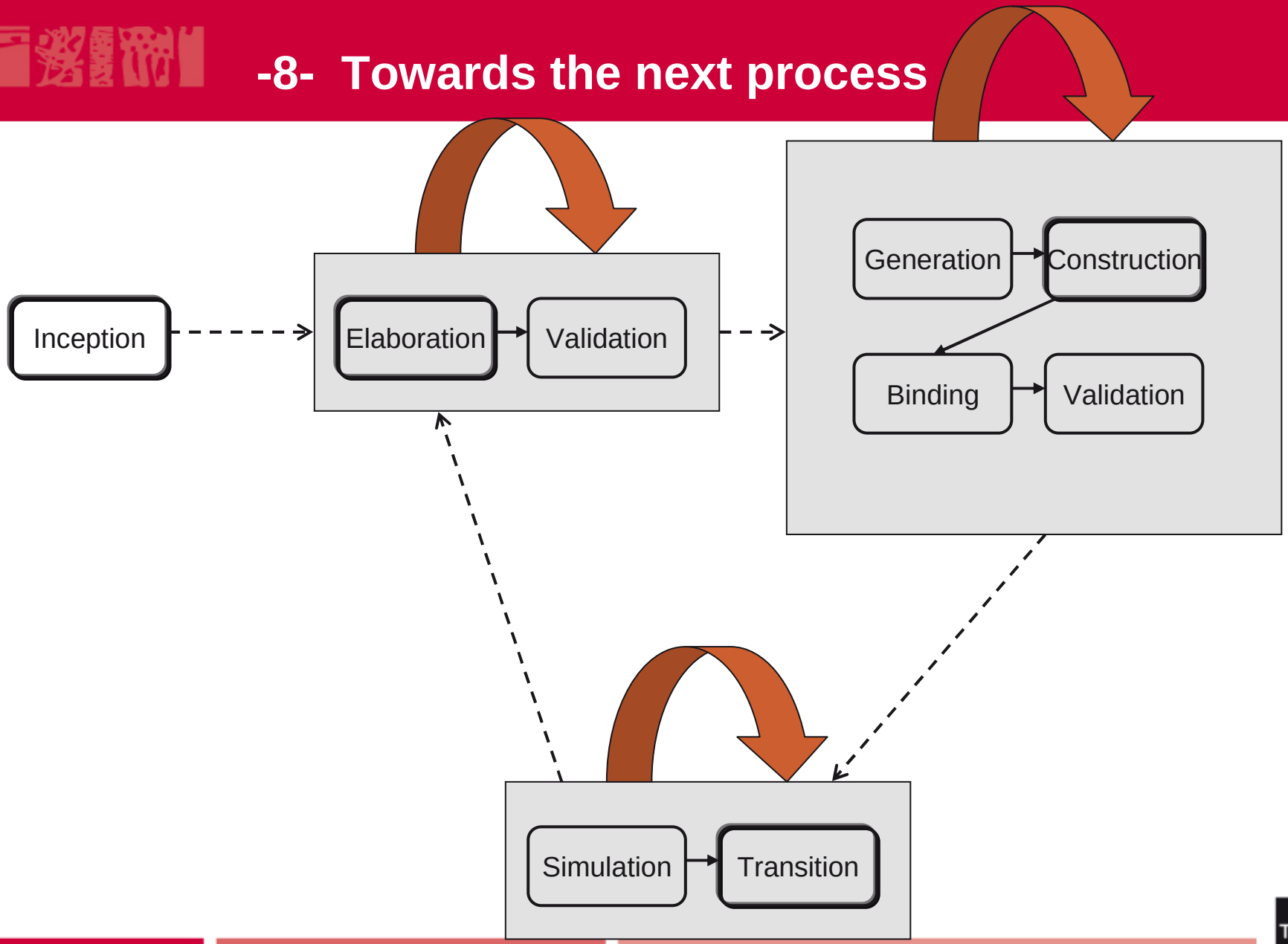
-7- UP Advantages

- **IBM Rational definitely banished the waterfall process**
  - Unfortunately this good resolution have not been followed by everyone, even in the research field.
- **The ``use case driven'' approach is definitively a very good approach that is even kept in the Agile methods**
  - allows the requirements to be traced
- **The ``architecture-centric'' process is adopted for all complex and large systems**
- **The possible customization enables an adaptable process framework in which each company may choose the most convenient elements.**

-7- UP Drawbacks

- **The homogeneous decomposition between Inception Elaboration and Construction is too much simplistic**
  - Because depending on activities types, cycles are more or less complex, therefore not homogeneous
- **The RUP is supported by a very heavy tool, which is not intuitive**
  - The learning period is long and requires significant investments
- **Depending on the environment, the parameterization may also be very long**
  - the parameterization gives the impression of genericity,
  - but the process is not fundamentally different for a any kind of project (telecommunications, automotive, aeronautics, financial, etc) : the phases and activities are the very same.
- **The RUP is only suitable for very big projects**
  - its intrinsic logic is so much linked to the IBM Rational world that it is mostly applied with the entire tool suite.
- **The entire process is rather a set of good recipes than the result of a rigorous ``rationale'' as the name should suggest**
  - The inception phase should be global with respect to a systemic approach.

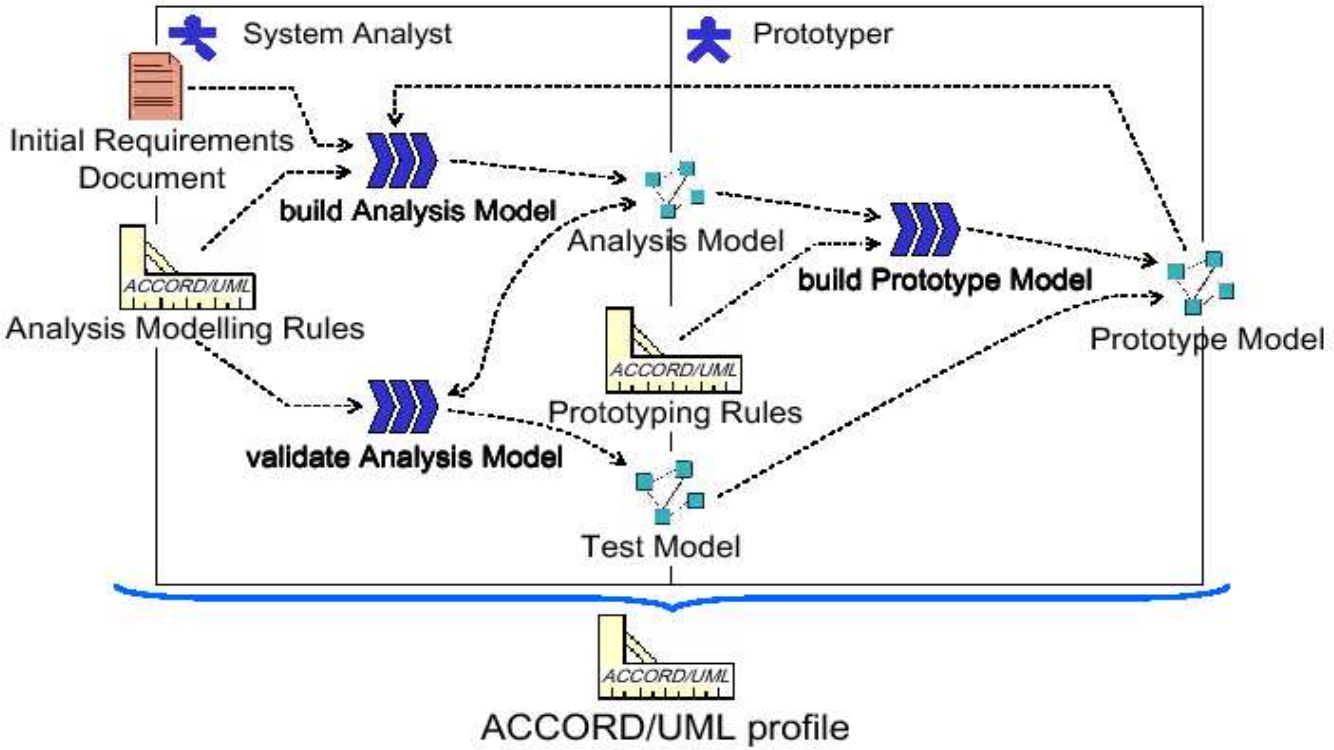
-8- Towards the next process



-9- Methodologies

- **A methodology is not a method**
  - above the method and provides foundations to define a set of methods
- **a method is defined**
  - when its approach is frozen (set of best practices & guidelines)
  - when it is made available for a larger community than this with which it has been initially created
- **New methodologies converge**
  - allow a full description of real-time features (high level of abstraction based on standard)
  - they are all providing tools that enable a seamless flow from models to code and vice versa
  - full model-driven approach
  - more and more reusability, more reliability
  - ➔object modeling techniques and formal methods have to be both used in the same frameworks
  - what formalism is driving the other one and when in the lifecycle ?

-10- The ACCORD/UML methodology



(Sébastien Gérard, CEA)

-11- ACCORD/UML Strong points

- **This methodology covers the whole lifecycle**
- **ensure a complementarity and a consistency between the different models**
- **It has adopted the organization in three views, functional, structural and dynamical**
  - the models are better organized through the development process

-11- ACCORD/UML Weak Points

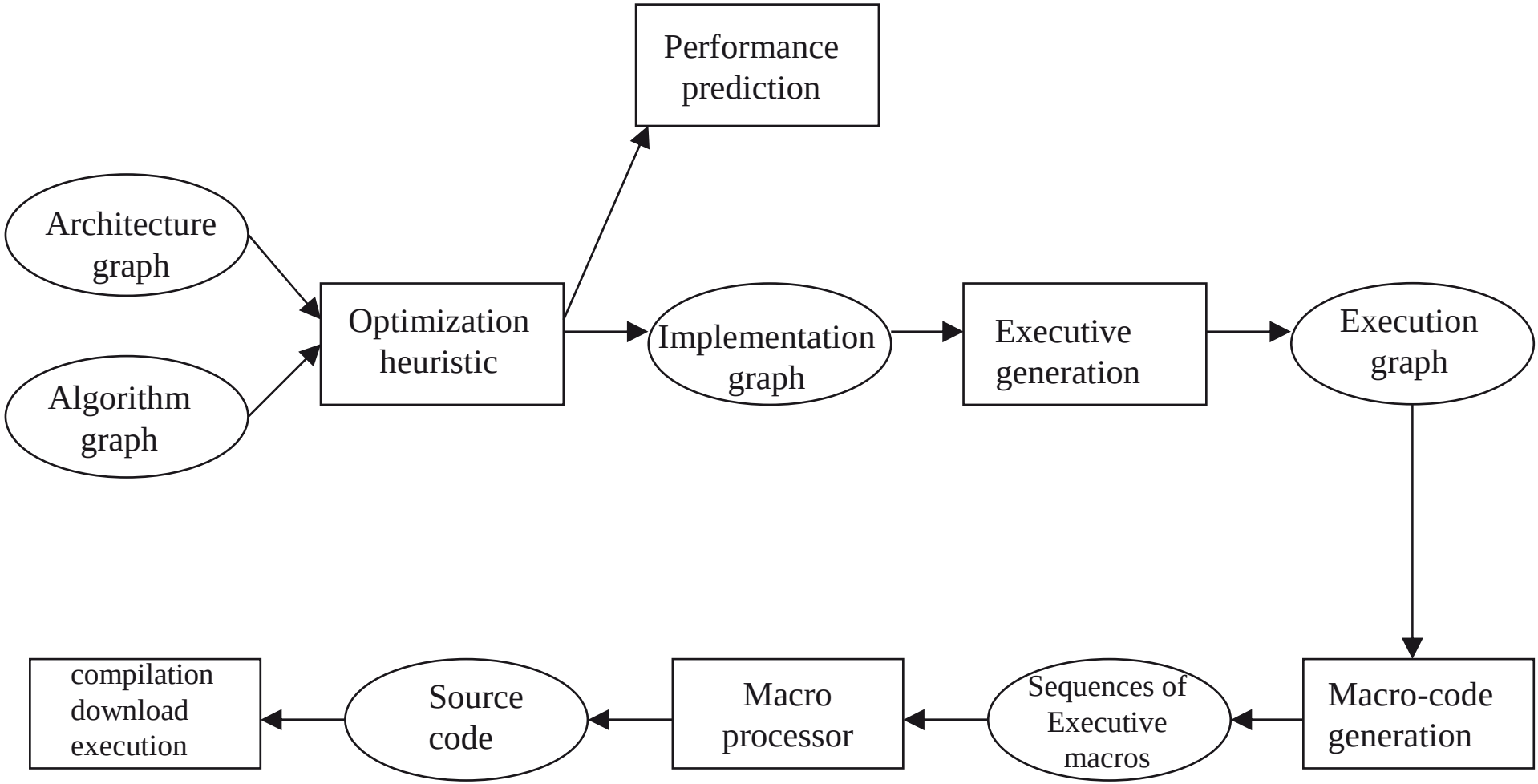
- **The support of MARTE by the ACCORD kernel is partial**
  - does not provide any opportunity to capture formal requirements
- **The action language is not formal (C++)**
  - it is not possible to formerly check or verify the actions
- **The lifecycle is obsolete and not very well adapted to MDD technologies**
- **There is no seamless flow between application analysis models and architecture models**
  - it is not clear how it is possible to refine the analysis models
- **ACCORD/UML favors the use of EAST-ADL to the detriment of all other ADLs, but is not really based upon several notations**
- **it appears like a single notation-based methodology**

-11- ACCORD/UML Shortcomings

- **The lack of formal methods in the whole lifecycle is obvious**
  - It is missing specially because critical real-time systems need to be formally checked
  - to have a support of formal languages in the earliest steps of the design

- **The issue of their integration is not even discussed**

-12- The AAA methodology



(Yves Sorel, Inria)

-13- AAA Strong points

- **A perfect integration between algorithms and architecture is reached through a unified model based on oriented graphs**

- **The implementation is done with graphs transformations :**
  - it distributes the actions on the different processors (the graph is partitioned)
  - it distributes the inter-processor communications on the inter-processor link
  - it schedules the actions assigned to a processor
  - it schedules the communications assigned to an inter-processor link

- **AAA/SynDEx provides a lot of interfaces with DSLs as Scilab/Scicos for the modeling and simulation of hybrids systems**
  - UML2/MARTE that allows a high-level modeling with its real-time profile
  - all the synchronous languages (Esterel, Lustre and Signal)
  - only one ADL : the EAST/ADL which is mostly dedicated to the automotive field,
  - the two languages AVS and CamlFlow for image processing and functional data-flow

-13- AAA Weak Points

- **The methodology coverage on lifecycle is partial**
  - it has to be integrated to others which may have a totally different approach
  - In particular, the adequation of AAA is based on an optimized implementation
  - ➔ this is something that has to be taken into account as a feedback in the resulting modeling of the architecture design
- **there are only interfaces to a software, and a lot of work remains to describe a global design method that would be based on the common use of all these languages**
- **The optimization process that allows a rapid prototyping is not easy to extend to the whole lifecycle of big projects**

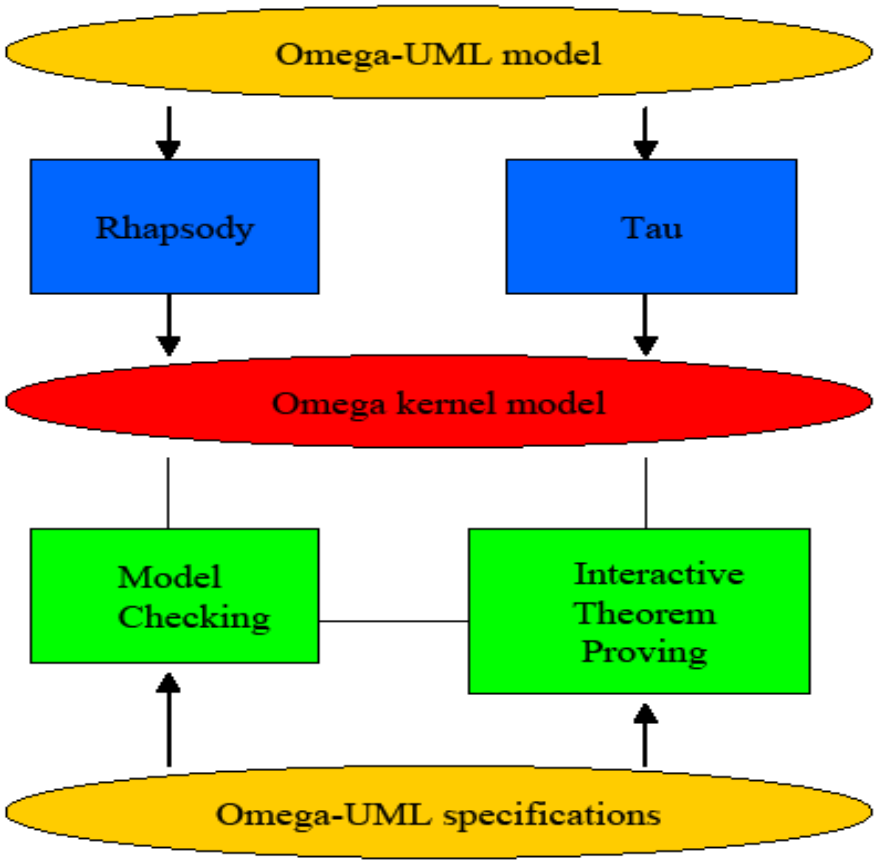
-13- AAA Shortcomings

- **AAA methodology only focuses on a small part of the lifecycle and is designed to optimize the implementation of distributed real-time embedded applications**
  - describing its shortcomings is not a challenge

- **What is a challenge is to integrate this methodology into a more generic approach that is consistent**

- **Above all, what is really missing is the requirements capture phase**

-14- The OMEGA methodology



(Susanne Graf, Verimag)

-15- OMEGA Strong points

- **The most mature methodology**
  - It has been taken into account **almost all the real-time design issues**

- **if any important issues have been properly (but separately) addressed (with different and adequate languages), there still remains some unaddressed issues**
  - the integration of a simple and understandable language for requirements
  - the use of a specific ADL
  - the overall process development orientation is not specific (the proposed process development is quite the RUP)

- **It goes further than most other methodologies**
  - it really provides a consistent set of languages and tools
    - allows model-checking
    - interactive verification based on the PVS theorem prover

- **The semantics of the OMEGA kernel model is expressed in PVS and the TLPVS package is also used for properties that are expressed in LTL.**

-15- OMEGA Weak Points (1)

- **Too much ``tool-oriented''**
  - use of OMEGA / use of the IF language and toolset
  - IFx ,extended version of IF
- **The methodology is not really driven by formal requirements**
  - Partial mapping between OCL and PVS
- **The IF specifications : a good intermediate representation between the user level modeling (SDL,UML, SCADE) and a semantic model that allows formal simulation and verifications**
  - far from being proved-based, makes the simulation and verification phases the main phases of a project
  - The earliest steps of requirements are not taken into account

-15- OMEGA Weak Points (2)

- **One of the major drawbacks is the ``push'' of verification techniques to the detriment of other formal approaches**
  - The concepts are tackled by the middle of the lifecycle (as the ``meet in the middle'' of the AAA/SynDEx methodology)

- **UML is the main language of the methodology (the standard and the OMEGA profile)**
  - in the requirement capture phase, nothing more than the use cases is exhibited
  - ➔we deduce that the proposed use cases are not formal
  - the scenarios are depicted using Live Sequence Charts, initially, the specification of the problem domain is not formal

- **There are no stated differences between the real-time families domains, which do not have the same constraints/goals**

- **There is no specific proposition concerning a concrete syntax for the action semantics**

-15- OMEGA Shortcomings

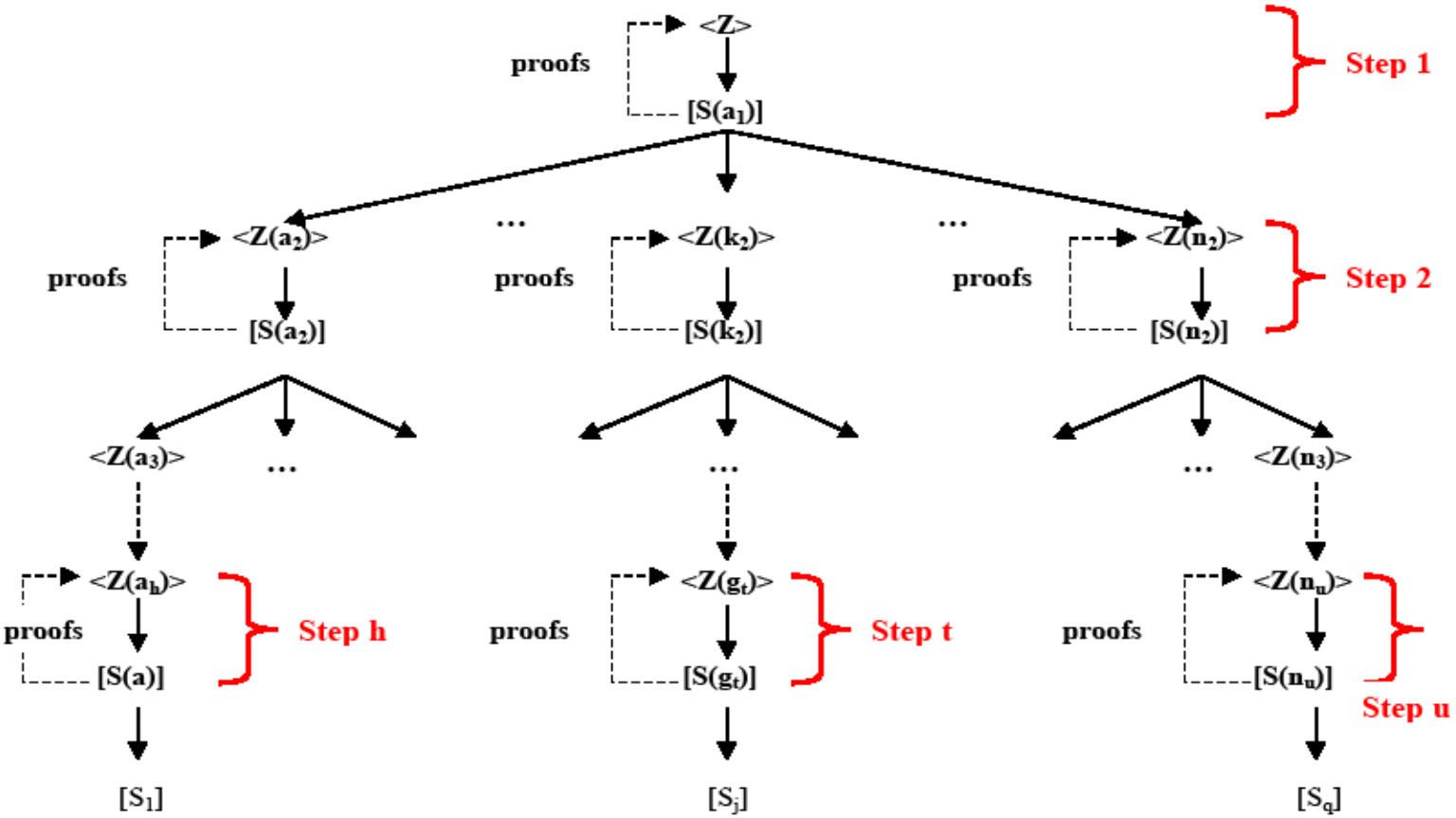
- **does not uniformly cover the whole lifecycle**
  - some parts are more formal
  - is too much verification-oriented

- **The communication, simplification and planning aspects of the modeling activities are ignored**
  - Models may provide a way forward but are not necessary built with a more abstract language that allows to execute actions
  - Models are here to show how the actions could be executed, which is totally different

- **With such a plethora of languages, the methodology does not provide anything like guidelines or any starting point to automate the different lifecycle phases**

- **The use of standards (MARTE, AADL) is not taken into account, and OMEGA is not standardized**

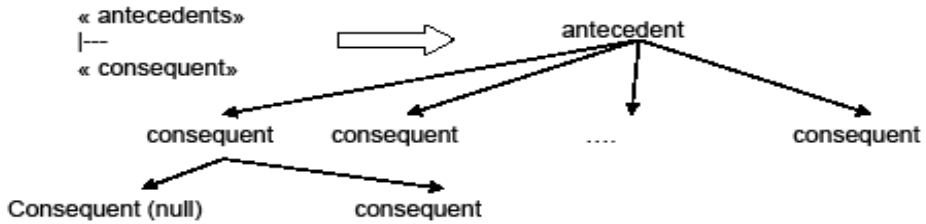
-16- The PBSE methodology



(Gérard Le Lann, Inria)

$$[S] = \cup_{i \in \{1,q\}} [S_i]$$

-17- Strong points/ Weak Points / Shortcomings



$$Z \mid\!- S$$

$$Z_1 \mid\!- S_1$$

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

-18- The next methodology

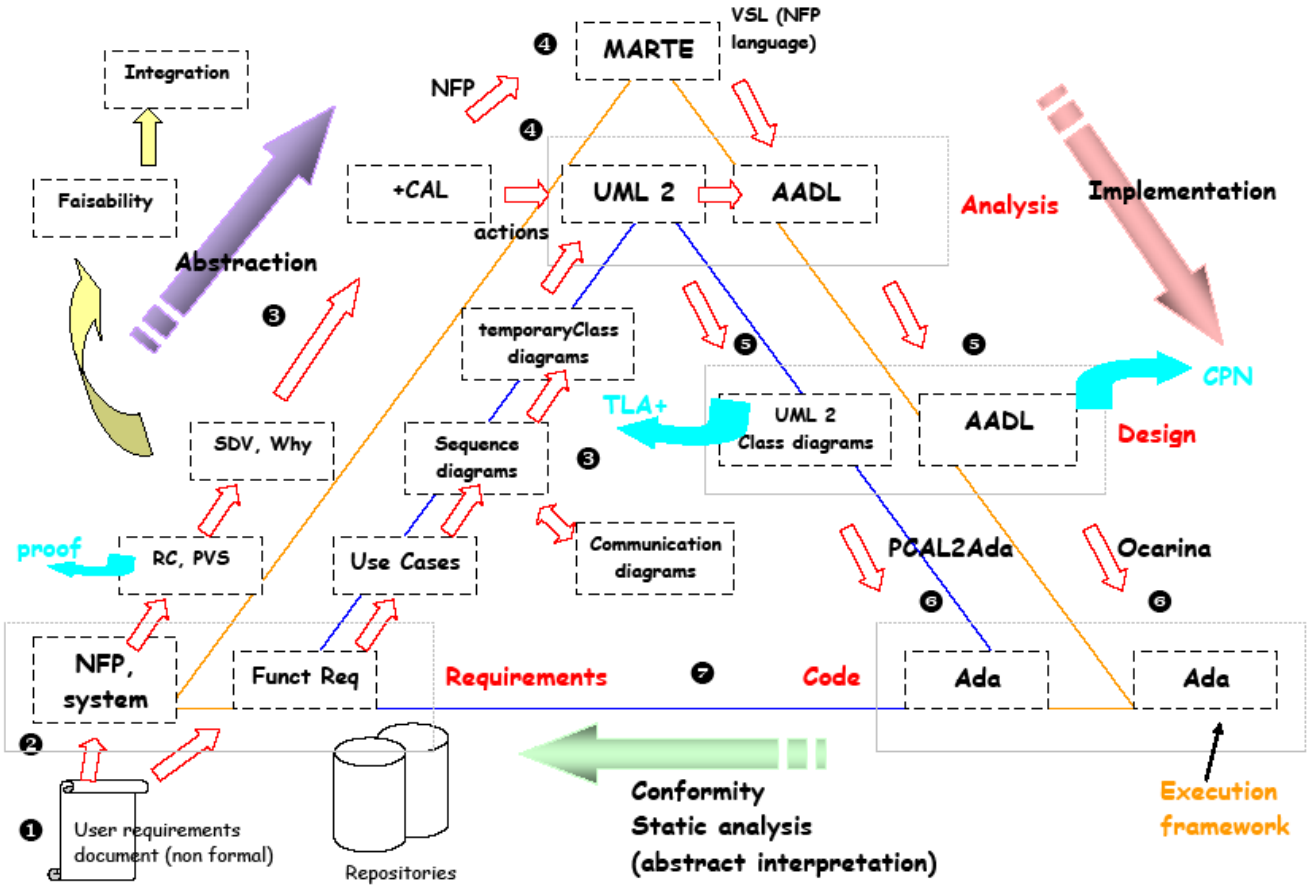
- **The next methodology will provide a seamless flow, as the ACCORD/UML does**

- **The ACCORD/UML methodology has shown its capabilities to integrate ADLs notations (EAST-ADL), and the ``AAA'' methodology for the algorithm architecture adequation**
  - It provides a preliminary Analysis model with analysis modeling rules, an intrisic action language, a prototyping model with prototyping rules that respectively allows model-checking and code generation
  - More of it, the ACCORD/UML have always based the modeling activities on the UML profiles for real-time, even the MARTE profile is not yet Integrated

- **These features are sufficient to base any method framework on the ACCORD/UML methodology**
  - So far, all the formal aspects are missing in the ACCORD/UML. Nothing more than the OCL allows verification (and the OCL is a very restricted language)
  - The action language is not formal, and no formal language is proposed in the first steps of the development process, so the critic non-functional properties cannot be proved

- **All the integration techniques that are used in the OMEGA methodology in order to take benefit of the formal methods have to be adapted in the ACCORD/UML methodology**
  - It does not necessary means that a language such as IF is to be inserted
  - but ``intermediate'' languages have to be used in order that the end-user have easily access to the provers and model-checking techniques without spending too much time on formal methods engineering.

-19- A new software engineering method



Conclusions and Future Works

- **Several modeling languages**
  - Functional aspects
  - Architectural aspects
  - Behavioral aspects
- **Modeling languages**
  - Models that make systems less complex
  - Executable models
- **Iterations are not performed with the same type and number of activities**
  - A growing complexity of languages integration
- **Languages integration**
  - What are the best practices ?