UML Temps Réel

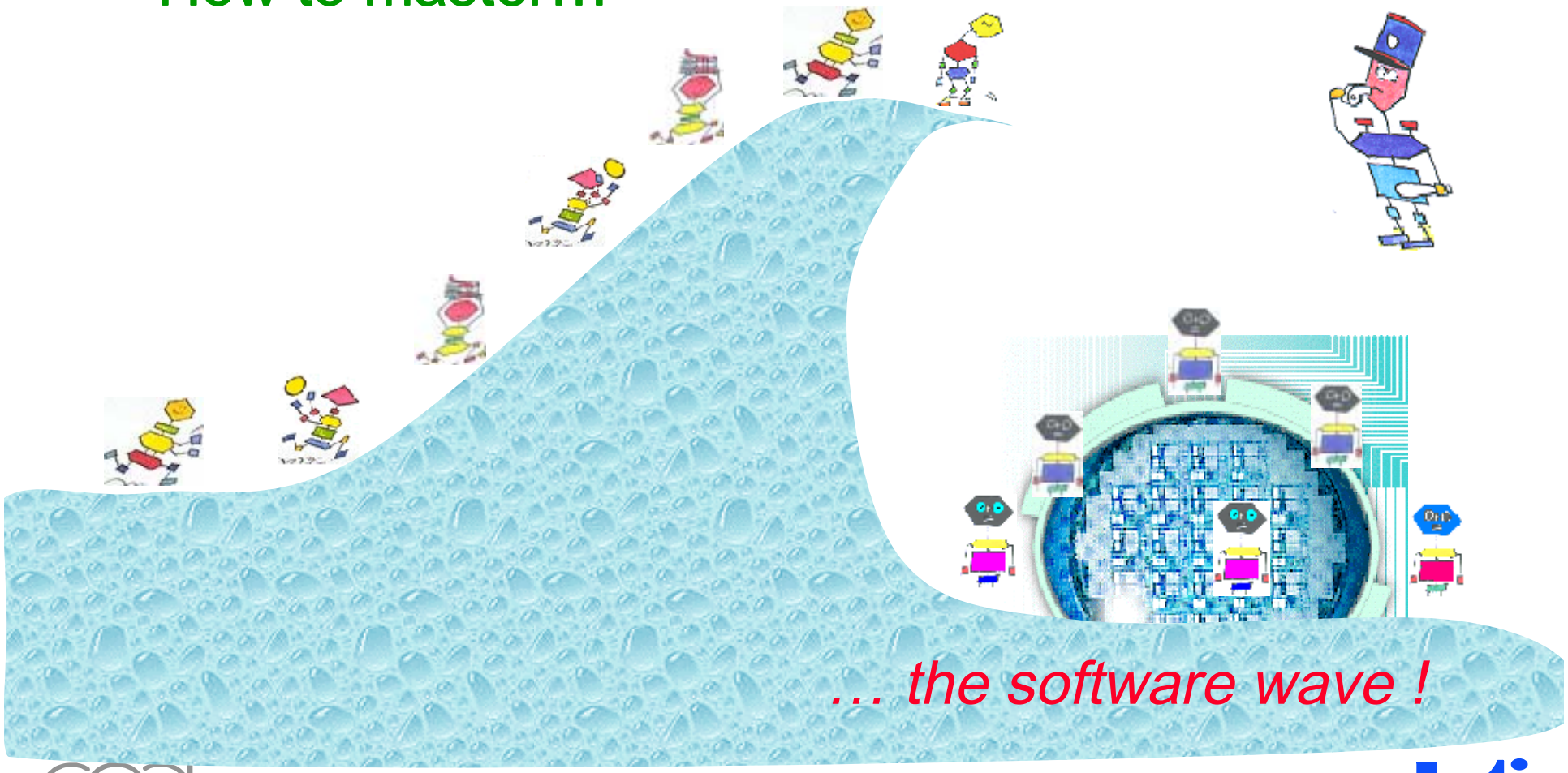
François Terrier, Sébastien Gérard

LETI (CEA - Technologies Avancées) DEIN
CEA/Saclay
F-91191 Gif sur Yvette Cedex France
Phone: +33 1 69 08 62 59 ; Fax: +33 1 69 08 83 95

Francois.Terrier@cea.fr ; Sebastien.Gerard@cea.fr

Embedded systems soon > 50 % of market

- ✓ **With more and more importance of software**
- ☞ How to master…



Beginning 99 : the PITAC report (US)

☞ Facts :

Craft practices → Low mastery of the products

… Demand for software far exceeds the Nation's ability to produce it.

… The Nation depends on fragile software.

… Technologies to build reliable and secure software are inadequate.

… The diversity and sophistication of software systems are growing rapidly.

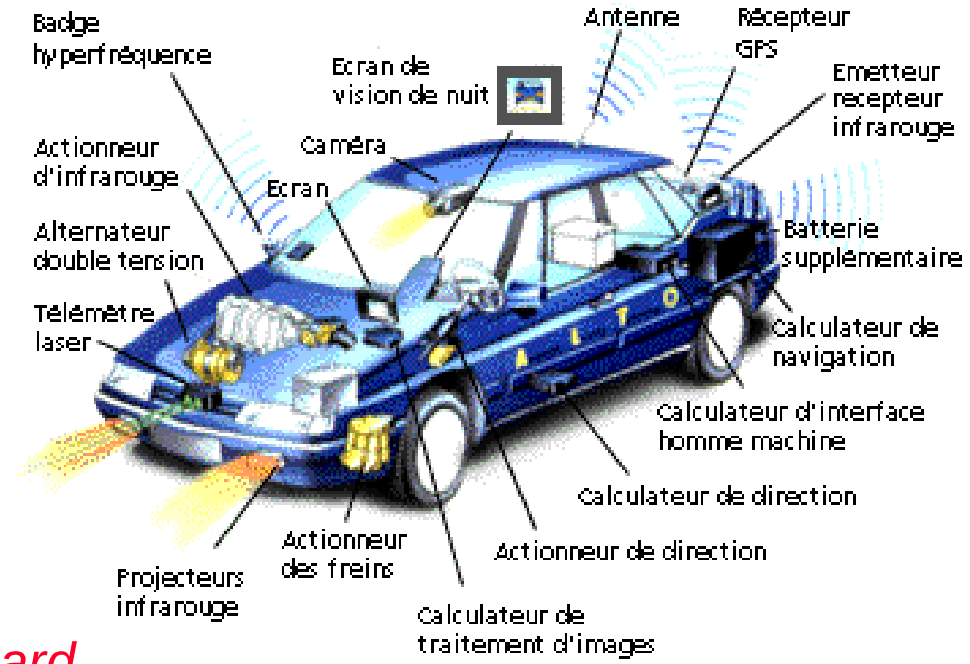
… common activities of ordinary people are based on software.

… The Nation is under-investing in fundamental software research.

… « We put the Nation at risk »

Focus on Electronics in the car industry

The car will be the first complex system
of the current life based on software



☞ **Situation :**

- ↘ Hardware cost
- ↗ Integration cost

☝ *Safety / security*

➢ *Independency between soft / hard*

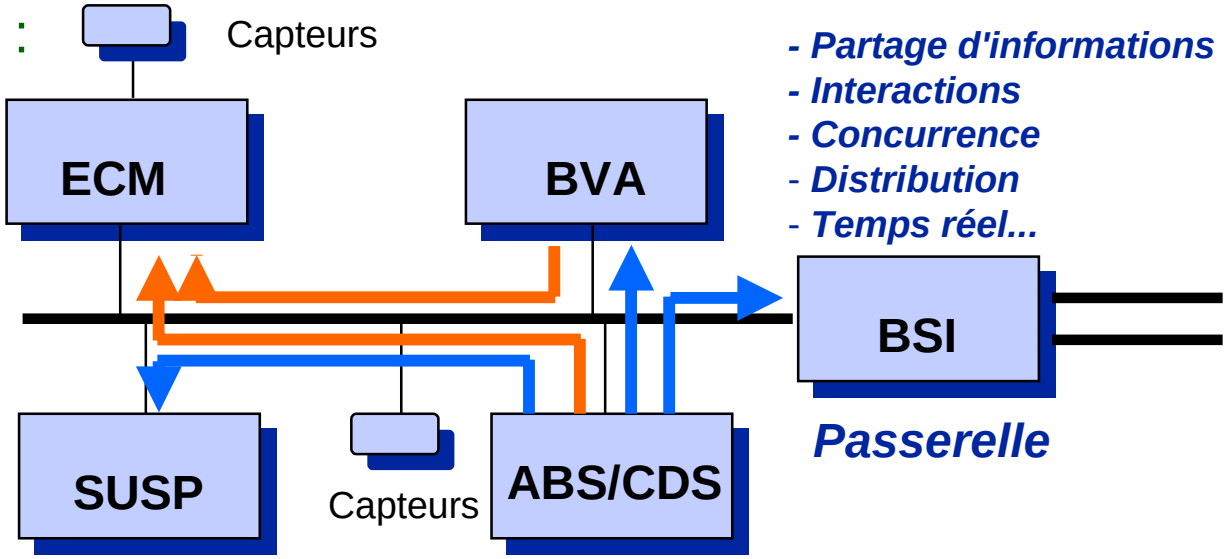
➢ *Flexibility : each person will have a different car…*

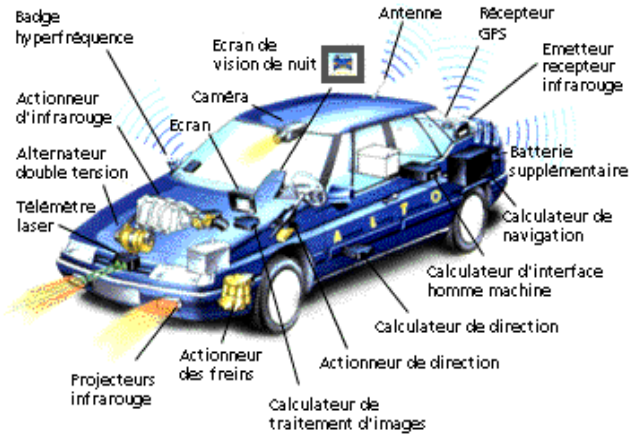
➔ Software innovation is required

Gros plan sur l'électronique automobile (source PSA)

☞ Au cœur de la question des systèmes embarqués

❖ L'électronique : 25% du coût d'une voiture d'ici 2005…





❖ Part du logiciel en ↗↗↗
« très bon rapport en valeur ajoutée… »

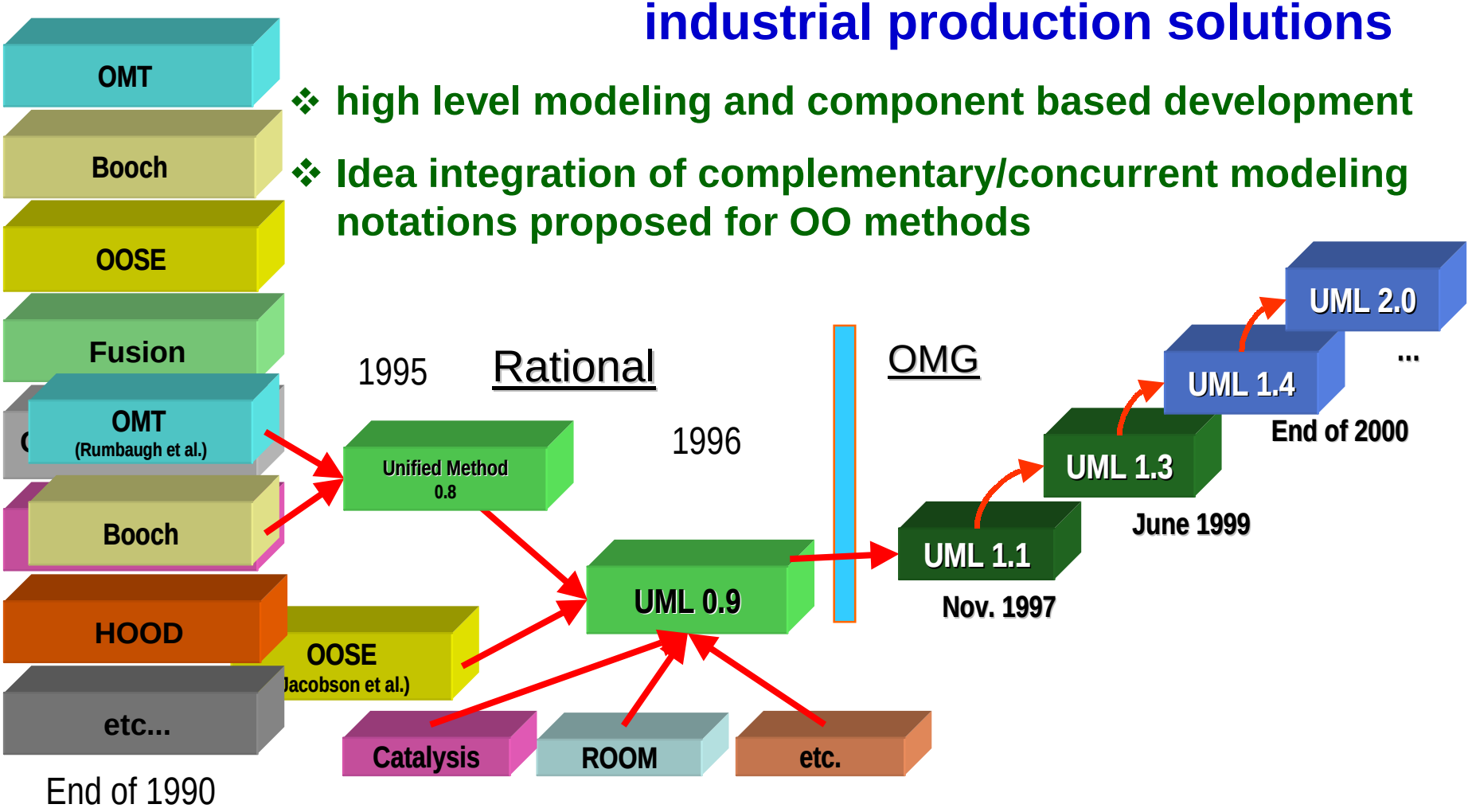
➢ *Problématique technique de système complexe*

✓ 60 processeurs sur les modèles haut de gamme !

Use of a « universal » modeling standard

☞ **We must go from craft practices to industrial production solutions**

- ❖ high level modeling and component based development
- ❖ Idea integration of complementary/concurrent modeling notations proposed for OO methods



But what about real-time systems ?

- **Importance of dynamic in such system requires the specialization of the modeling language**

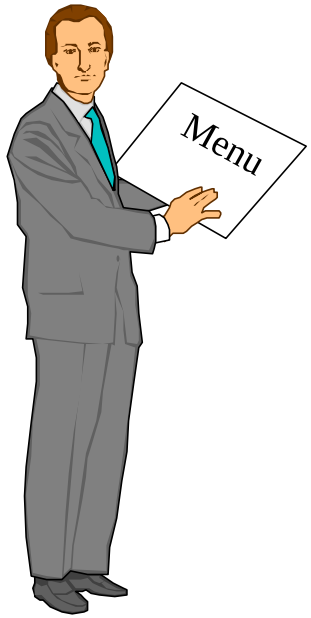
- **Solutions have been developed to integrate current practices into the UML OO framework:**

  - ✓ **More or less advanced levels of integration of real time and object paradigms**

- **Variability of the practices of real time domain depending on the context:** ***small embedded system or installation control and command, production automaton, distributed systems, safety critical systems, telecom, high performance computing…***

  - ❖ *Low level of automatic integration of the « good practices »*

Plan of the presentation

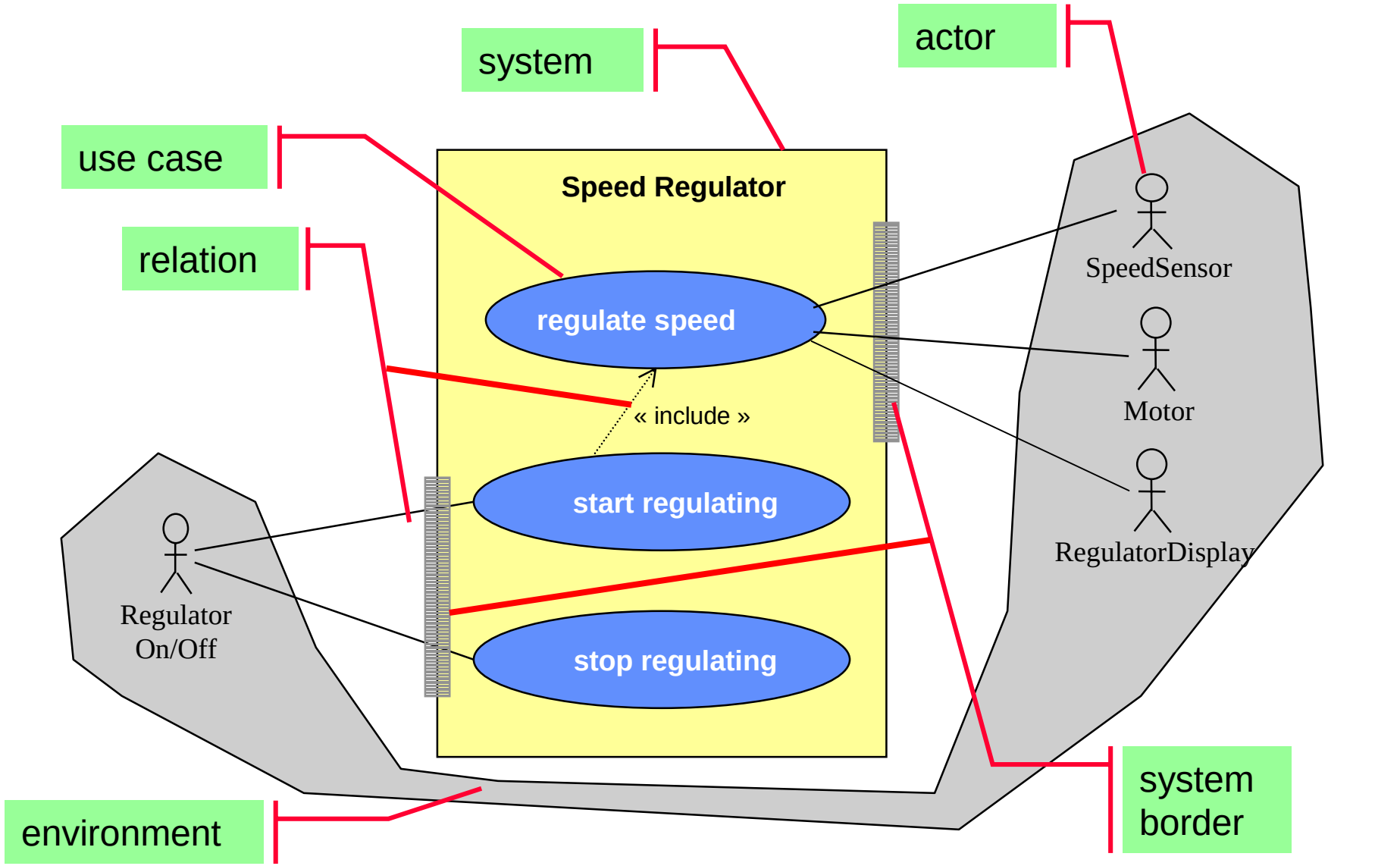


- Current status of UML 1.3
- Some proposals
  - ARTiSAN / Real Time Studio
  - RT-UML / Rhapsody tool
  - UML/SDL tools association
  - UML-RT / ROSE-RT tool
- Toward a stronger paradigm integration
  - The ACCORD/UML approach
- Information & Prospects

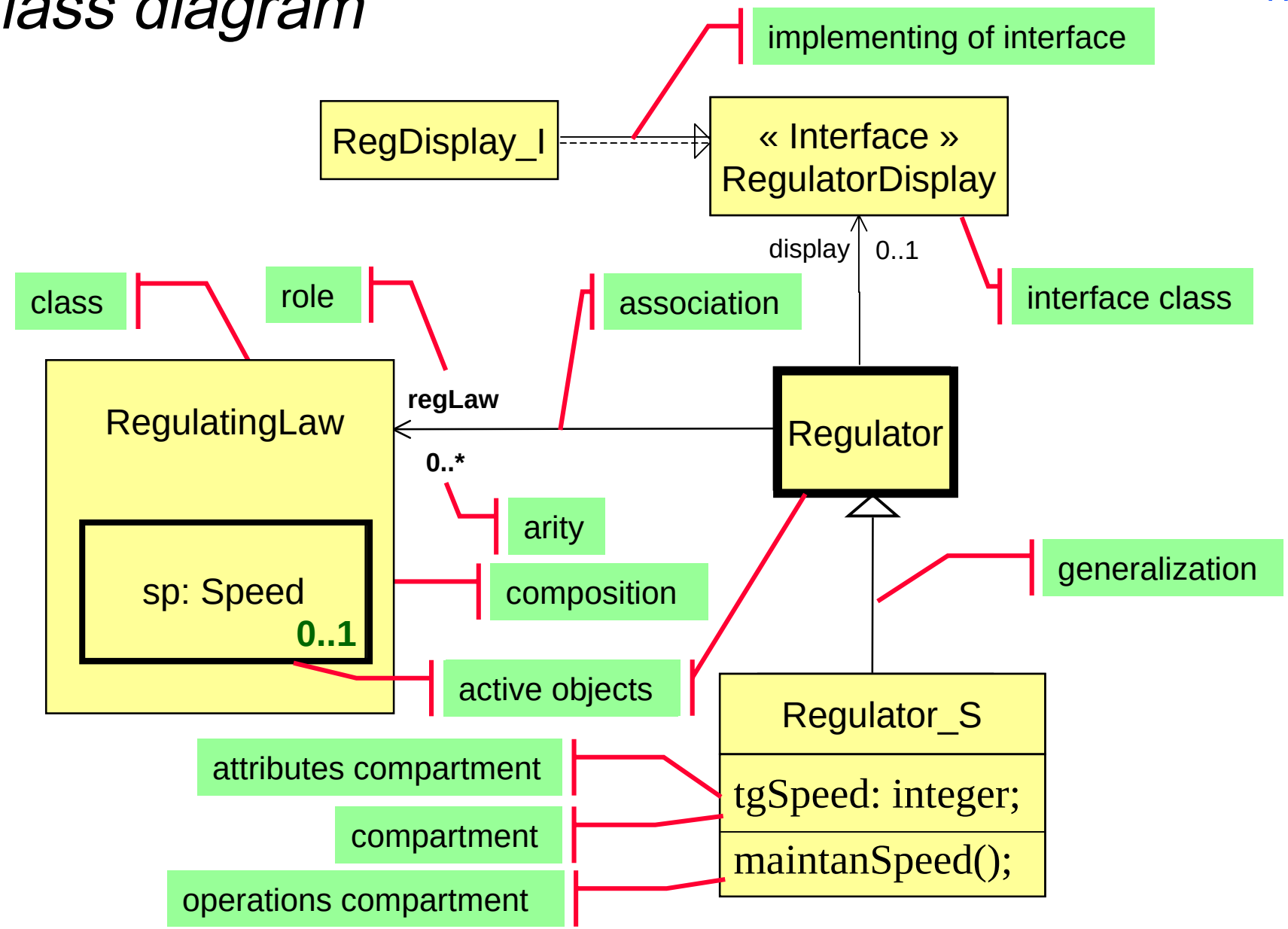
UML 1.3: essential models

- Use case diagram
  - Requirement modeling
- Class diagrams
  - Static structure
- Activity, sequence or collaboration diagrams
  - Interaction
- State machine diagrams
  - Behavior view
- Component, deployement diagram
  - Structure of material implantation

Use case diagram

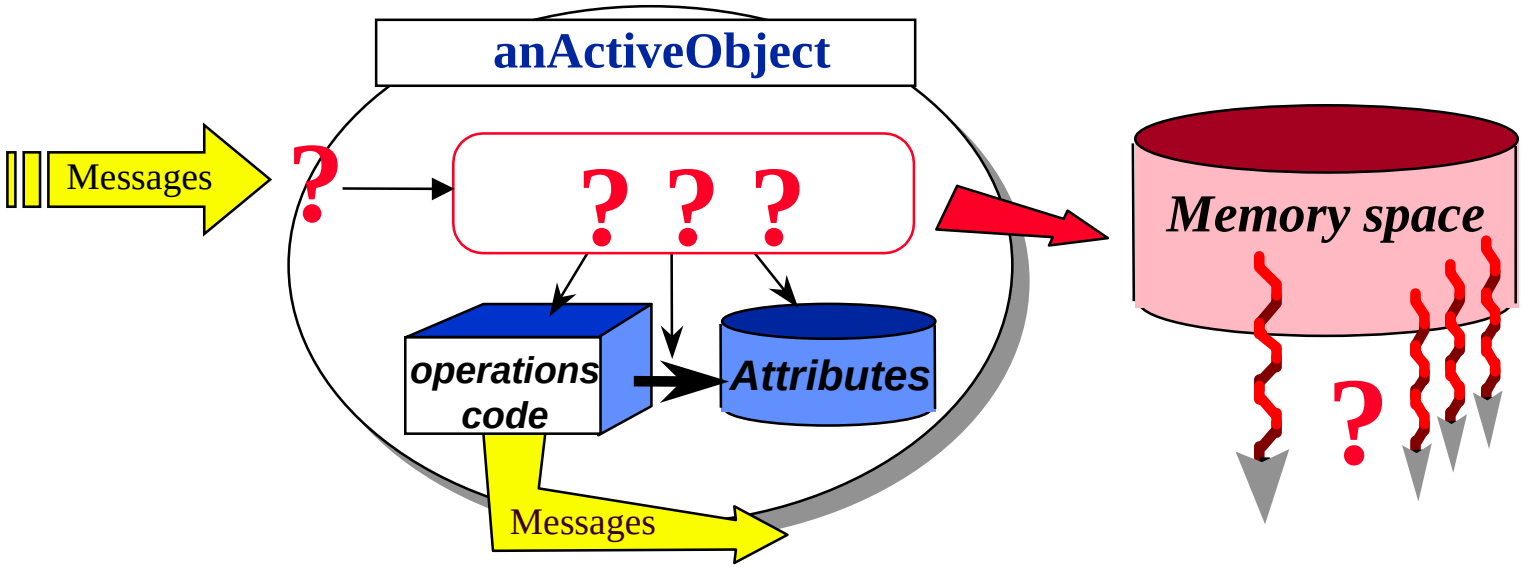


Class diagram



The UML concept of active object

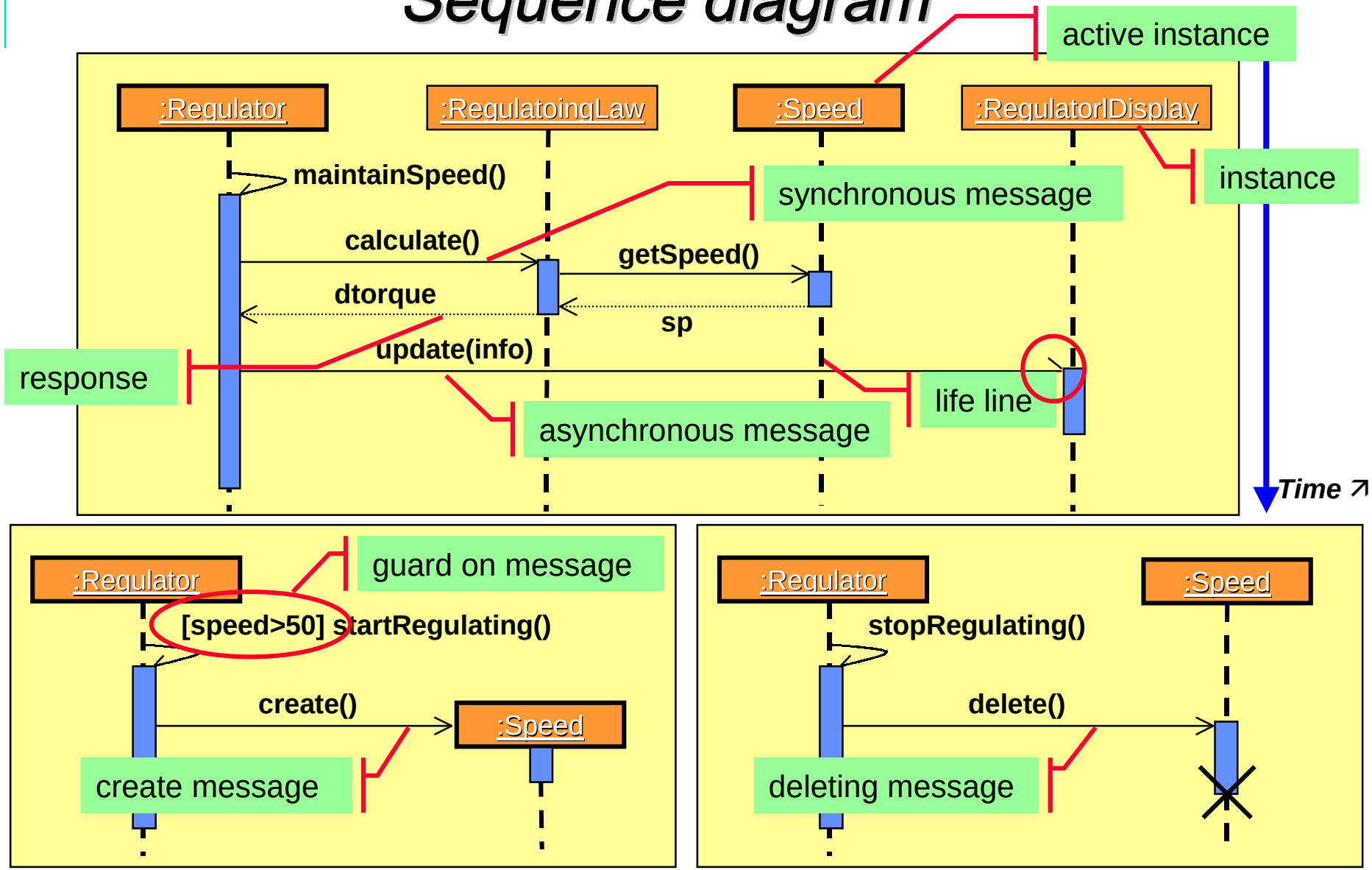
- They are objects having an *(only ?)* independent processing resource (« thread », « process » or other)



- Behavior not well defined: *connection between messages processing and use of the processing resource are not defined*

- *are they just an Object Oriented view of « tasks » ?*

Sequence diagram



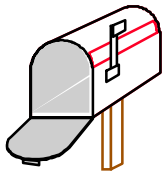
UML communication mechanisms

- Communication: only by message passing
  - A message = an action + an event
    - Point to point communication or possibility to have a set of targets
- Two types of message sending
  - Operation call (CallAction + CallEvent)
    - Synchronous/asynchronous, input and output parameters
  - Signal sending (SendAction + SignalEvent)
    - Asynchronous communication, input parameters only
- No specific communication mechanisms for the active objects…

UML state machines

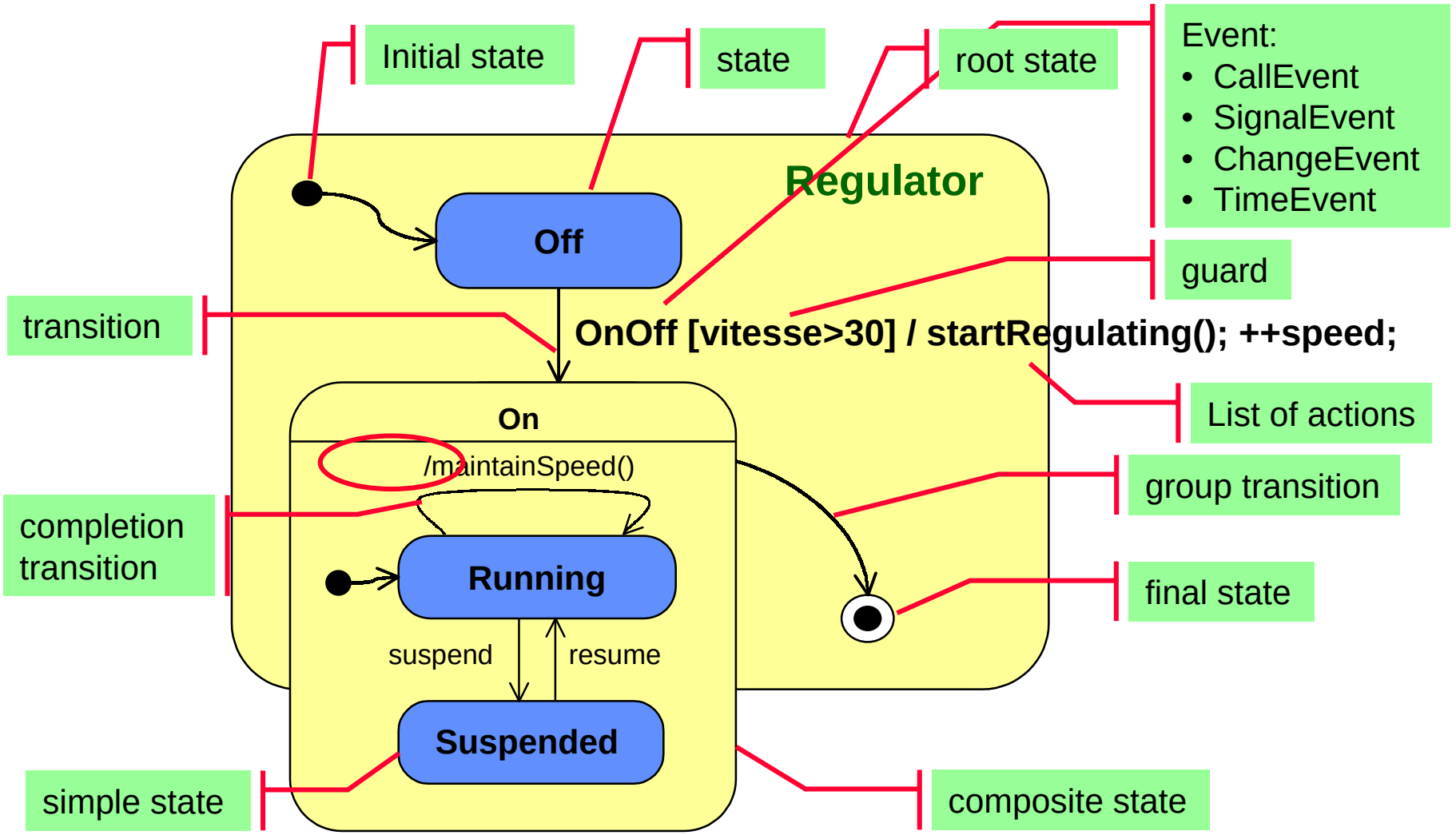
- Each object may have a state machine (or even several)
  - Transitions can have four triggering types of events (or none):
    - Object operation call: "CallEvent"
    - Signal receipt: "SignalEvent"
    - Condition becoming true: "ChangeEvent"
    - Date occurrence: "TimeEvent"

} *messages*

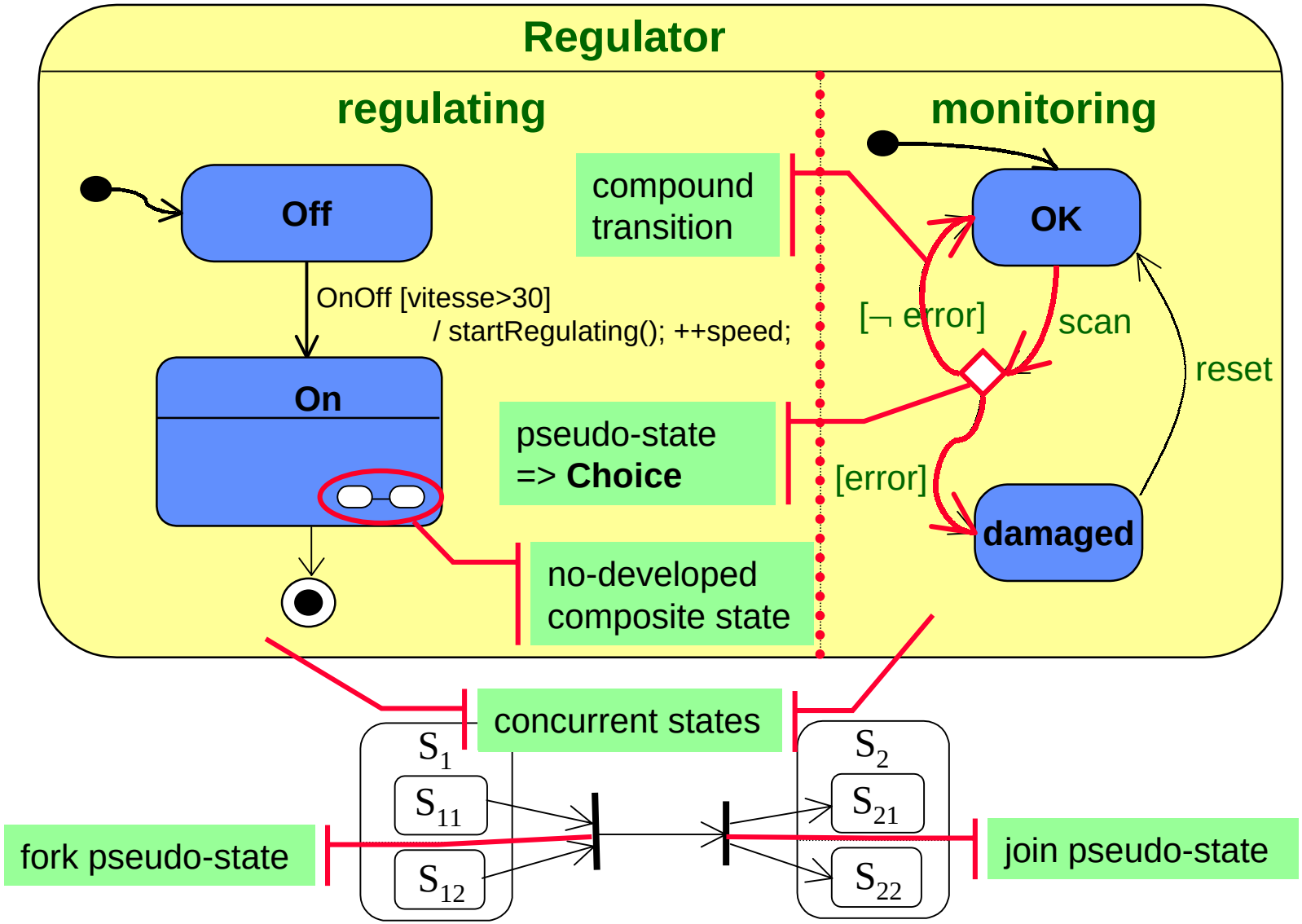


- A state machine has a queue to store incoming events
  - Storing mechanisms and its extraction protocol has to be defined (implemented) by developers
- Current event processing has to be completed before handling of next incoming event
  - Run To Completion (RTC) assumption
  - No distinction between internal and external events

State machine diagram (1/2)



State machine diagram (2/2)



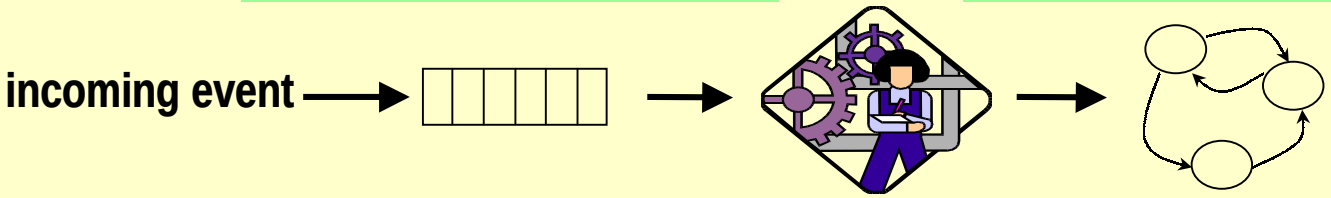
State machine semantics

- UML state machines = hypothetical machine that:
  - queues event
  - dispatches event
  - processes event

Run-To-Completion:
only one event at a time

Queue
=> Saves incoming events

Processor
=> Consumes selected event



Dispatcher
=> Selects and dequeues an event

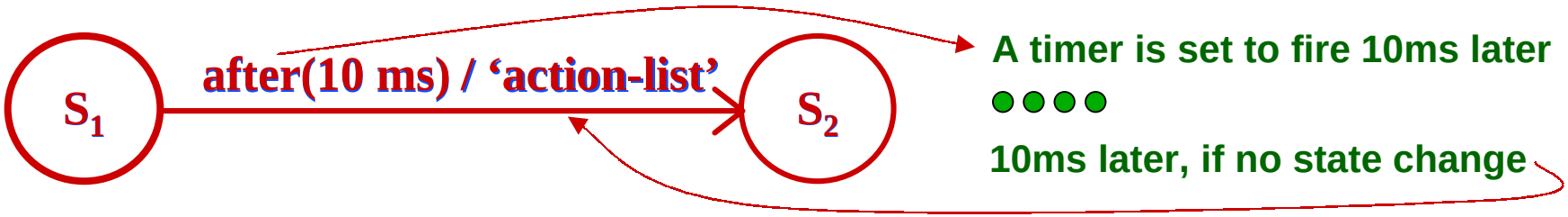
– FIFO dequeuing (commonly used by OO tools)

UML 1.3 & real-time specification (1/3)

☞ Timer setting on transitions of state machines

- ❖ TimeEvent
  - after ⇒ relative moment in time
  - when ⇒ absolute moment in time

☞ Semantics

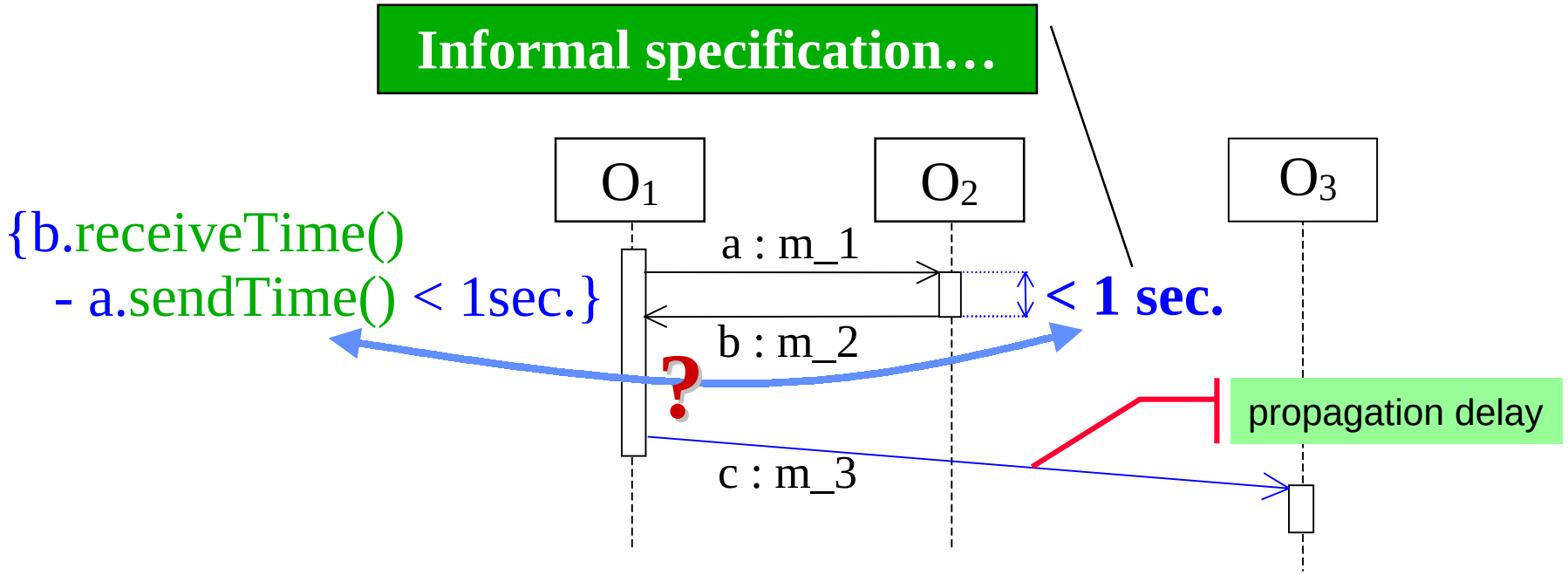


☞ Drawbacks

- ❖ Specification is performed through implementation mechanisms
- ❖ TimeEvent handled as other events ⇒ non-determinist

UML 1.3 & real-time specification (2/3)

- Dates & timing intervals in sequence diagrams



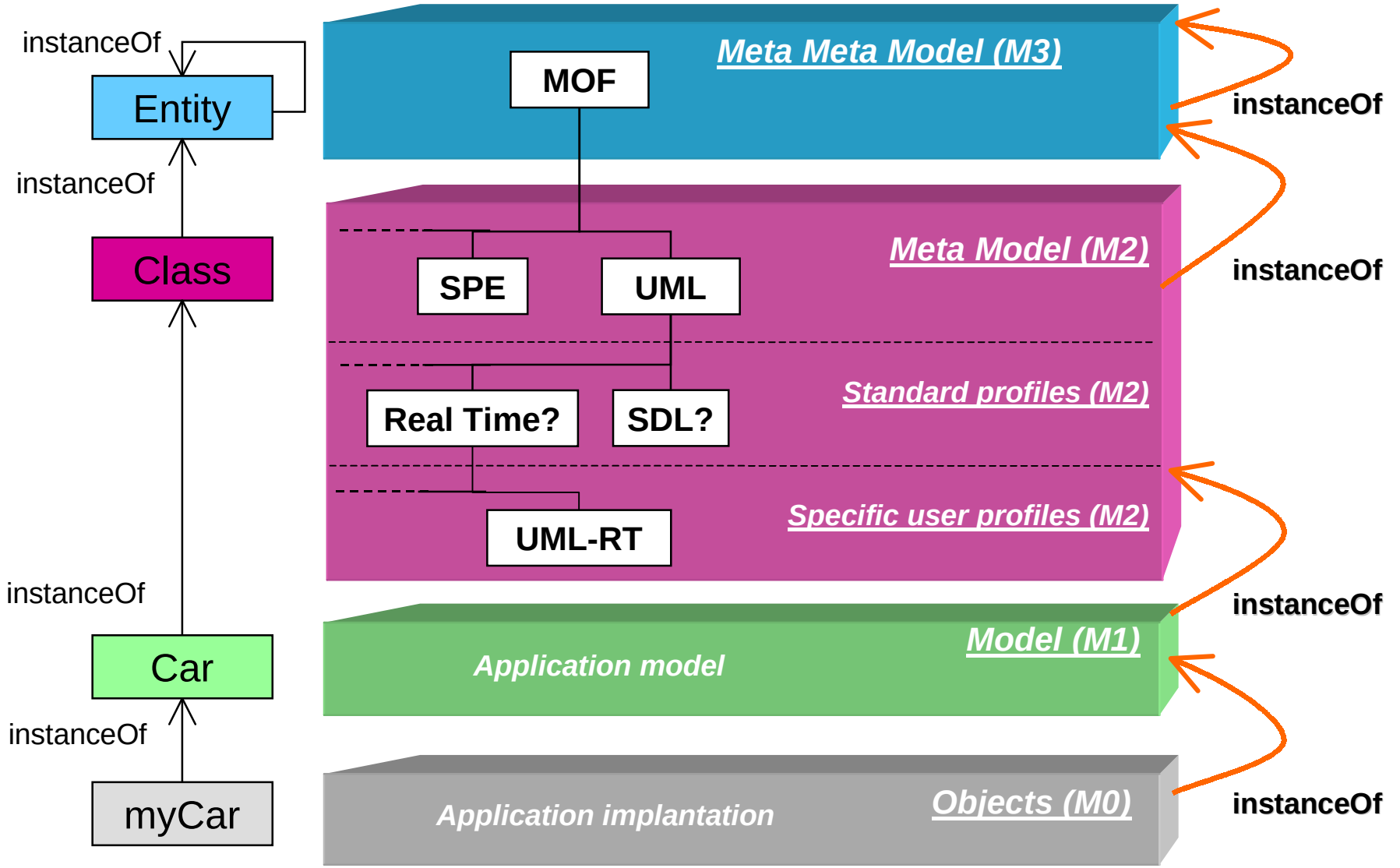
  - No relation is defined between these specifications neither with the rest of the model nor with the system scheduling policy

UML 1.3 & real-time specification (3/3)

☞ Ongoing works at the OMG

- Proposal of an « Action language semantics »
  - Must integrate proposals from all the submitters (« SDL » domain consortium, Rational and other tool vendors)
  - Standard only on semantics not on notations…
  - Several times postponed…
- Proposal of a real-time profile on
  - Time semantics, scheduling and real-time concepts at implementation level → a UML virtual machine…
- Works are also in progress on Profile formalization, and on pattern description and use…

UML meta-model: four levels of instanciation



UML 1.3: specialization mechanisms

- Tagged value
  - ✓ Properties added to
- Constraint
  - E.g., {documentati
- Stereotypes
  - ✓ Addition of « well-Fo
  - E.g., {destroyed}, {new} or {transient} on Instance
  - ✓ Indirect add of elements to the meta-model
    - E.g., « thread » or « process » on Classifier

Set of tagged values and of constraints specializing an element of the meta-model

Organisation needs

⇨ notion of Profile in UML 1.3

UML profile definition

☞ Objective

Specialization of a standard meta-model (e.g., UML) into a specific meta-model dedicated to a given application domain.

☞ A profile ca

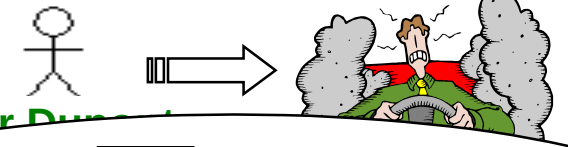
- ✓ Selected ele
- ✓ Extension me
- ✓ Descriptions o
- ✓ Additional notatio
- ✓ Rules for model translation, validation, presentation

Fundamental meta-classes on

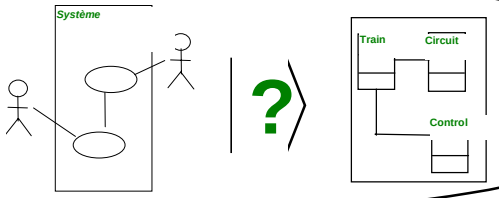
Stereotypes, tagged values,

Clarification of
« Semantics Variation Points »

e.g.:



e.g.:



UML-based existing approaches for RT

☞ Real-time systems [J. A. Stankovic 1988]

« Real-Time systems are those systems in which correctness of the system depends not only on the logical results of computations, but also on the time at which results are produced. »

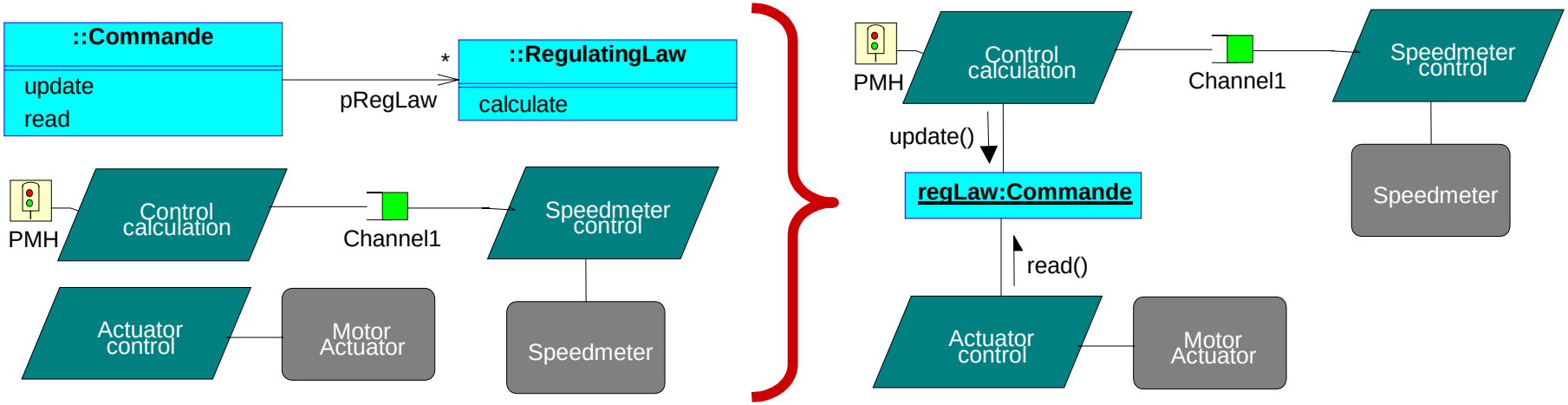
RT qualitative properties			RT quantitative properties
Concurrency	Behavior	Communication	Time

☞ Four main industrial approaches

- UML-RT (ROOM) & Rose-RT
- RT-UML & Rhapsody
- UML-SDL & Telelogic-Tau Suite
- ARTiSAN & Real Time Studio

ARTiSAN: *two orthogonal models, weak integration*

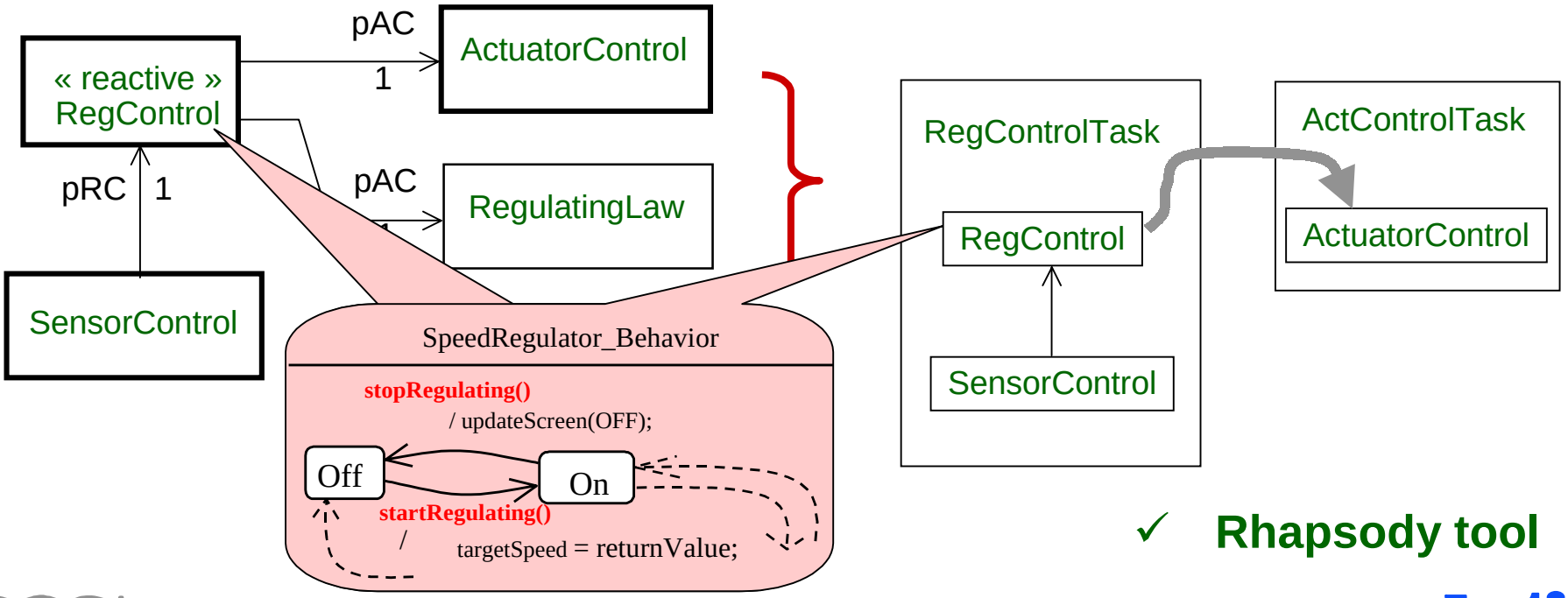
- This tool aims to offer in the same environment:
  - A classical UML modeling facilities
  - A classical task model called the concurrent model
  - An implementing stage (assignment of the objects to tasks)



- Task model has to be manually implemented
- Relation between task and object model is weak
  - Communication between object of different tasks must be implemented by the user with low level concepts
- Timing constraints must be translated on the implementation

RT-UML: *two « orthogonal » but « integrated » models*

- Tasks are identified by declaration of active objects
  - Behavior specified by state machine of reactive objects
  - Reactive objects are assigned to active objects…
  - Communication by message between reactive objects



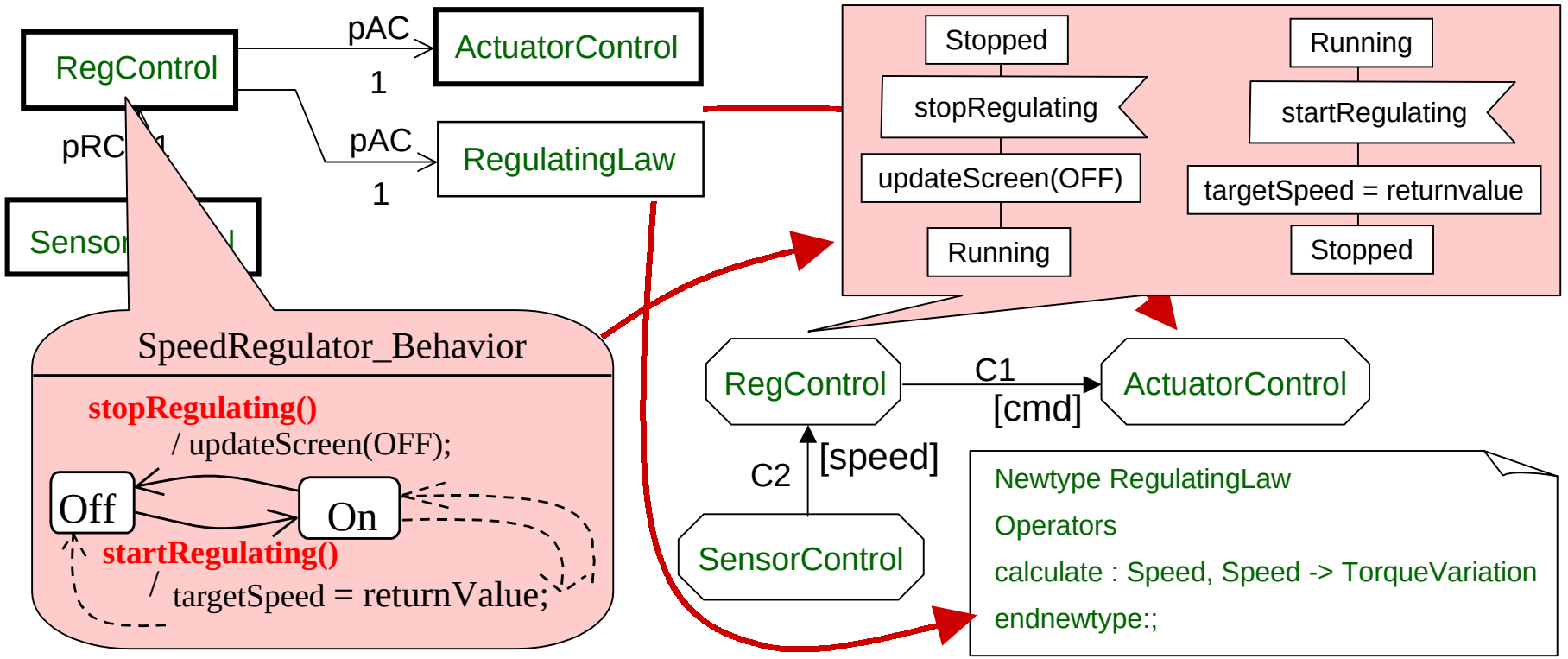
✓ **Rhapsody tool**

RT-UML: non homogenous features

- Different processing semantic between signals and operation calls
  - Signals: parallel execution in thread of receiver object
  - Operation calls: execution in thread of sender object
    - Under control of the object state machine « triggered op. call »
    - Without control of the object state machine…
  - Concurrency is only managed through RTC assumption
  - Return value of operation call can be defined… under responsibility of the caller
- Priorities can be assigned to the active objects
  → Not on the event themselves…
- Timing constraints must be implemented

UML/SDL: association of UML analysis and SDL design

- UML modeling is used at the first modeling stage
  - Use cases, sequence and class diagrams are defined
  - Active object declarations → SDL processes
  - Passive objects → Abstract Data Types of SDL

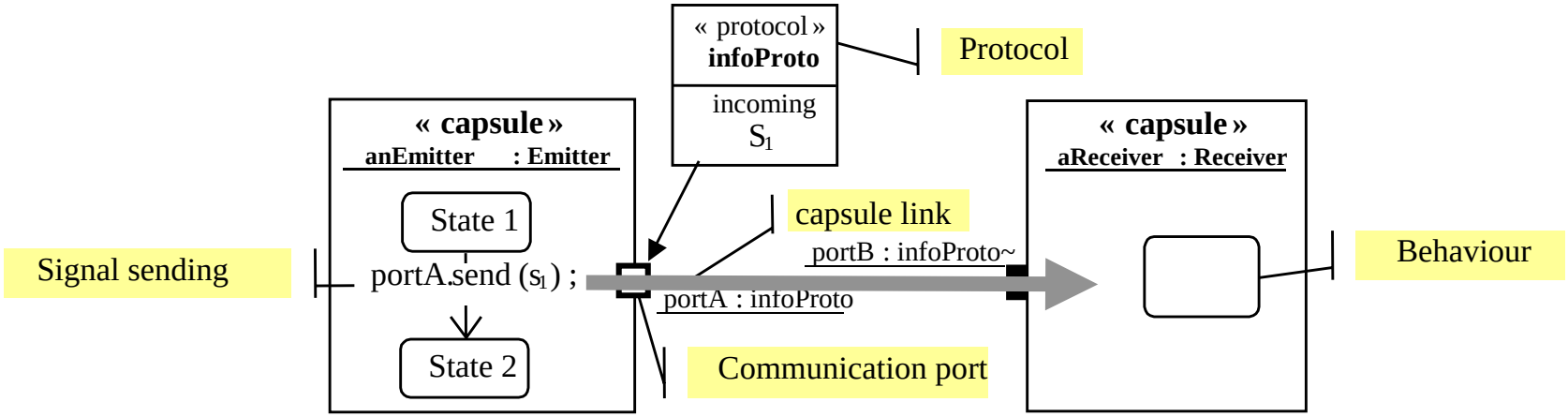


UML/SDL: mapping is under user responsibility

- The developer has to clarify the mapping of:
  - UML state machines → SDL specifications
  - Use of shared passive objects in UML model → SDL
  - Active object communication → SDL signal exchange
- Independency between model and implementation?
- Timing specifications:
  - → only with low level implementation mechanisms
    - Timers, SDL priorities…

UML-RT: attempt to integrate task and object paradigms

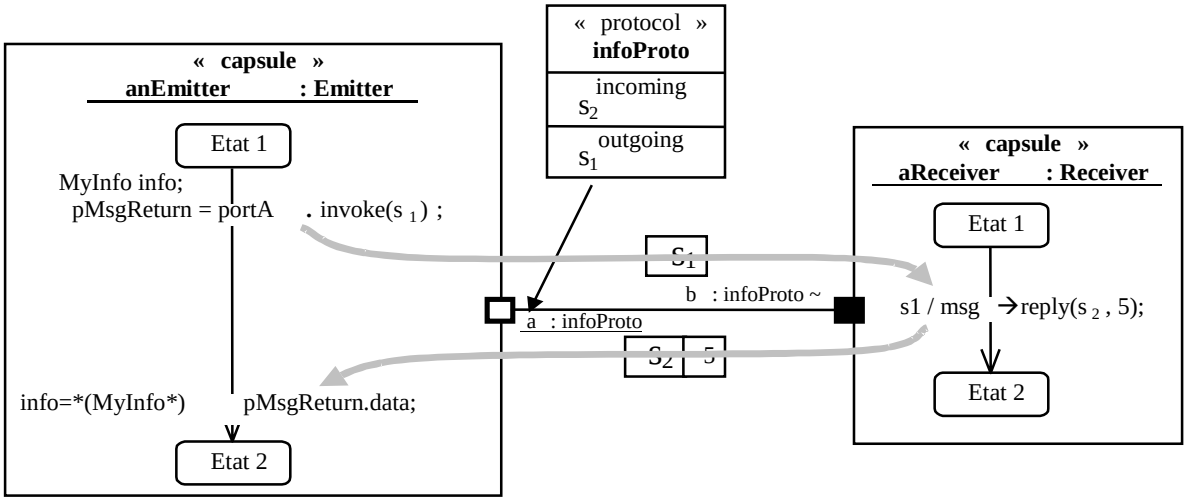
- « capsule » stereotype identifies active objects
  - They will be assigned to task at implementation stage
  - They have state automaton
- Communication is performed through « ports »
  - Sending of signal via port of communication



- ✓ Defines the set and protocol of the exchanged messages

UML-RT (ROOM/ObjecTime → ROSE-RT): Synthesis

- Communication based on specific concepts
  - Output parameters: return values managed by sender



- Priorities can be assigned on messages
  - One message queue by priority (five levels) :

    At implementation level mapping with task priorities must be managed by hand

- Timing specification through low level mechanisms

Synthesis on current offers

☞ Weak integration of object and real time…

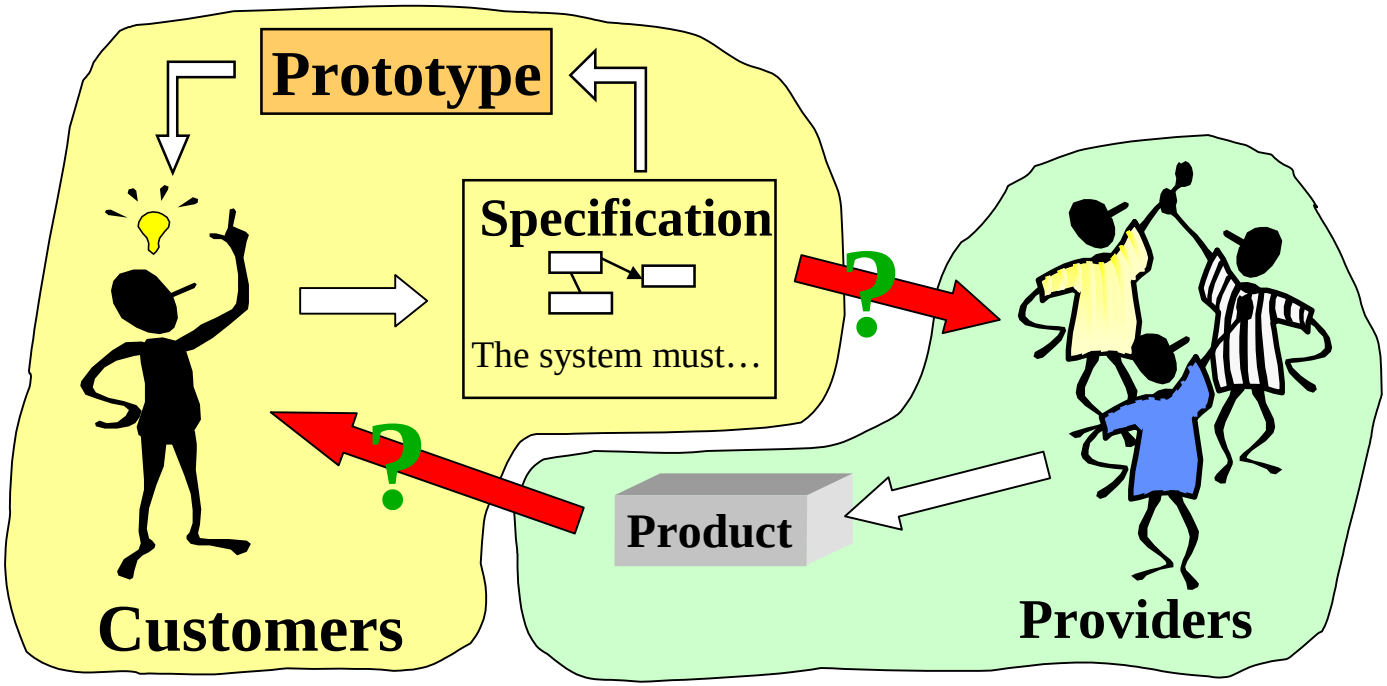
- Two very different models (e.g., ARTiSAN, UML/SDL)
- Behavior lies on operation and signal processing but with poor links to the usual object interface
- Focus is made more on signal than on operations that leads to behavior specification mixing up control action at object level and processing actions at operation level
- Output parameters often hard to manage

☞ Poor facilities to express timing constraints

- Specification of timers or of priorities
- Implementation of real time constraints kept to the users
- Sometimes difficulties to map model constraints on RT-OS
  → model / task priorities with OS priority management policies

Larger market → New users → New needs

- **Customers want to specify good RT models**
  - **They want also to be able to prototype/develop the systems**

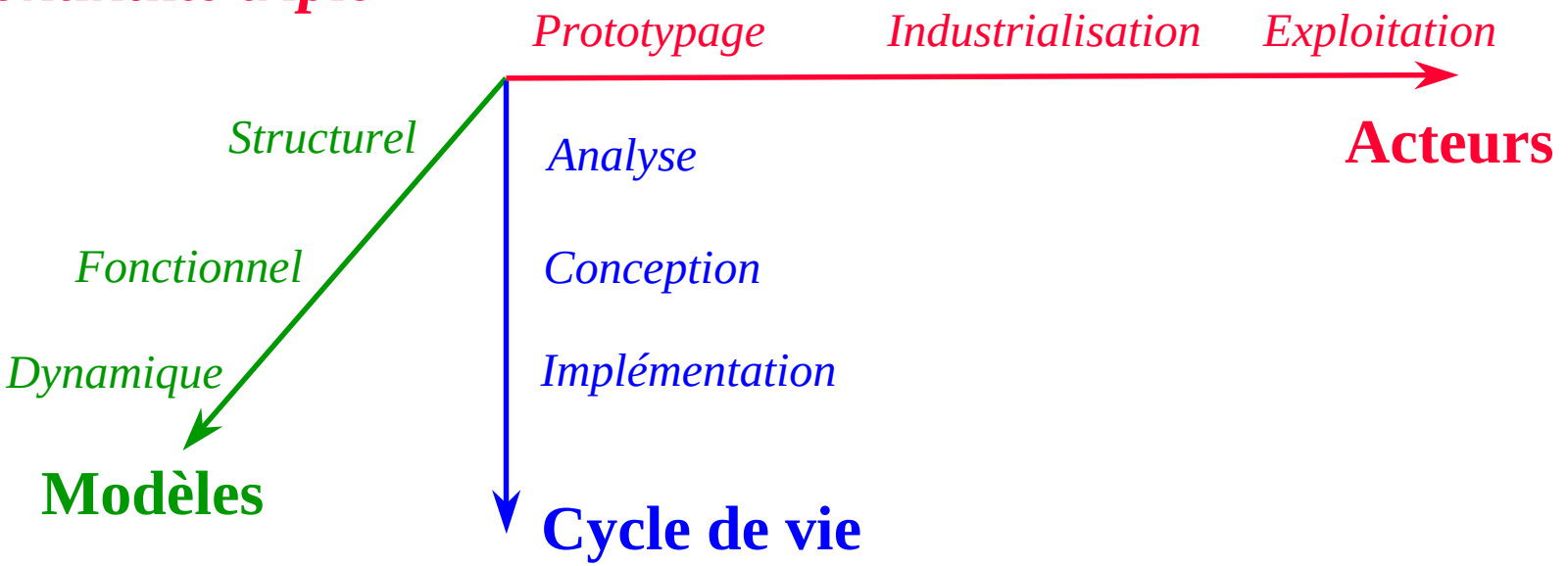


- **Notations are not sufficient: method of use is required**
  - **Continuity and "tracability" of the model is mandatory**
  - **Availability of Model and application validation is critical**

ACCORD : une méthode reposant sur un standard

☞ Objectif principal : *assurer la création d'un modèle d'application* unique, complet, homogène, cohérent et standard

⇨ ***Continuité triple***



⇒ Puis, fournir : *un support de développement pour la méthode*

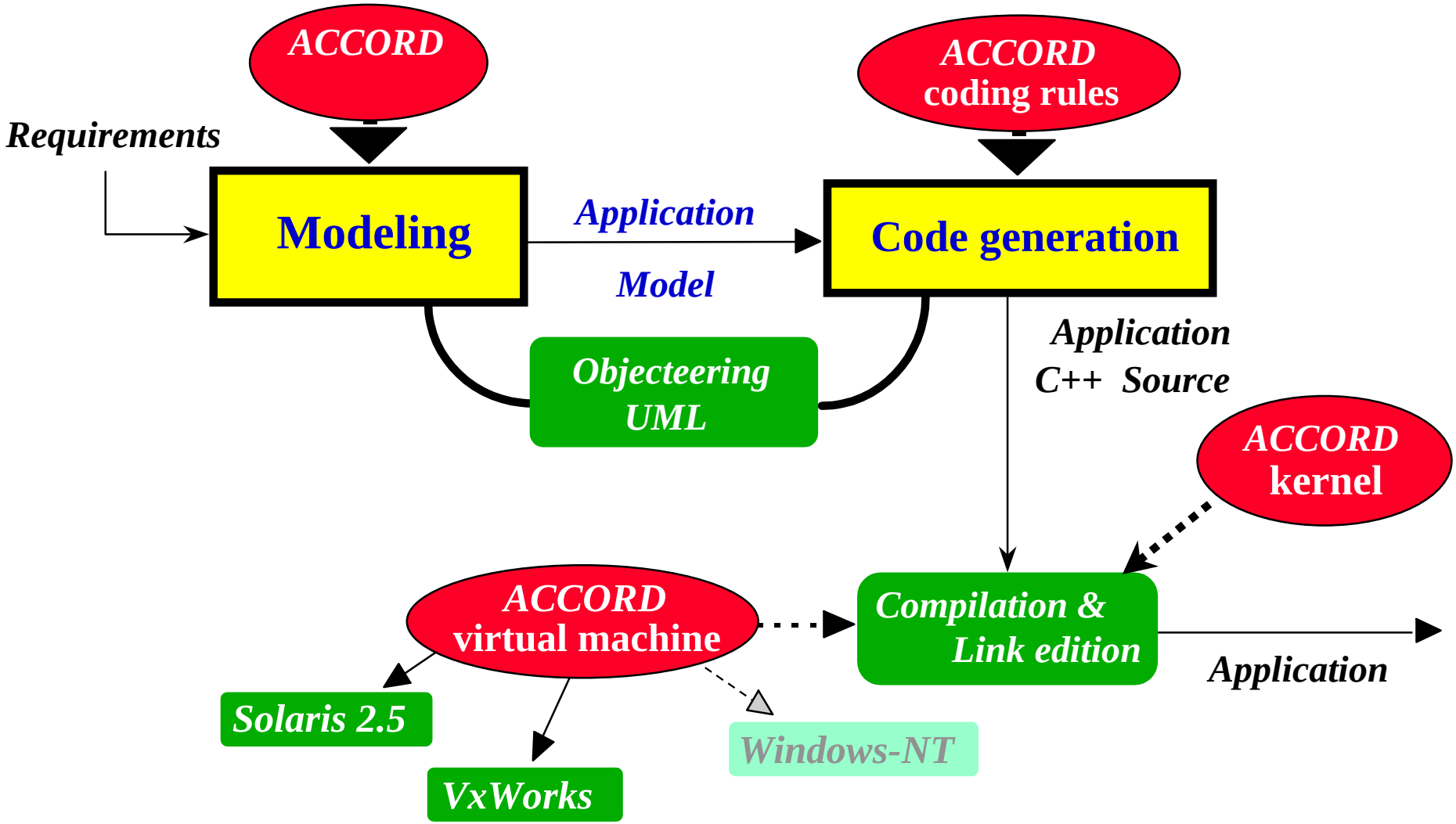
ACCORD/UML: introduction of real-time objects

- High level specification of real time behavior
  - Declarative expression of the real-time constraints
  - Hide RT-OS implementation concepts
- « Formal » models
  - Validation on model
  - Automatic prototyping code generation
  - Tuning of implementation through directives, patterns…
- Keep the usual object programming practices

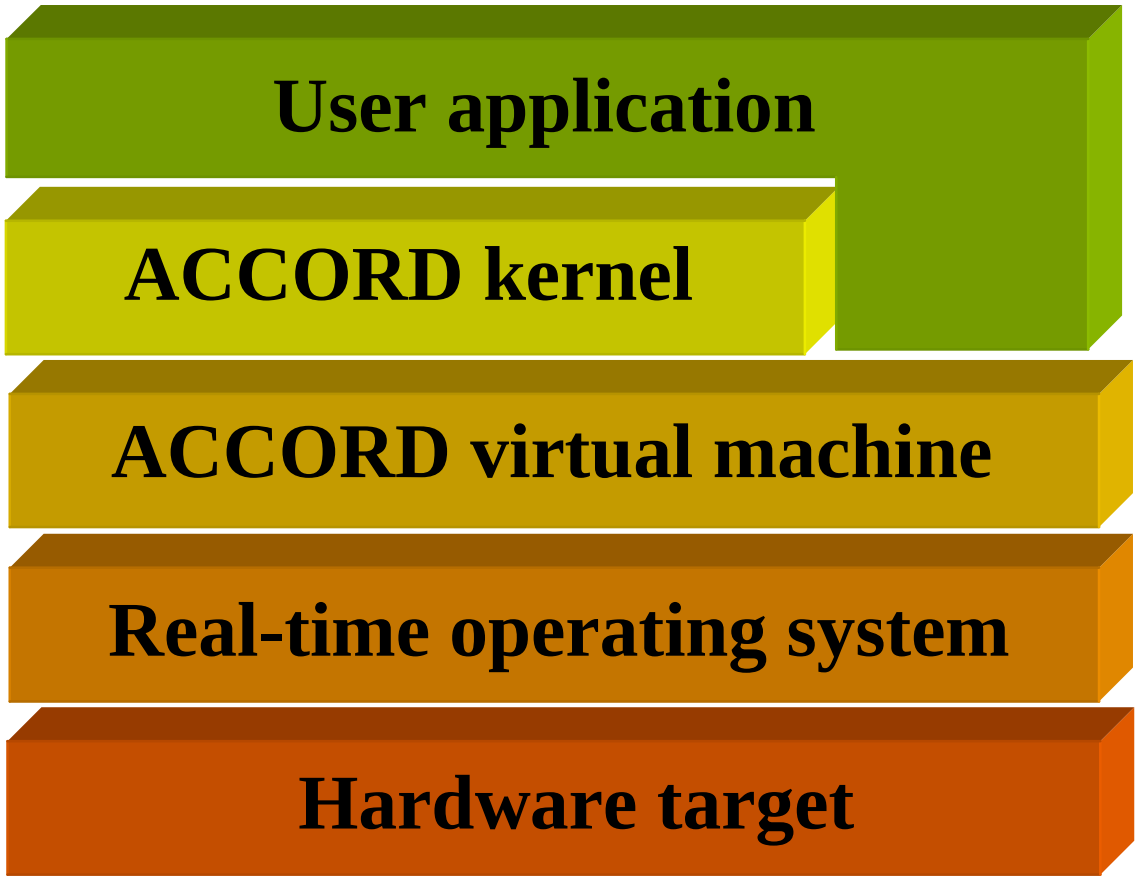
« meta-modeling » using in the ACCORD platform

- Introduction of high level modeling concepts
  - Real-Time Objects
  - Signals …
- Modeling rules introduction
  - Model structuring
  - Behavior specification
  - Signals using
- Mechanisms definition of operatring
  - Design patterns dedicated to Signals, Real-Time objects …
  - Automatic code generation
- Models analysis for validation
  - Test cases automatic generation

ACCORD : development framework



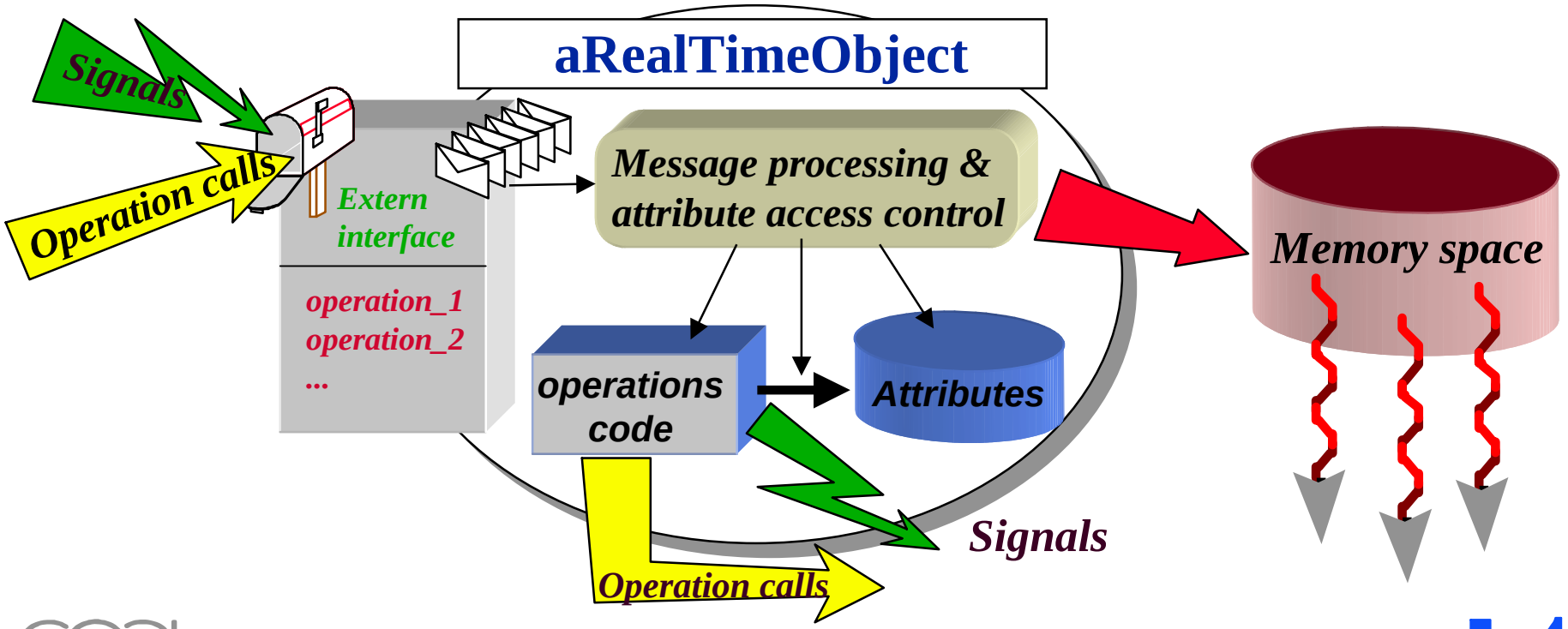
ACCORD executing architecture



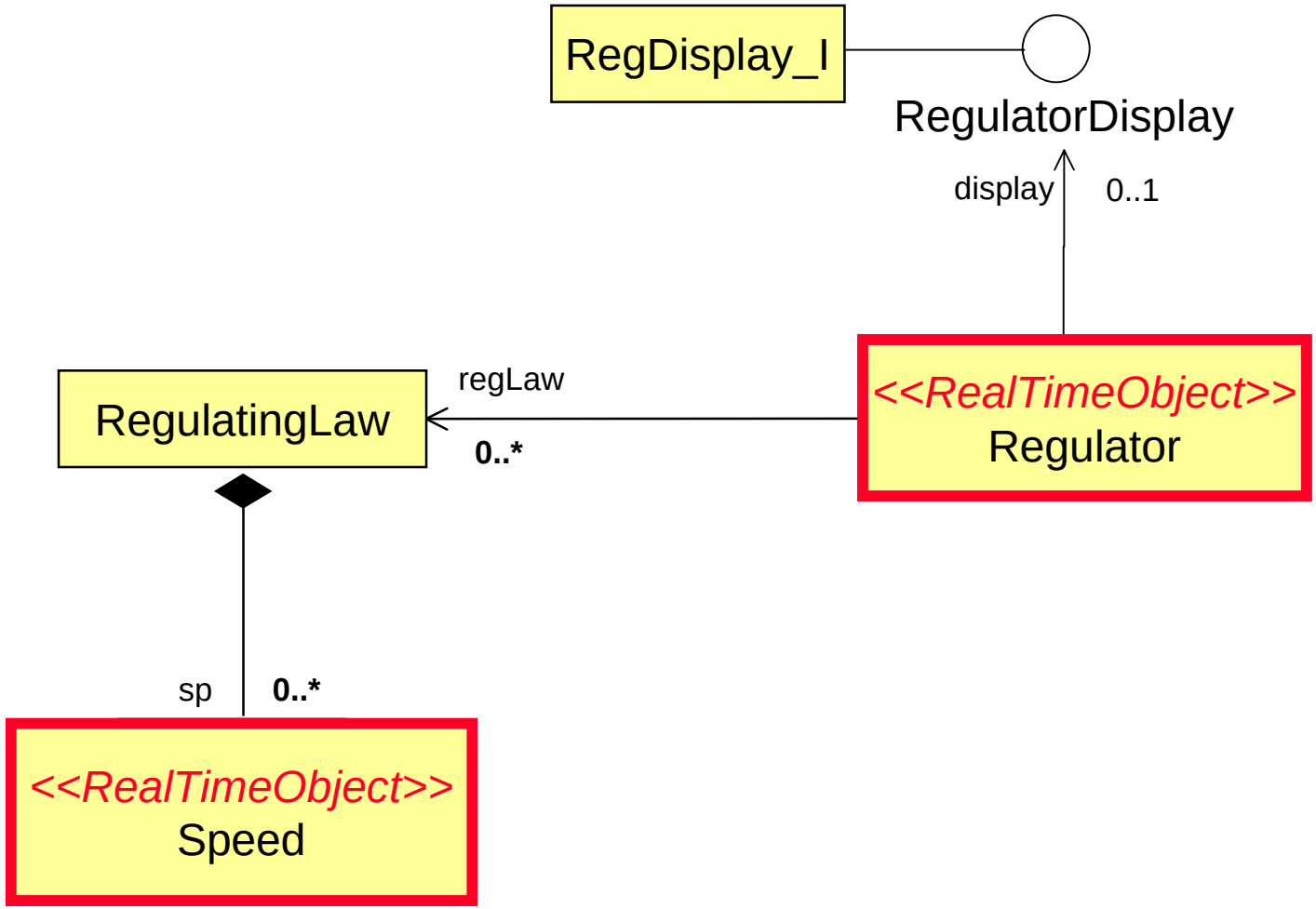
ACCORD/UML: « task » model

- Task are attached to implementation of RT objects
  → support parallel processing of the messages they receive

- User point of view : an autonomous computing entity with a standard UML object interface

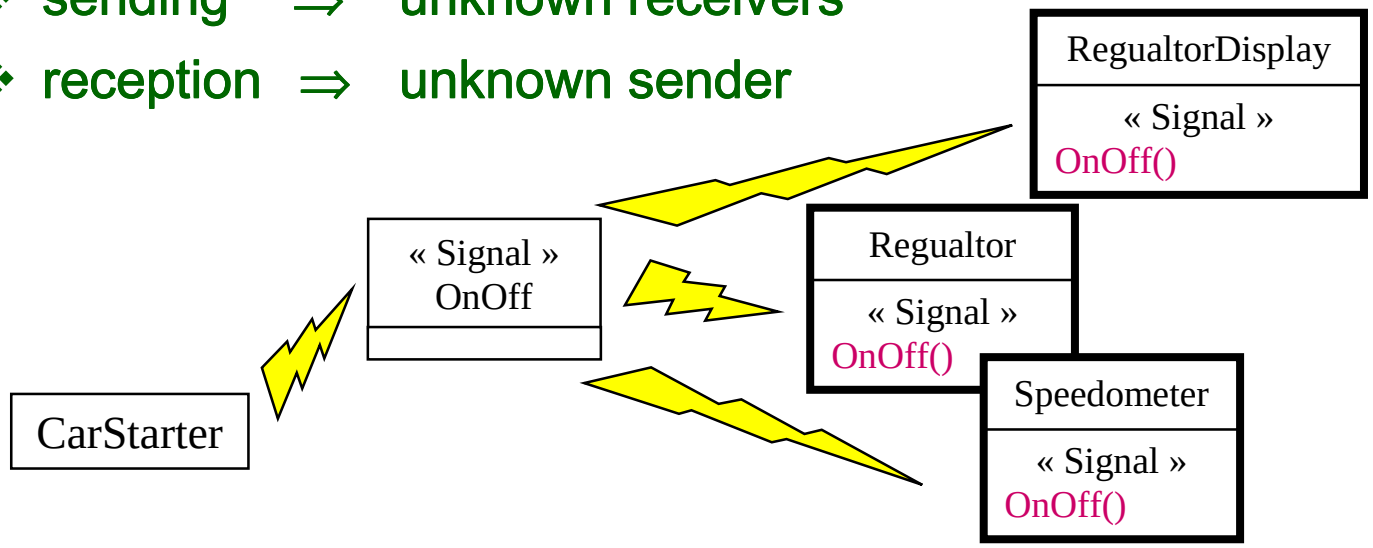


New stereotype in Class diagram of ACCORD/UML



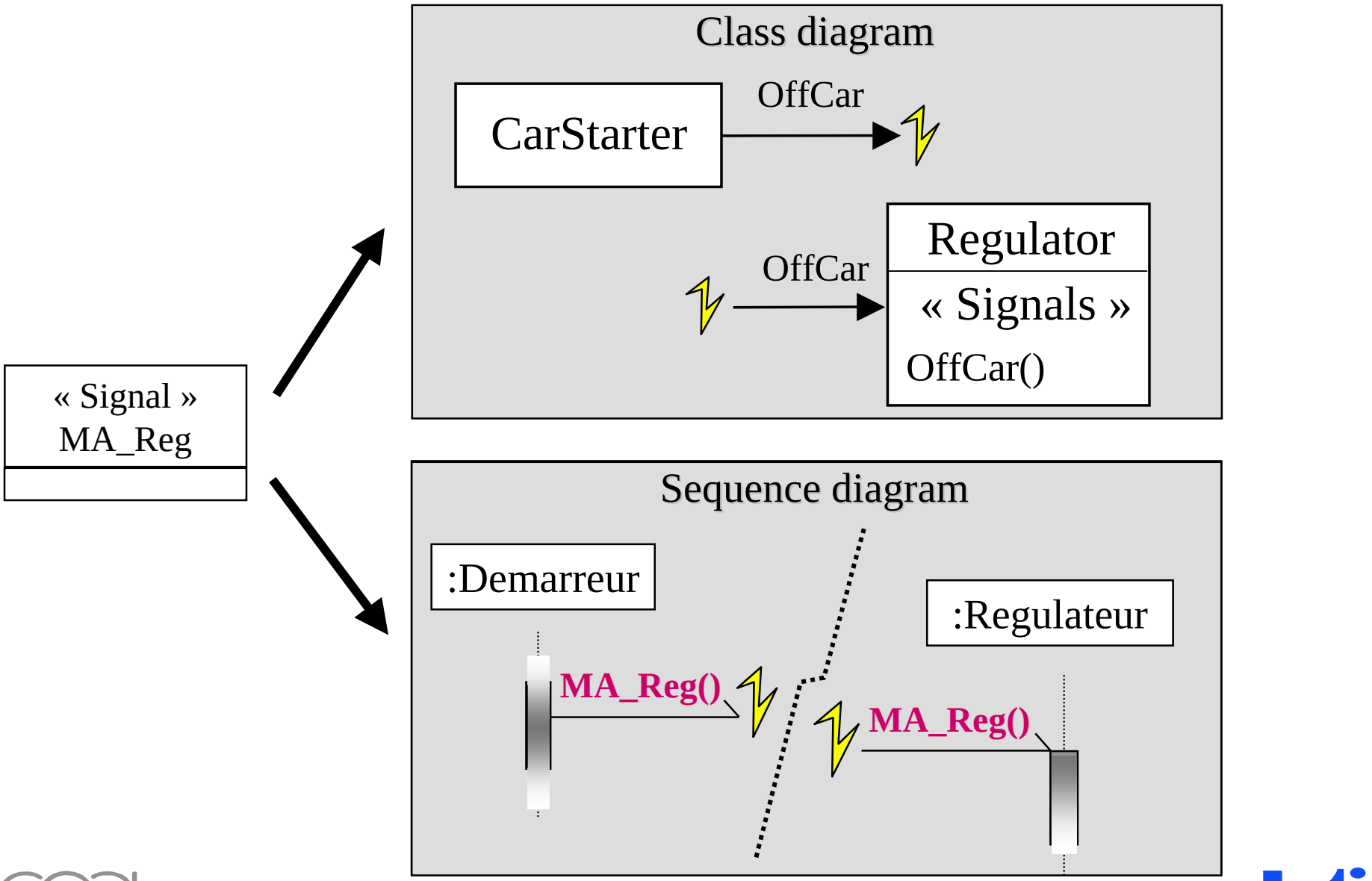
Signal specialization of ACCORD/UML (1/2)

- Usual signal using ⇒ overlapping between asynchronous operation call & signal sending
- ACCORD/UML ⇒ broadcasting communication
  - Broadcasted to all objects declaring sensible
  - sending ⇒ unknown receivers
  - reception ⇒ unknown sender



- Reaction to a signal receipt = method execution

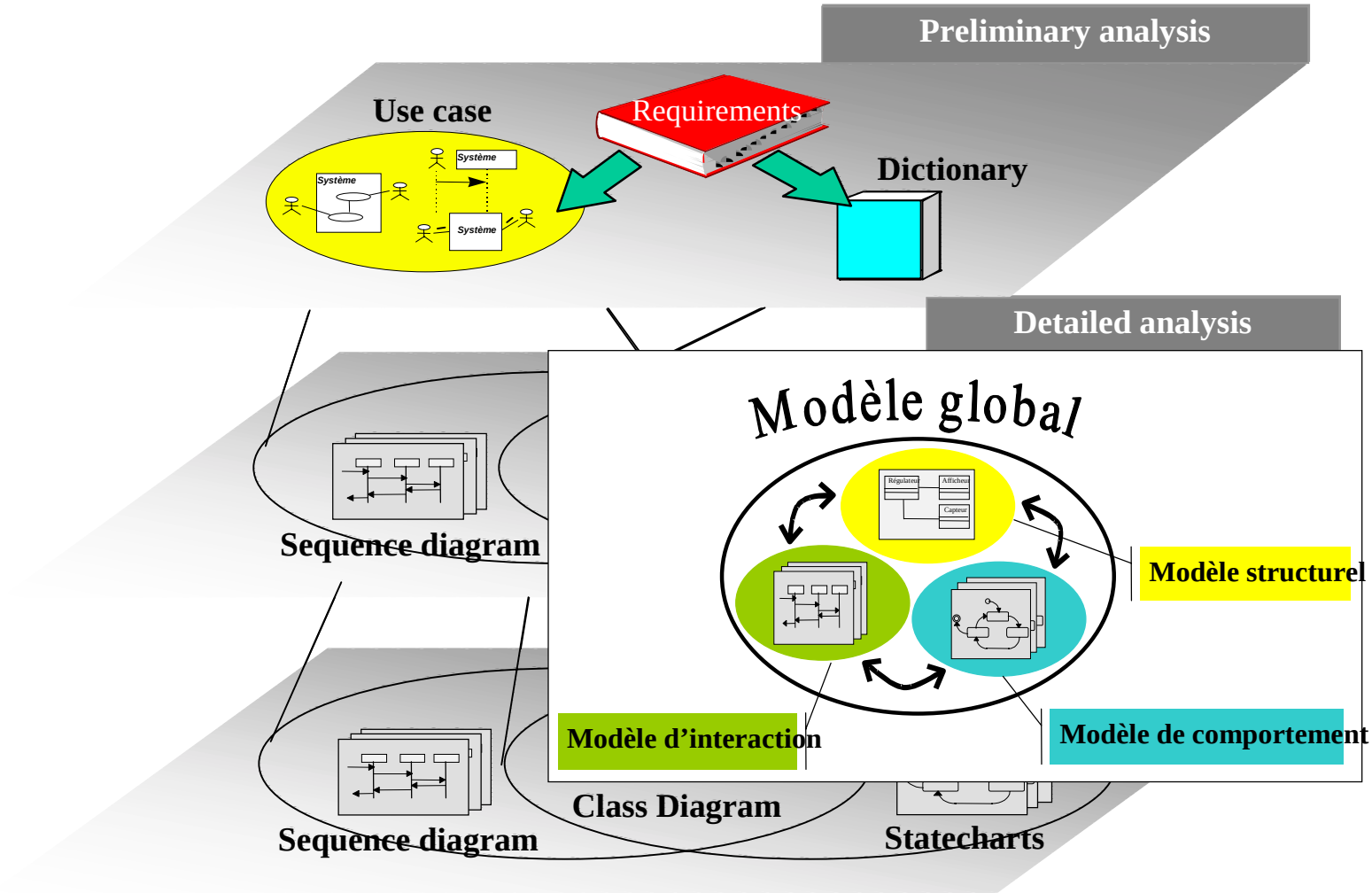
Signal specialization of ACCORD/UML (2/2)



« meta-modeling » using in the ACCORD platform

- *Introduction of high level modeling concepts*
  - *Real-Time Objects*
  - *Signals …*
- Modeling rules introduction
  - Model structuring
  - Behavior specification
  - Signals using
- *Mechanisms definition of operatring*
  - *Desing patterns dedicated to Signals, Real-Time objects …*
  - *Automatic code generation*
- *Models analysis for validation*
  - *Test cases automatic generation*

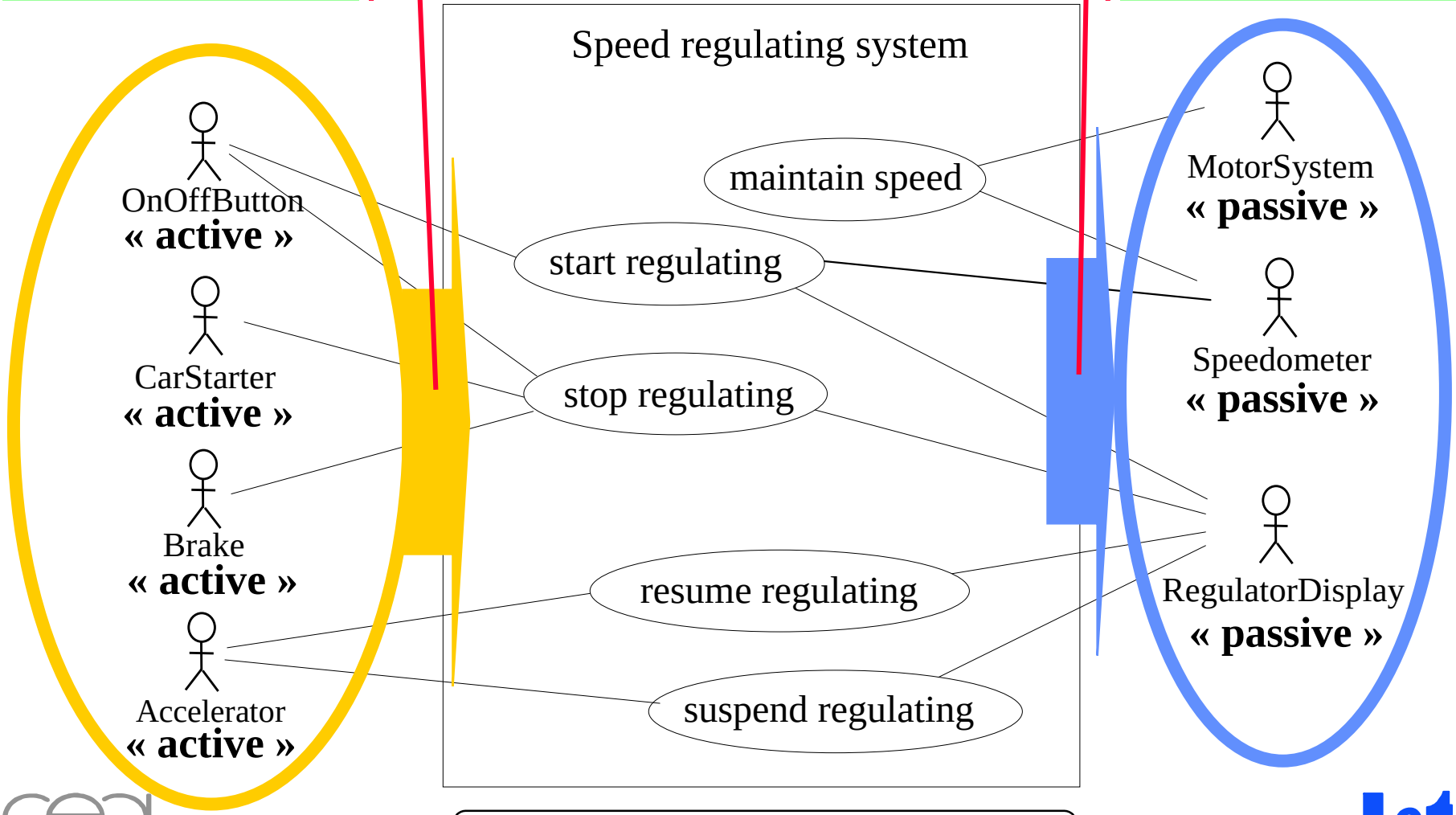
ACCORD development process



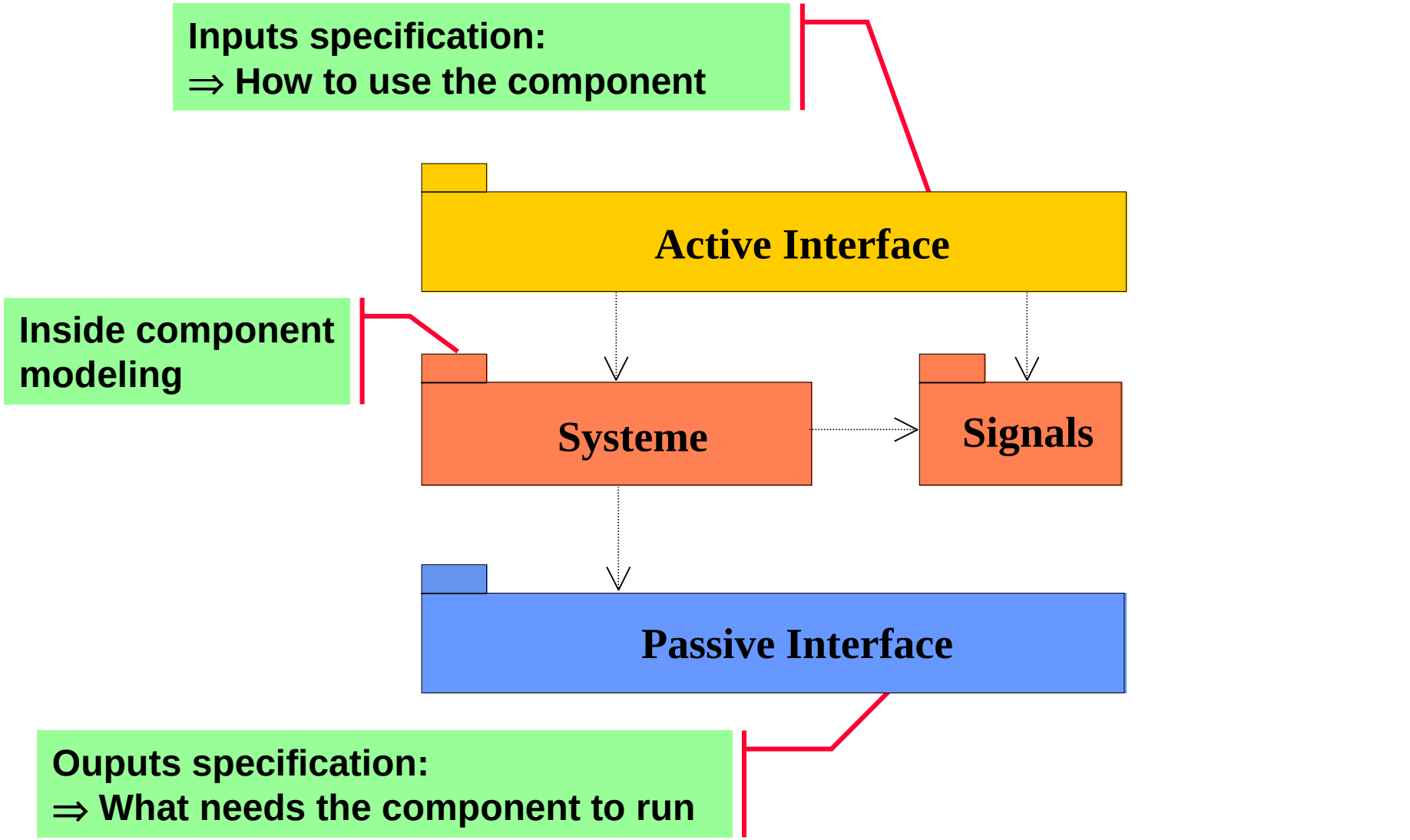
Active / Passive actors

Interactions from environnement to system

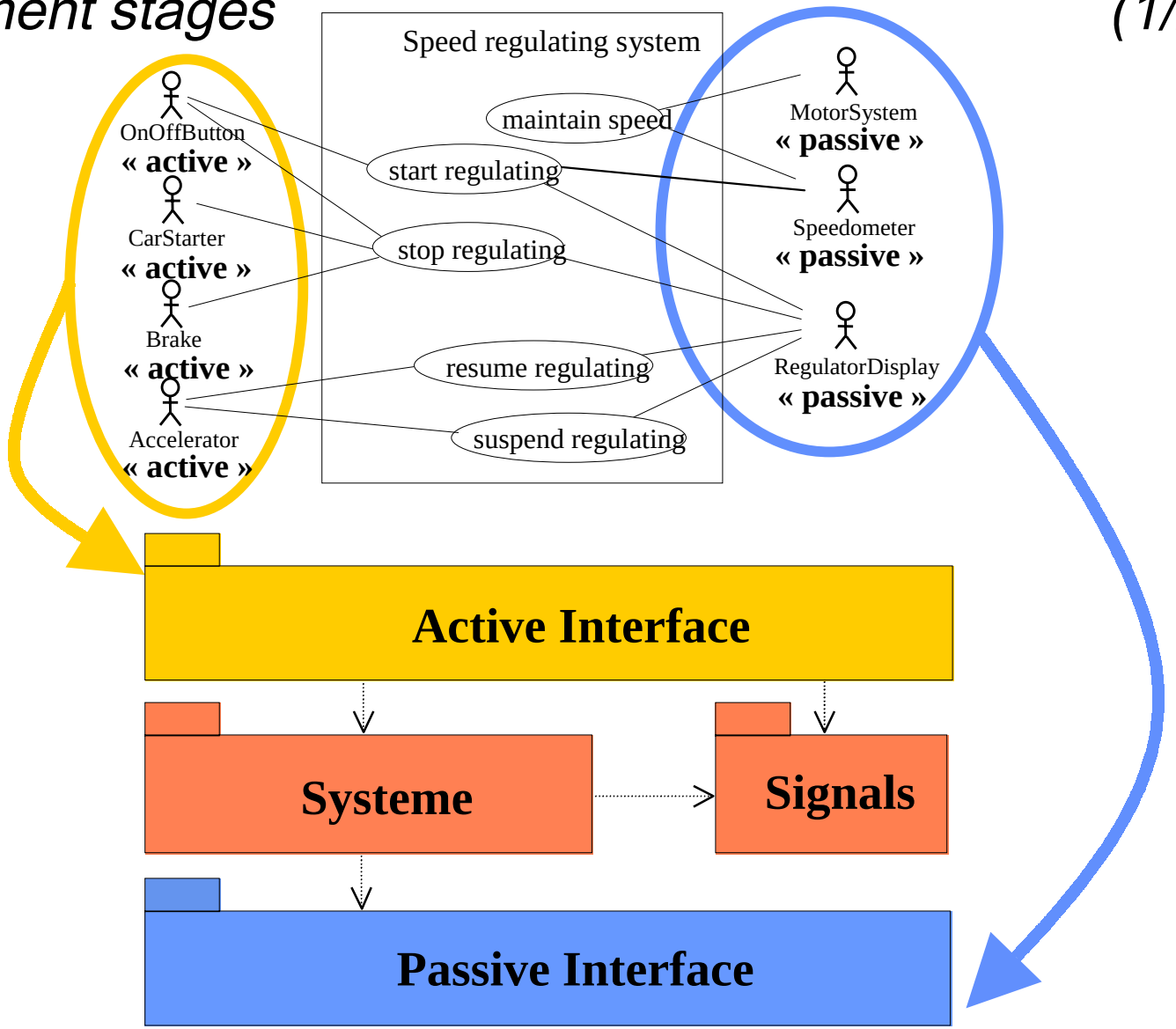
Interactions from environnement to system



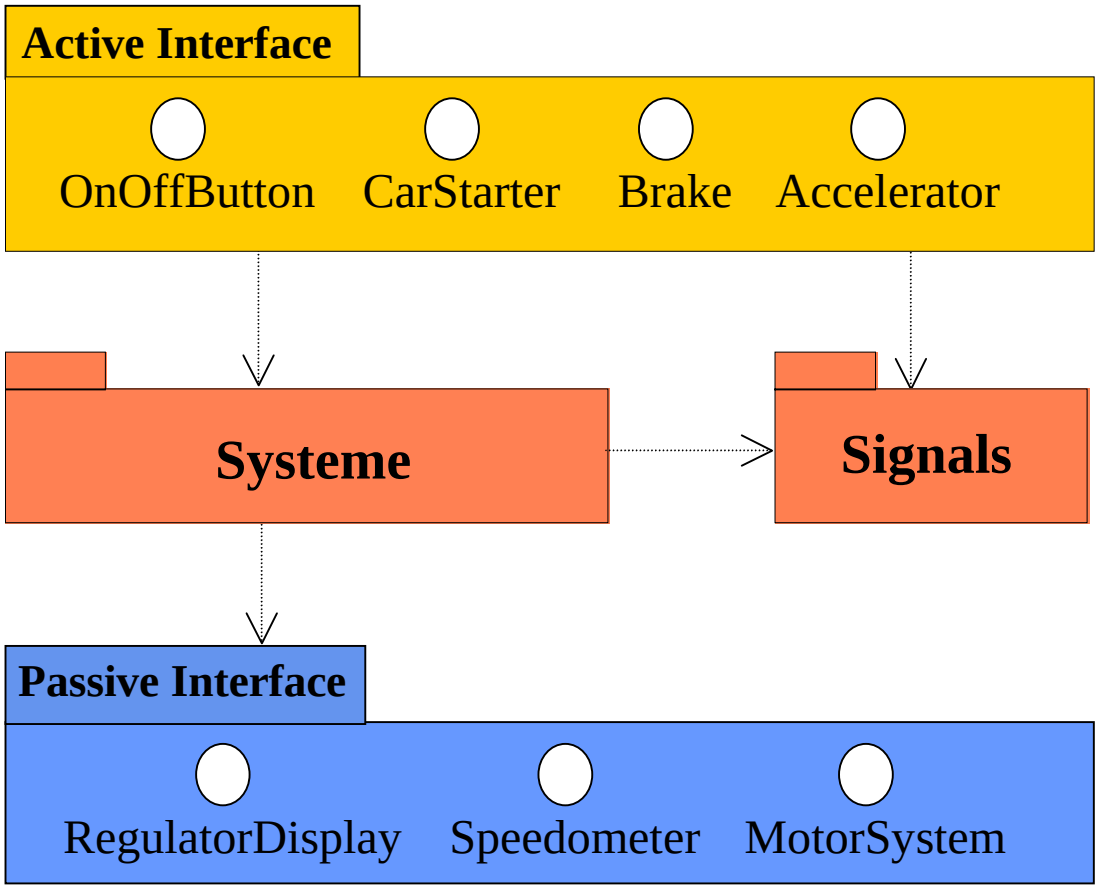
Generic structure of an ACCORD/UML component



Consistency rule and model transformation between development stages (1/2)



Consistency rule and model transformation between development stages (2/2)

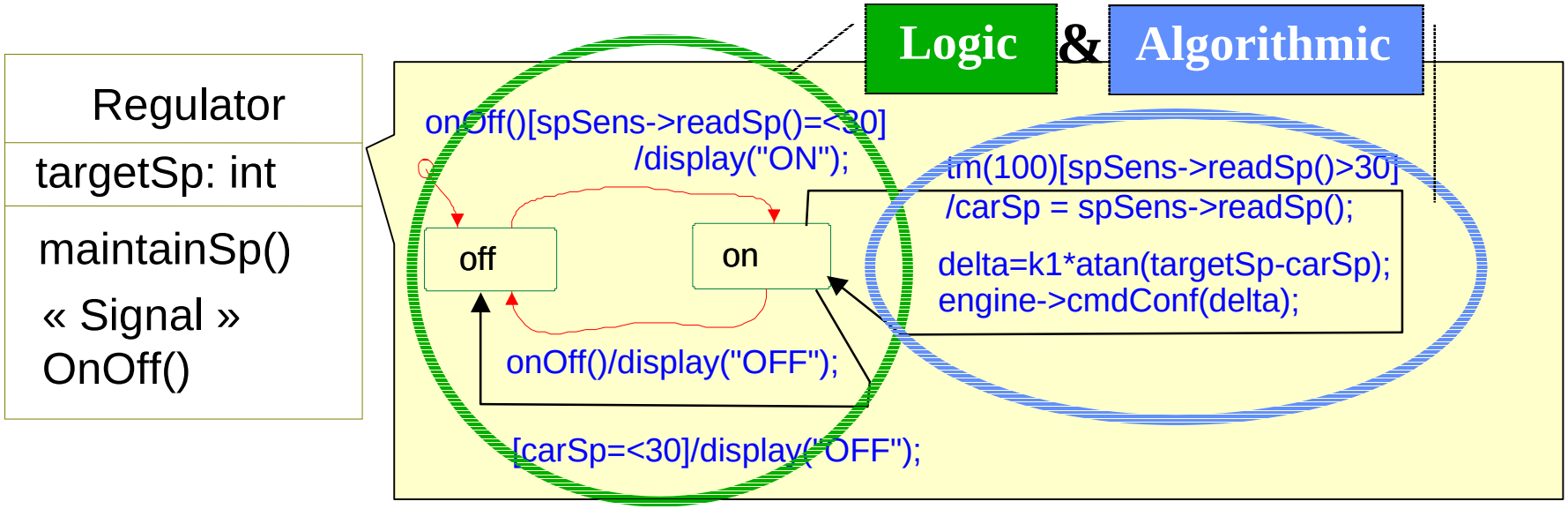


« meta-modeling » using in the ACCORD platform

- *Introduction of high level modeling concepts*
  - *Real-Time Objects*
  - *Signals …*
- Modeling rules introduction
  - Model structuring
  - Behavior specification
  - Signals using
- *Mechanisms definition of operatring*
  - *Desing patterns dedicated to Signals, Real-Time objects …*
  - *Automatic code generation*
- *Models analysis for validation*
  - *Test cases automatic generation*

Usual modeling of the application behavior

Drawbacks: ➔ mix of control logic & algorithmic specifications
➔ loose of clear relation with object interface



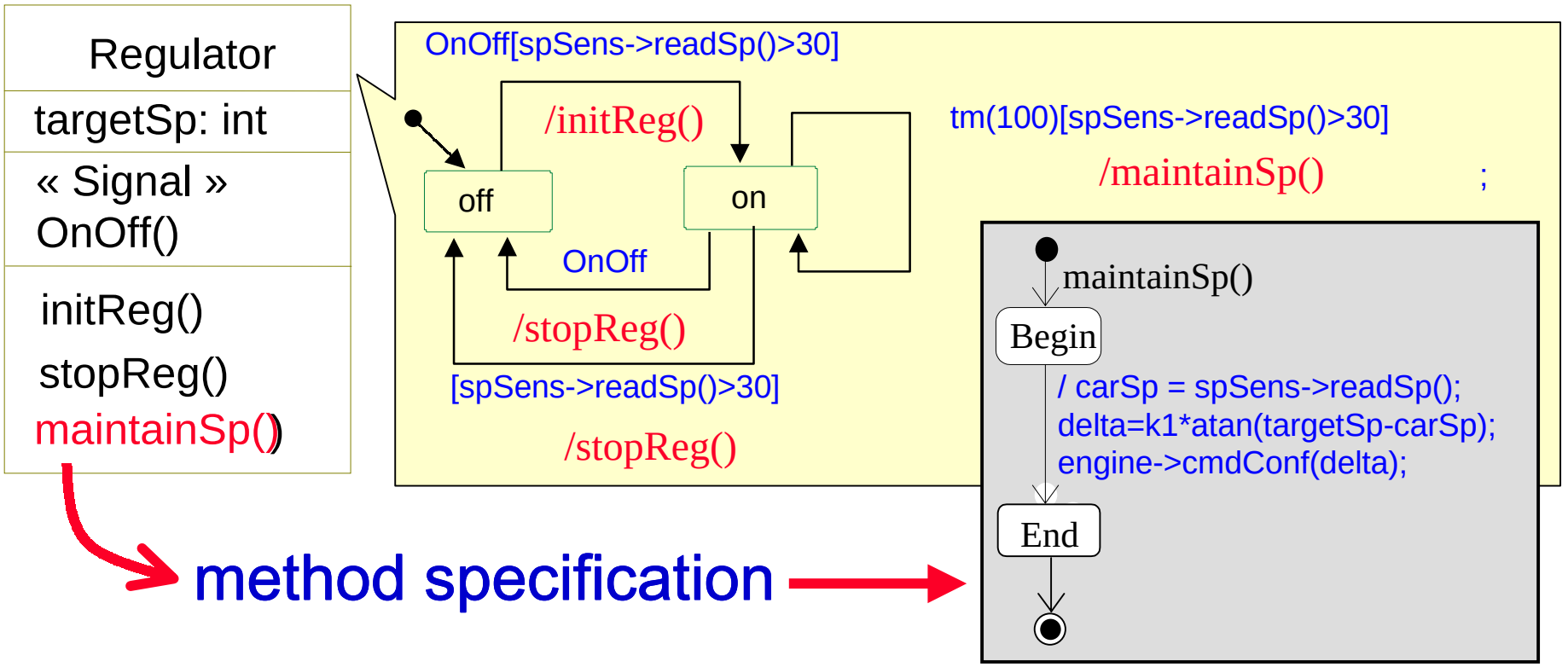
Readability ↘

Reusability ↘

Approach, step1: separation of concerns

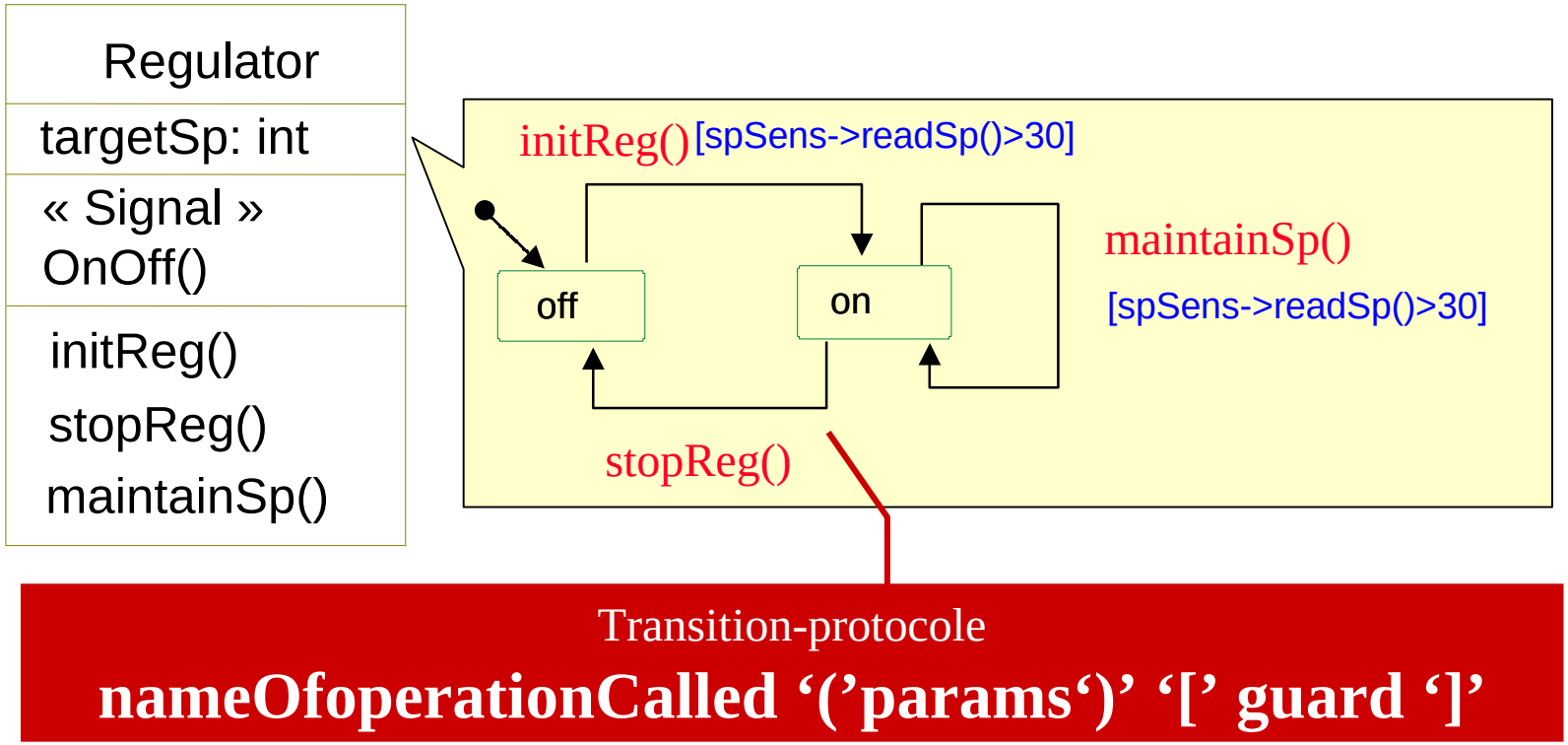
☞ The control state machines

⇒ **assign all action sequences to object operations**



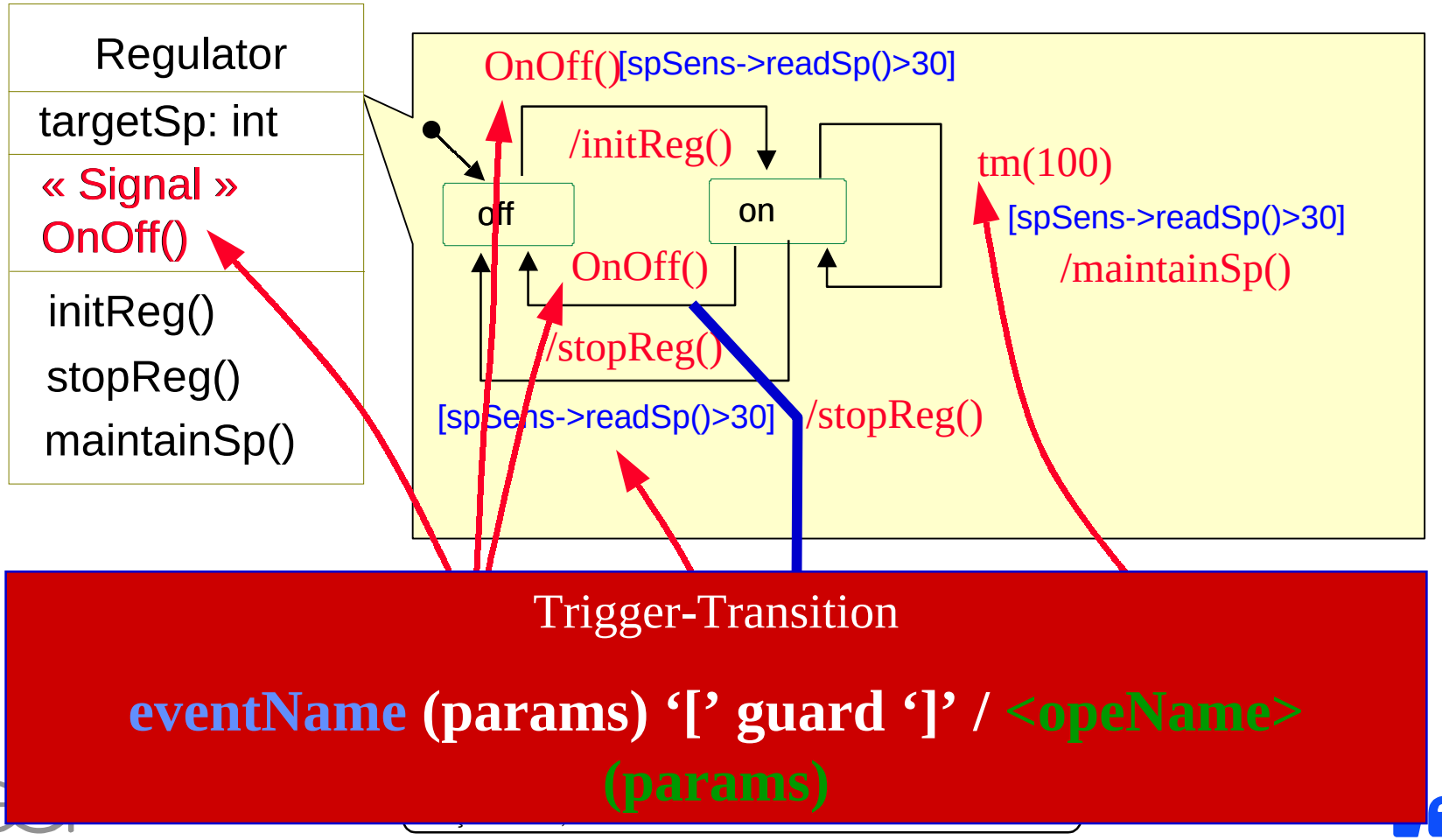
Approach, step2: definition of « legal » use

- **Protocol automaton:**
  focus on possibility to process operations

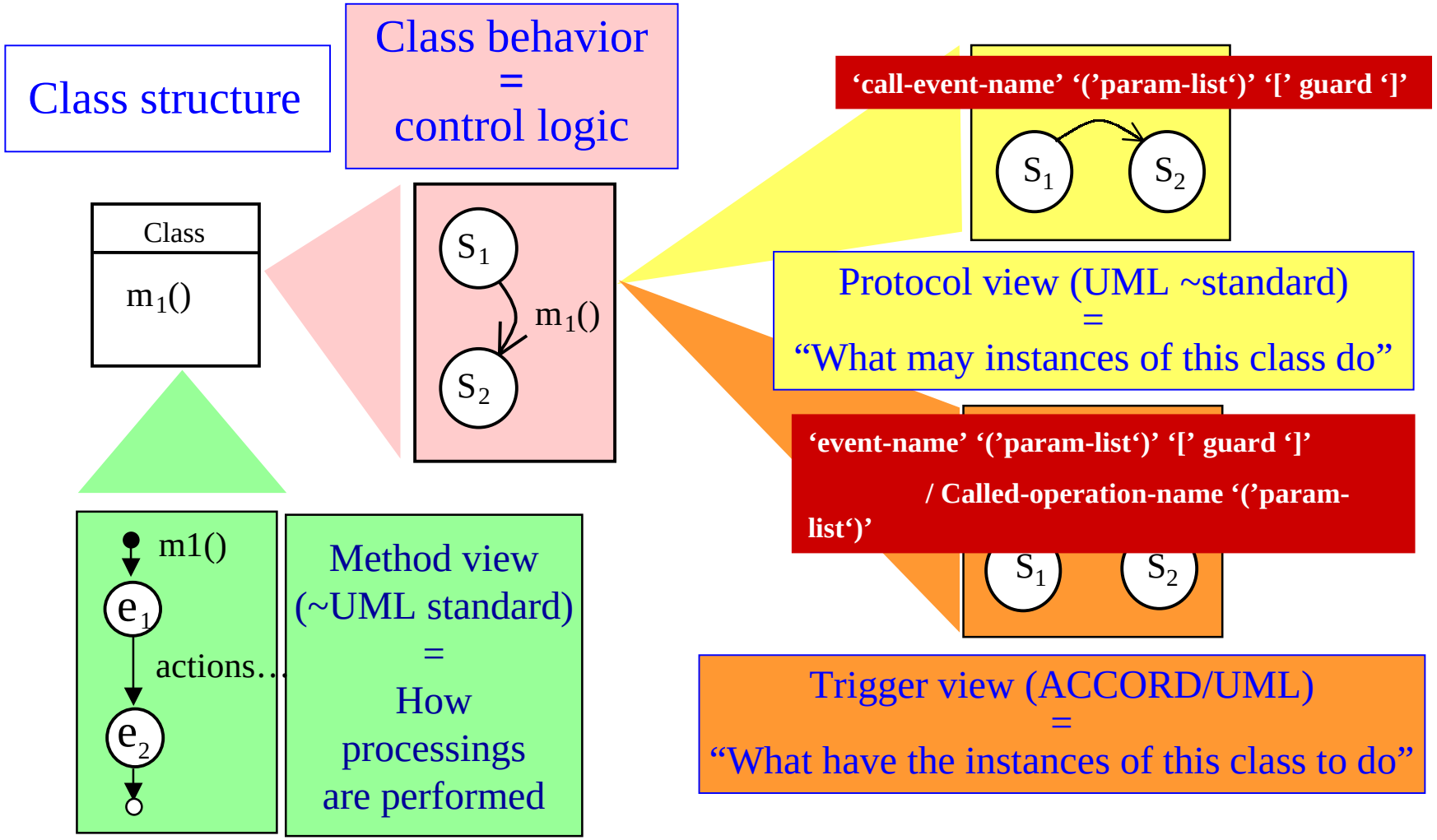


Approach, step3: specify reactivity

- **Trigger automaton:** ***focus on object reactivity***



Class state machine use with two different views



Examples of Well Formedness-Rules of ACCORD/UML statemachines using

SendAction

[1] The target of a *SendAction* is the instances set of the system that own a reception declaration towards the signal sent.

self.target.body = “all”

[2] *Parameter* direction of a *SendAction* has to be « in ».

self.parameter → forAll(p | p.kind = # in)

SignalEvent

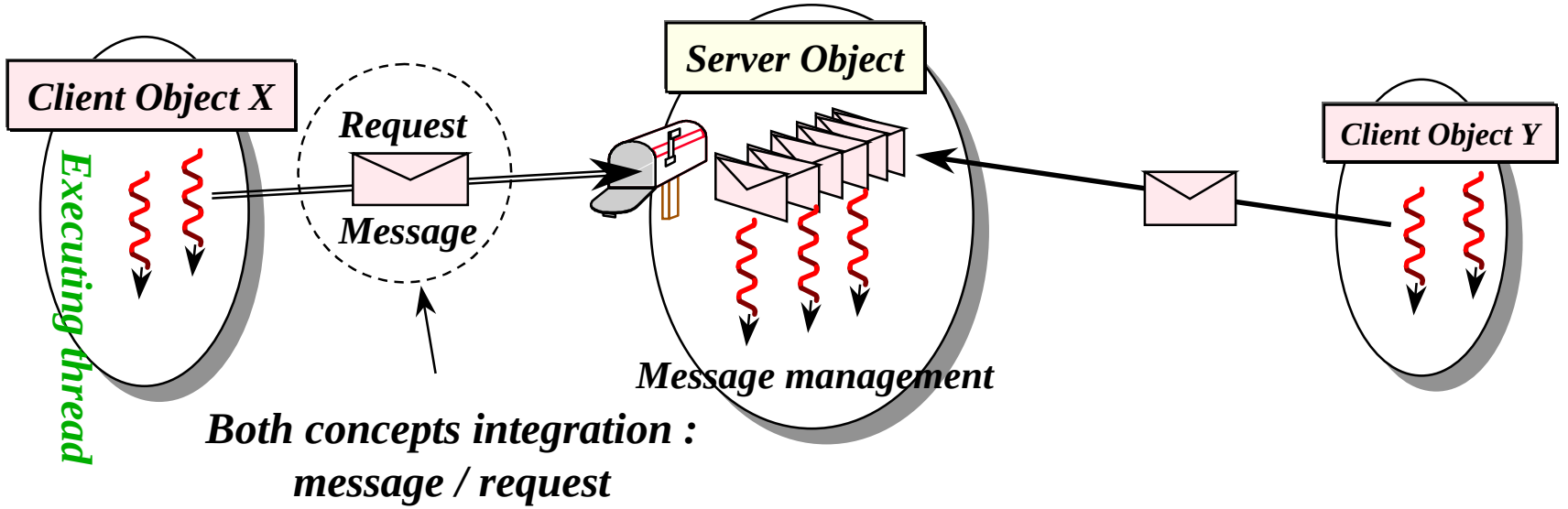
[1] *SignalEvent* owns as many parameters as its associated signal owns attributes

self.parameter → size = self.signal.allAttributes → size

Communication between Real-Time Objects

- By messages that convey real-time constraints
  - Deadlines, priorities…
  - Operation calls: Asynchronous message passing
    - Output parameters create synchronization on their use
  - Signals: Asynchronous anonymous broadcast (atomic)
    - Only input parameters → signal attributes
- Shared passive object can be defined
  - Concurrency control is added

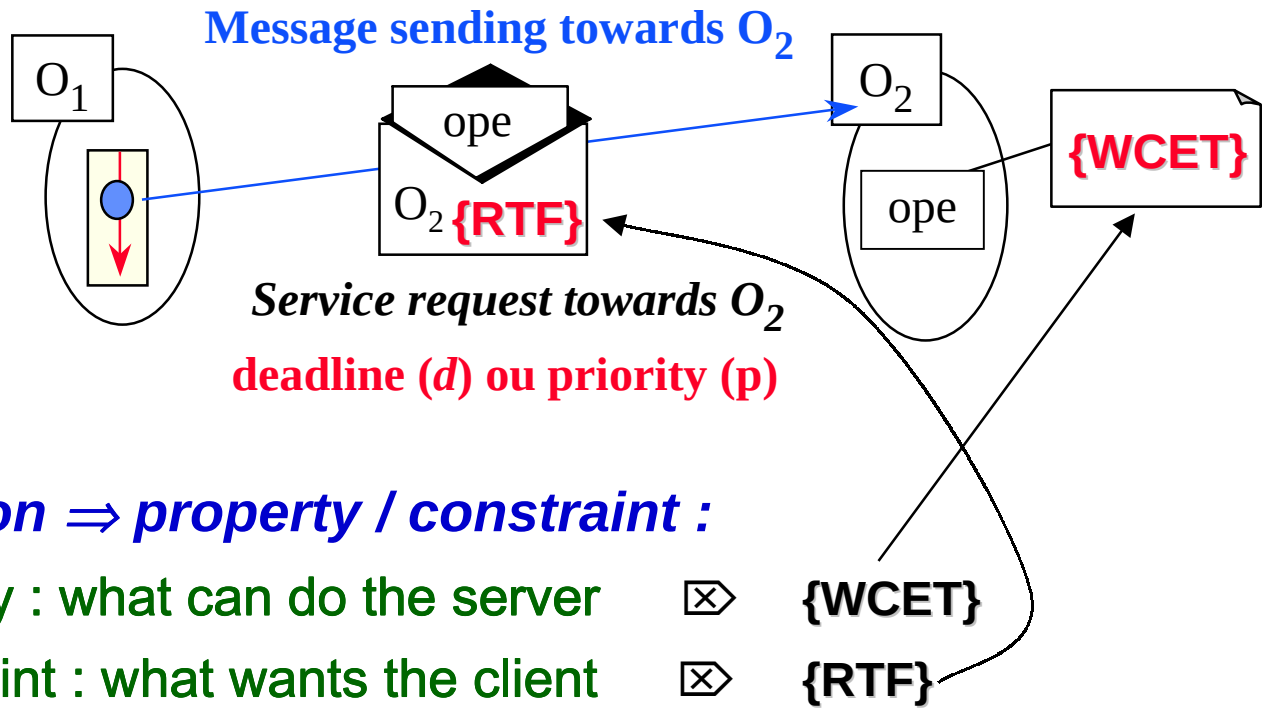
The Real Time-Object : a task server



- The concurrency granularity is the message
  - activity / task ⇔ message management
  - executing thread ⇔ method associated to operation
- Communication : asynchronous message sending
  - “needed”synchronization : output parameters

Real-Time constraint & ACCORD/UML (1/2)

☞ Real-time constraint ⌦ *on messages !*



☞ ***Distinction ⇒ property / constraint :***

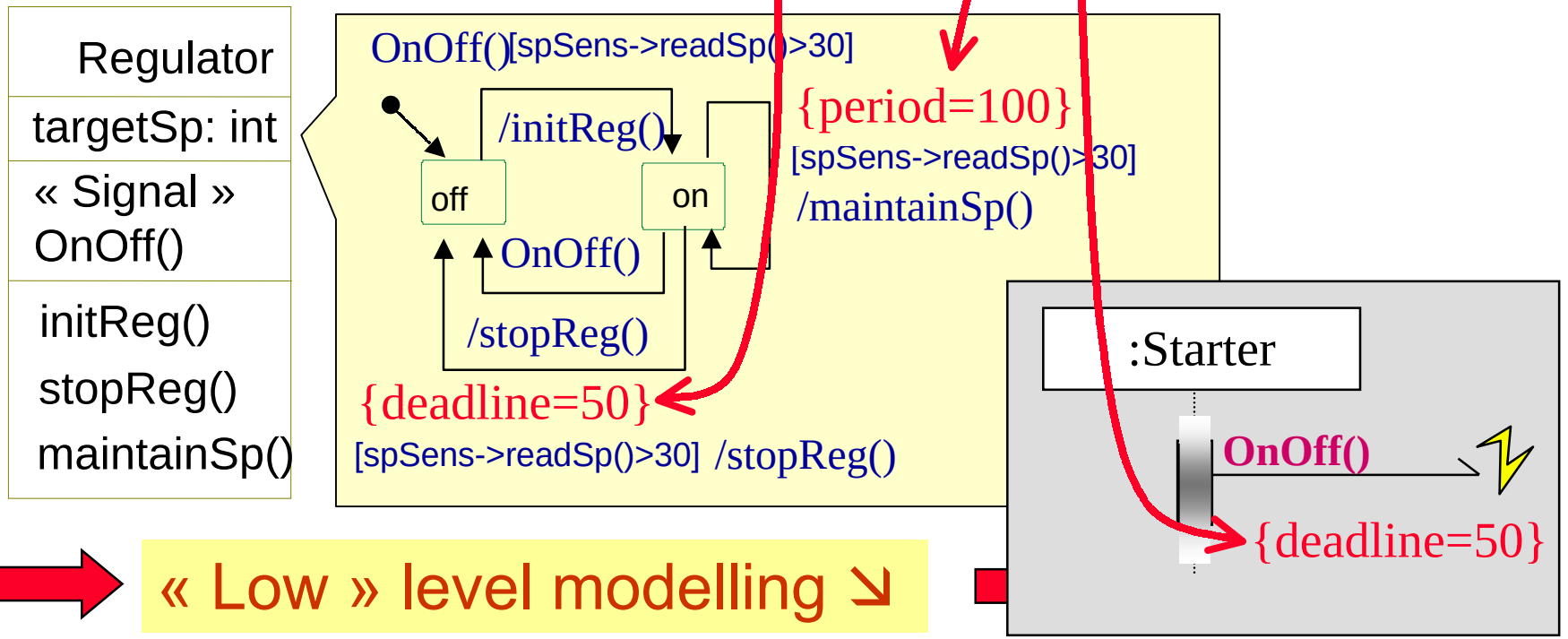
- ❖ Property : what can do the server ⌦ **{WCET}**
- ❖ Constraint : what wants the client ⌦ **{RTF}**

☞ ***Possibility of treatments triggering with different RT constraints***

Approach, step4: specification of time constraints

Use of UML constraints

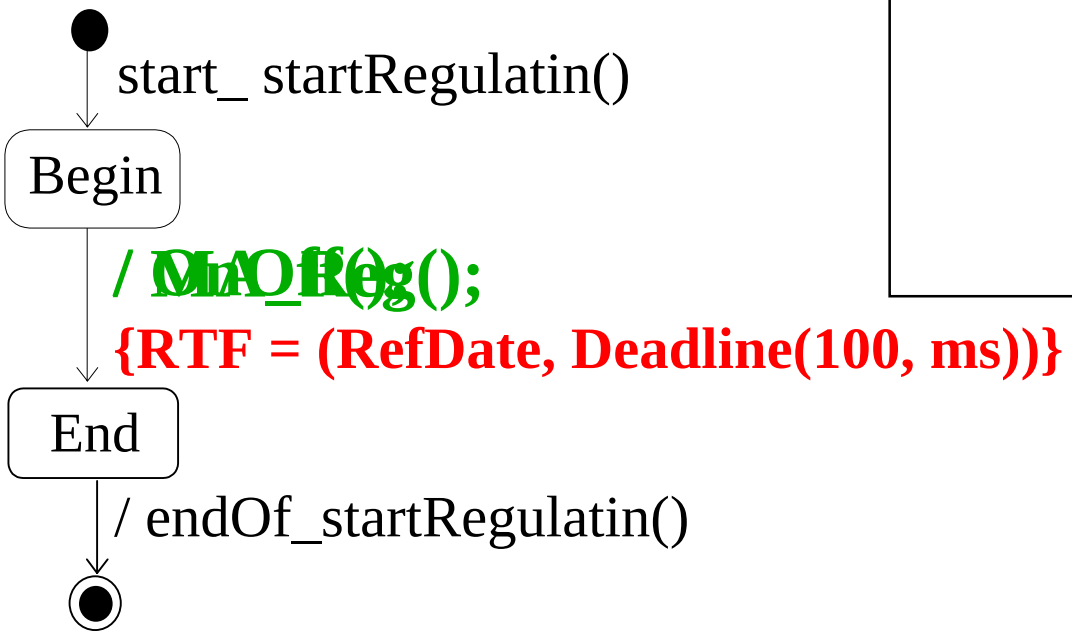
- ➔ on internal processings
- ➔ when sending messages (signals, operation call)



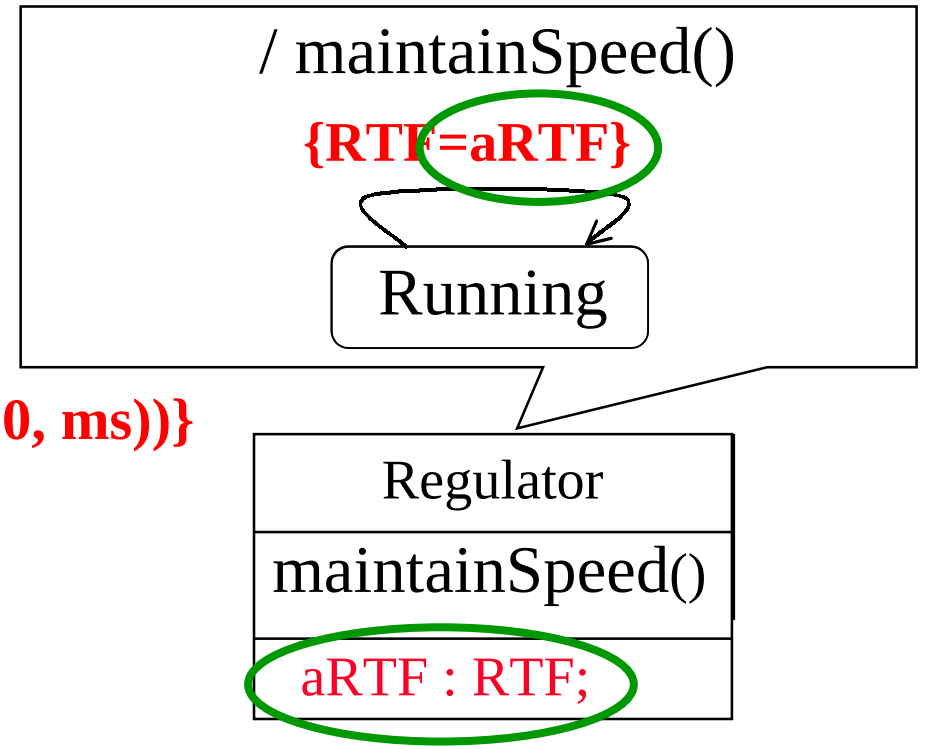
➔ « Low » level modelling ↘

RT specification in ACCORD/UML

RTF on a message sending



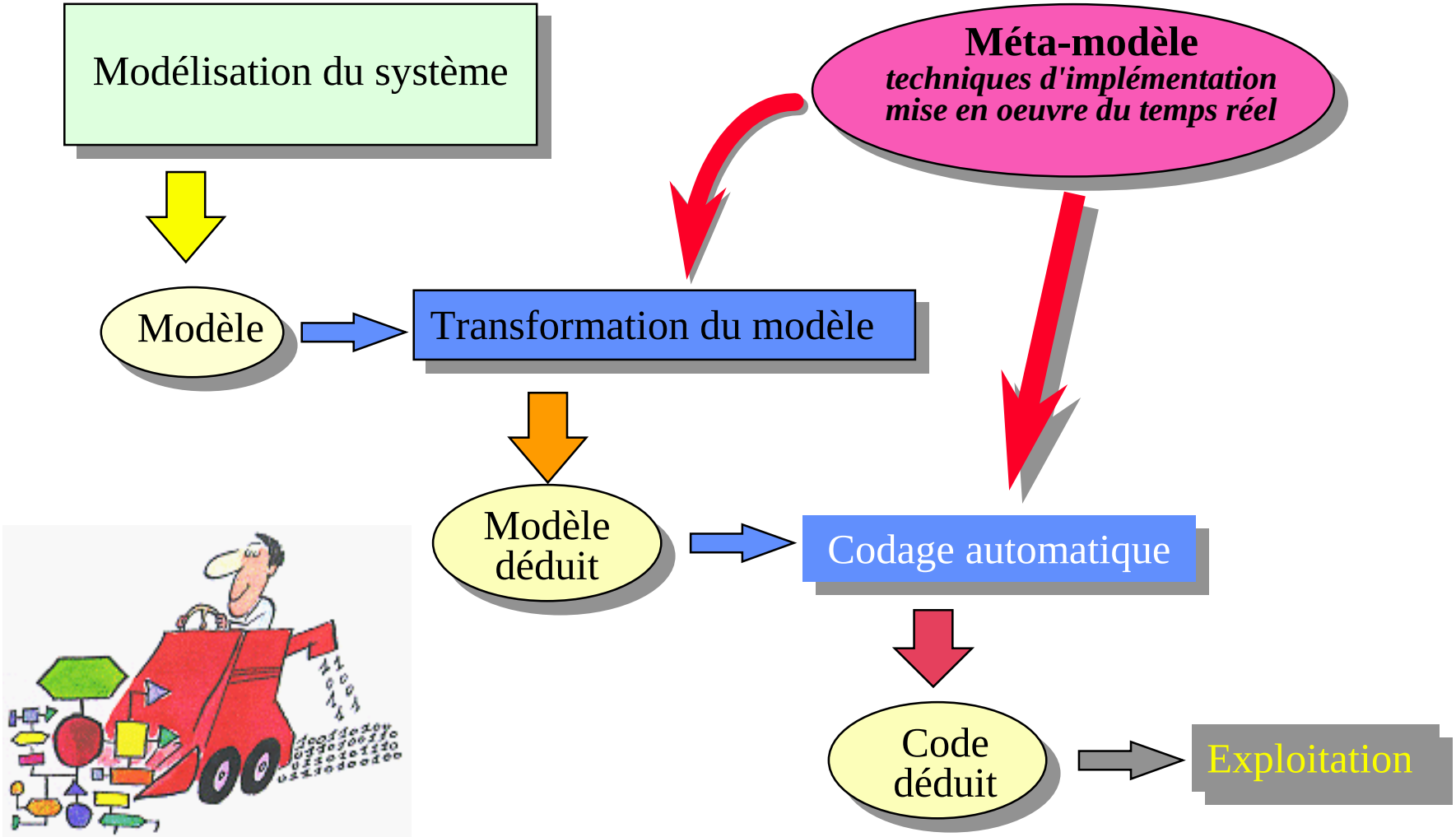
Periodic behavior



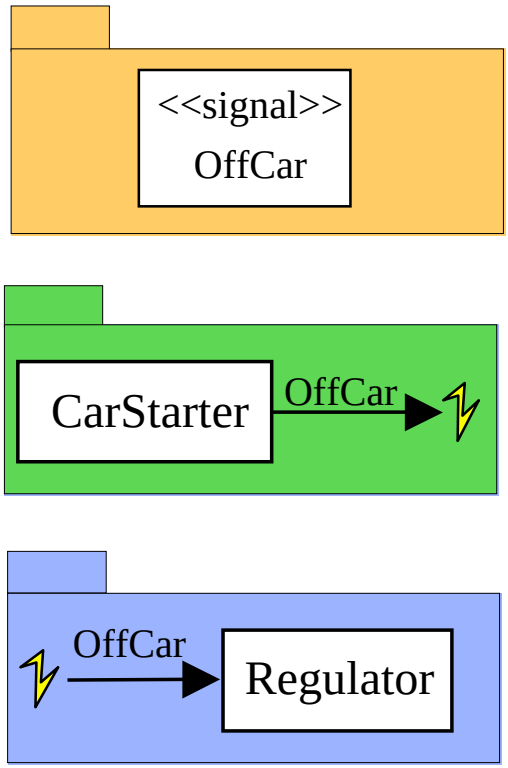
« meta-modeling » using in the ACCORD platform

- *Introduction of high level modeling concepts*
  - *Real-Time Objects*
  - *Signals …*
- *Modeling rules introduction*
  - *Model structuring*
  - *Behavior specification*
  - *Signals using*
- Mechanisms definition of operatring
  - Desing patterns dedicated to Signals, Real-Time objects …
  - Automatic code generation
- *Models analysis for validation*
  - *Test cases automatic generation*

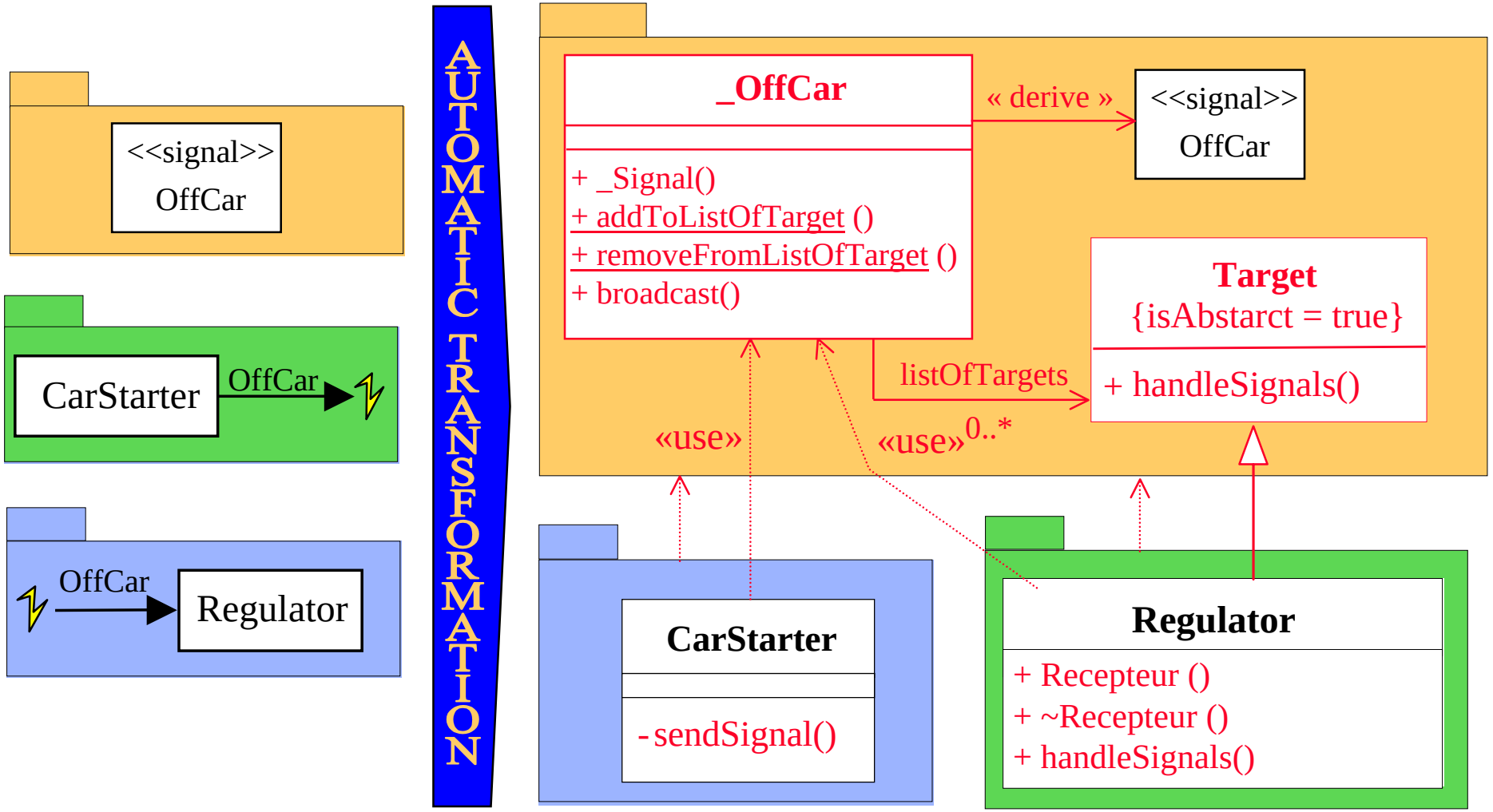
Transformation du modèle applicatif



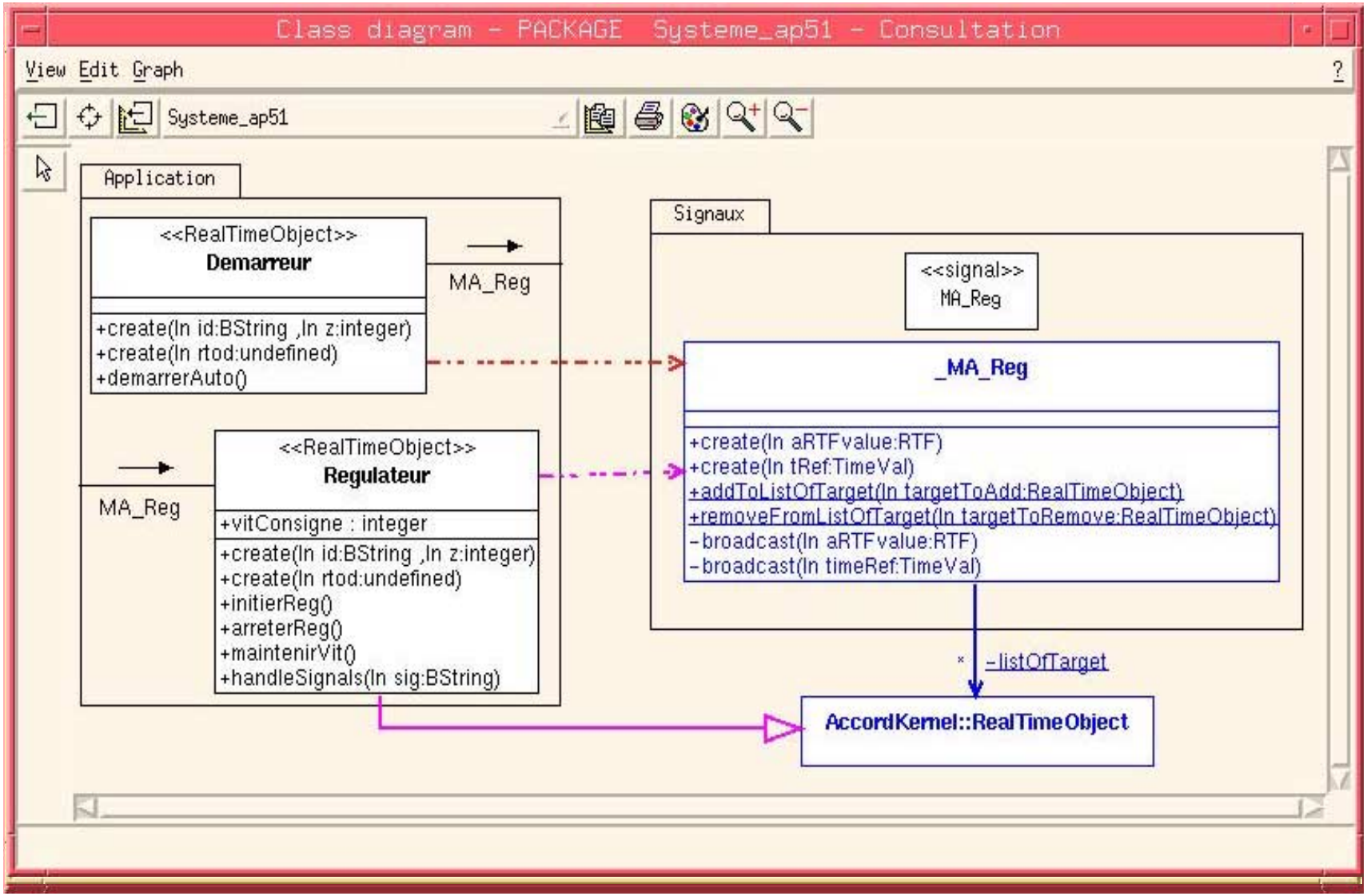
Model transformation, signal pattern



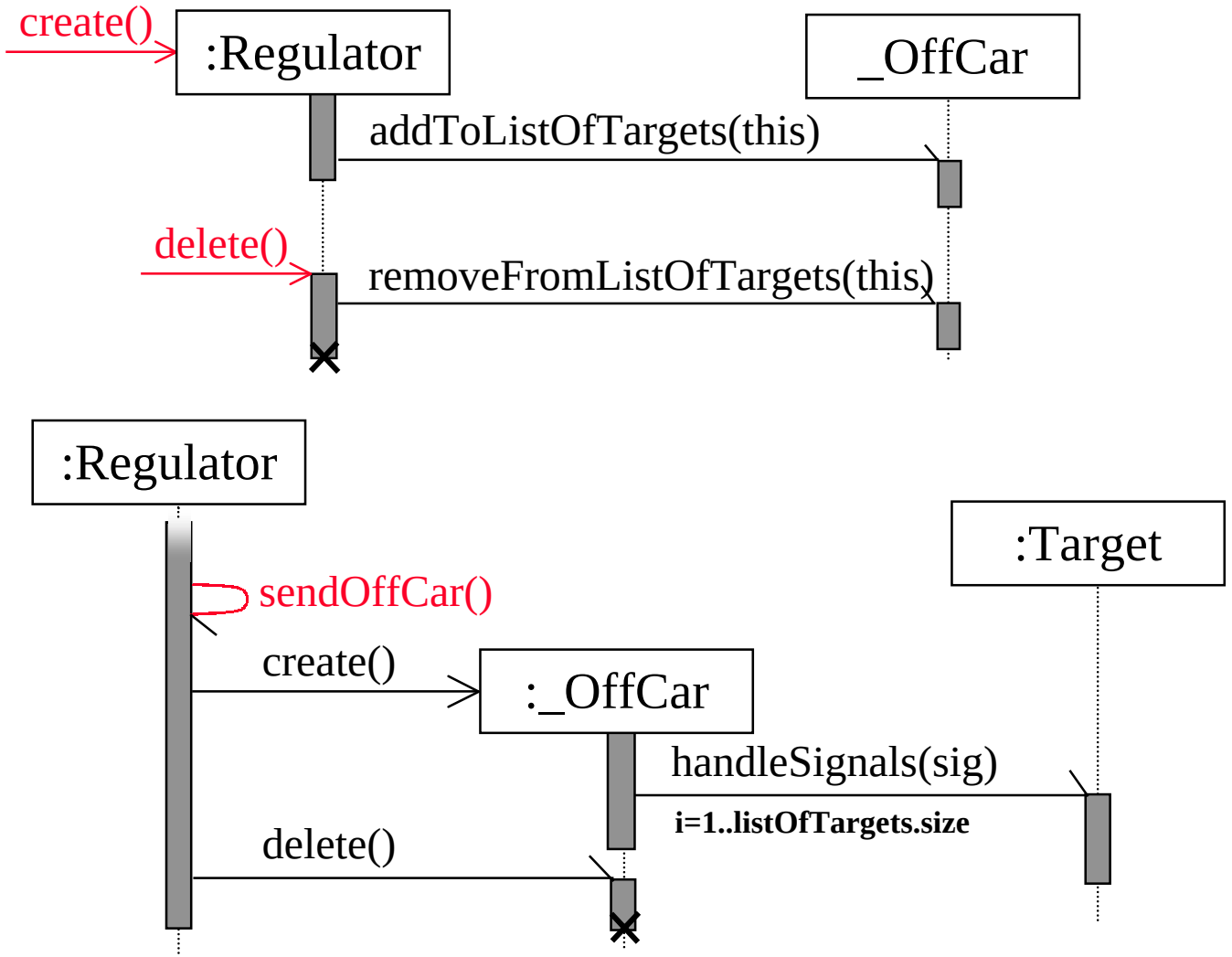
Model transformation, signal pattern



Mise en œuvre de la communication par signal



Dynamique du pattern des signaux



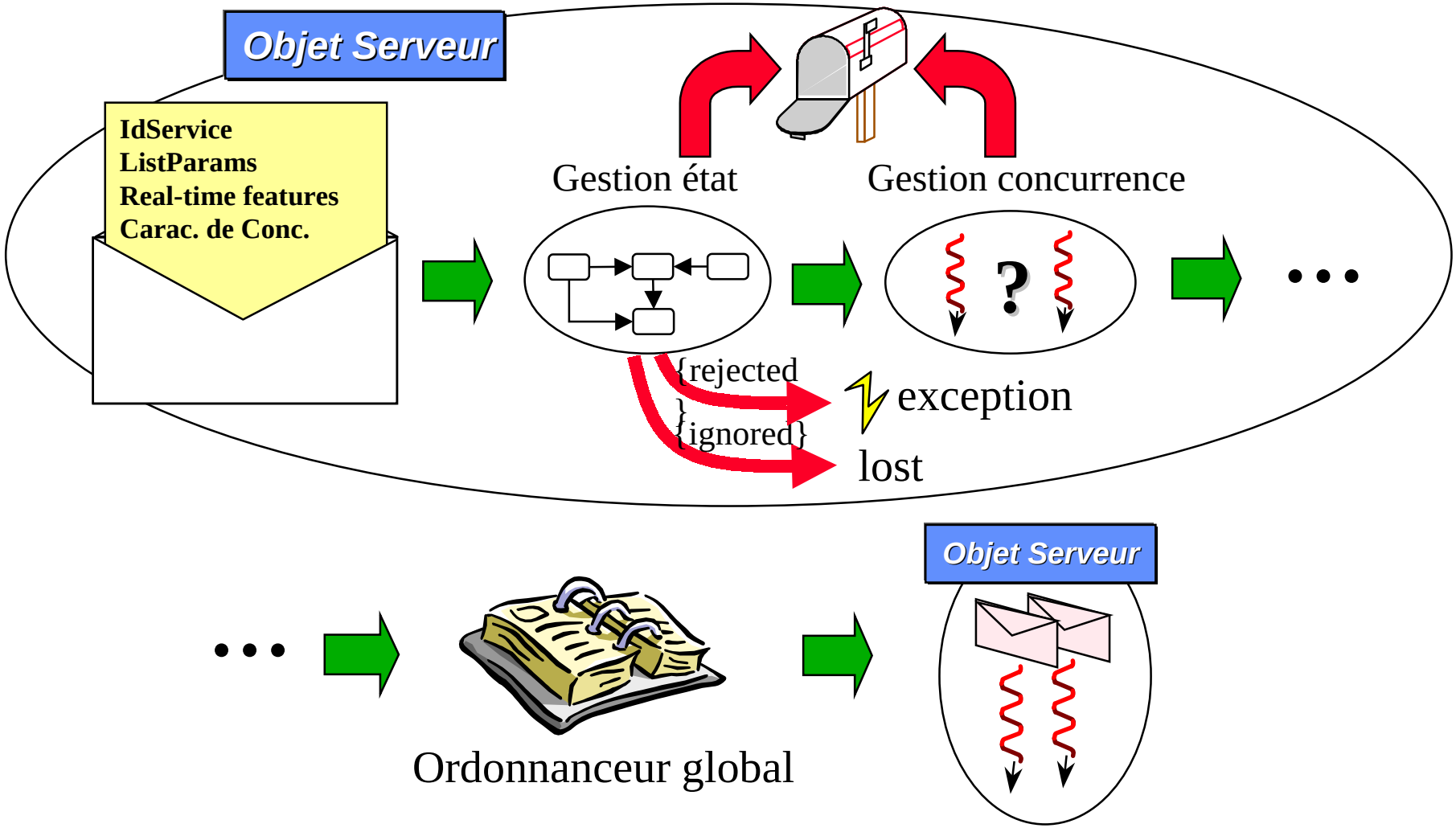
TOP (1) : "a real Time Object Pattern"

- Buts ⇒ créer du parallélisme via messages dans un fil d'exécution
  - synchroniser les accès aux ressources encapsulées par les objets
  - gérer les contraintes d'échéance ou de priorité sur les traitements des messages
  - sélectionner le message à traiter et demander son traitement au "système"

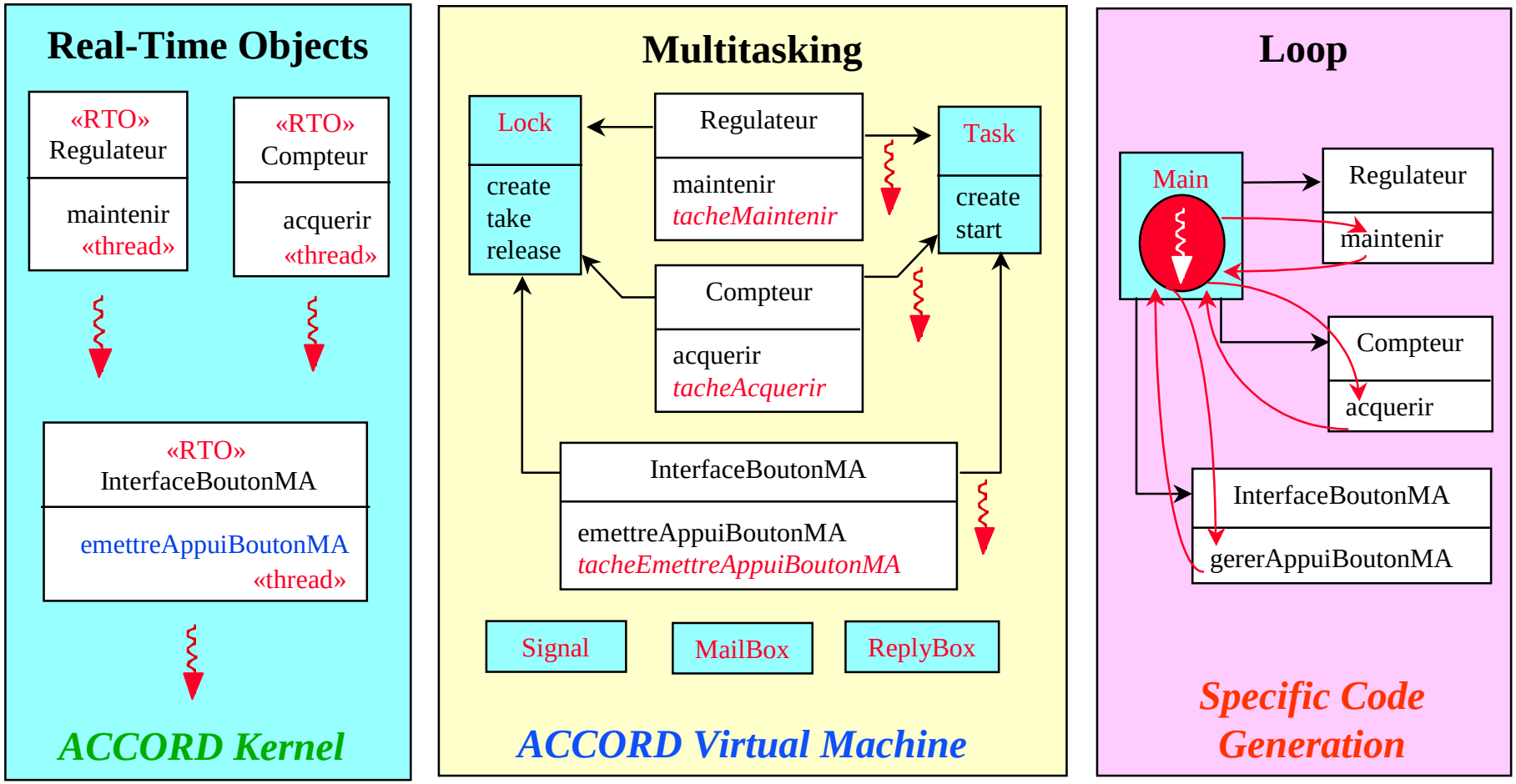
- Motivation ⇒ simplifier la mise en oeuvre des instructions systèmes...
  - intégrer dans un unique modèle (objet) les aspects "structurels" et "fonctionnels"

- Applications ⇒ applications temps réel à contraintes "souples"
  - logiciel multi-tâches avec/sans contraintes d'échéances/priorités conçu en tech. OO
  - tâches applicatives dont les tps d'exé sont grands / aux opérations systèmes
  - machine cible avec système d'exploitation temps réel fournissant du "multi-threading" (plusieurs tâches à commutation de contexte rapide fonctionnant dans un espace d'adresse commun)

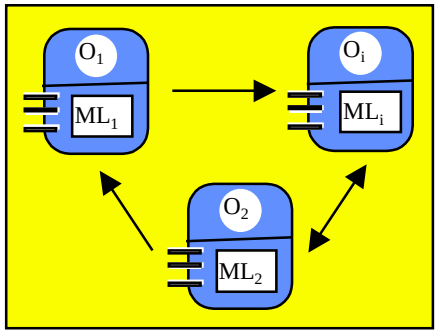
TOP (2) : Modèle d'Exécution Multi-Tâches



Une même spécification pour des implantations différentes



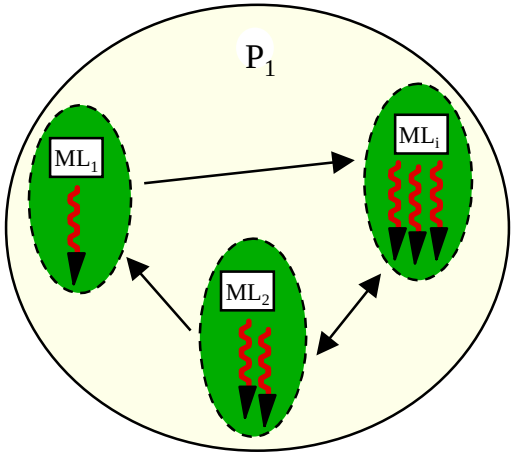
Passage du modèle à sa mise en œuvre multi-tâches



1. *Les données sont encapsulées et en accès protégé (sémaphore associé).*

2. *Un fil d'exécution par traitement de message d'un objet temps réel.*

3. *Les tâches son activées, synchronisées et ordonnancées en fonction des contraintes temps réel exprimées dans le modèle.*



Example of a synchronization management : classical view

❶ **Data declaration :**

```
...
double tgSpeed;
```

❷ **Data protection creation :**

```
...
SEM_ID semTgSpeed;
semTgSpeed = semMCreate(...);
```

❸ **Writing access :**

```
...
void maintain ()
{    semTake (semTgSpeed, WAIT_FOREVER );
          if ( carSpeed > tgSpeed ) ...
     semGive (semTgSpeed ); }
```

❹ **Task creation :**

```
...
taskSpawn( "T1", priority, ..., maintain , ... );
...
void maintain () { ... }
```

- ☞ **This approach needs low level code integration**
- ☞ **The programmer have to forget nothing …**
- ☞ **The user have to know the task implementing**

Example of a synchronization management : ACCORD view

❶ **Data declaration and protection creation :**

→ *Encapsulated in a class*

❷ **Shared data access :**

→ *the owning class have direct access*

→ *else, access via a special operation*

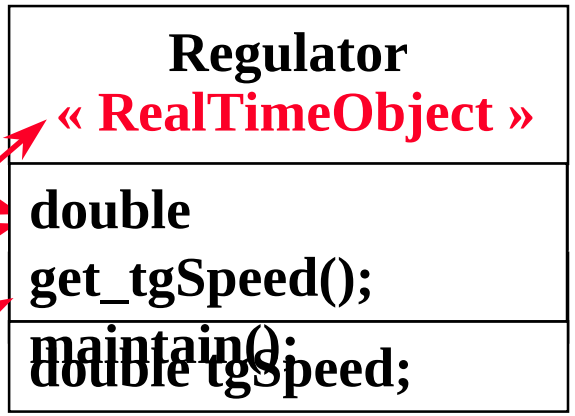
aReg.get_tgSpeed ()

❸ **Task creation :**

→ *To declare the object as task server*

→ *To add operations to the object*

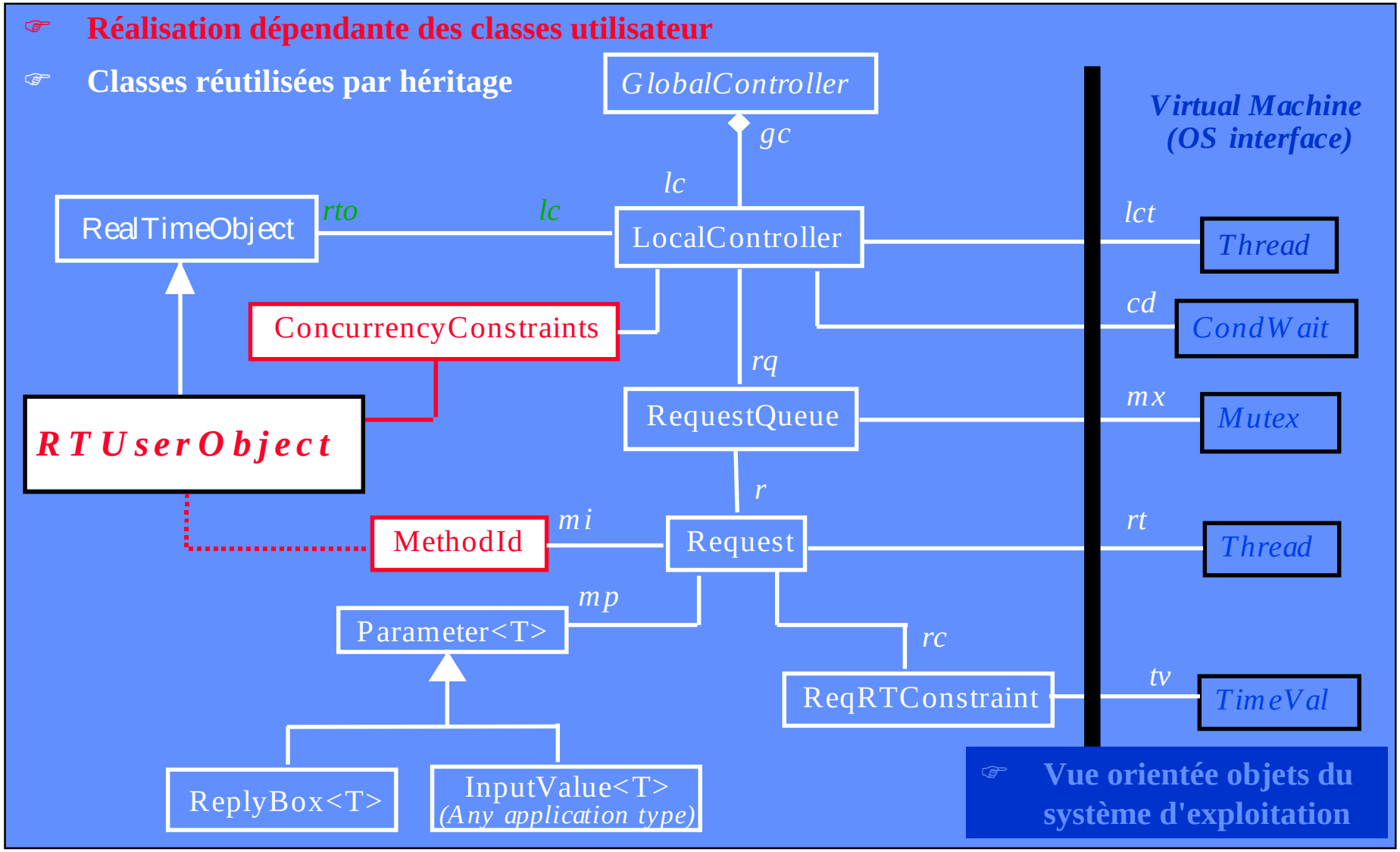
↳ **void maintain () { if (carSpeed > get_tgSpeed ()) ... }**



☞ **Automatic implementing (transparent to developer)**

☞ **Only need declaration of constraints to respect**

TOP (3) : Structure



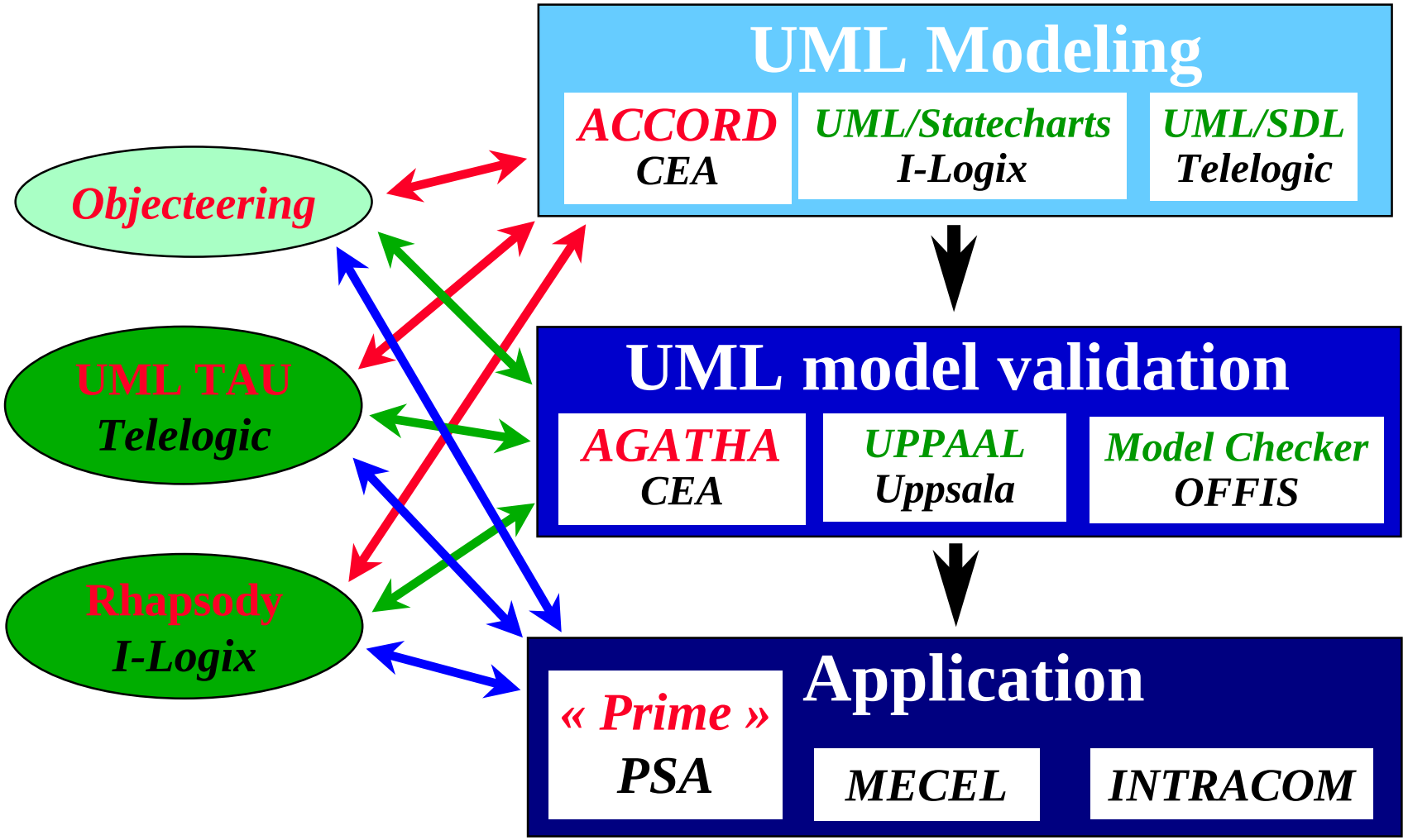
ACCORD/UML: Real-Time implementation

- Assignment of task to message processing by RTO
  - Default: reaction on signal and internal processing
    (always defined by the execution of an object operation)
  - User: implementation directives on message processing
  - Constraints specification only (deadline, period, priority)
    - Associated, when possible, to operation properties like execution times
- Scheduling mechanisms are provided
  - Event queue management based on message RT constraints
  - Concurrency management based on operation constraint declaration

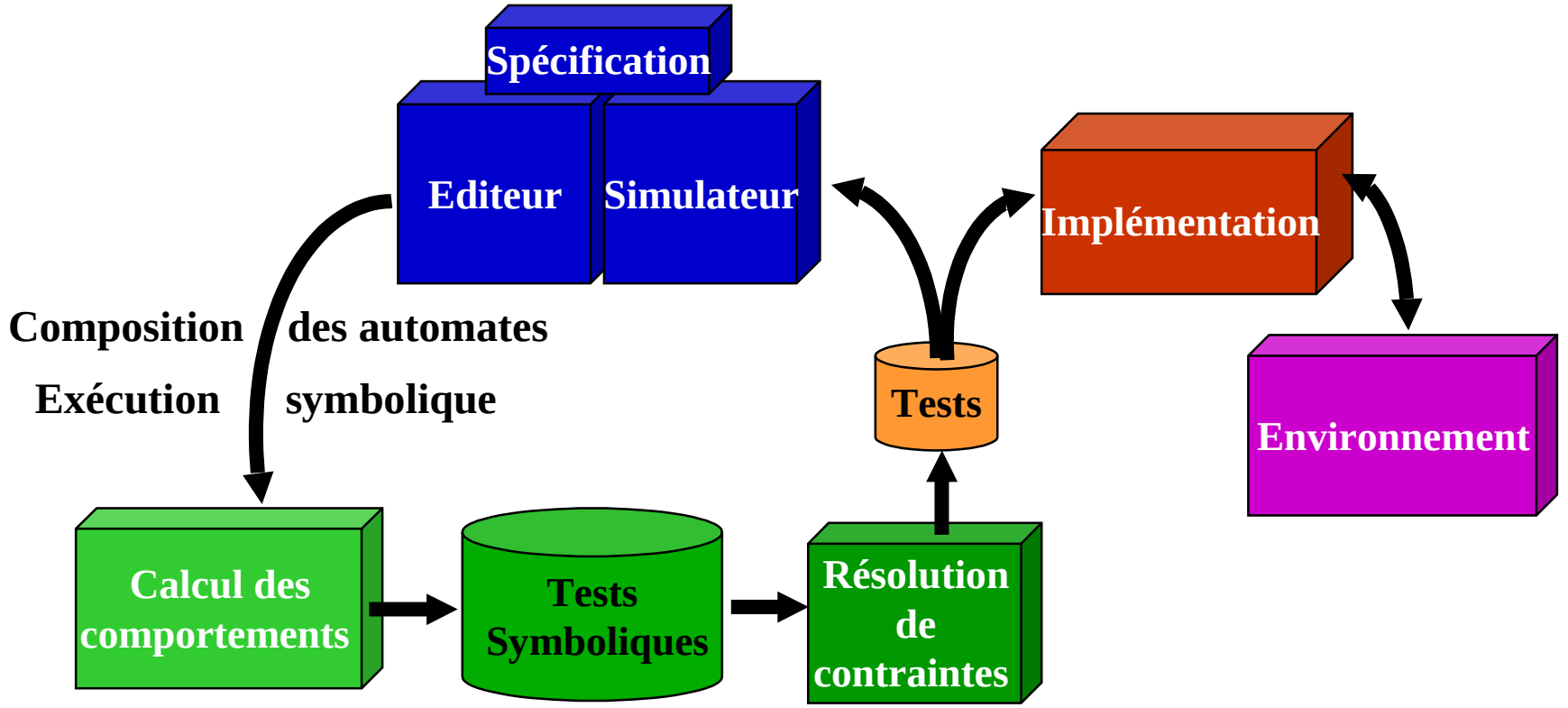
« meta-modeling » using in the ACCORD platform

- *Introduction of high level modeling concepts*
  - *Real-Time Objects*
  - *Signals …*
- *Modeling rules introduction*
  - *Model structuring*
  - *Behavior specification*
  - *Signals using*
- *Mechanisms definition of operatring*
  - *Desing patterns dedicated to Signals, Real-Time objects …*
  - *Automatic code generation*
- Models analysis for validation
  - Test cases automatic generation

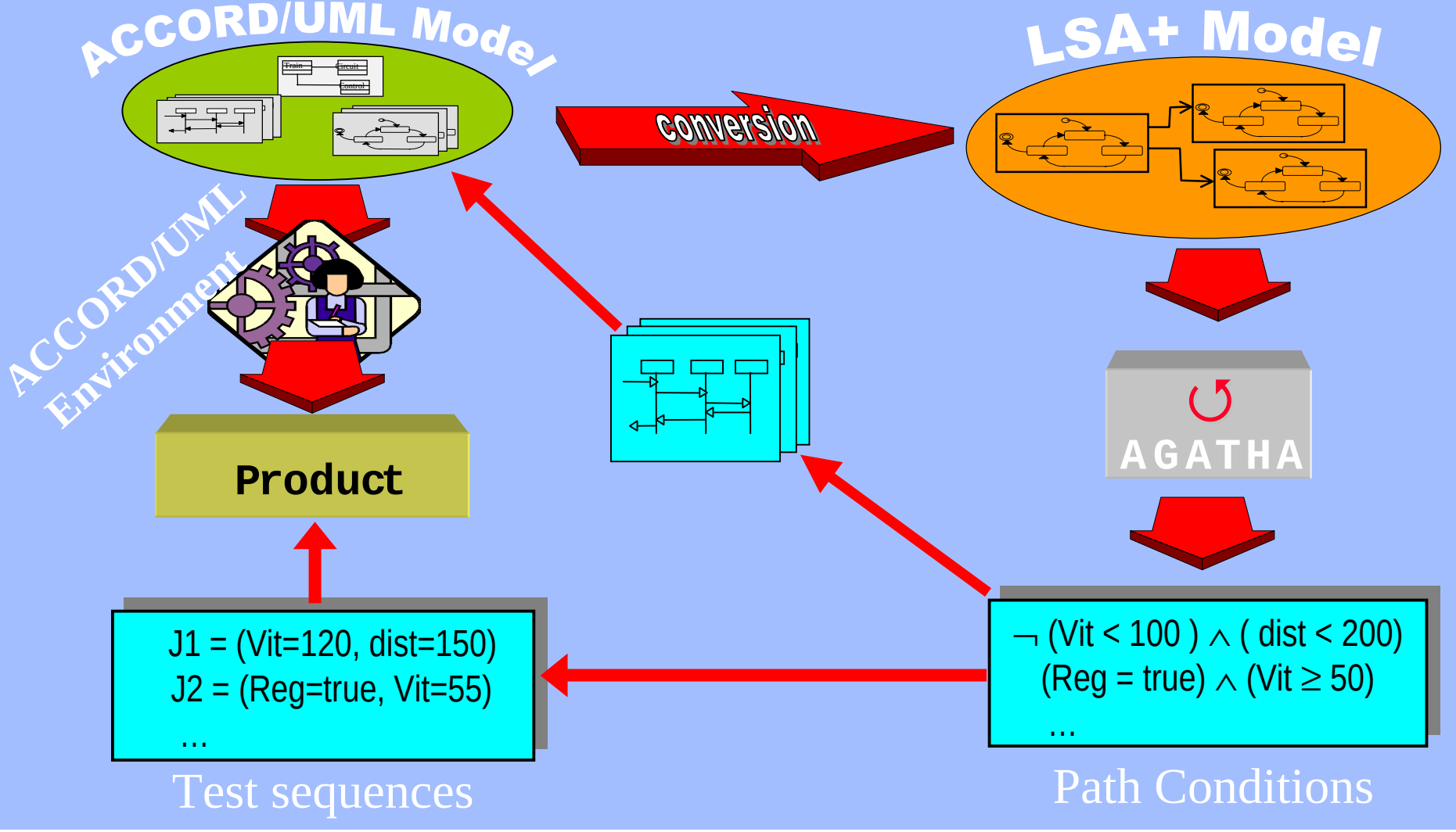
AIT-WOODDES : Workshop for Object Oriented Design & Development of Embedded System



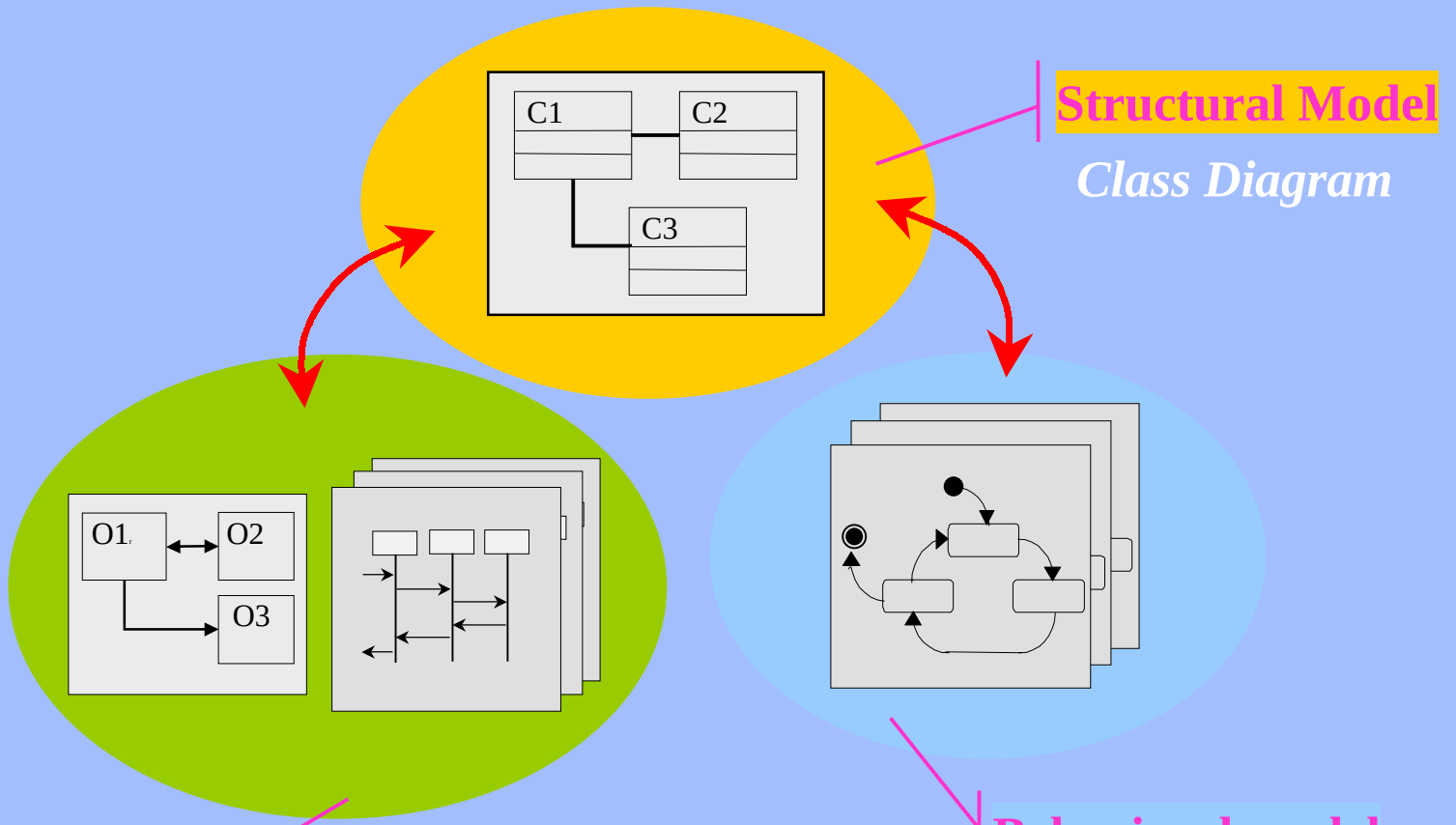
AGATHA : Un prototype de générateur de tests



UML to AGATHA connection



UML views



Structural Model

Class Diagram

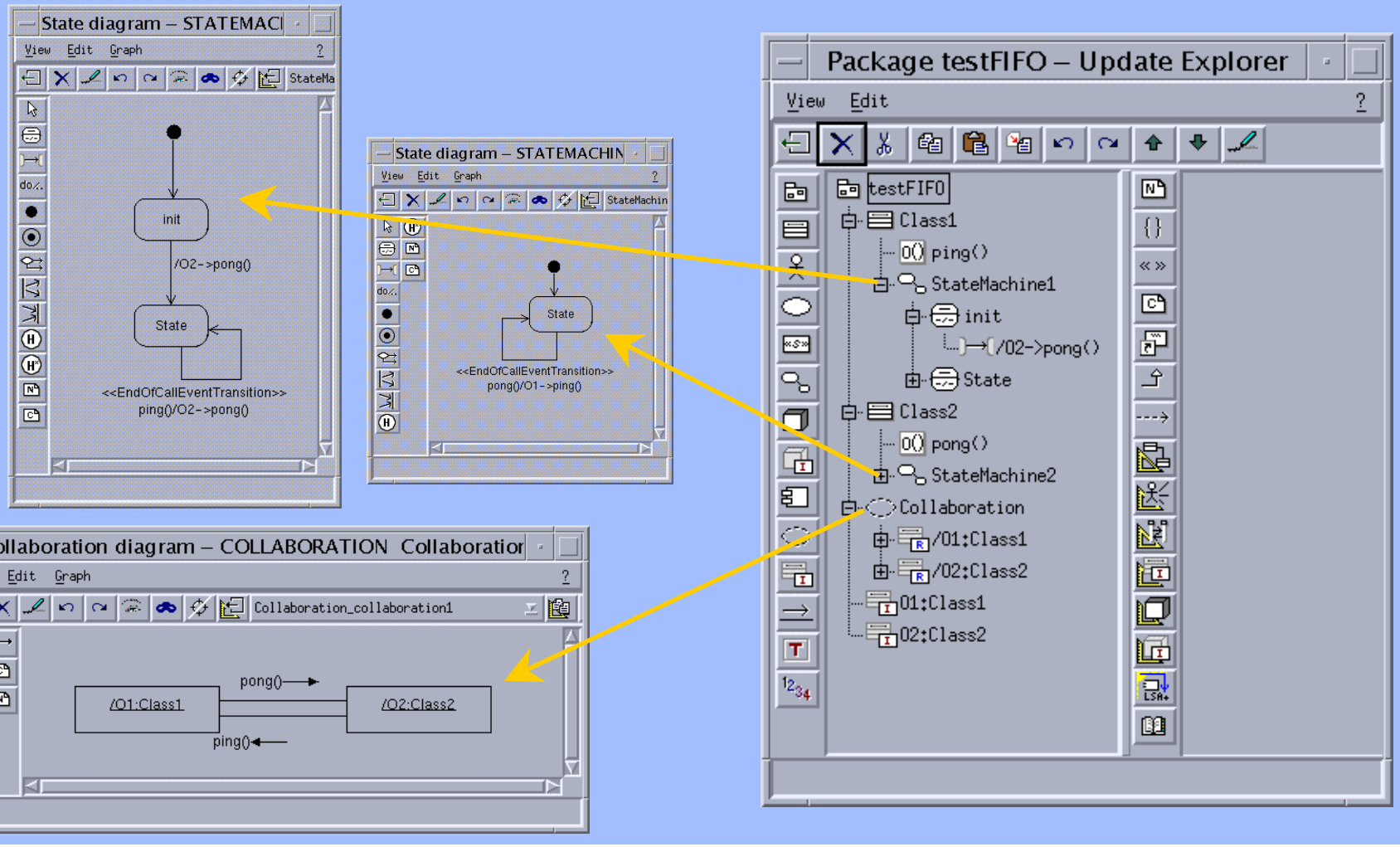
Behavioral model

Statechart

Interaction model

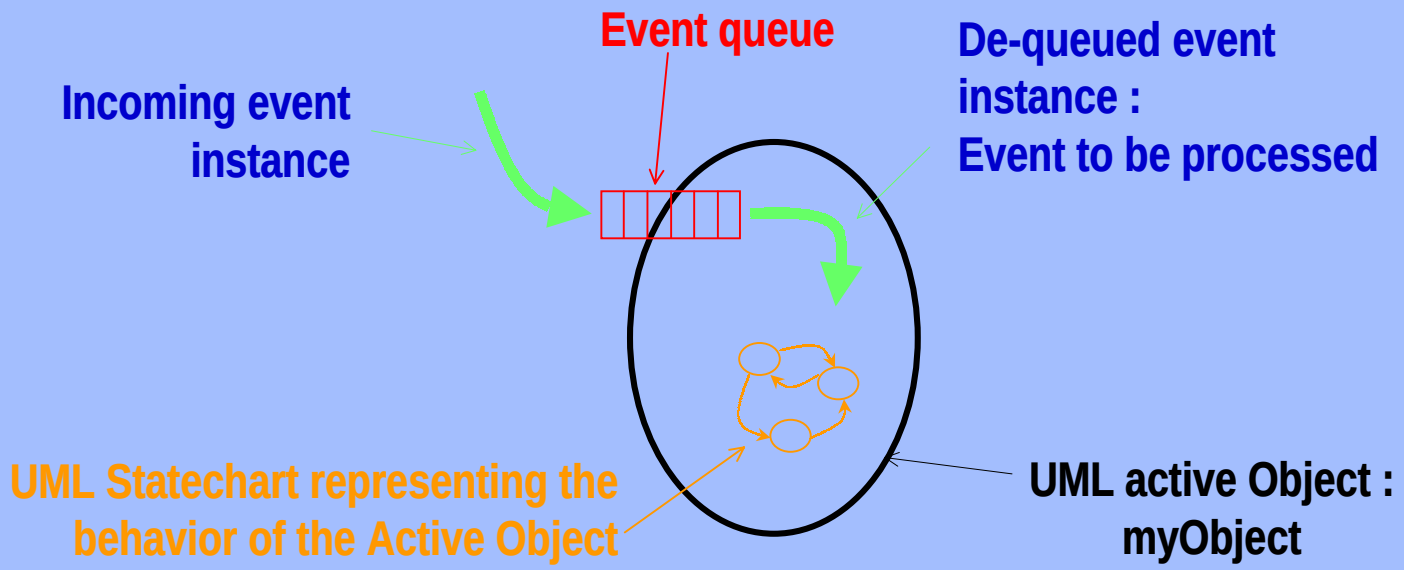
Collaboration & Sequence Diagrams

Objecteering 4 UML Modeler



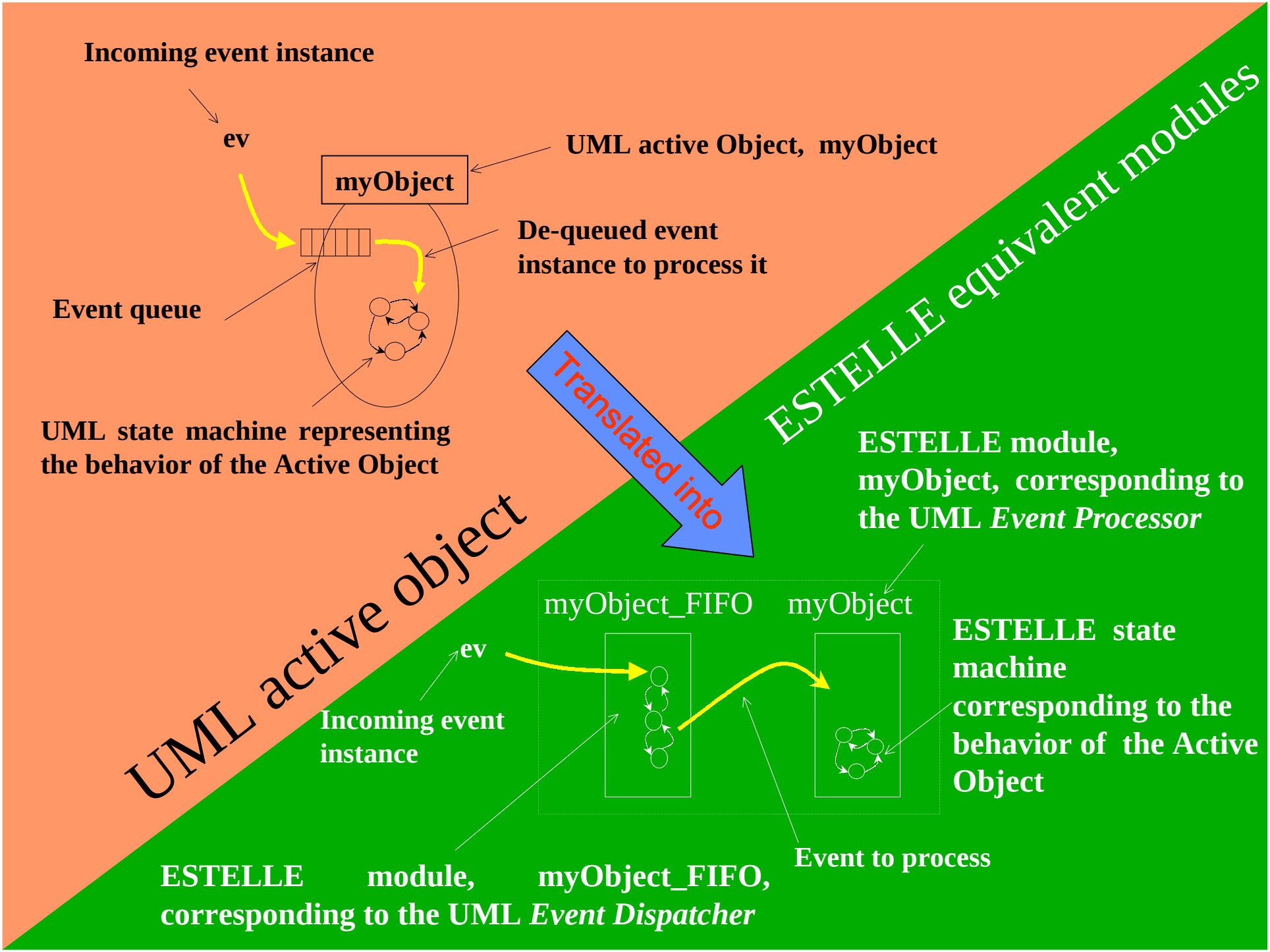
UML active objects principals

- UML Communication between objects
  - UML state machines define an hypothetical machine that holds, de-queues and processes events instances
  - FIFO dequeuing (commonly used by Object Oriented tools)



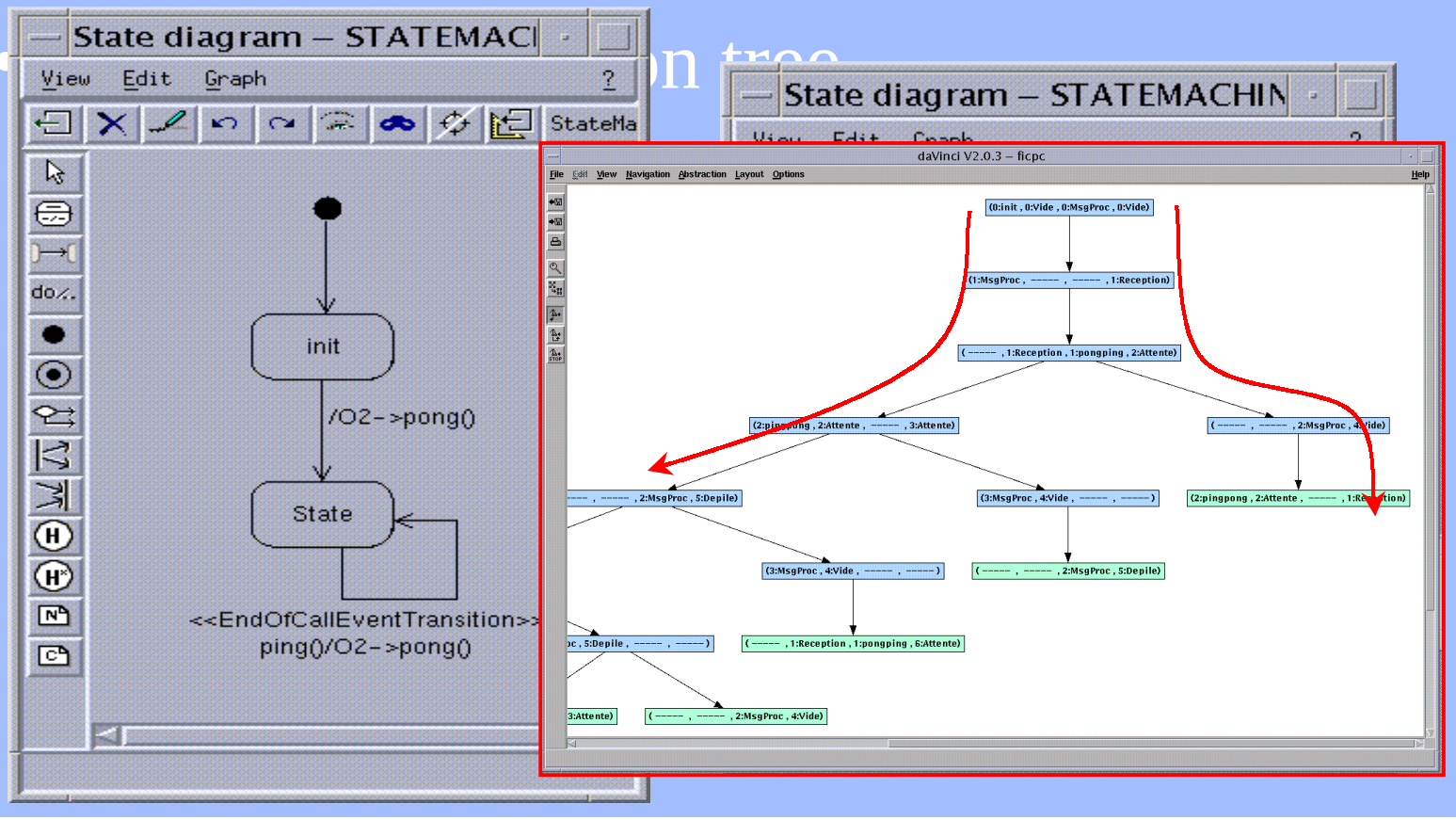
The RTC assumption

- ☞ When does the dispatcher is allowed to de-queue a message ?
- ☞ An Object does only process 1 message at a time
- ↳ The Dispatcher de-queue a new message when the current message processing is ended
- ↳ Synchronisation message between LSA+ modules

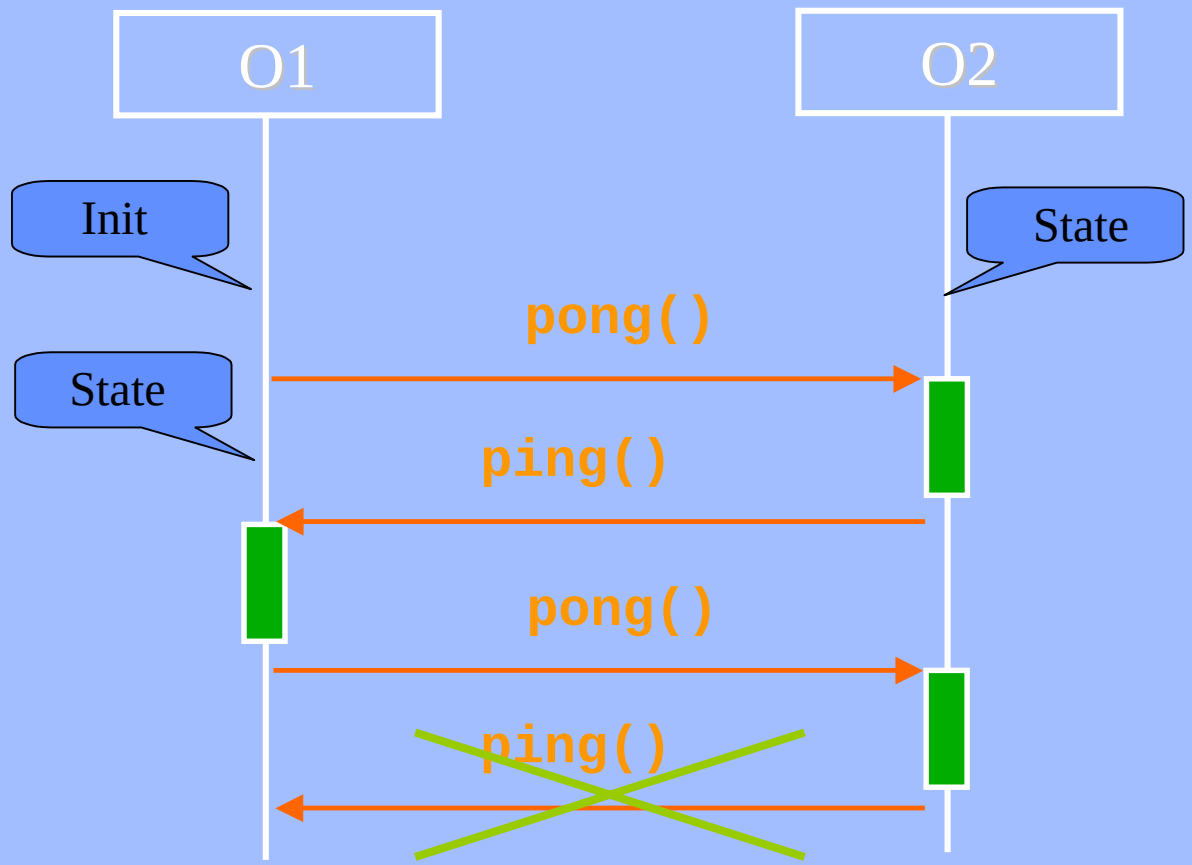

Incoming event instance
ev
myObject
UML active Object, myObject
De-queued event instance to process it
Event queue
UML state machine representing the behavior of the Active Object
Translated into
ESTELLE equivalent modules
UML active object
ESTELLE module, myObject, corresponding to the UML Event Processor
myObject_FIFO
myObject
ev
Incoming event instance
ESTELLE state machine corresponding to the behavior of the Active Object
Event to process
ESTELLE module, myObject_FIFO, corresponding to the UML Event Dispatcher

Ping-Pong Example

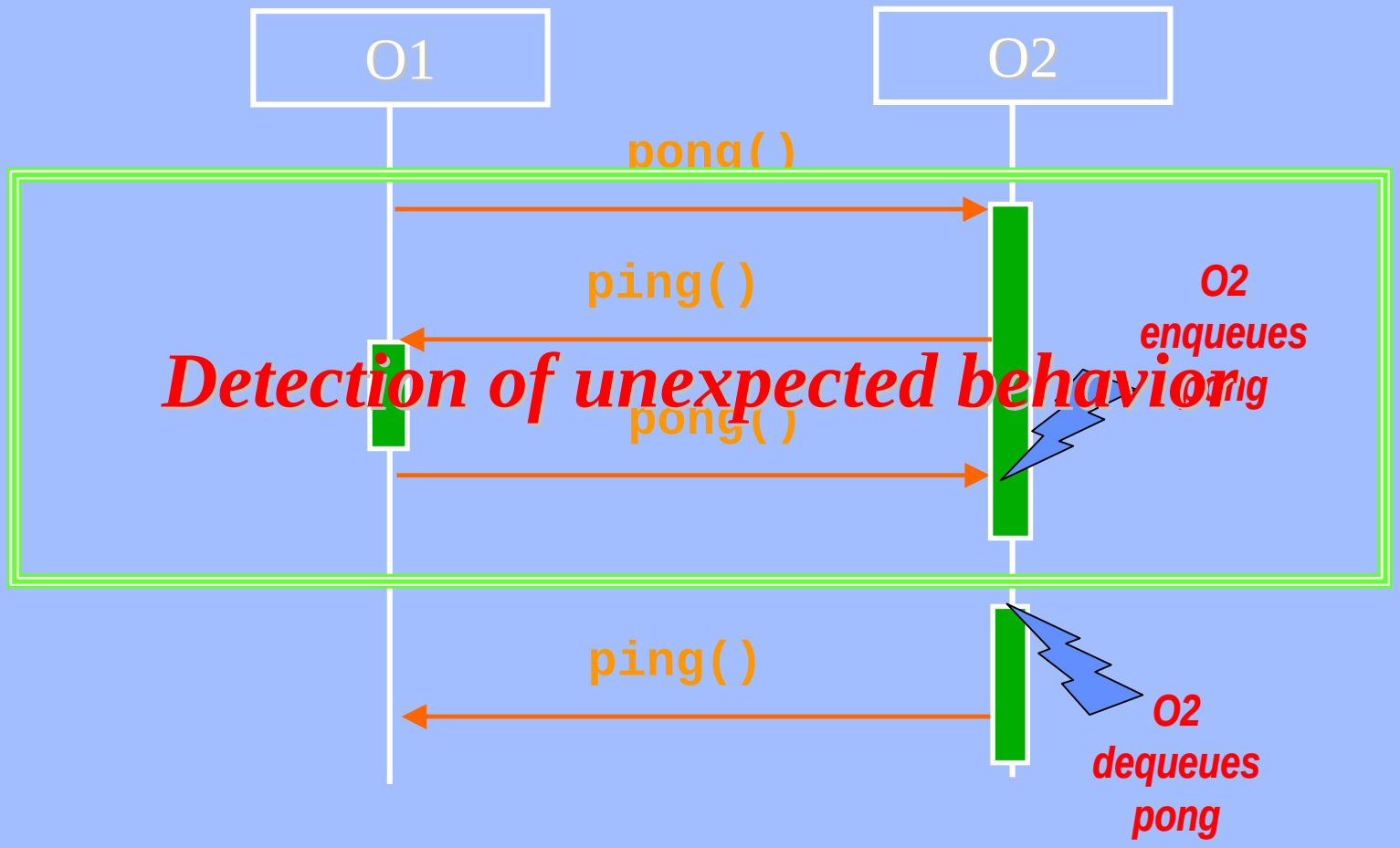
- Example of 2 simple parallel automaton



Intuitive communication case



Other execution



ACCORD/UML: synthesis

- Keeps to modeling concepts the common OO view
  - It can be used by non real-time specialists
- Provides models without implementation details
  - They can be changed without changing the structure of the models (nor in class, or sequence or state diagrams)
- Facilitate automatic generation of prototype code
- Allows model validation
  - through simulation or formal analysis (execution model can be deduced from the specification)

Conclusions

- **Profil ACCORD/UML => expression haut niveau de:**
  - Contrainte Temps-réel, Concurrence, Communication, Comportement
- **AGL industriel : Objecteering + modules d'extension**
  - Support des notations dédiées, Règles de modélisation
  - Génération de code TR mutli-tâches (C++ sous Solaris et VxWorks)
  - Framework multi-tâches temps-réel
- **Évaluation sur un cas d'étude PSA: Régulateur**
- **Contrat avec PSA pour le transfert**
- **Évaluation en cours sur application Télécom**

Travaux en cours et perspectives...

Poursuite des travaux

- Liens outils existant (Rhapsody, TAU UML) ⇒ AIT-WOODDES ⇒ proposition OMG
- Relation modèles flots de données/synchrones ⇒ RNTL ACOTRIS
- Déploiement/distribution et lien Soft/Hard ⇒ DAPNIA (A. Shebli)
- Techniques de validation par le test d'une application UML
  ⇒ Thèses de Phan Trung Hiêú (Propr. TR) et N. Rapin (composants)

Perspectives ouvertes

- Formalisation et génération de modèles d'implantation
- Définition d'une bibliothèque de patrons dédiés TR
- Intégration des contraintes liées aux systèmes critiques
- Liaison avec le Co-Design et les modèles continus

Some web sites

- AIT-WOODDES: *« Workshop for Object Oriented Design and Development of Embedded Systems », IST project of the 5th PCRD*
  http://wooddes.intranet.gr/project.htm
- SIVOOES: *ECOOP'2000 workshop on « Specification, Implementation and Validation of Object-oriented Embedded Systems »*
  http://www-dta.cea.fr/leti/UK/Pages/Tech_info/Sivooes.htm
- UML'2000: http://www.cs.york.ac.uk/uml2000
  Workshop on Formal Design Techniques for Real-Time UML
  http://wooddes.intranet.gr/workshop.htm
- Action semantics: *AD/98-11-01*, http://www.omg.org, http://uml.simware.com
  http://people.ce.mediaone.net/weigert/actionsemantics/home.html
- ARTiSAN: http://www.artisansw.com
- RT-UML (Rhapsody): http://www.ilogix.com
- UML-SDL (UML TAU Suite): http://www.telelogic.com
- UML-RT (ROSE-RT): http://www.rational.com
- Objecteering: http://www.softeam.com