Component models for embedded systems: *from UML to Autosar*

François Terrier

with contributions from
Sylvain Robert, Ansgar Radermacher, Frédéric Loiret

CEA-List

francois.terrier@cea.fr

Local context of researches

SYSTEM@TIC PARIS-REGION Pôle de compétitivité

Usine Logicielle

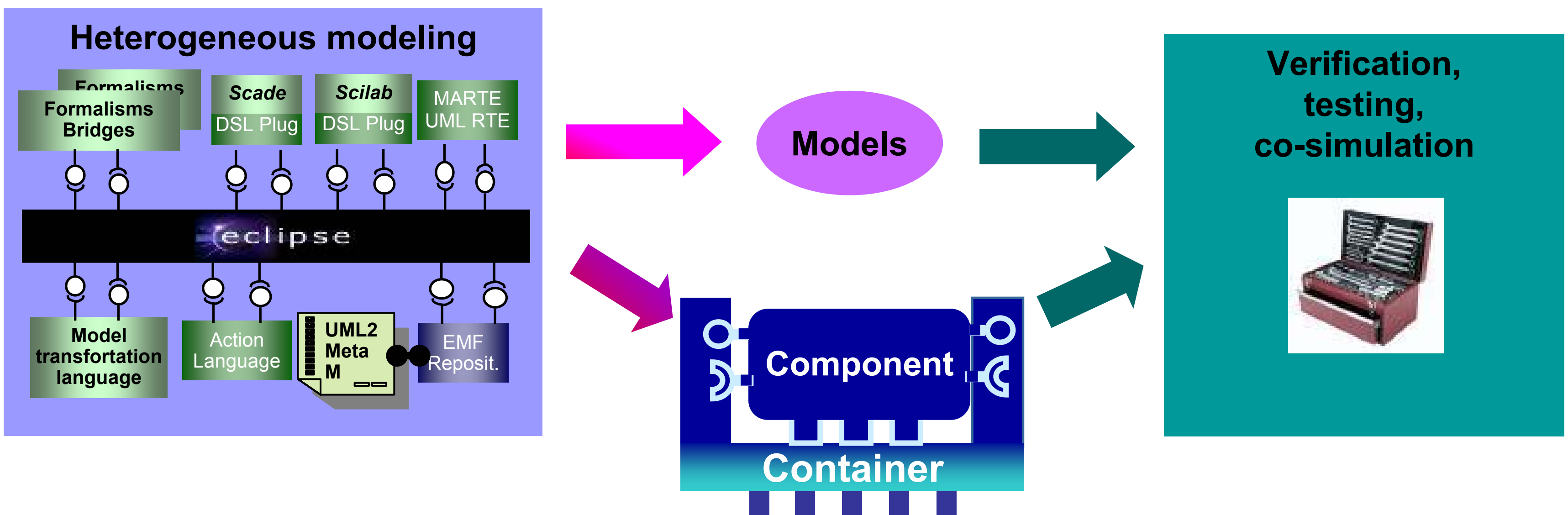
NUM@TEC AUTOMOTIVE

SYSTEM@TIC
PARIS-REGION
Pôle de compétitivité

Concevoir et maîtriser les systèmes complexes

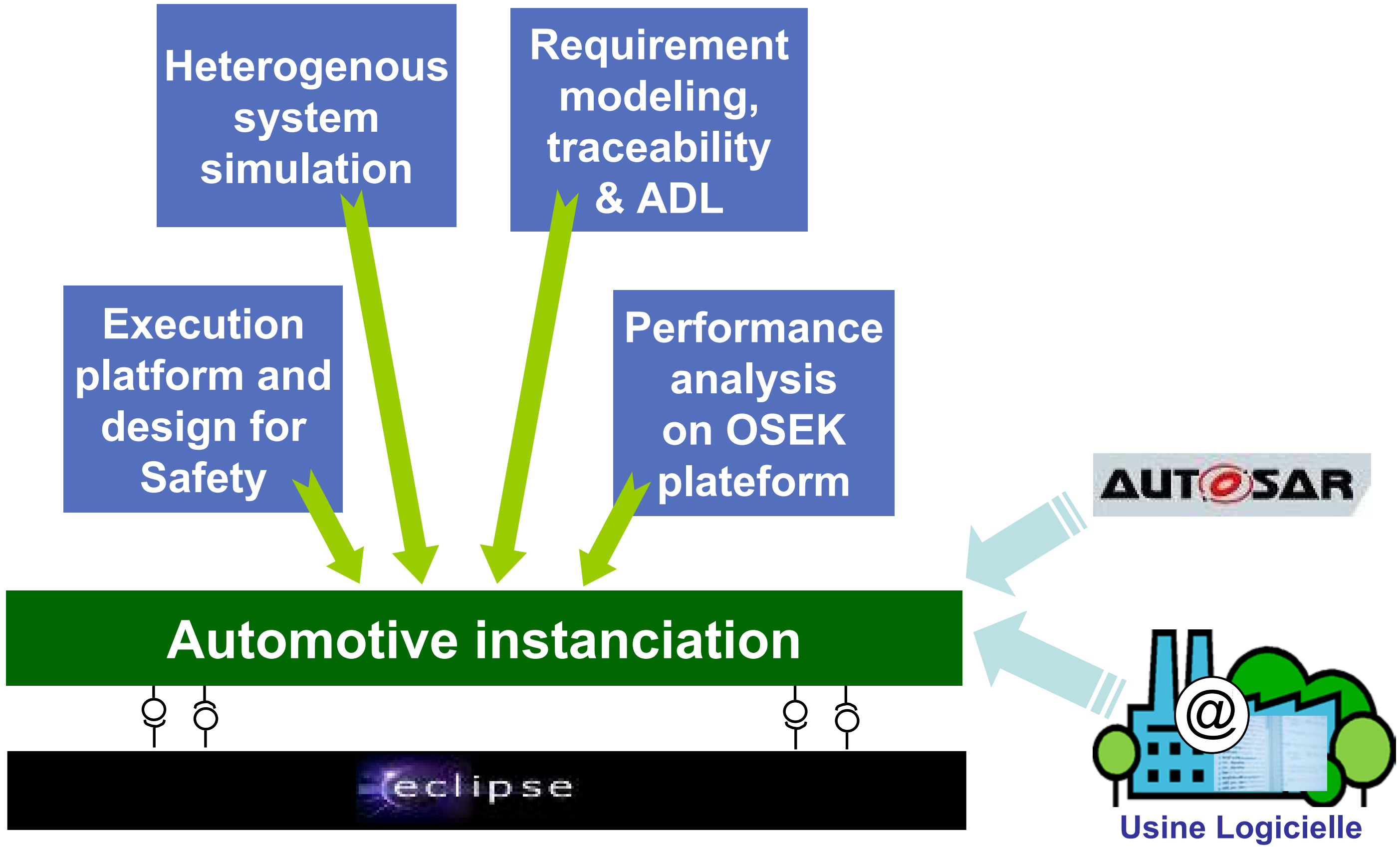
Usine Logicielle

Multi domain tools for Model Driven Engineering
➔ *Heterogeneity & interoperability management*



Execution infrastructure built through generation & libraries
Integration of fault tolerance services

➔ Research program on embedded systems for automotive & transportation



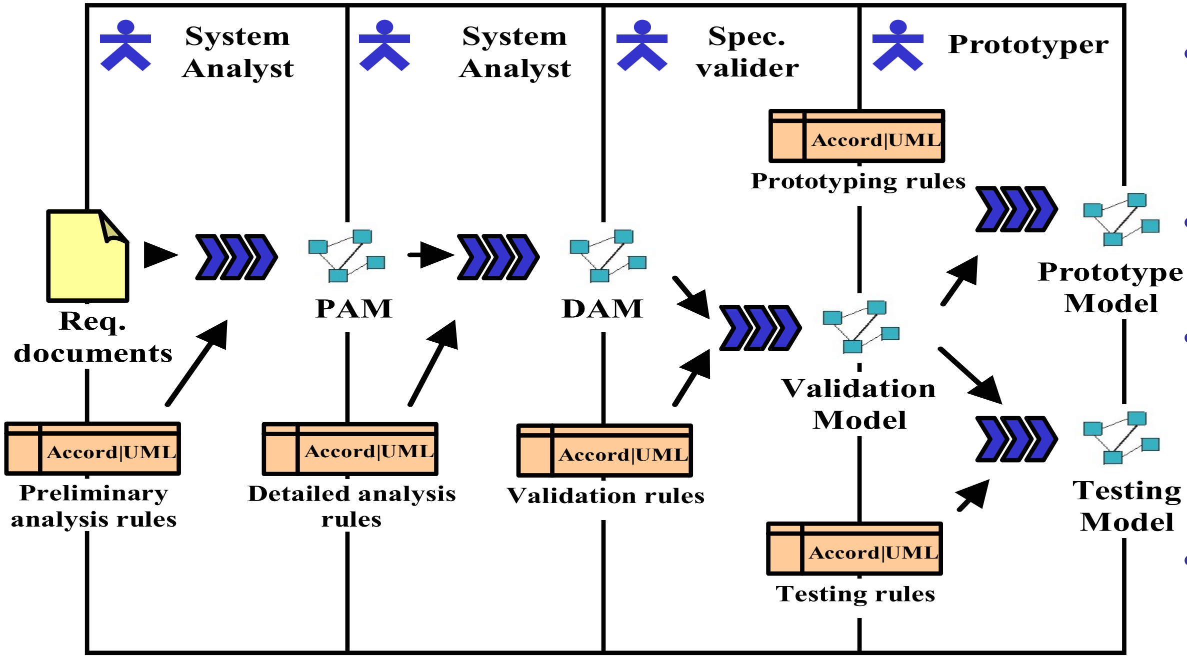
UML & Component model

… Starting with a UML profile for RT!

Building a MDE tool chain for RTES

- a conceptual framework
- a development process and method,
- a set to software engineering tools
- an execution platform

→ to assist in developing applications from requirements to deployment

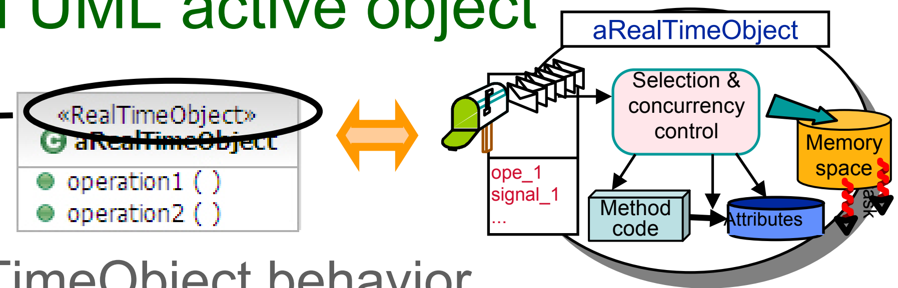


- **UML and profile based approach**
  - UML models
  - Modelling rules
- **UML 2.0 Profiles**
  - For RTE concepts
- **Tools to support methodology**
  - Automated refinement
  - Pattern appliance
  - Model validation
- **Dedicated RT Kernel**
  - Code generation

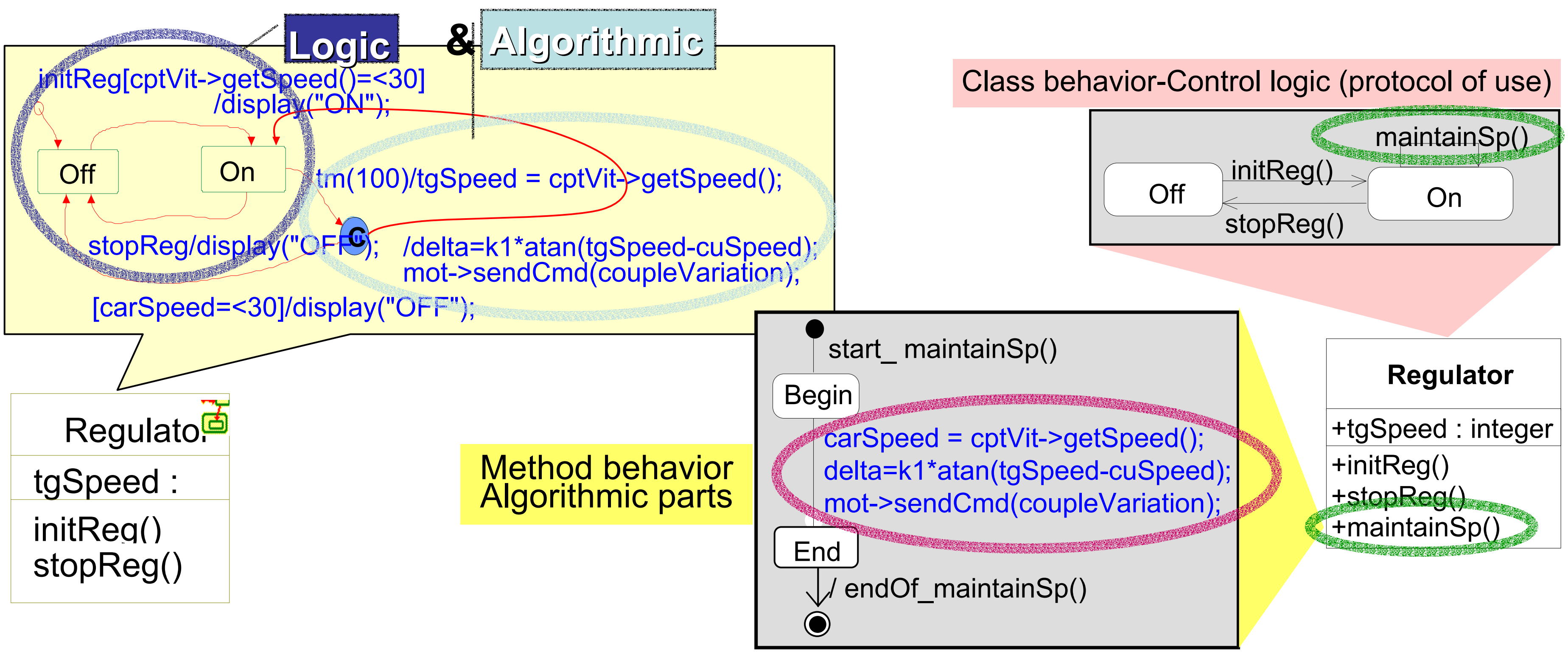
Introduce High level concepts

- RealTimeObject: extend UML active object

UML stereotype



- ✓ Chose way to model RealTimeObject behavior
  - → Use of protocol state machines *(→ now in UML2, see DIPES'2000…)*

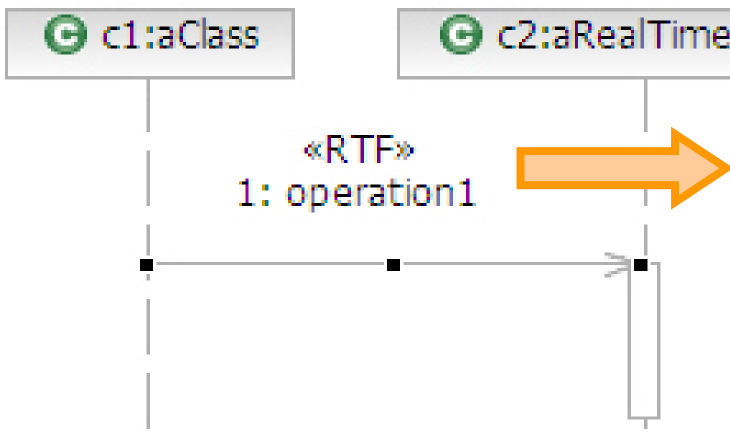


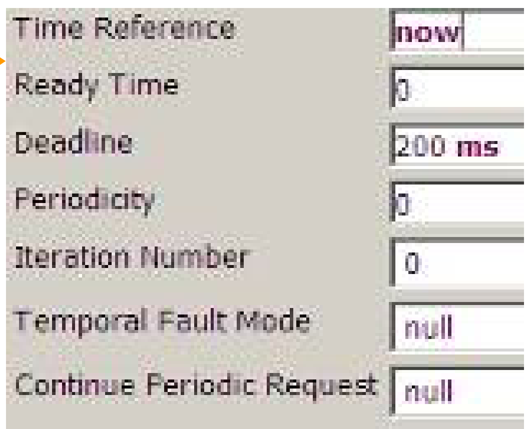
Fix execution model

- Specify queue management policy
- Specify signal management
- Specify concurrency constraints
- …

Refine UML protocol statemachines

→ Attach selection criteria on each message in the queue





- RealTimeFeature

→ Declare constraints instead to implement them for implementation/platform independence purpose…

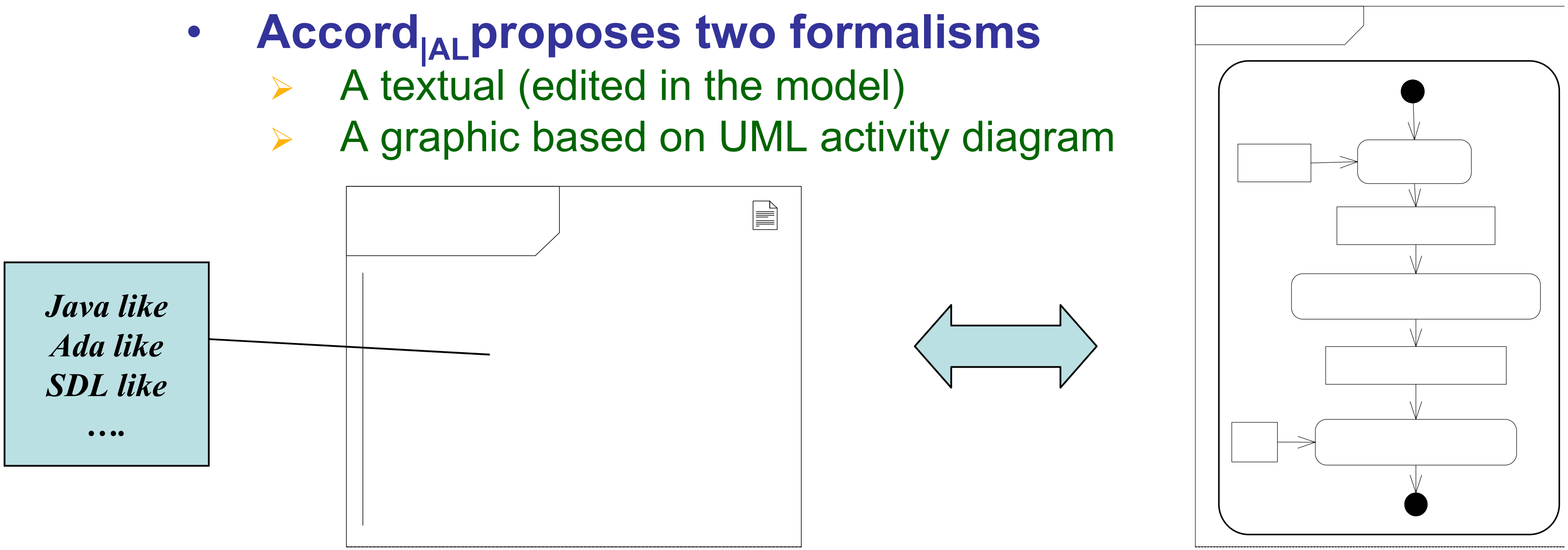
Building complete models

➔ **Separate control (object life cycle) from data processing:**

- Control mechanisms are modeled using state machines
- Data processing actions are modeled using UML activity diagrams

- **Require addition of explicit notations and some basic actions**
  - Mathematical actions are modeled using MathML language syntax

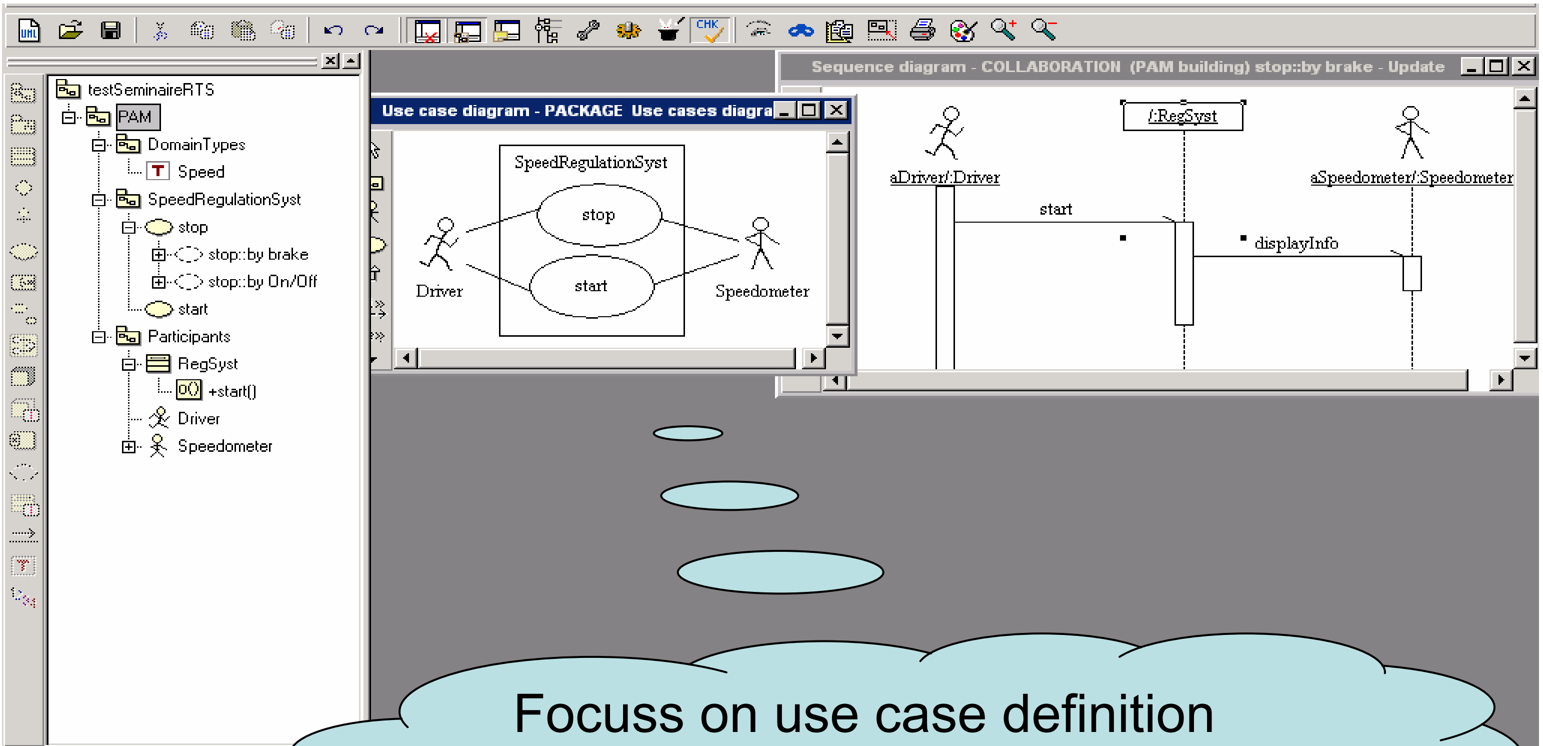
- **Accord$_{|AL}$ proposes two formalisms**
  - A textual (edited in the model)
  - A graphic based on UML activity diagram



➔ In the profile, each action is defined by 3 elements + examples: *semantics, textual notation (in EBNF), graphic notation*

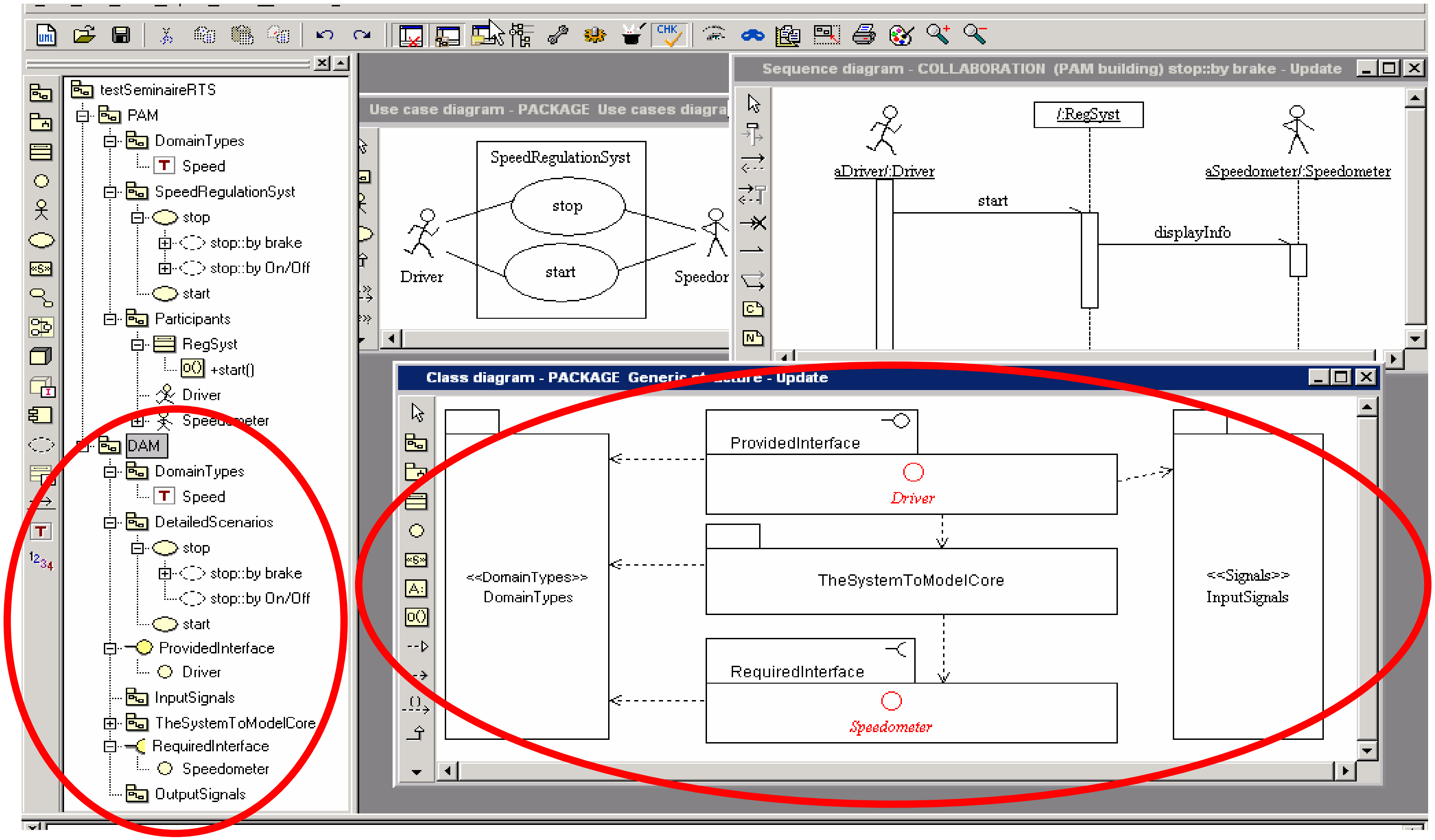
Modeling rules for preliminary model definition

→ Interactions with the developed system seen as a black box

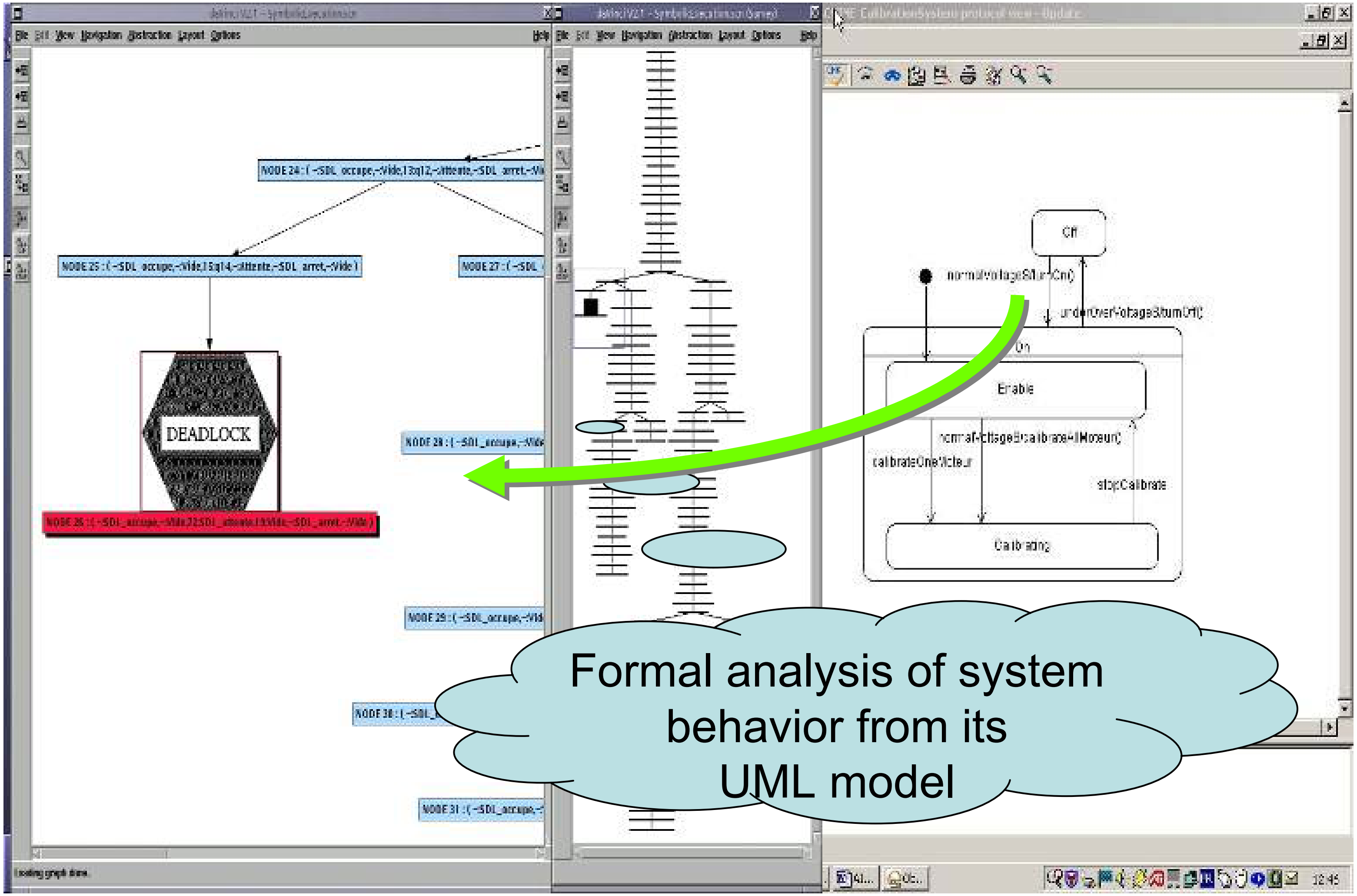


Focuss on use case definition
and collaboration specifications

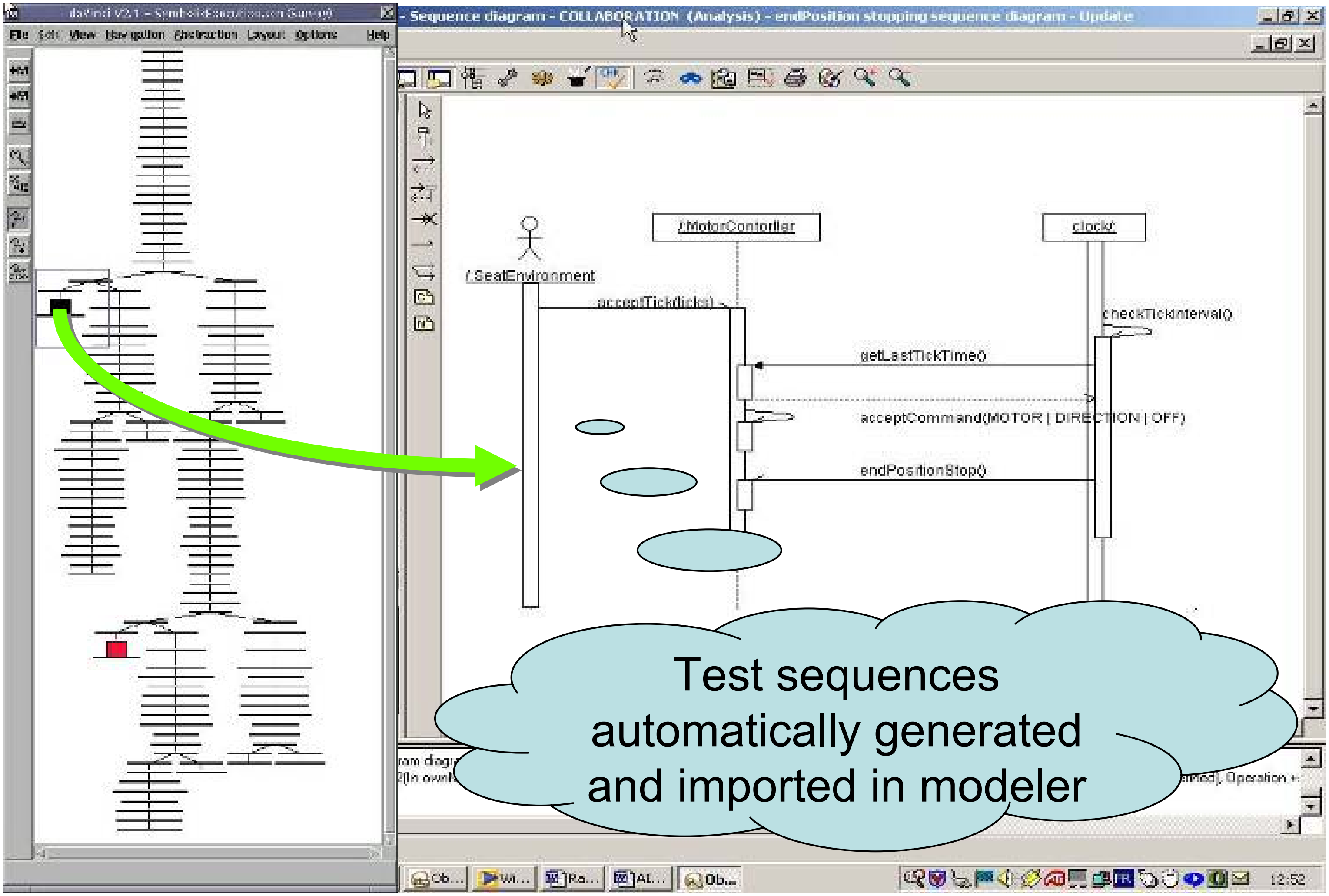
Assistance and automation:
generation (& trace) of the detailed model squeleton



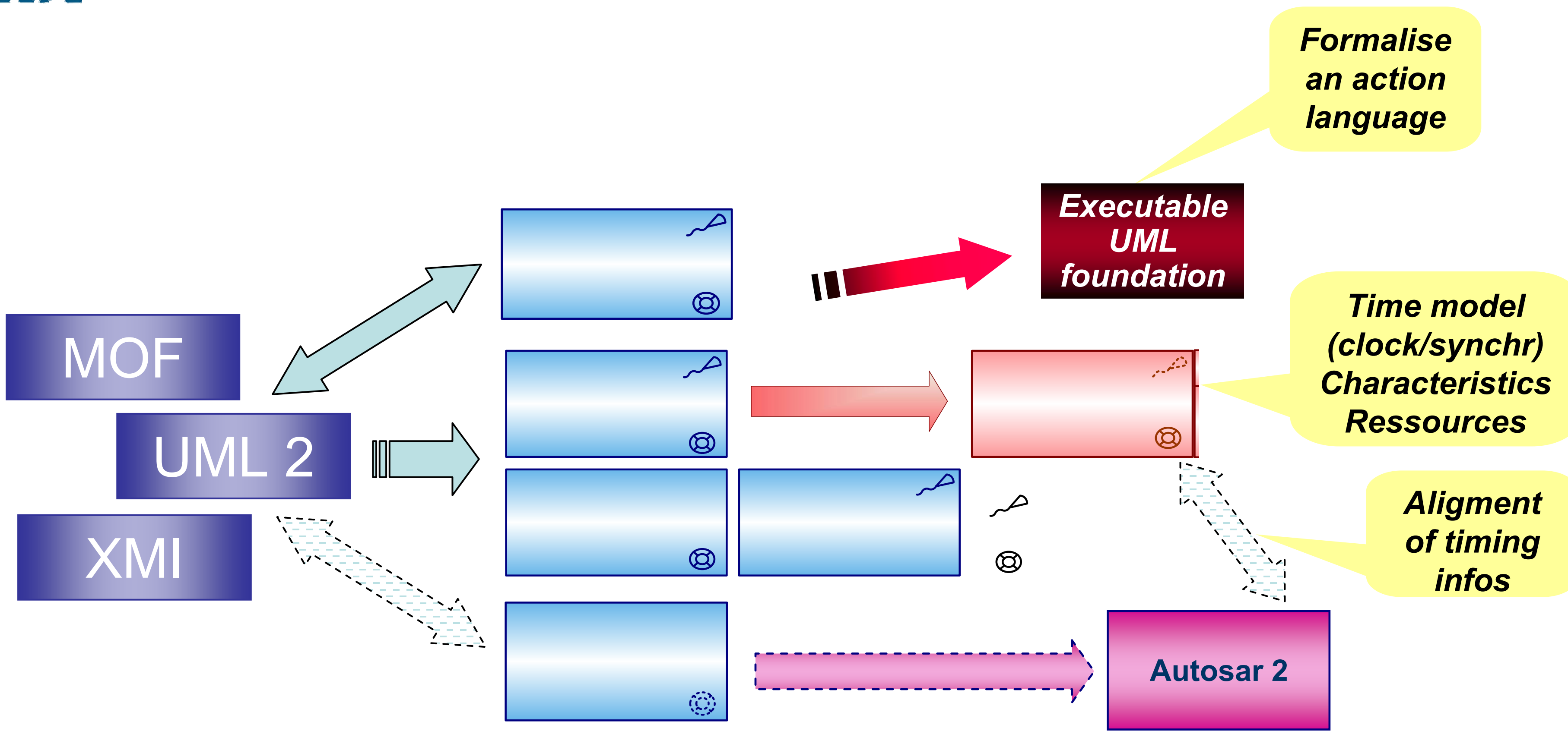
Model translation into formal model
Behavior analysis through symbolic execution



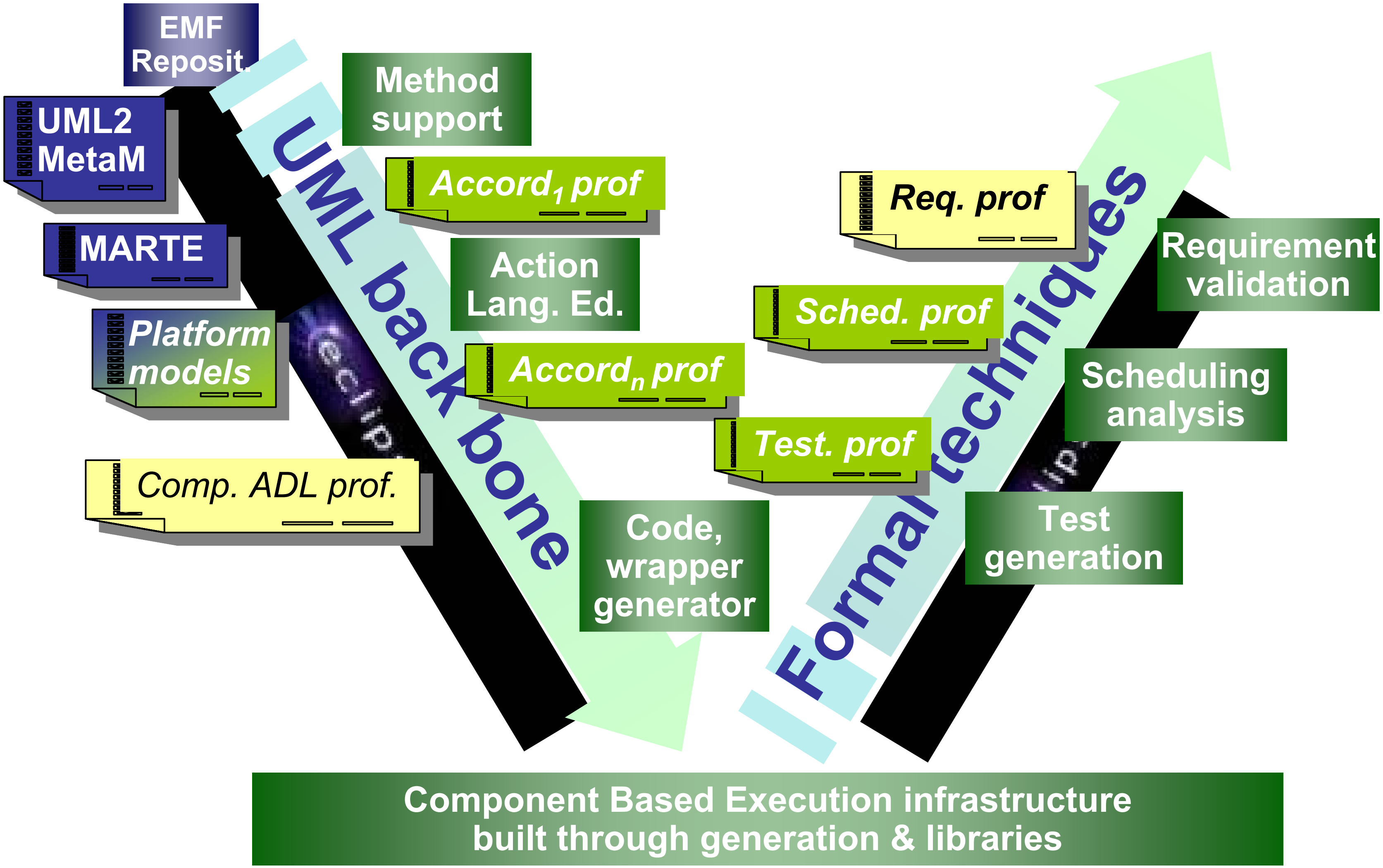
Feedback for behaviour representation



UML for RTES: a set of ongoing actions



Overview of the tool set



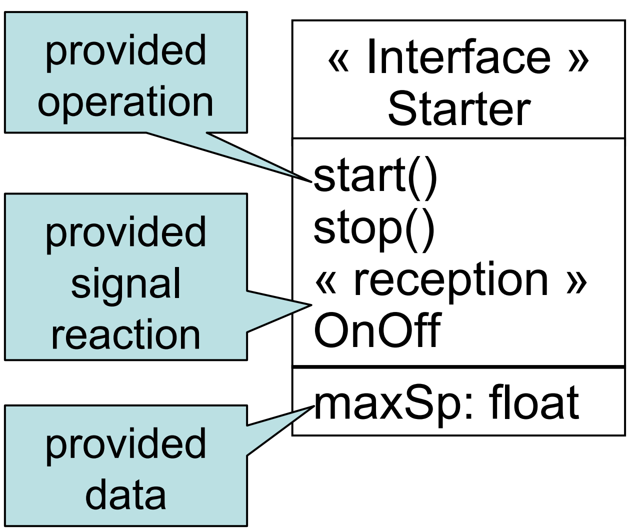
Component diagram

➔ Model the system architecture identifying

- **Modular and replaceable parts of a system**
  - Content is encapsulated
  - Can be replaced during design time or execution time
- **Provided and required interface describing:**
  - Some structural points (attributes, associations, …)
  - Its behaviour (operation, reception, state-machine, …)
- **Two possible views**
  - Extern (“black box”): contract of use, visible behaviour
  - Intern (“white box”)
    - ✓ Shows elements being purely intern to the component (« private »)
    - ✓ Shows how behaviour defined by the interface are implemented
- **Connexion mechanisms**
  - Interface dependencies (association, use, realization)

UML 2.0 Interface

- **Specify operation, signal, attribute, behaviour**
  - **No instances (~abstract class)**
- **“Provided” ⇔ *realized* by a classifier** ***(Class, Component…)***
  A classifier can realize several interfaces
- **Required ⇔ *used* by a classifier**



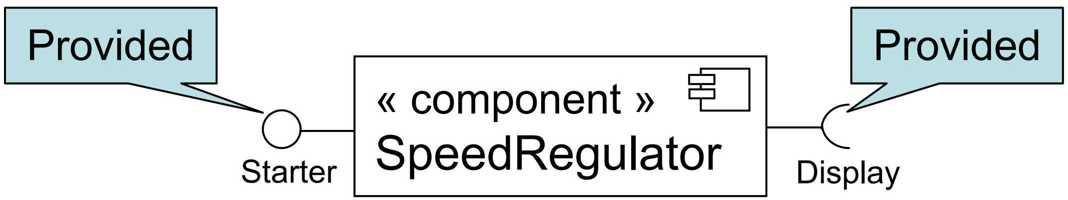
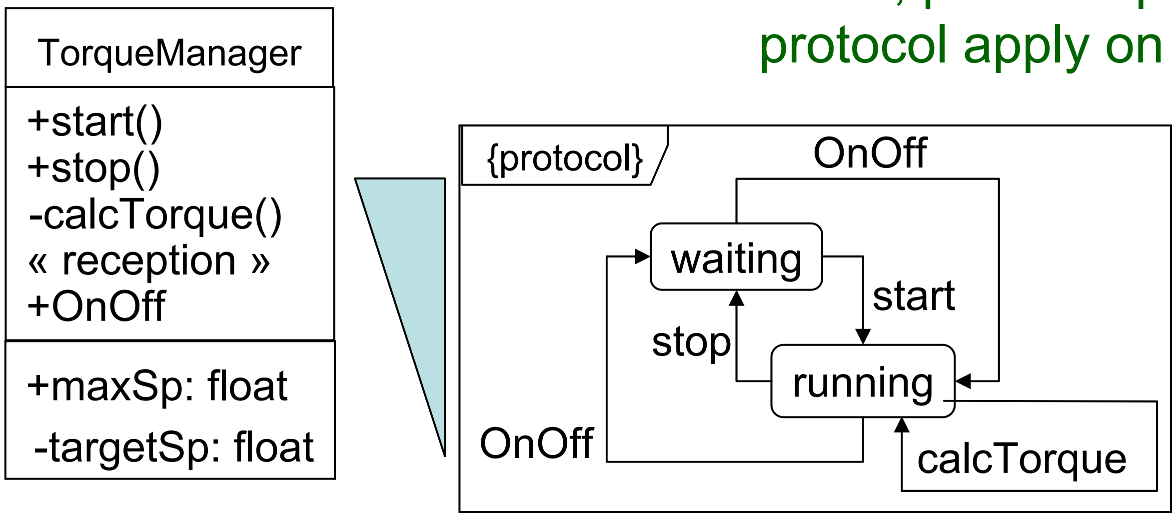


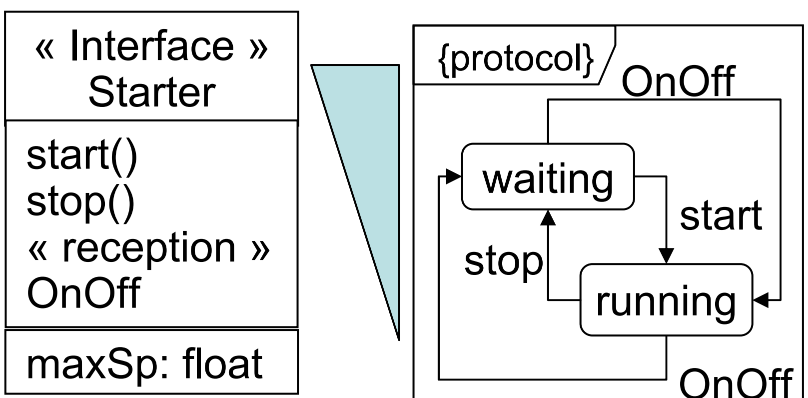
Figure2: condensed notation

Interface can have constraint of use

- **Conformance between Interface / Realisation ⇔ protocol state-machine conformance**
  - State invariant, pre- and post-conditions of interface protocol apply on realization state-machine



➔ New states, transitions, operations, receptions are allowed



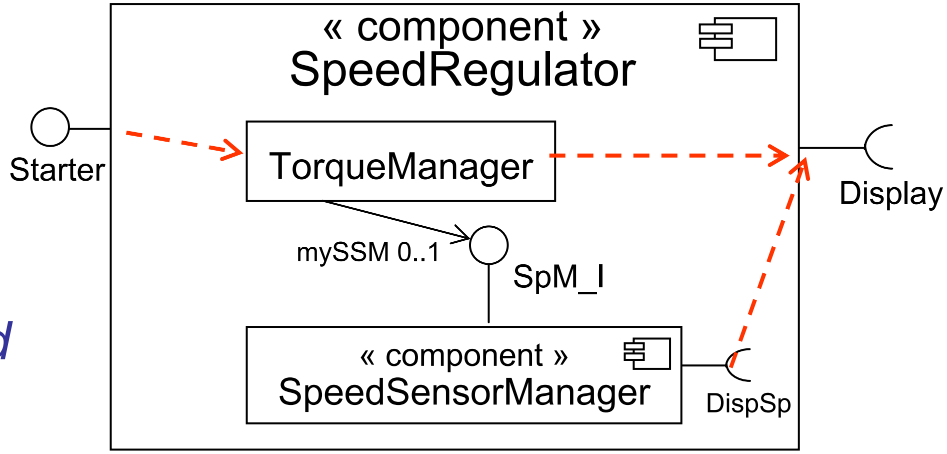
➔ Possible formal interpretation:
- Real. state Inv. ⇒ Interf. state Inv.
- For each mapped operation
  - Interf. Pre ⇒ Real. Pre
  - Real. Post ⇒ Interf. Post

Connectors

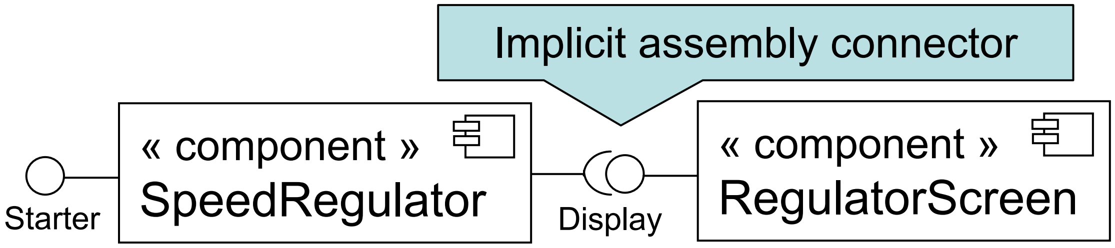
- **Delegation** connector links interfaces of a component with contained parts

➢ Used to model behaviour implementation in nested components

➔ *Implementation conformity required*



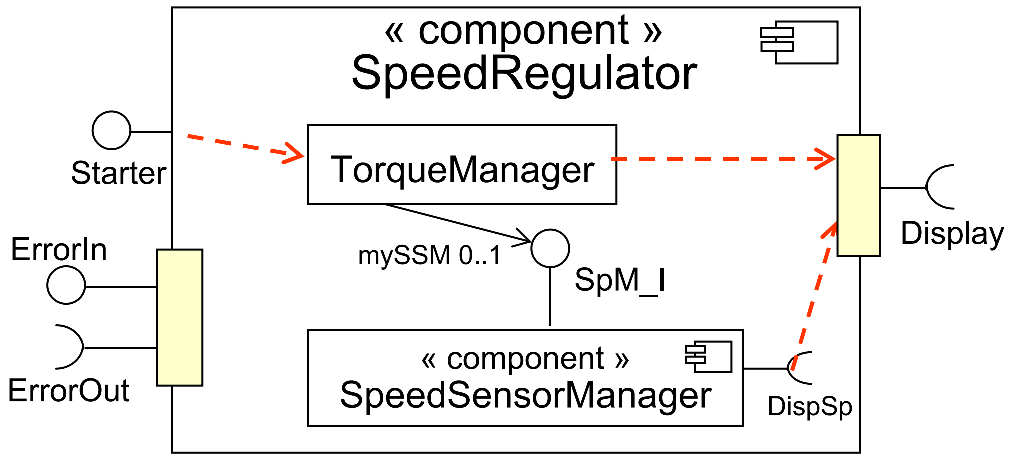
- **Assembly** connector links required and provided interfaces



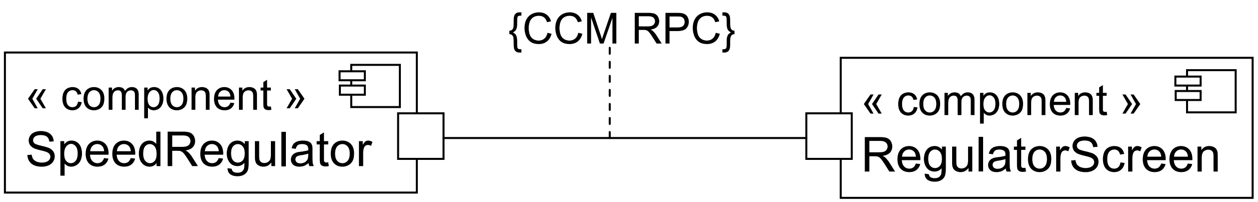
➔ *Conformity of the interfaces required*

Ports

- **Ports to structure usages of the interfaces**



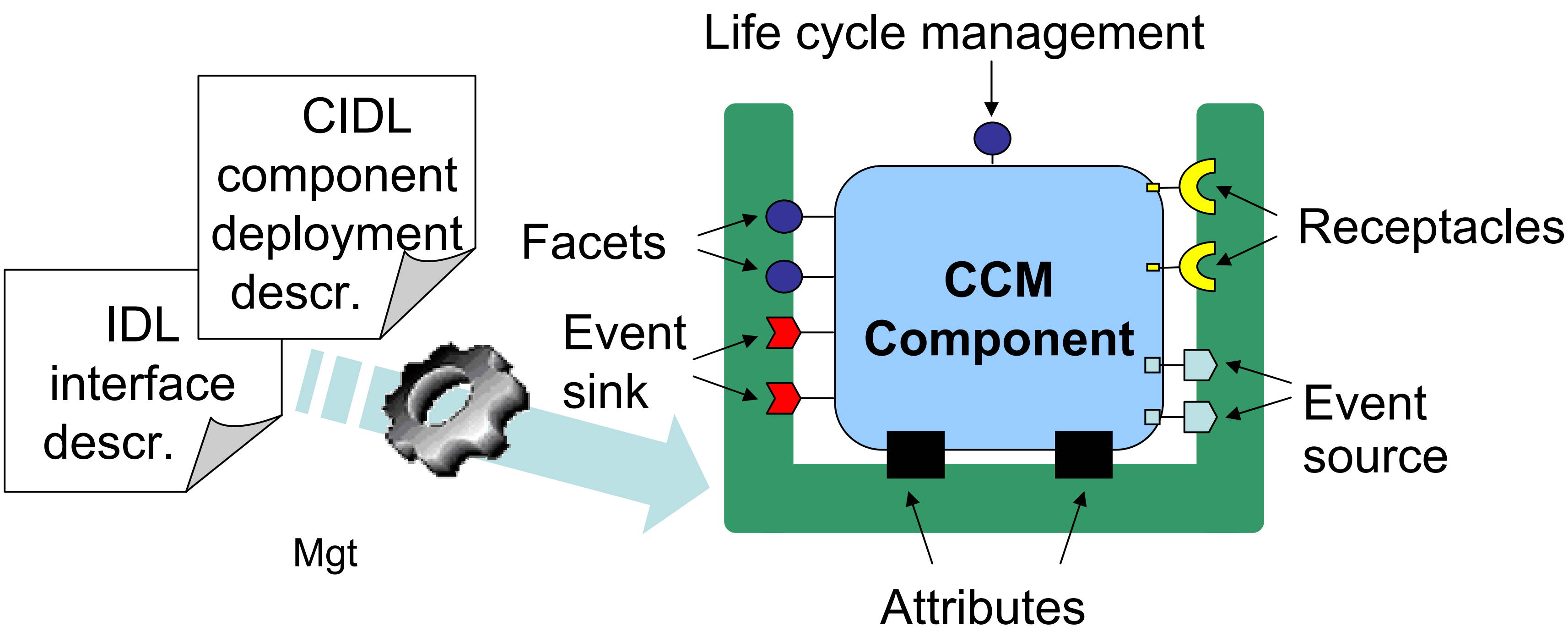
- **Ports making explicit communication links**



- **From Models to Implementation:**
  - Use of a MW component model

A CCM component and its container

- **Principle of CCM component definition**



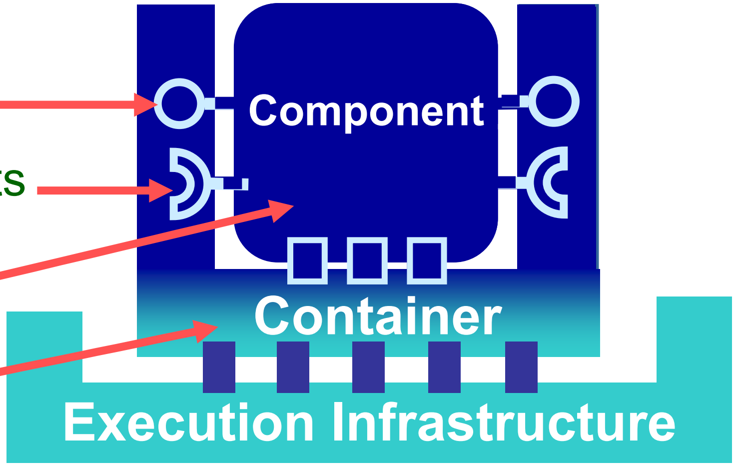
- **CCM component model**

Container/Component model

- **Container associated to component aims to**
  - Localise functional product upgrade in the ***component***
  - Localise dependencies to platforms in the ***container***
  - Provide acces to infrastructure services

- **Explicit description of:**
  - provided services to other components
  - requested services from other components

  - **Separation of concerns:**
    - ✓ business logic
    - ✓ 'technical' properties



- ✓ Containers are provided as part of the infrastructure
- ✓ Based on descriptors ➔ move from programmatic to declarative
- ✓ Easier deployment and reuse, needed for reconfiguration

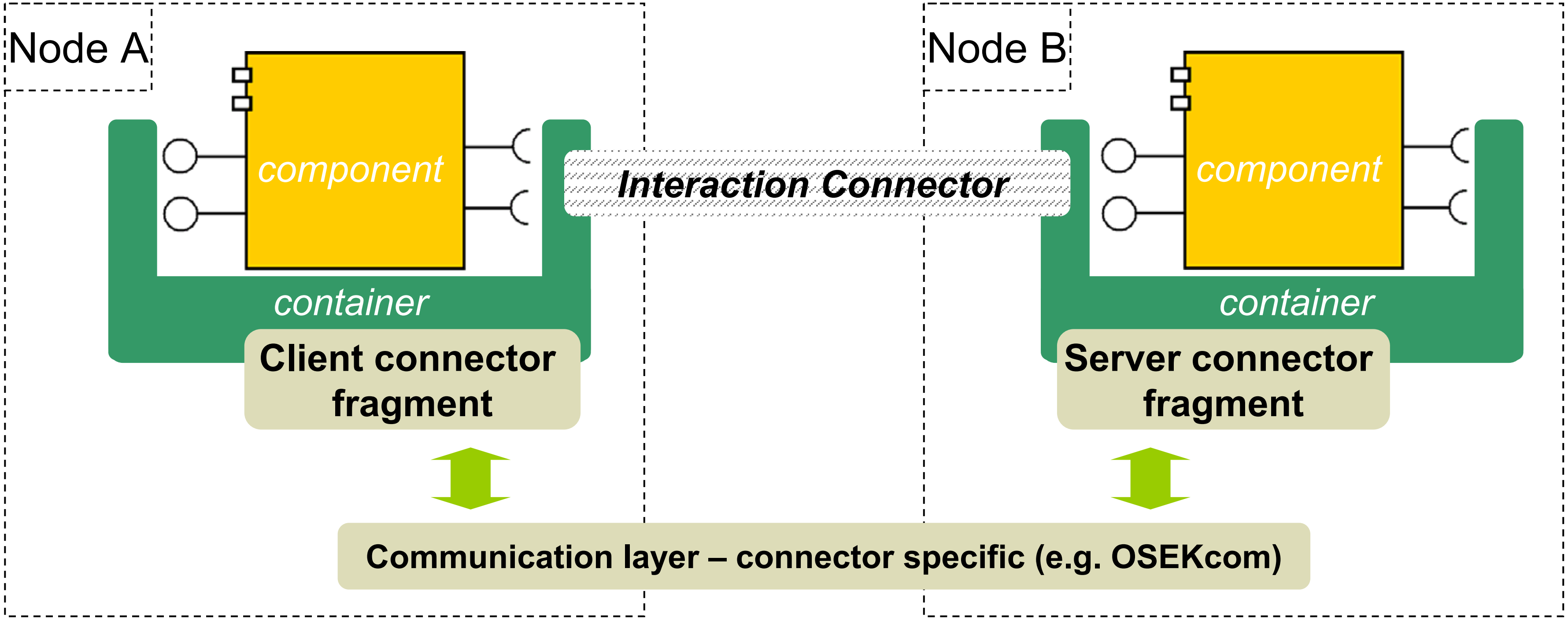
Containter limitation on interactions

- **CCM interactions extensions:**
  - Complex RTE interactions (Streaming, Event passing with priorities, Buffering, Various pub/sub, Deferred synchronous call, Blackboard)
  - Modular, extensible interactions
- **Make interactions independent from CORBA**
  - Embedded ➔ Constrained HW platforms
- **Have minimal impact on CCM**
  - Reuse of existing items
- **Methodological benefits:**
  - Interactions management peculiar to business domain
  - Expertise capitalization

Introducing connectors: C^3M

Component-Container-Connector Model

- **Software entity managing inter-components interaction:**
  - May be considered as part of the container
  - Fragmented
  - Communication layer specific to the connector
  - (potentially) complex intermediary processing



Introducing connectors: C³M

- **Conceptual mapping with UML components**

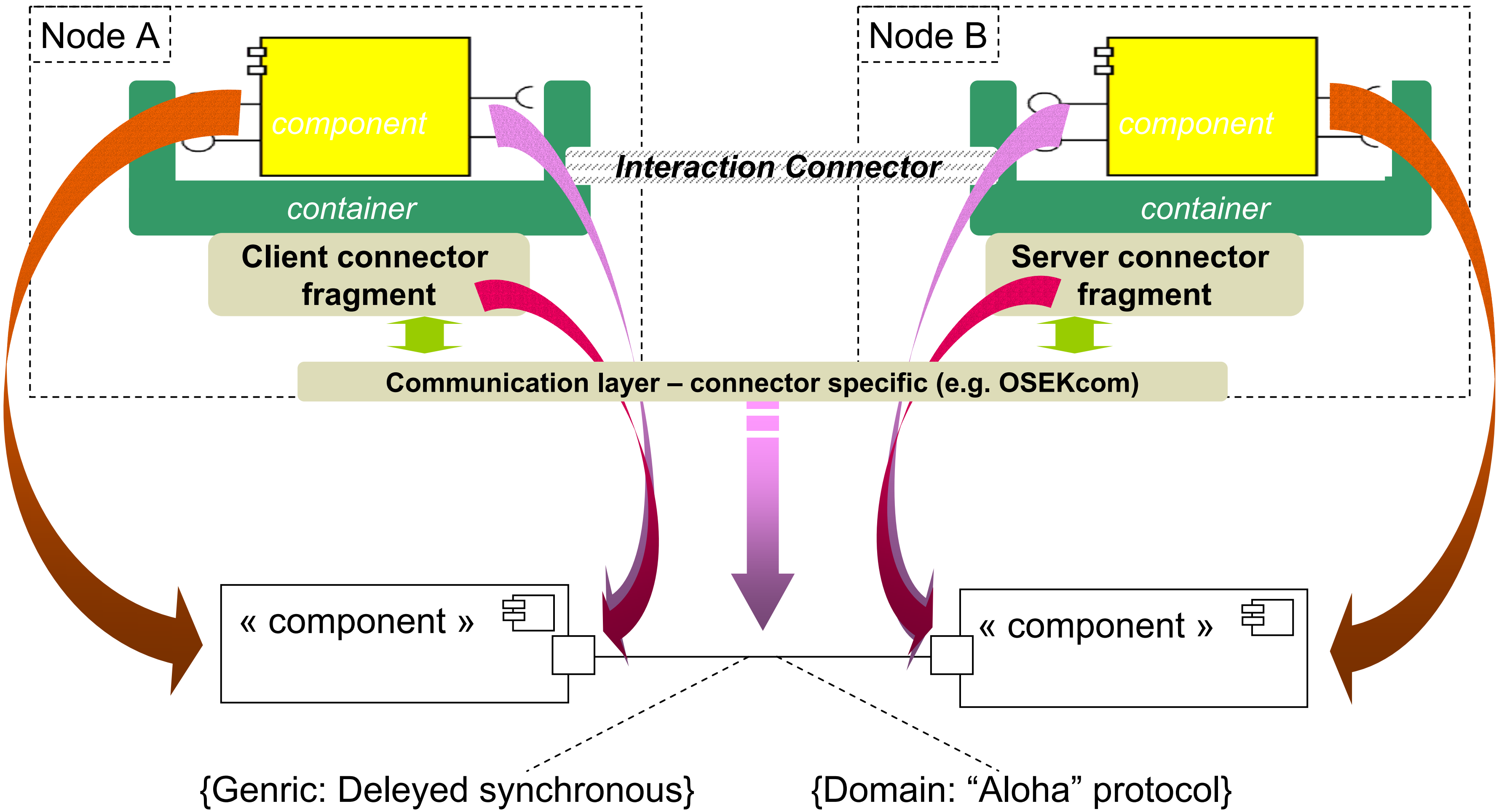


Illustration with the OSEK platform

- **Execution infrastructure for highly constrained hardware platforms**
  - an operating system (OSEK-OS)
    - multi-tasking operating system
    - highly static, all resources declared at compile time (OIL file)
  - coupled with a communication environment (OSEK-COM)
    - simple message-based communication

➔ **From CCM to OSEK: Mapping a (highly dynamic) component-based approach (CCM) on a basic (and highly static) RT/OS!**

- How preserving the CCM development process?

Illustration with OSEK - Execution model

- *Activities* instead of components
  - Identification of activities (control flows) in application architecture
    - ✓ Basically linked to application entry points
  - Activities timing features description (e.g. end-to-end deadline)
- Mapping to tasks
  - Components are design-time development artifacts, with no runtime counterpart
  - Component code is kept intact

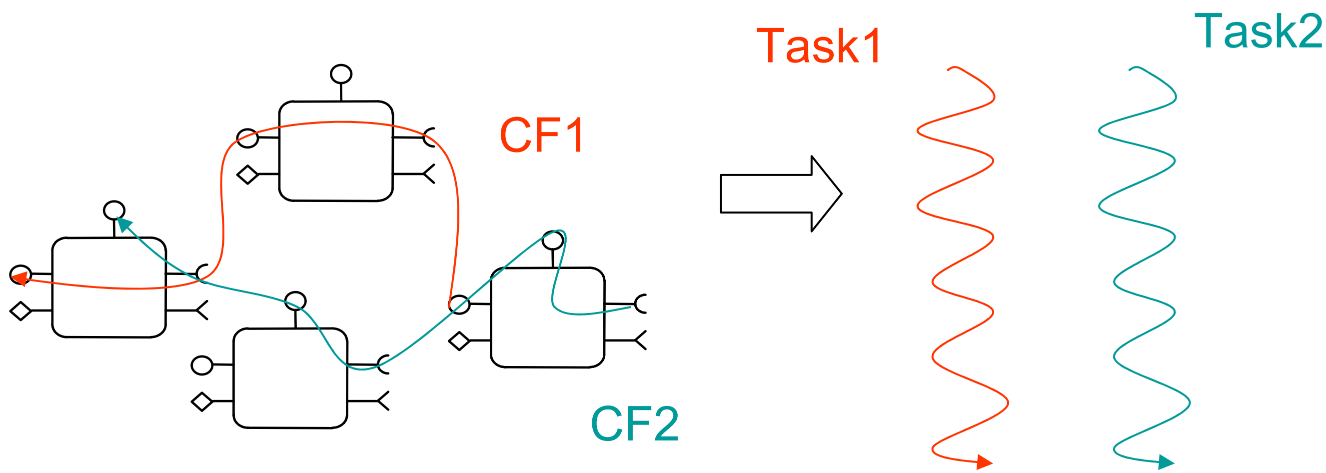


Illustration with OSEK - Communications

- **Each Interaction mechanism is realized by a connector. Ex: synchronous call:**
  - The connector fragment at caller side sends an event
  - The connector fragment at target side receives this event

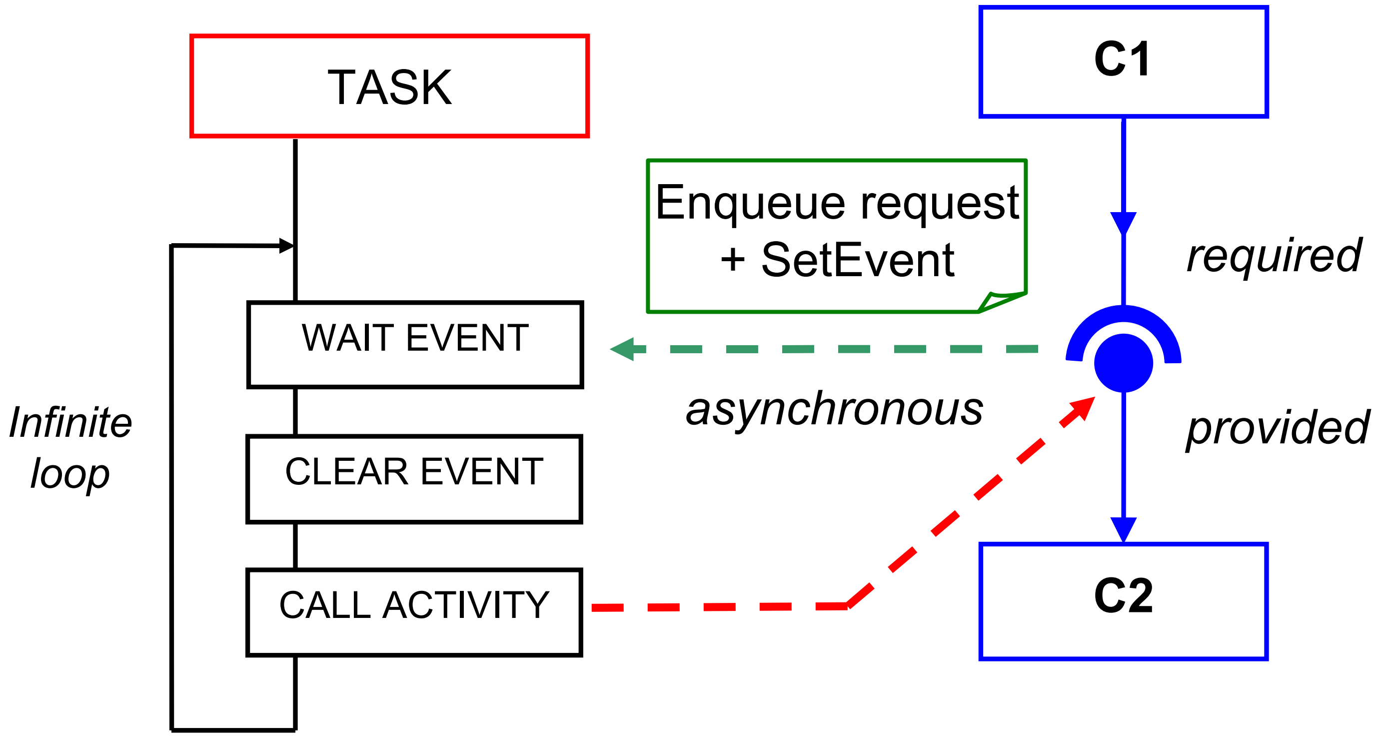


Illustration with OSEK - container services

- **Periodic activation:**
  - Achieved through the use of alarms and counters
  - Interaction with a timer module (part of framework)

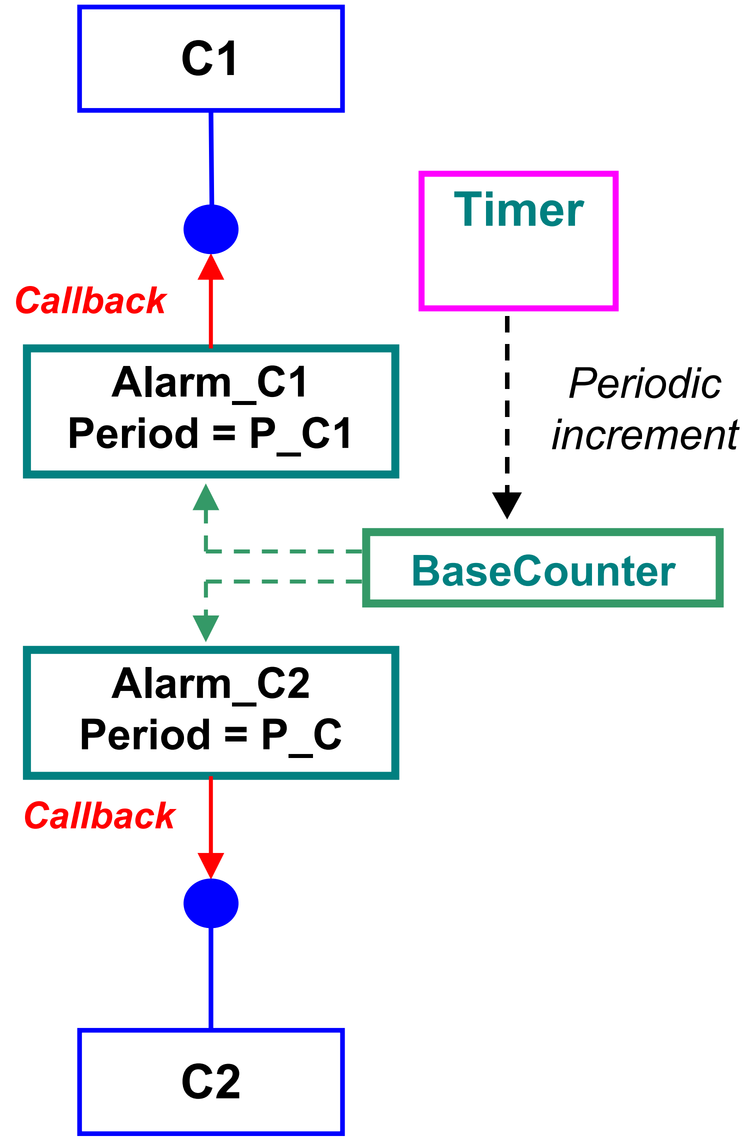
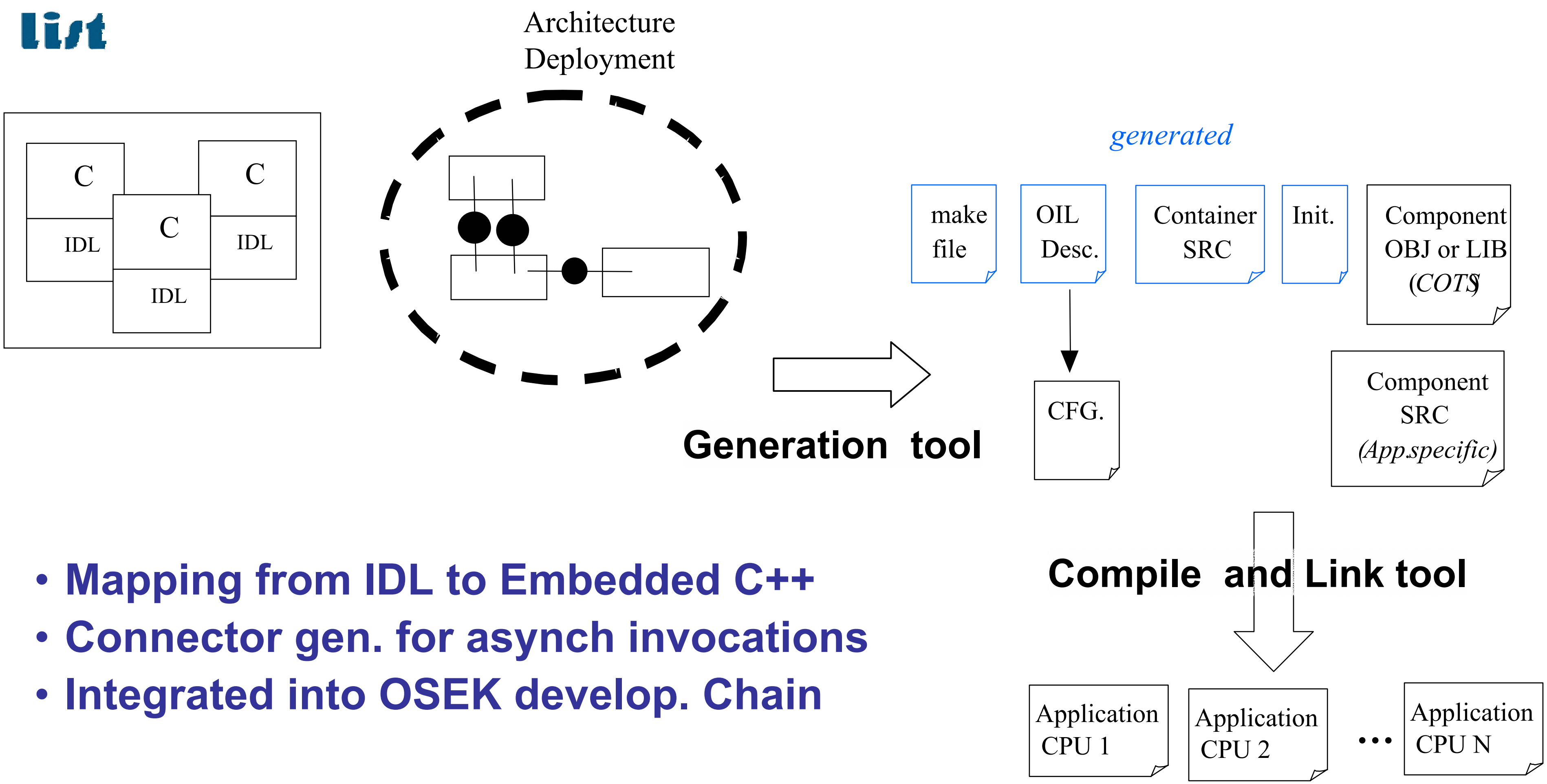


Illustration with OSEK – complete generation chain



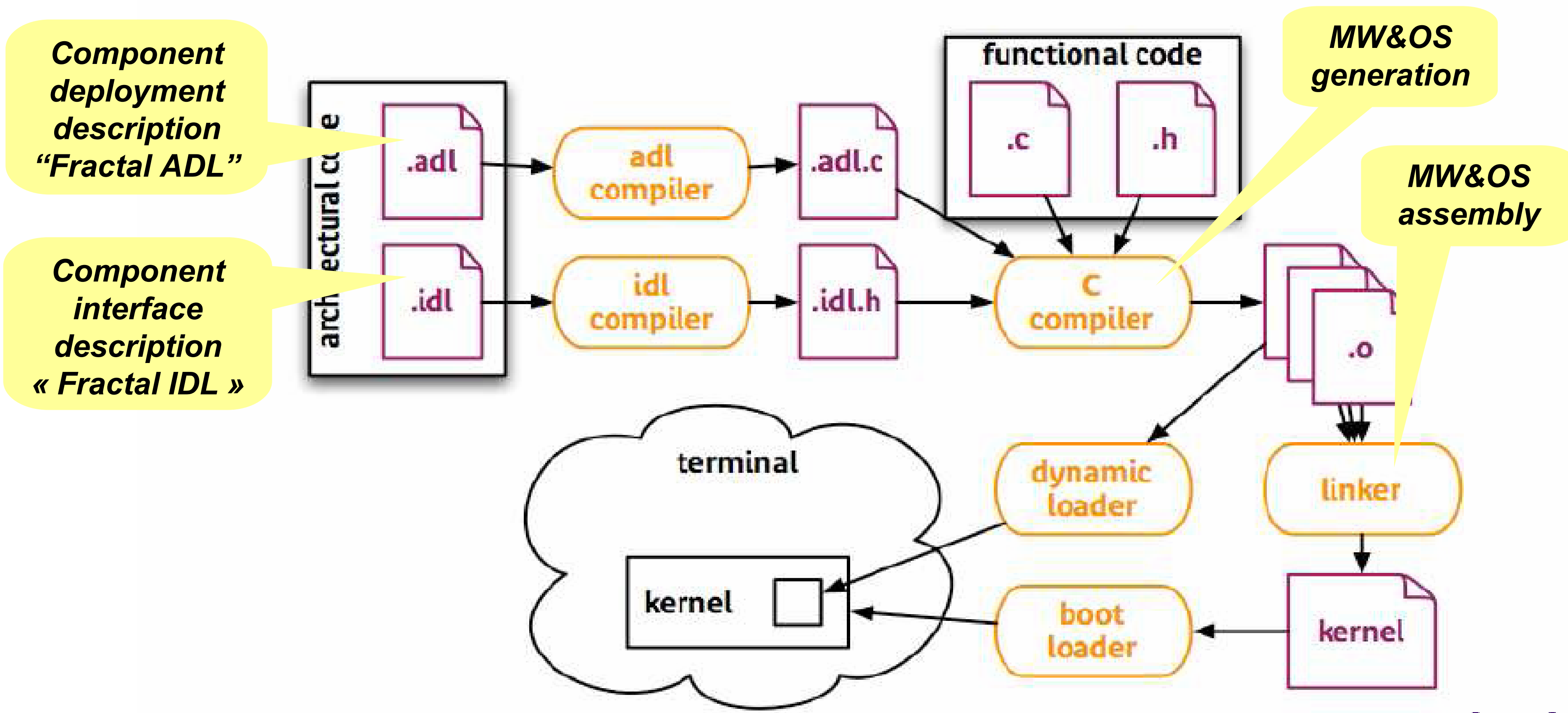
- **Mapping from IDL to Embedded C++**
- **Connector gen. for asynch invocations**
- **Integrated into OSEK develop. Chain**

- **Achieved small footprint (1 component)**
  - component ROM : 2,71 kBytes RAM : 17 Bytes
  - container ROM : 23,8 kBytes RAM : 1,43 kBytes

Conclusion

- **The (MDE) process is similar to several approaches such as (e.g.)**
  - AADL & tools
  - Fractal / Think
  - Autosar + tools

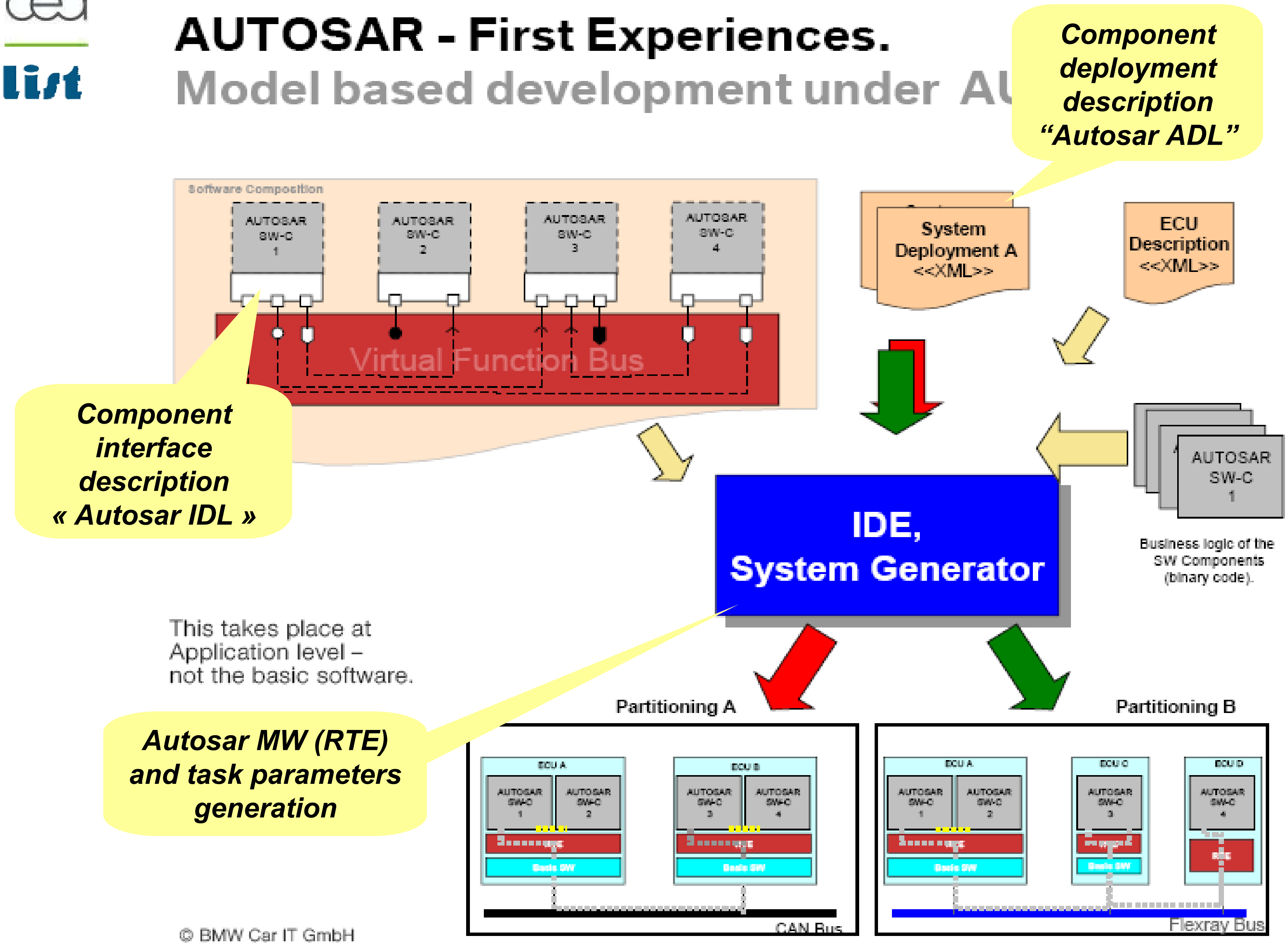
THINK build process



Collège Scientifique
France Télécom
Recherche & Développement

AUTOSAR - First Experiences.

Model based development under AU



Starting action…

➔ **Try to push some convergence on component Models and technologies**