



Introduction à UML

Shebli Anvar – DSM/DAPNIA – CEA Saclay
François Terrier, Sébastien Gérard DRT/LIST – CEA/Saclay

F-91191 Gif sur Yvette Cedex France
Francois.Terrier@cea.fr ; Sebastien.Gerard@cea.fr ;
Shebli.Anvar@cea.fr

Définitions

- ◆ UML = **Unified Modeling Language**
 - Langage unifié pour la modélisation objet
 - Langage de modélisation des applications construites à l'aide d'objets, indépendant de la méthode utilisée
- ◆ Différence Langage – Méthode
 - Langage de modélisation = notations, grammaire, sémantique
 - Méthode : comment utiliser le langage de modélisation (recueil des besoins, analyse, conception, mise en œuvre, validation...)
- ◆ Objet = représentation du problème basée sur des entités (concrètes ou abstraites) du monde réel

La complexité des logiciels

- ◆ Le logiciel est complexe par nature → gérer cette complexité
- ◆ Les systèmes peuvent être décomposés selon
 - ce qu'ils font (approche fonctionnelle)
 - ce qu'ils sont (approche objet)
- ◆ L'approche objet gère plus efficacement la complexité

Historique des langages OO

- ◆ Langages de programmation orientés objets
 - Simula (1967)
 - Smalltalk (1970)
 - C plus Classes (1980)
 - C++ (1985)
 - Eiffel (1988)
 - Java (1995)
- ◆ SGBD orientés objets
 - Utilisation des objets avec un langage OO
- ◆ Genèse des méthodes d'analyse
 - Implémentation
 - Conception (solution informatique)
 - Analyse (comprendre et modéliser le problème)
 - ...

Les méthodes d'analyse

◆ Méthodes orientées comportement

- on s'intéresse à la dynamique du système
ex : réseaux de Pétri

◆ Méthodes fonctionnelles :

- s'inspirent de l'architecture des ordinateurs
- on s'intéresse aux fonctions du système
ex : SADT

◆ Méthodes orientées données :

- on ne s'intéresse pas aux traitements
ex : MERISE

◆ Méthodes orientées objets :

- on ne sépare pas les données et les traitements
ex : Booch, OMT

L'unification

◆ des méthodes

- La guerre des méthodes ne fait plus avancer la technologie des objets
- Recherche d'un langage commun unique
 - ◆ Utilisable par toutes les méthodes
 - ◆ Adapté à toutes les phases du développement
 - ◆ Compatible avec toutes les techniques de réalisation

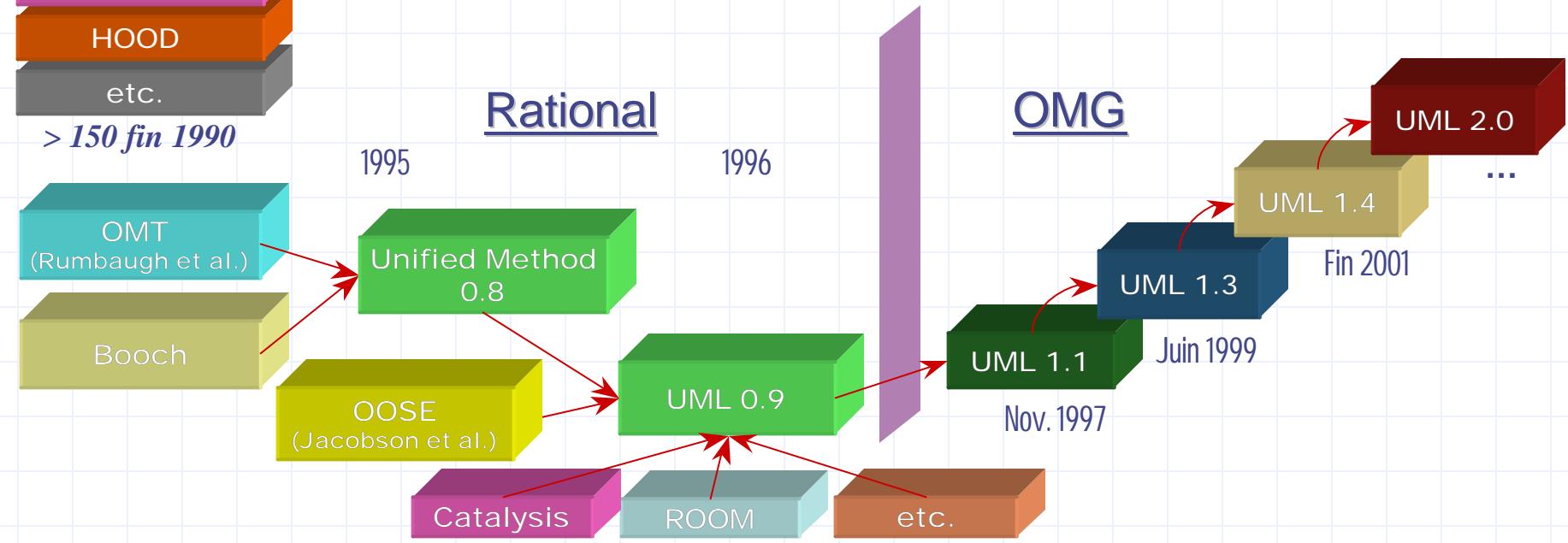
◆ sur plusieurs domaines d'applications

- Logiciels → Ingénierie des logiciels
- Logiciels et matériels → Ingénierie des systèmes
- Personnes → Ingénierie des affaires

Intérêt d'un standard de modélisation universel



- ❖ Passer de l'artisanat à la production industrielle
 - Modélisation haut niveau
 - Développement basé sur composants
 - Intégration de procédés de modélisations complémentaires
 - Notation unifiée pour toutes les méthodologies OO



Unified Modeling Language

- ◆ Langage = syntaxe + sémantique
 - Syntaxe
 - ◆ Règles selon lesquelles les éléments du langage (ex. les mots) sont assemblés en des expressions (ex. phrases, clauses).
 - Sémantique
 - ◆ Règles permettant d'attribuer une signification aux expressions syntactiques

→ UML Notation Guide

→ UML Semantics

OMG UML 1.4 Specification: <http://www.omg.org>

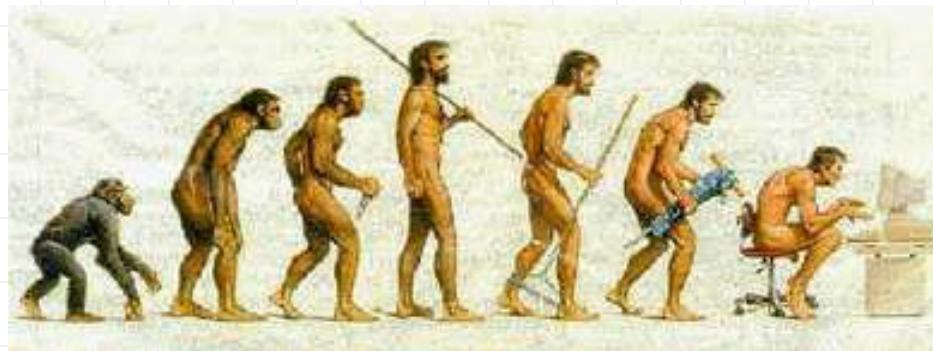
- ◆ UML Summary
- ◆ UML Semantics
- ◆ UML Notation Guide
- ◆ UML Standard Profiles
- ◆ UML CORBA Facility Interface Definition
- ◆ UML XML Metadata Interchange DTD
- ◆ Object Constraint Language

Autres documents

- ◆ CORBA 2.4.2 (inclus Real-Time CORBA specifications)
- ◆ Meta-Object Facility (MOF)
Base commune CORBA, UML, etc.
- ◆ UML v. 1.4
 - OMG UML Tutorials:
<http://www.celgent.com/omg/umlrtf/tutorials.htm>
 - OMG UML Resources: <http://www.omg.org/uml/>
 - Pierre-Alain Muller, Essaim Mulhouse : www.uml.crespim.uha.fr
- ◆ Profile for Action semantics
http://www.kc.com/as_site/home.html
- ◆ Profile for Scheduling, Performance and Time
- ◆ UML Profile for CORBA

Objectifs

- ◆ Représenter des systèmes entiers
- ◆ Choisir la granularité de la description
- ◆ Établir un couplage explicite entre concepts et artefacts exécutables
- ◆ Programmation sans programmer : créer un langage de modélisation utilisable à la fois par les humains et les machines



Caractéristiques du langage de modélisation UML

- ◆ Générique et Expressif
- ◆ Syntaxe et sémantique définis
- ◆ Flexible (configurable, extensible)
 - Définition du Métamodèle
 - Norme non figée
 - On peut adapter le langage à des domaines particuliers sans ajouter de nouveaux types de diagrammes
 - Introduction d'une nouvelle notion en la définissant comme particularisme d'une notion existante

Portée

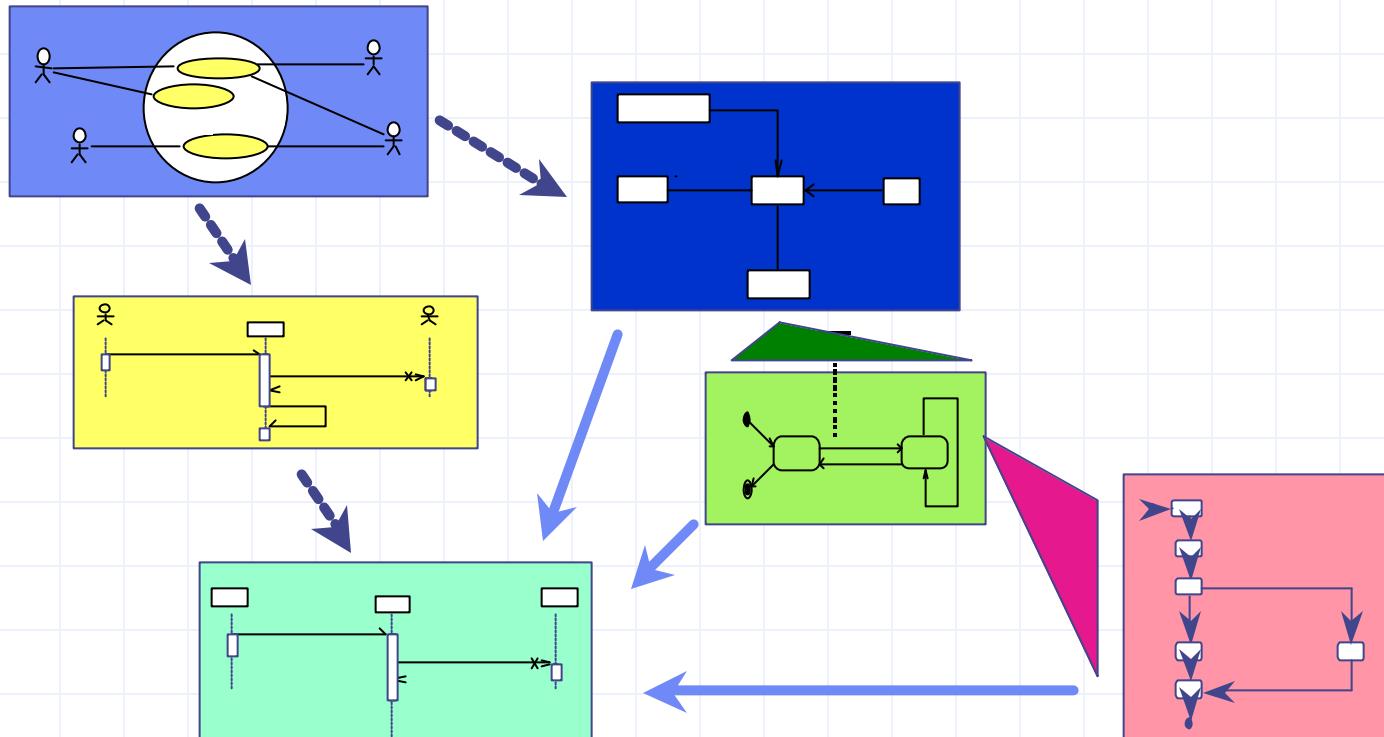
- ◆ Formalisme unique pour tout type d'application
 - gestion, scientifique, temps réel, industrielle, multimédia...
- ◆ Reste au niveau d'un langage
 - ne propose pas un processus de développement
 - ni ordonnancement des tâches,
 - ni répartition des responsabilités,
 - ni règles de mise en œuvre
(Certains ouvrages et AGL basés sur UML ajoutent cet aspect fondamental en méthodologie)

Démarche

- ◆ Ensemble de point de vues complémentaires
- ◆ Développement par raffinages successifs

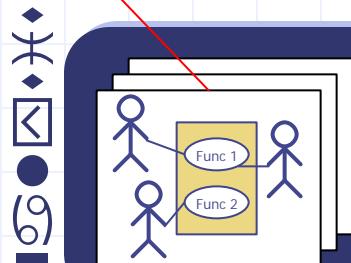
Des modèles complémentaires

- ◆ Des règles de cohérence pour une modélisation non ambiguë
- ◆ Des vues complémentaires pour un modèle complet
- ◆ L'analyse formelle du modèle devient possible

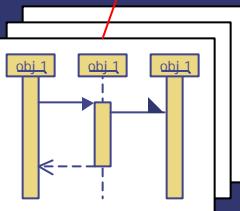


Des modèles de plus en plus détaillés

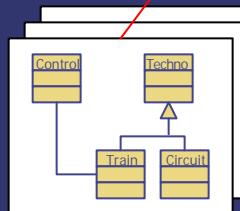
UML Use Case
Diagrams



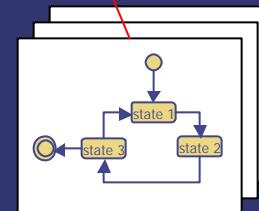
UML Interaction
Diagrams



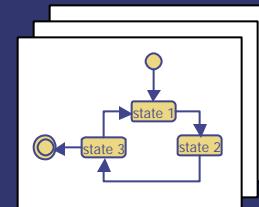
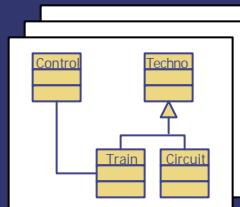
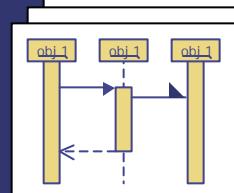
UML Class
Diagrams



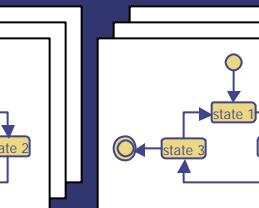
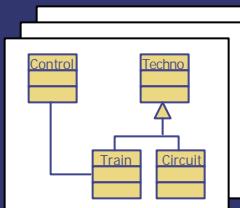
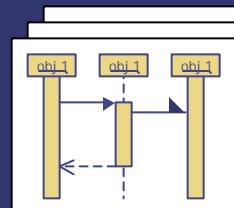
UML Statechart
Diagrams



Modèle d'ensemble



UML Activity
Diagrams



En résumé

- ◆ UML est un langage de modélisation objet
- ◆ UML est une notation, pas une méthode
- ◆ UML convient pour toutes les méthodes objet
- ◆ UML est dans le domaine public

UML est la notation standard pour documenter
les modèles objets

Caractéristiques fondamentales des objets

- ◆ Objet

- État
- Comportement
- Identité

- ◆ Communication entre objets

L'état

- ◆ L'état regroupe les valeurs instantanées de tous les attributs d'un objet
- ◆ L'état évolue au cours du temps
- ◆ L'état d'un objet à un instant donné est la conséquence de ses comportements passés
- ◆ L'état conditionne son comportement futur

Le comportement

- ◆ Décrit les actions et les réactions d'un objet
 - du point de vue externe (opérations)
 - Du point de vue interne (méthodes)
- ◆ L'état et le comportement sont liés
 - Le comportement dépend de l'état
 - L'état est modifié par le comportement

L'identité

- ◆ Tout objet possède une identité qui lui est propre et qui le caractérise
- ◆ L'identité permet de distinguer tout objet de façon non ambiguë, indépendamment de l'état
- ◆ L'identité d'un objet est immuable tout au long de sa vie

Communication entre objets

- ◆ Application = société d'objets collaborant
- ◆ Les objets travaillent en synergie afin de réaliser les fonctions de l'application
- ◆ Le comportement global d'une application repose sur la communication entre les objets qui la composent

Les classes

- ◆ La classe est une description abstraite d'un ensemble d'objets « de même type »
- ◆ La classe peut être vue comme la factorisation des descriptions communes à un ensemble d'objets
- ◆ La classe est également un espace de nommage (*Namespace*)

Les 9 diagrammes

Besoins des utilisateurs

Diagramme des cas d'utilisation

Structure statique

Diagramme de classes
Diagramme objet

Dynamique des objets

Diagramme états-transition
Diagramme d'activités

Interactions entre objets

Diagramme de séquence
Diagramme de collaboration

Réalisation et déploiement

Diagramme de composants
Diagramme de déploiement

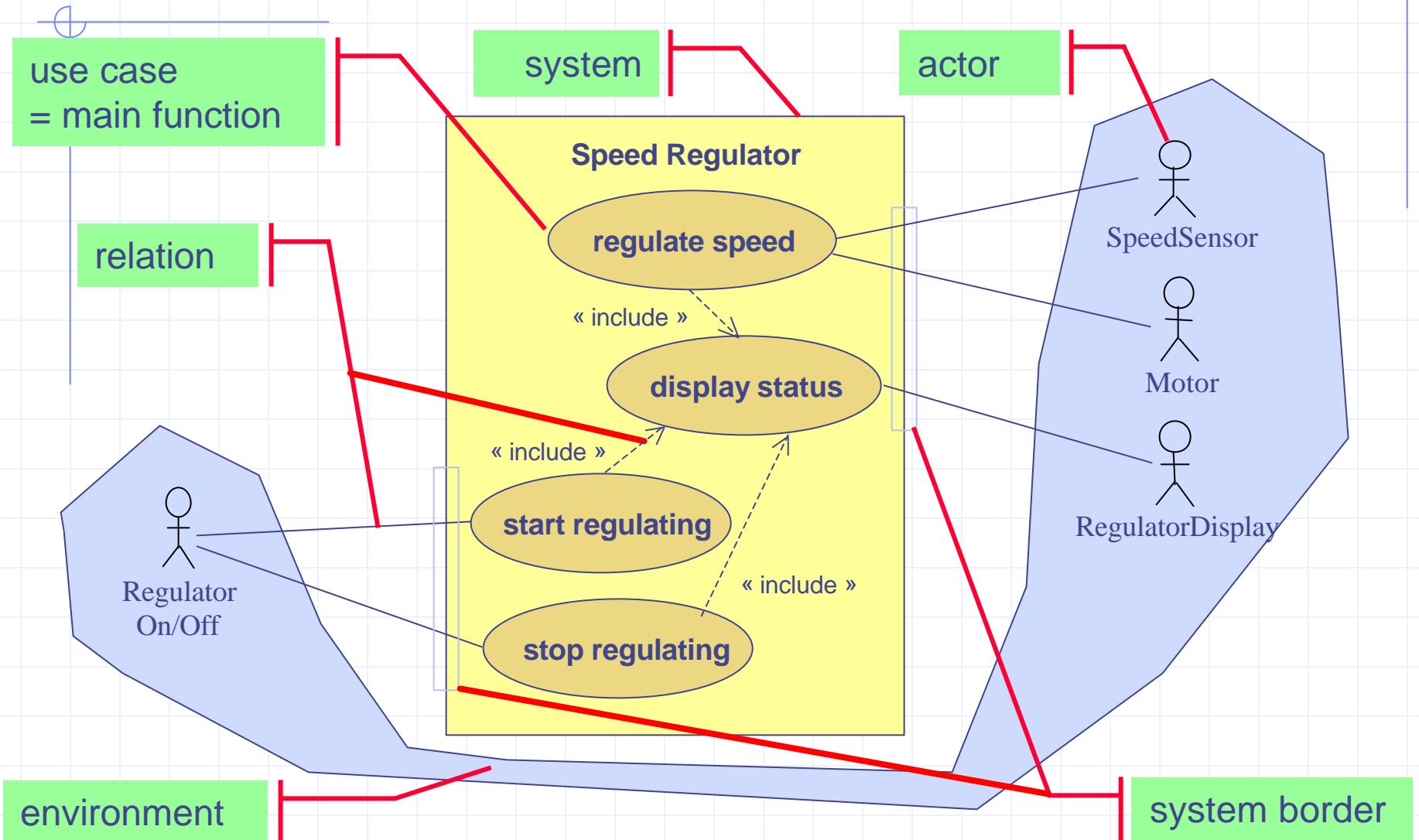
Points de vues de modélisation

- ◆ Spécifier le système
- ◆ Modéliser des objets communicants
- ◆ Modéliser la structure de l'application
- ◆ Modéliser le comportement des objets
- ◆ Modéliser les traitements
- ◆ Modéliser l'instanciation de l'application

Points de vues de modélisation

- ◆ Spécifier le système
- ◆ Modéliser des objets communicants
- ◆ Modéliser la structure de l'application
- ◆ Modéliser le comportement des objets
- ◆ Modéliser les traitements
- ◆ Modéliser l'instanciation de l'application

Use case diagram

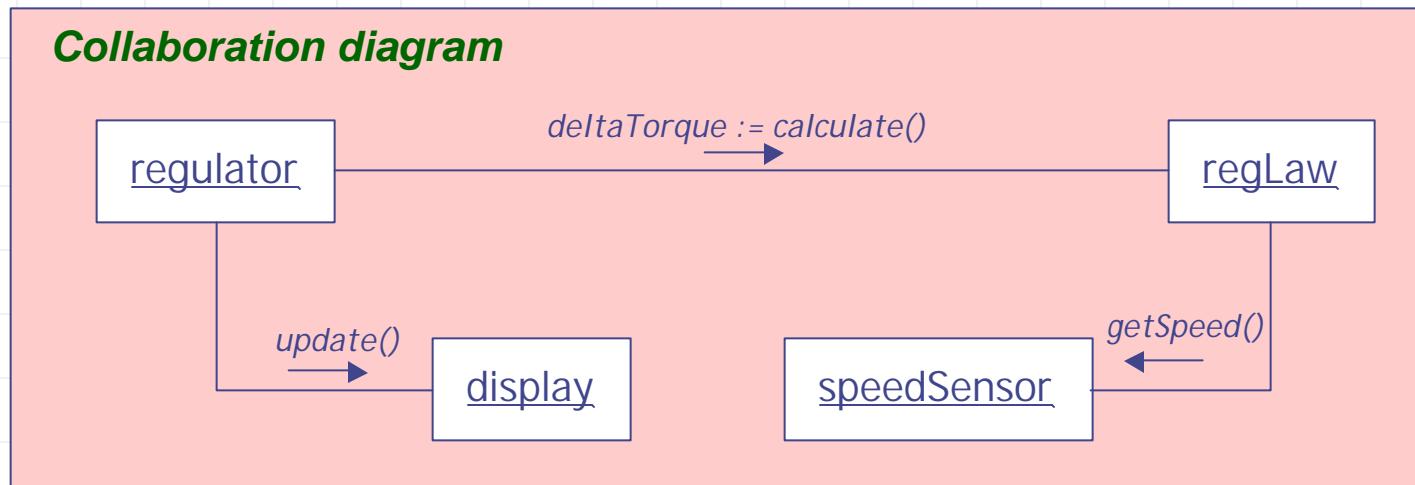


Points de vues de modélisation

- ◆ Spécifier le système
- ◆ Modéliser des objets communicants
 - Instances, liens, messages
 - Mécanismes de communication
 - Diagrammes de collaboration
 - Diagrammes de séquence
- ◆ Modéliser la structure de l'application
- ◆ Modéliser le comportement de l'objet
- ◆ Modéliser les traitements
- ◆ Modéliser l'instanciation de l'application

Instances, liens, messages

- ◆ On identifie et nomme les objets (instances) qui interviennent dans le système
- ◆ D'abord, les objets « physiques » puis les objets plus abstraits
- ◆ On spécifie les liens entre objets, puis les messages transitant par ces liens



Mécanisme de communication

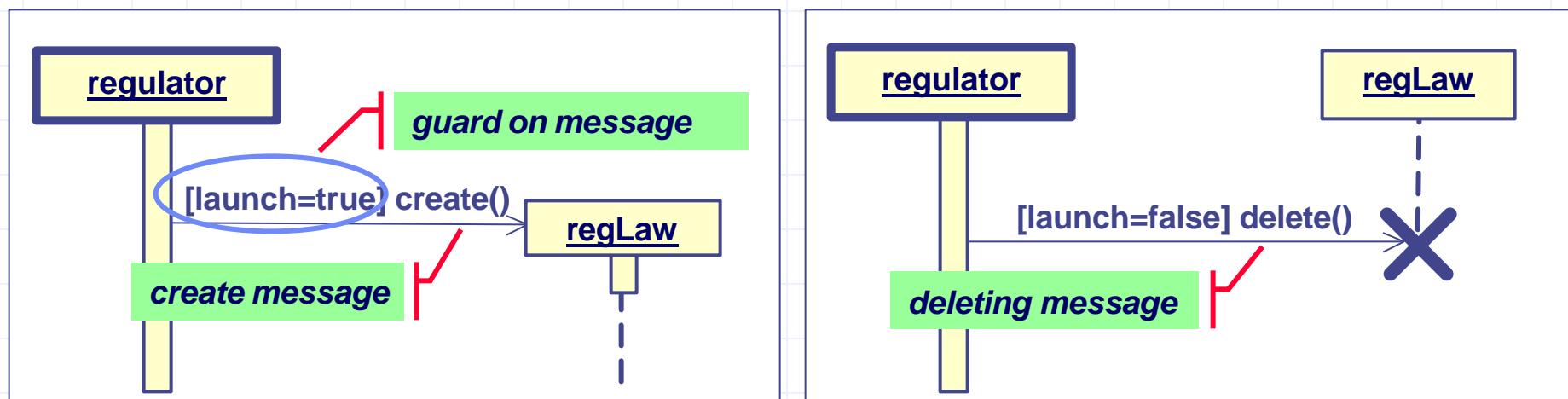
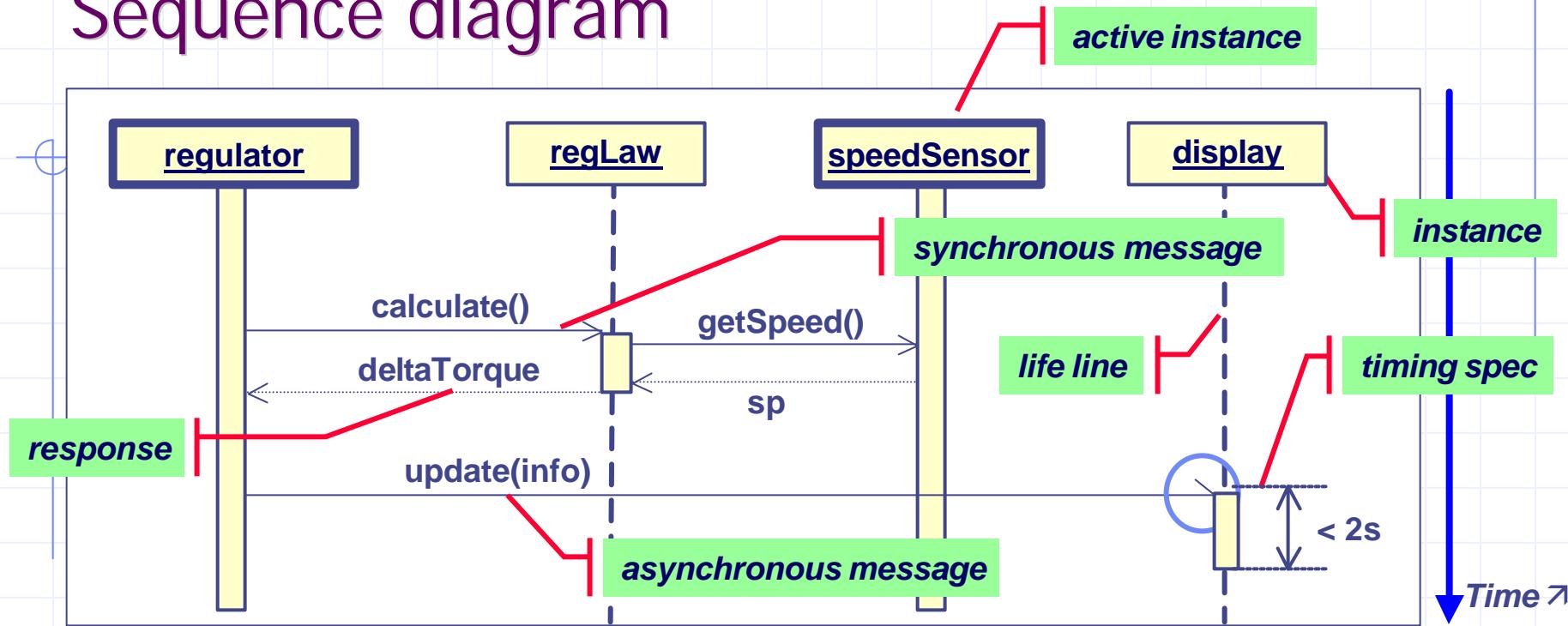
◆ Communication: uniquement par messages

- Message = une action + un événement
 - ◆ En général point à point
 - ◆ Possibilité d'un ensemble de cibles

◆ Deux types d'envoi de message

- Appel d'opération (CallAction + CallEvent)
 - ◆ Synchrone/asynchrone, paramètres input et output
- Signal (SendAction + SignalEvent)
 - ◆ Asynchrone, paramètres input seulement

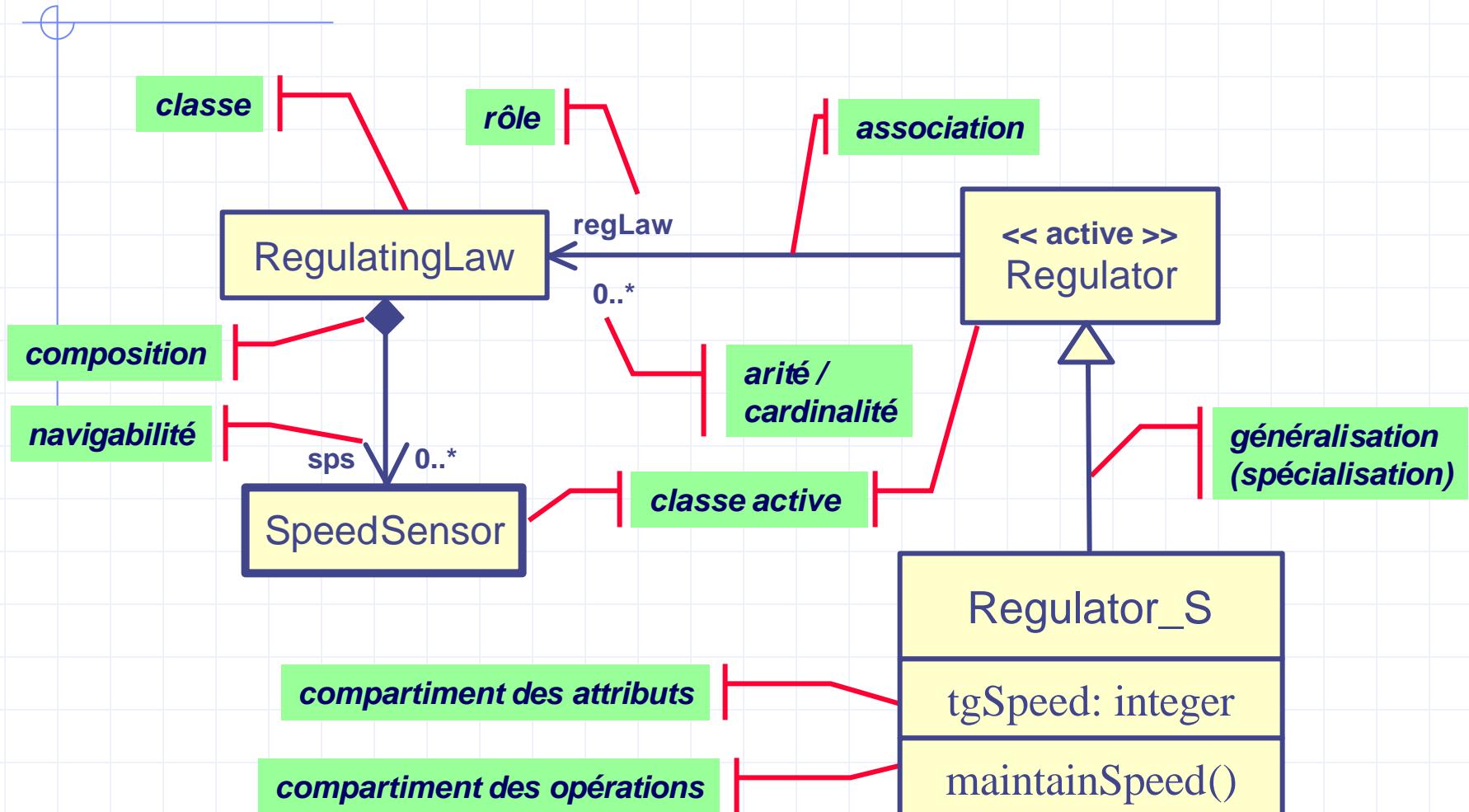
Sequence diagram



Points de vues de modélisation

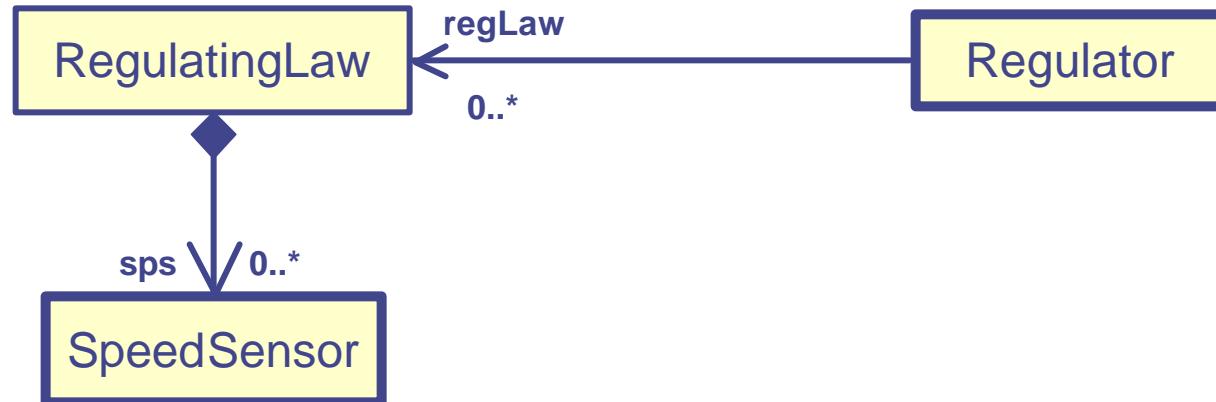
- ◆ Spécifier le système
- ◆ Modéliser des objets communicants
- ◆ Modéliser la structure de l'application
 - Diagrammes de classes
 - Paquetages
- ◆ Modéliser le comportement de l'objet
- ◆ Modéliser les traitements
- ◆ Modéliser l'instanciation de l'application

Class diagram

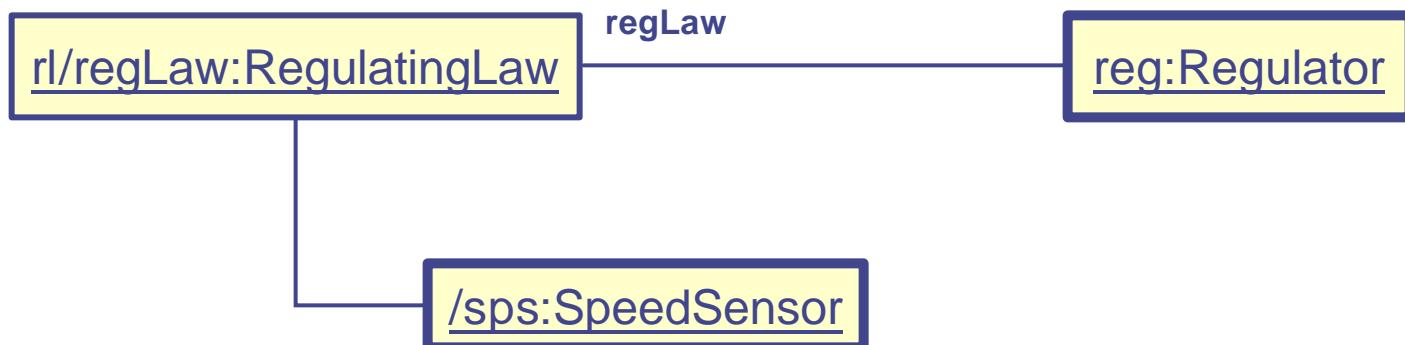


Indication de type d'instance

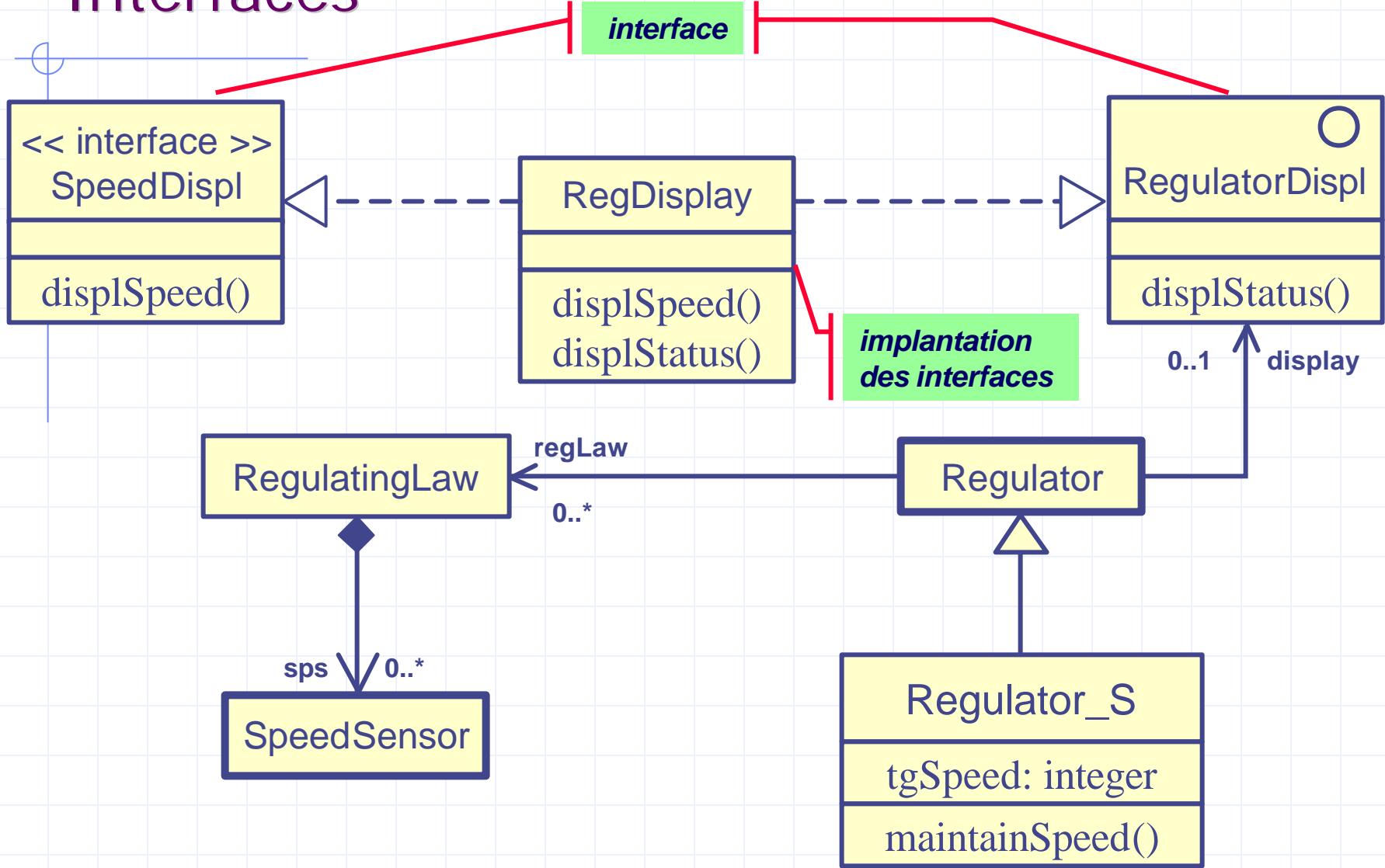
Collaboration diagram



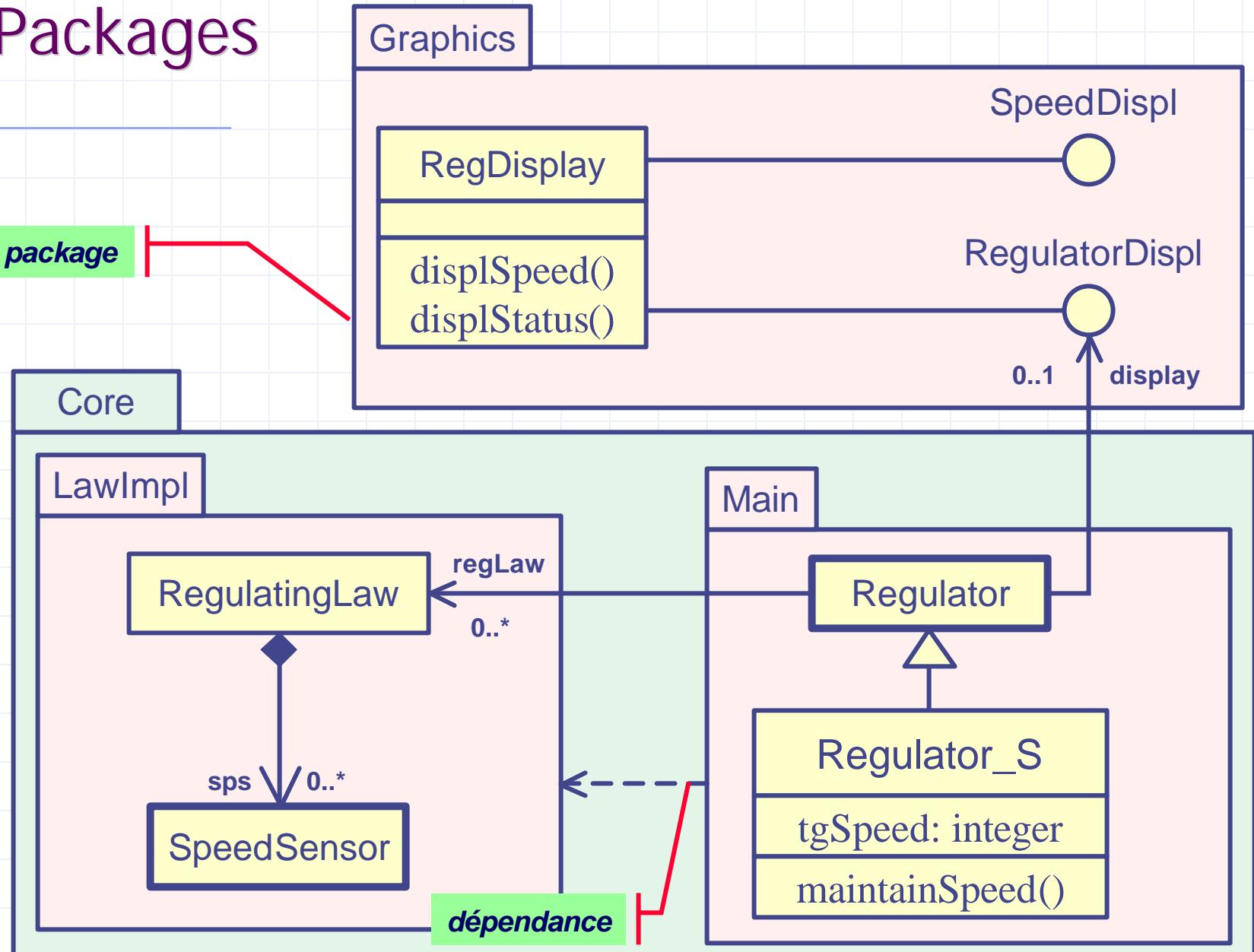
Instance / role diagram



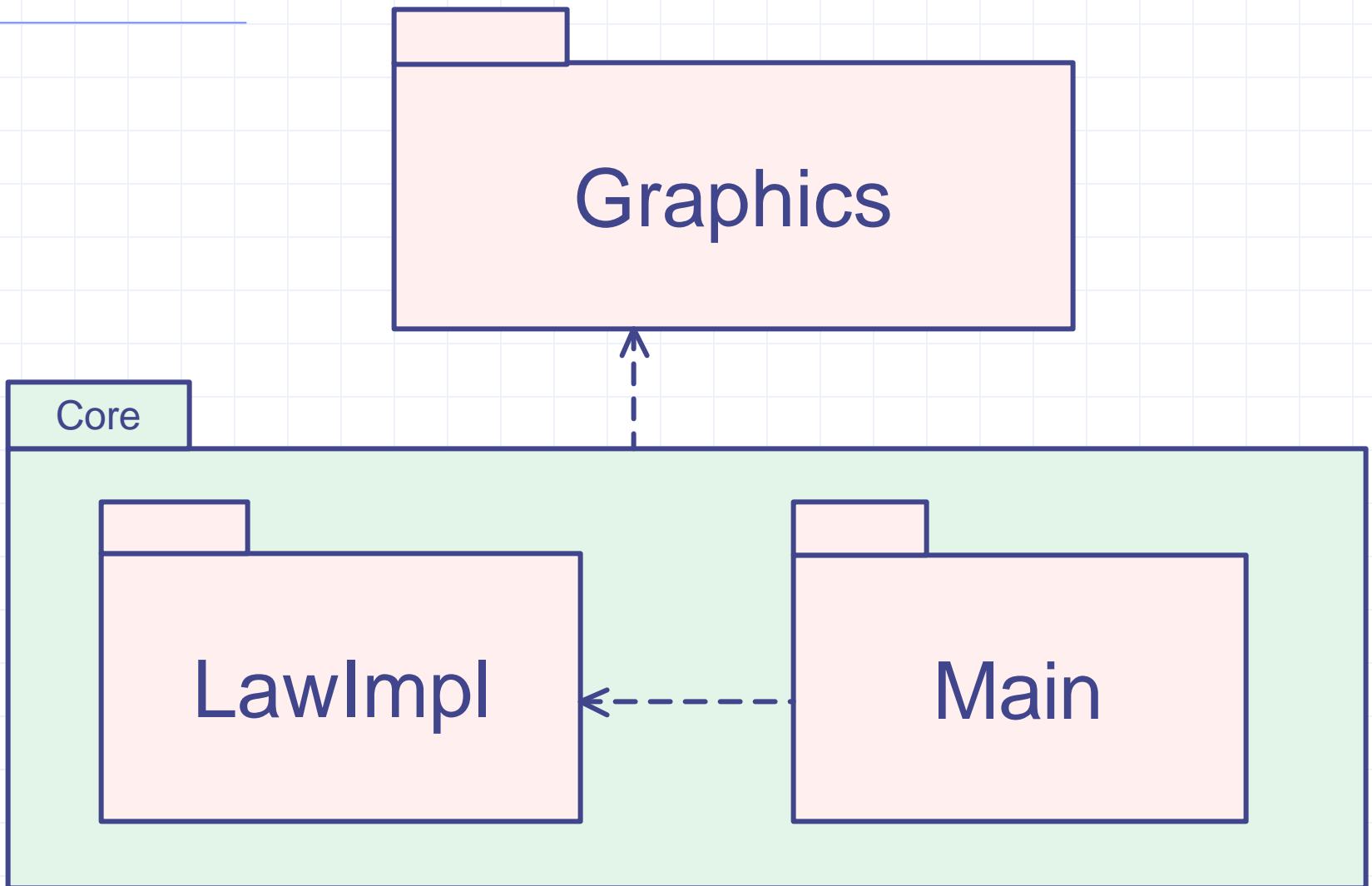
Interfaces



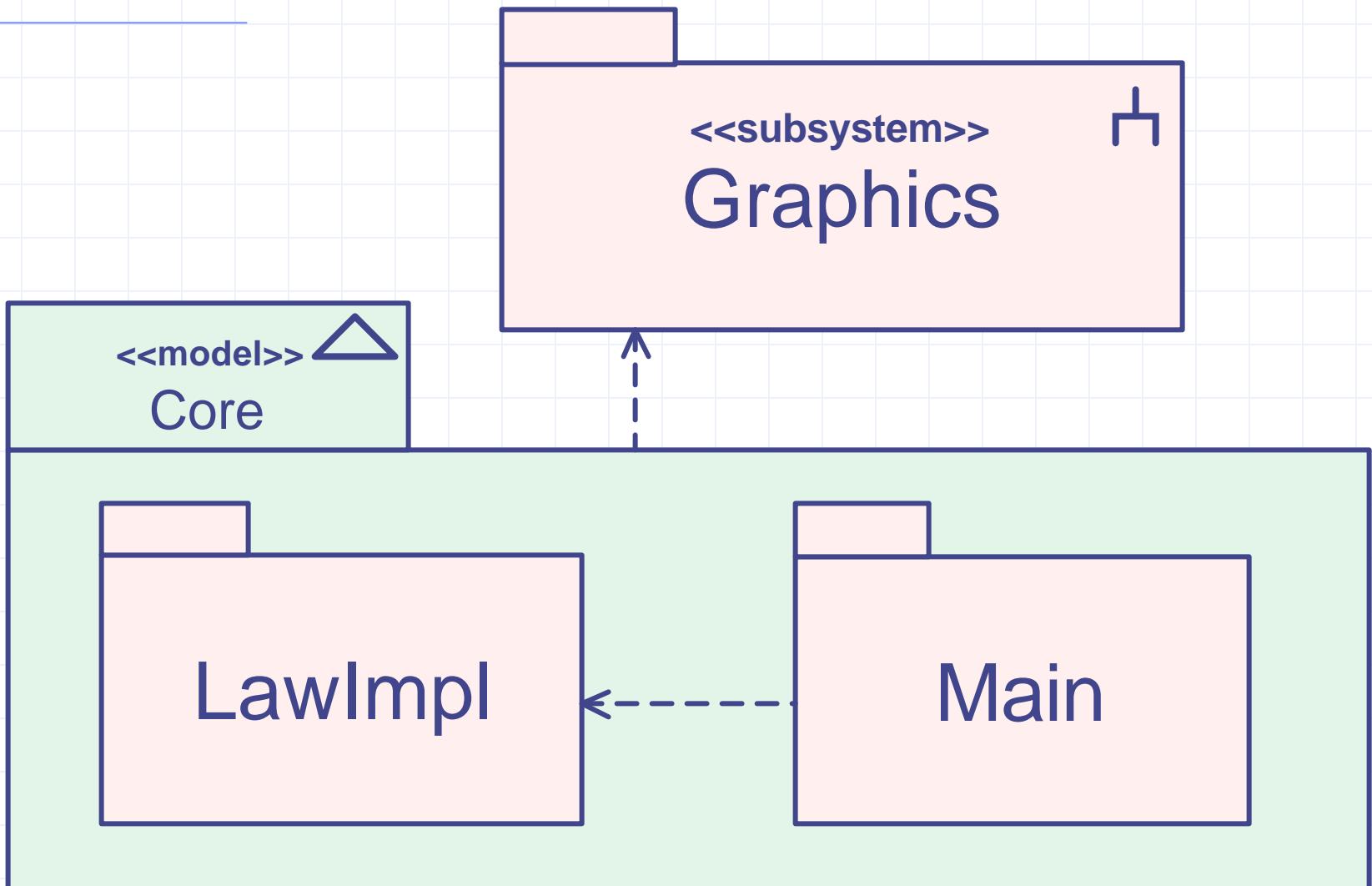
Packages



Packages



Packages

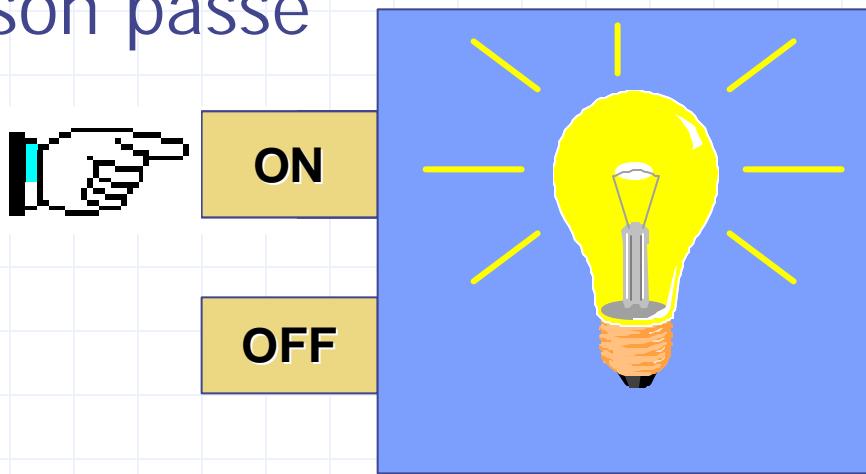


Points de vues de modélisation

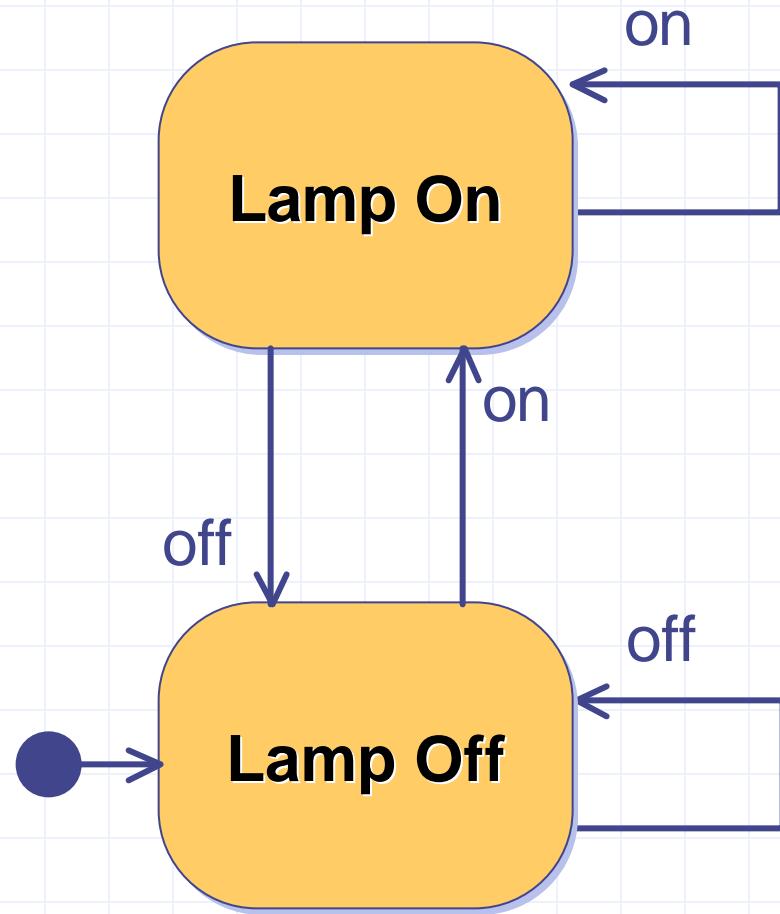
- ◆ Spécifier le système
- ◆ Modéliser des objets communicants
- ◆ Modéliser la structure de l'application
- ◆ Modéliser le comportement de l'objet
 - Machines d'état
- ◆ Modéliser les traitements
- ◆ Modéliser l'instanciation de l'application

Automate

- ◆ Une machine dont le comportement en output résulte de
 - Les inputs courants
 - Les inputs passés
- ◆ Caractérisée par un état interne représentant son passé

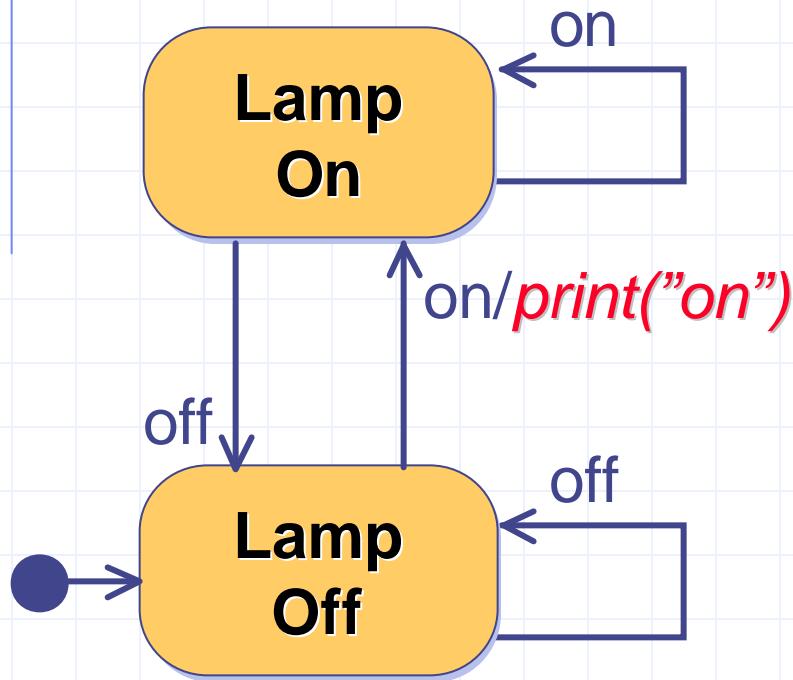


Statechart Diagram

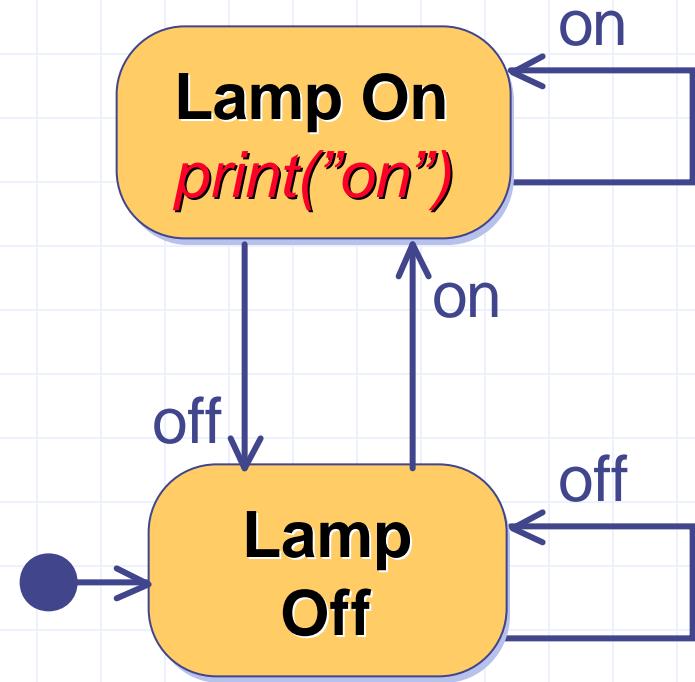


Outputs et Actions

- ◆ Le changement d'état peut générer des outputs



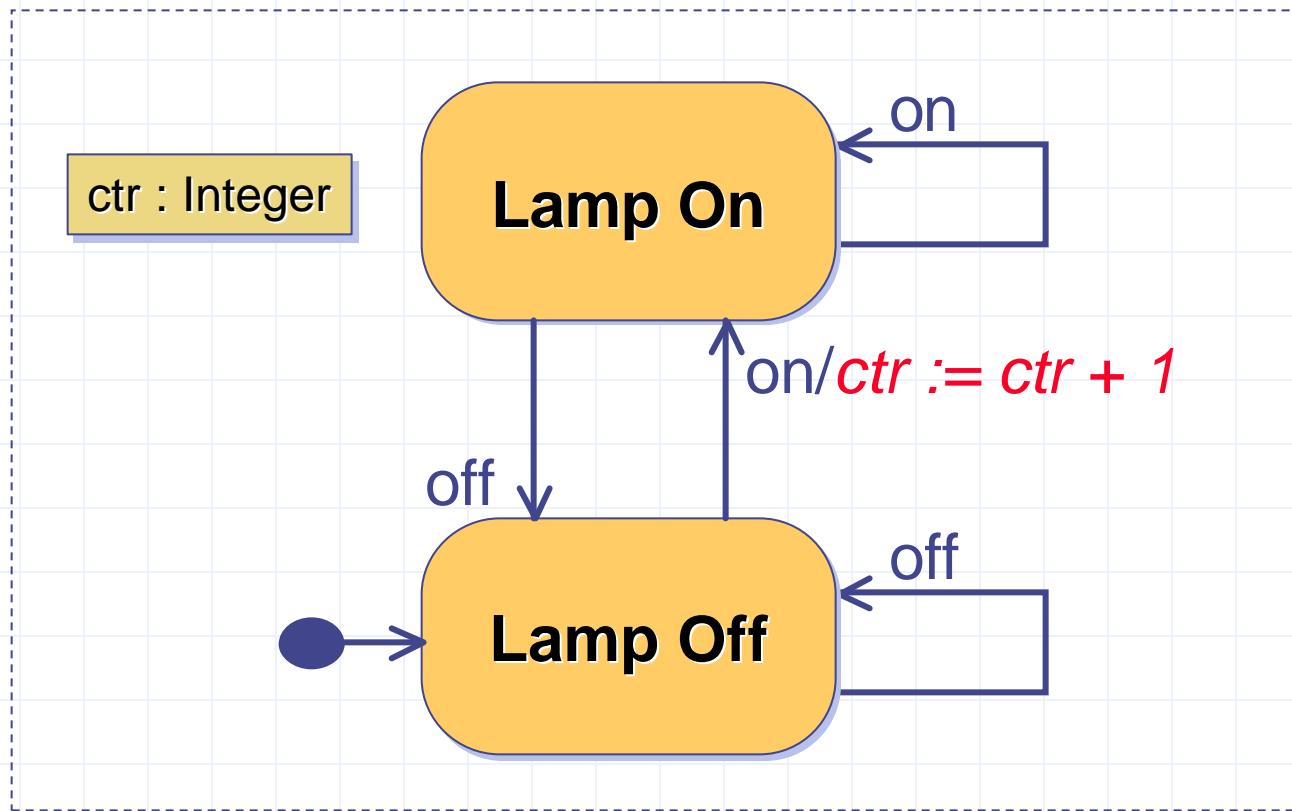
Mealy automaton



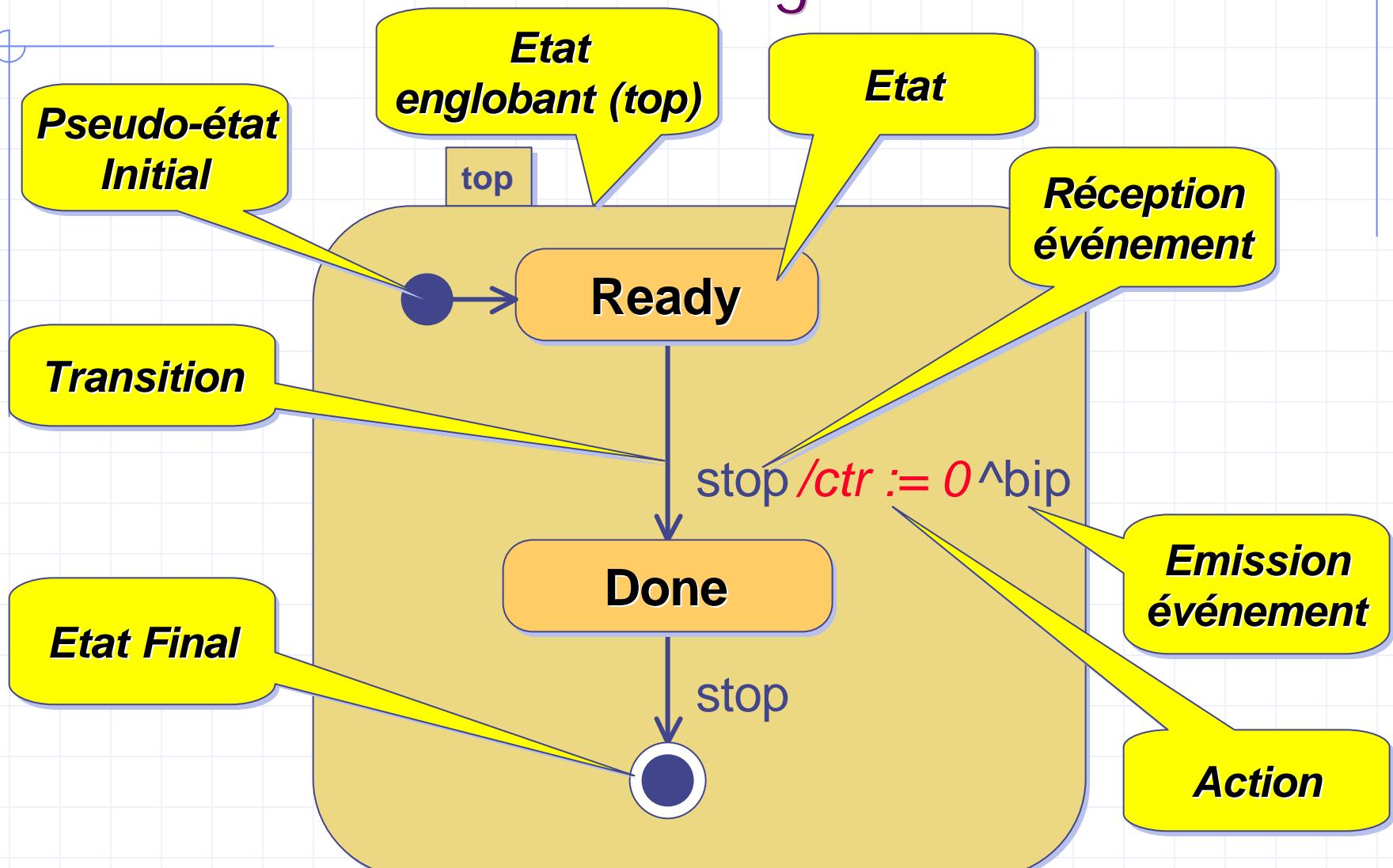
Moore automaton

Machine d'états étendue

- ◆ Addition de variables ("extended state")



Basic UML Statechart Diagram



Comportement “Event-Driven”

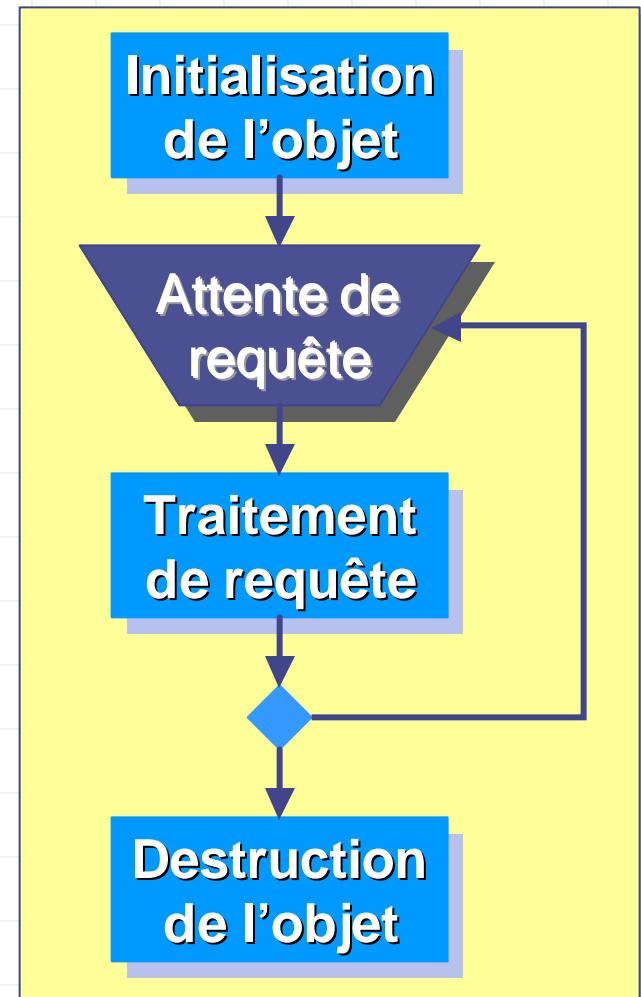
- ◆ Evénement = un type d'occurrence observable:
 - Interactions:
 - ◆ Appel d'opération synchrone (call event)
 - ◆ Réception de signal asynchrone (signal event)
 - Evénements temporels (time event)
 - ◆ Expiration d'un délai (after(x))
 - ◆ Survenance d'une date (when(x))
 - Changement de valeur d'une entité (change event)
- ◆ Une instance d'événement survient à un instant donné et n'a en lui-même aucune durée

A quelle entité le comportement est-il attaché?

- ◆ En principe, tout ce qui manifeste un comportement “event-driven”
- ◆ En pratique, une machine d'état est attachée à une classe, afin de contraindre:
 - le comportement interne de ses instances
 - les interactions entre objets
- ◆ L'intérêt principal de la machine d'état en UML apparaît dans le cas des objets actifs

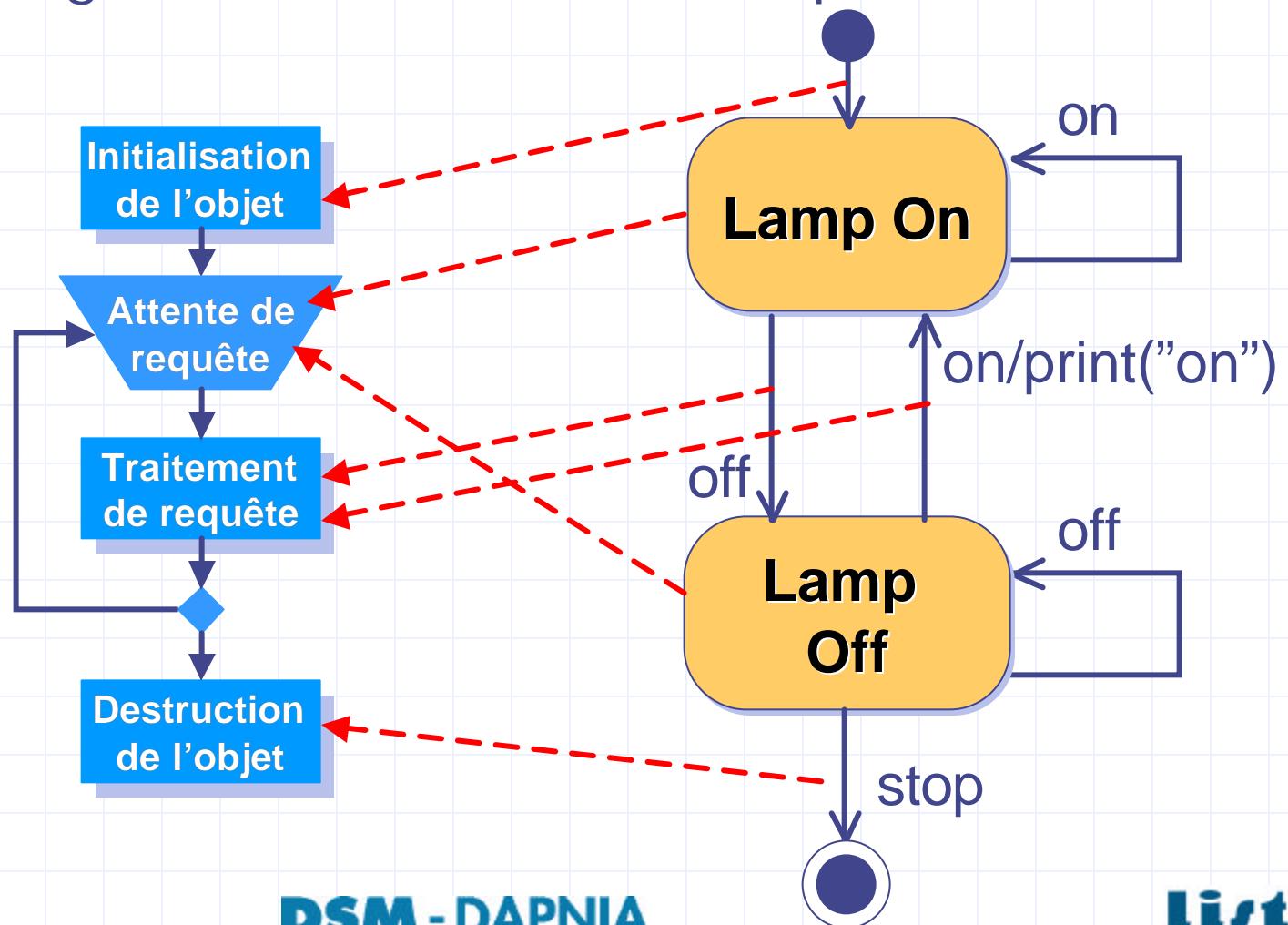
Modèle général du comportement d'un objet actif

- ◆ Modèle serveur simple



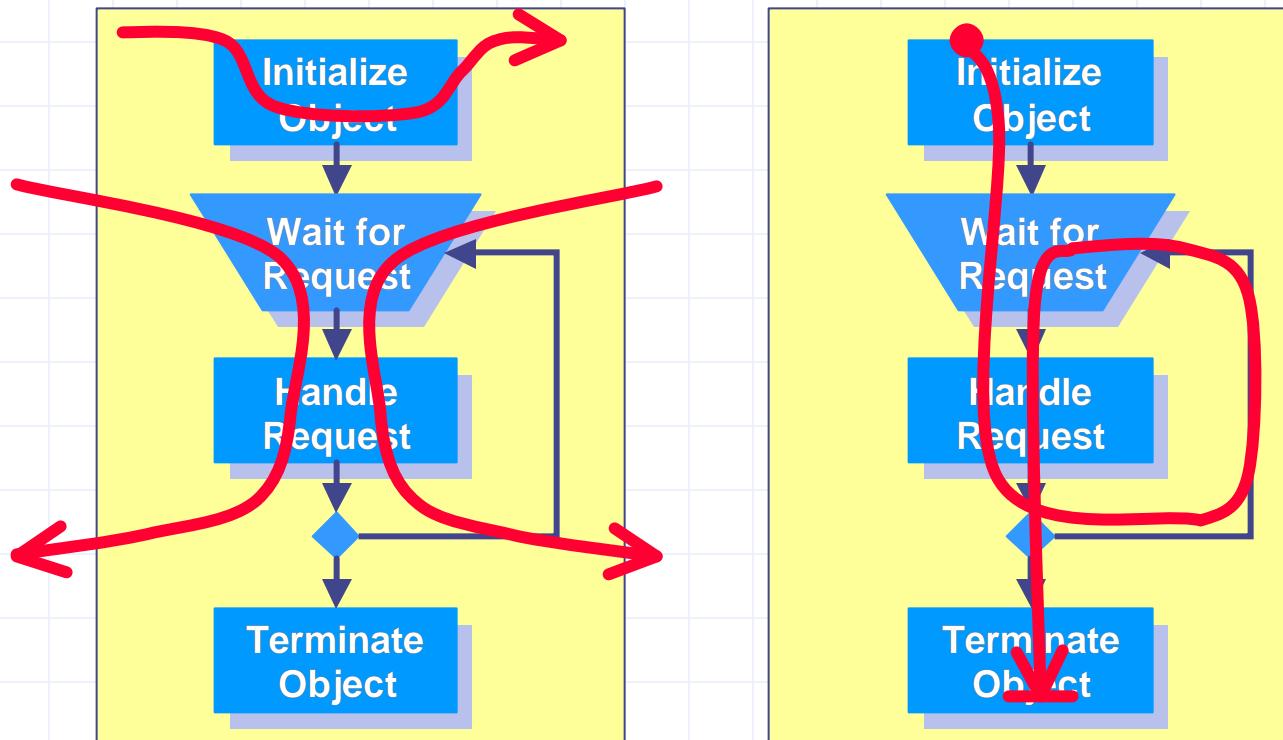
Modèle général du comportement d'un objet actif

- ◆ Mapping machine d'états \Leftrightarrow simple serveur

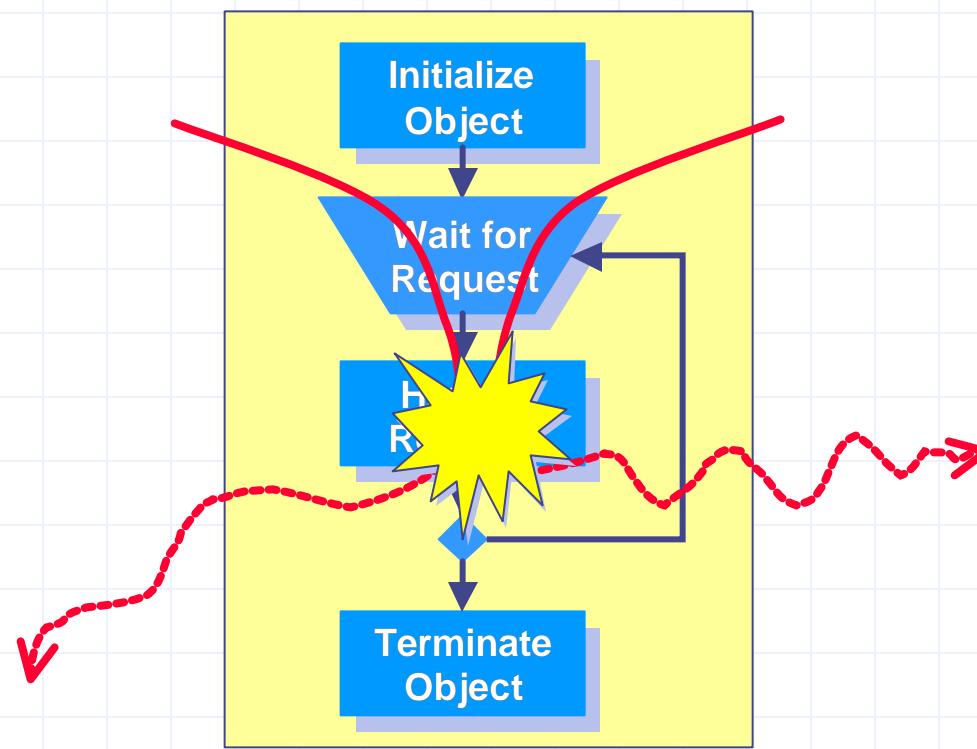


Objets et fils d'exécution

- ◆ Objet passif : activation dépend d'un fil externe
- ◆ Objet actif : possède son propre fil d'exécution



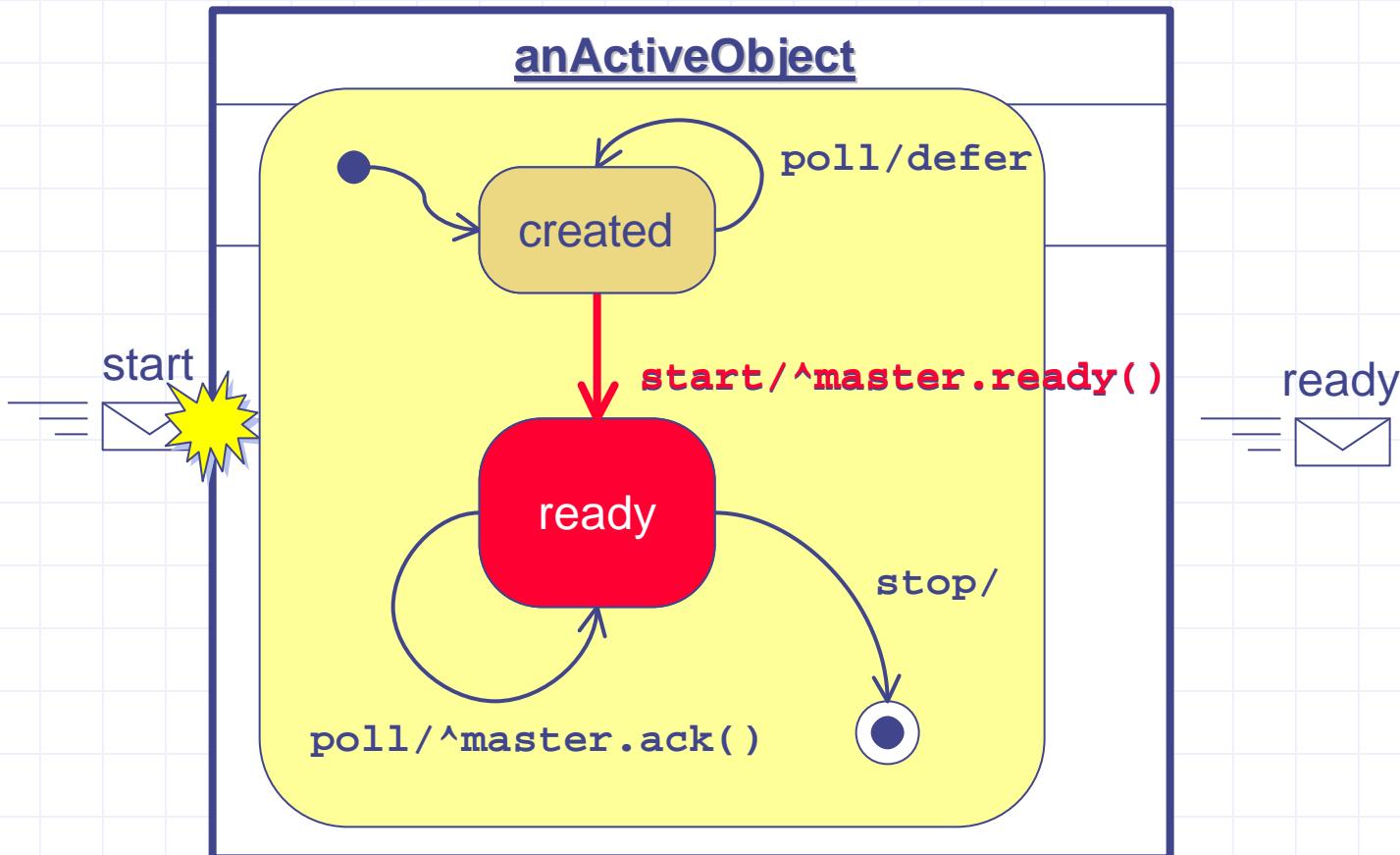
Dynamique d'un objet passif



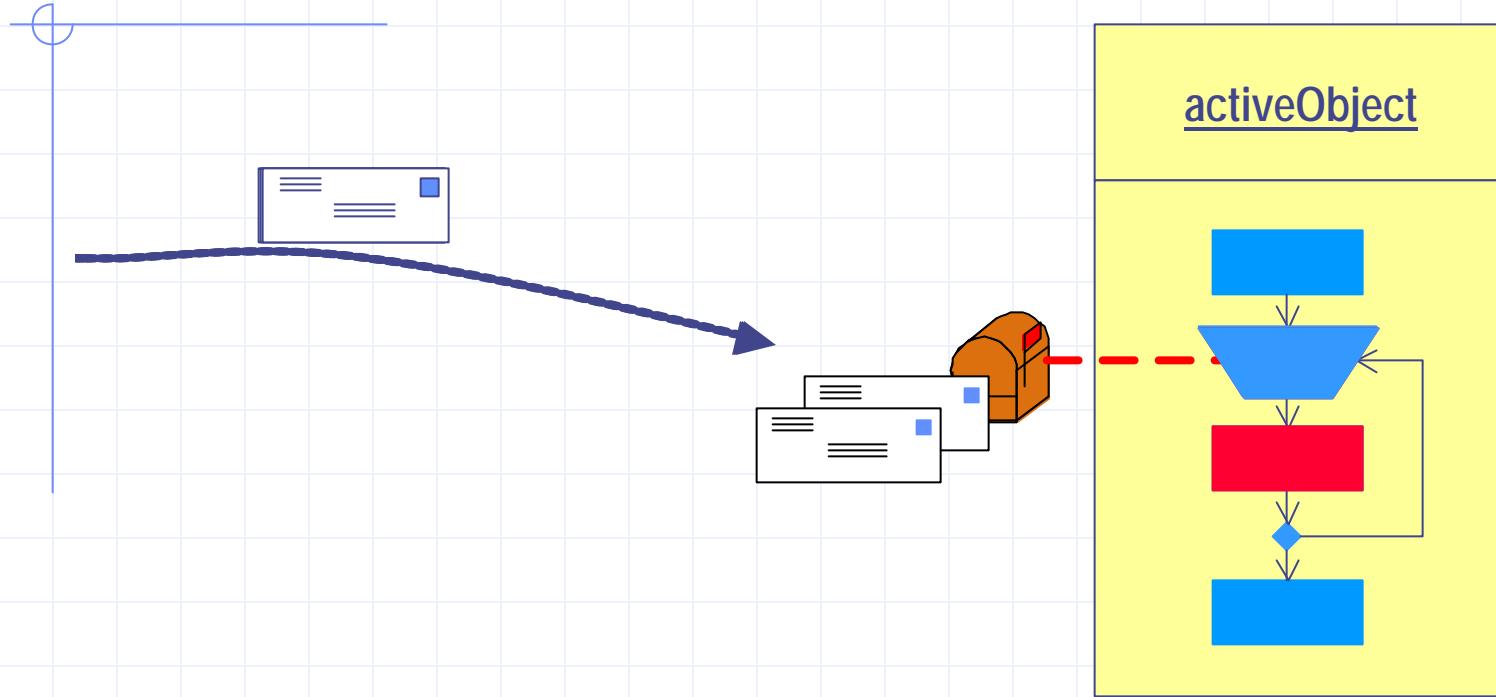
- ◆ L'encapsulation ne protège pas des conflits de concurrence !
⇒ Synchronisation explicite nécessaire

Objets actifs et machines d'états

- ◆ L'objet encapsule son propre fil d'exécution
(n'exporte que des opérations de requêtes)

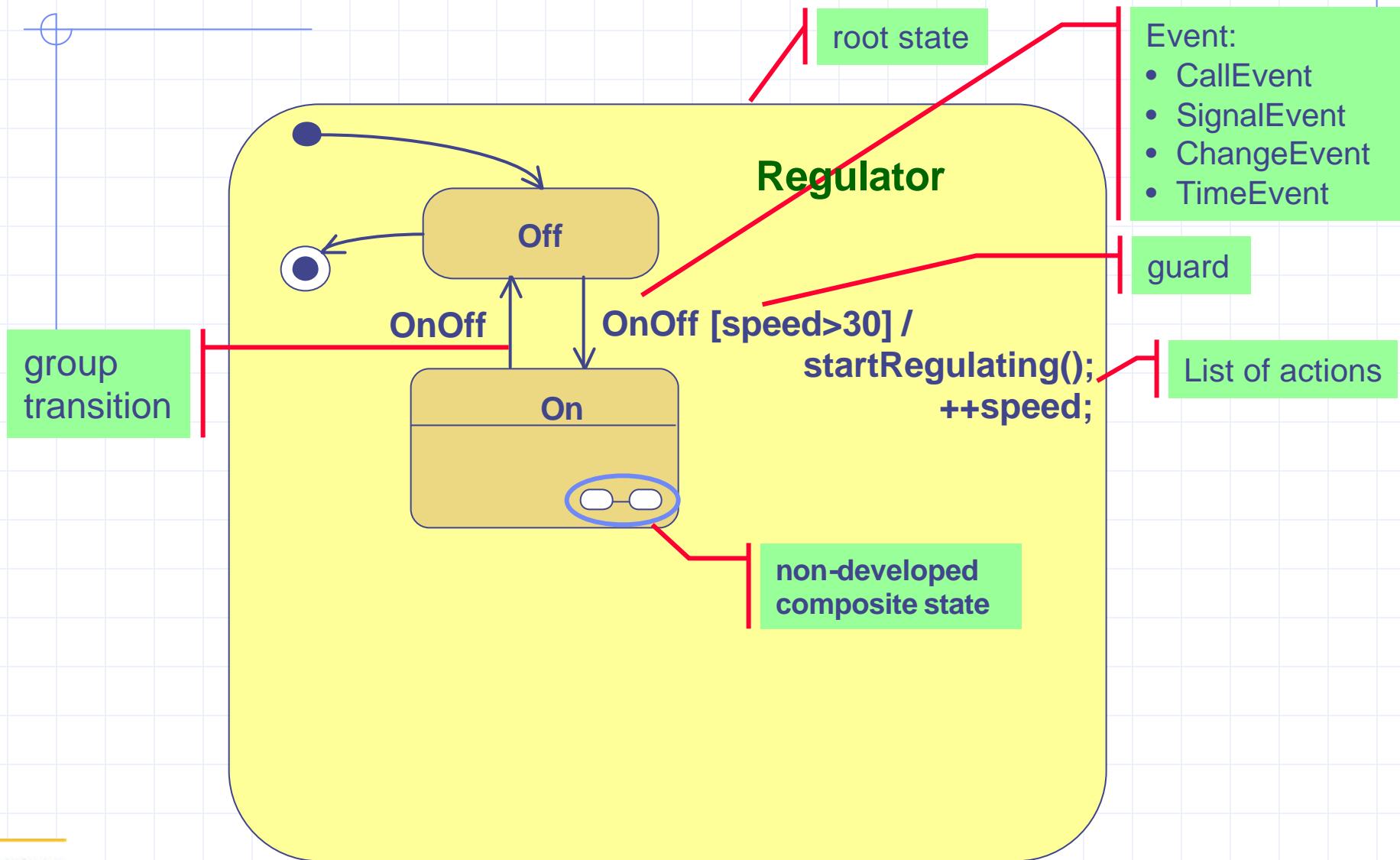


Dynamique d'un objet actif

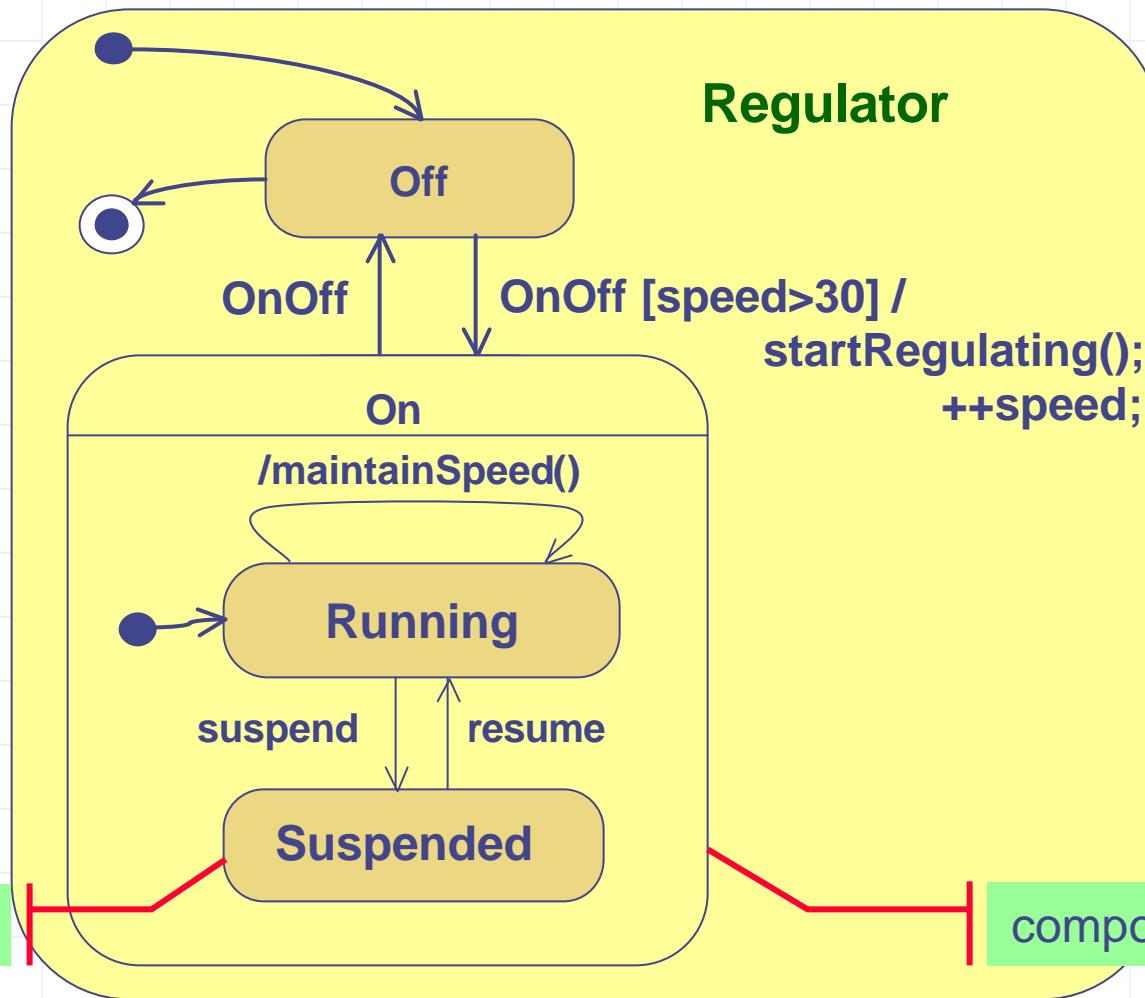


- ◆ Modèle “exécution jusqu’à complétion”:
 - traitement sérialisé des événements
 - élimination des accès concurrents internes
 - minimisation du “context-switching”

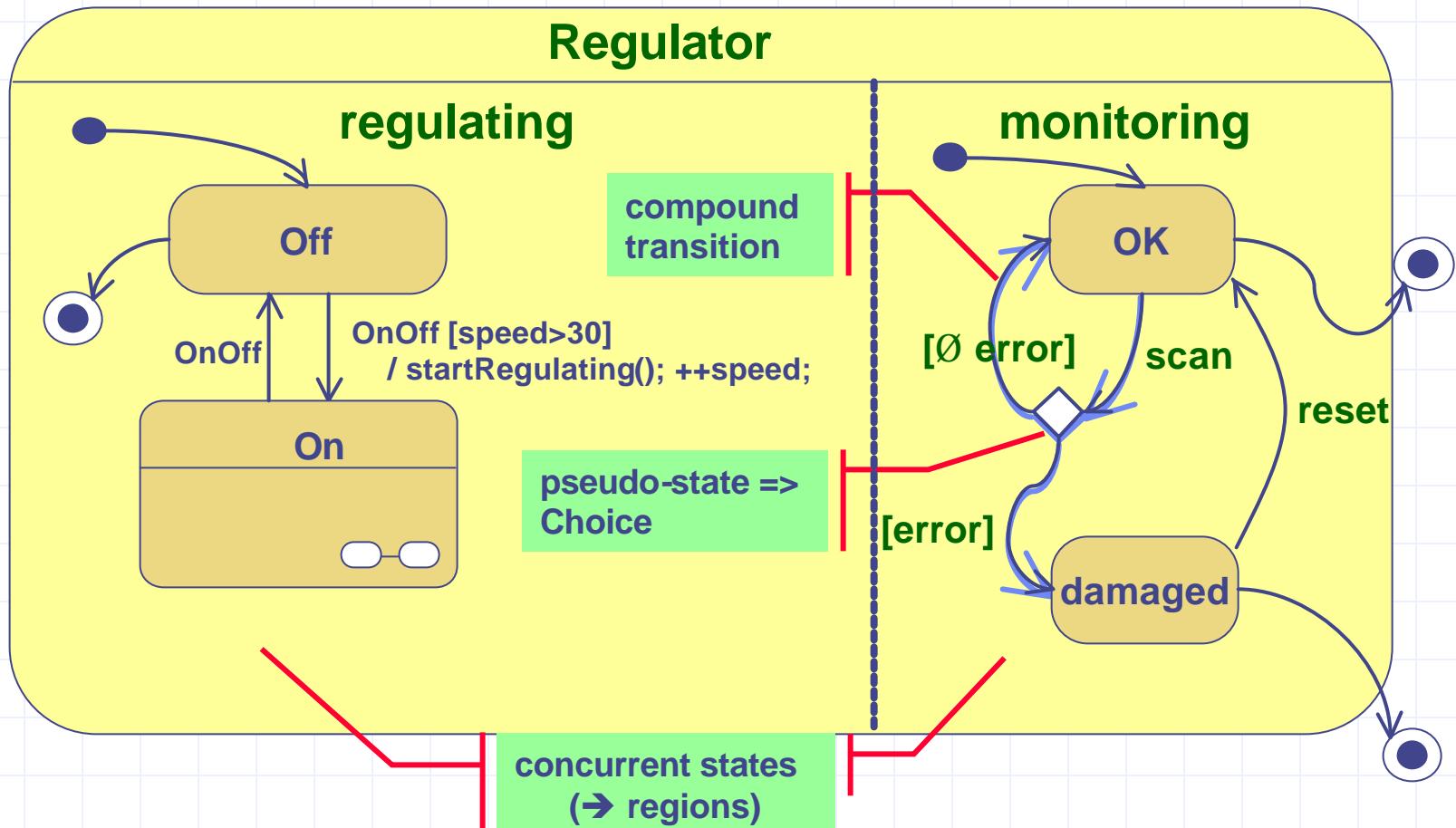
Statechart diagram



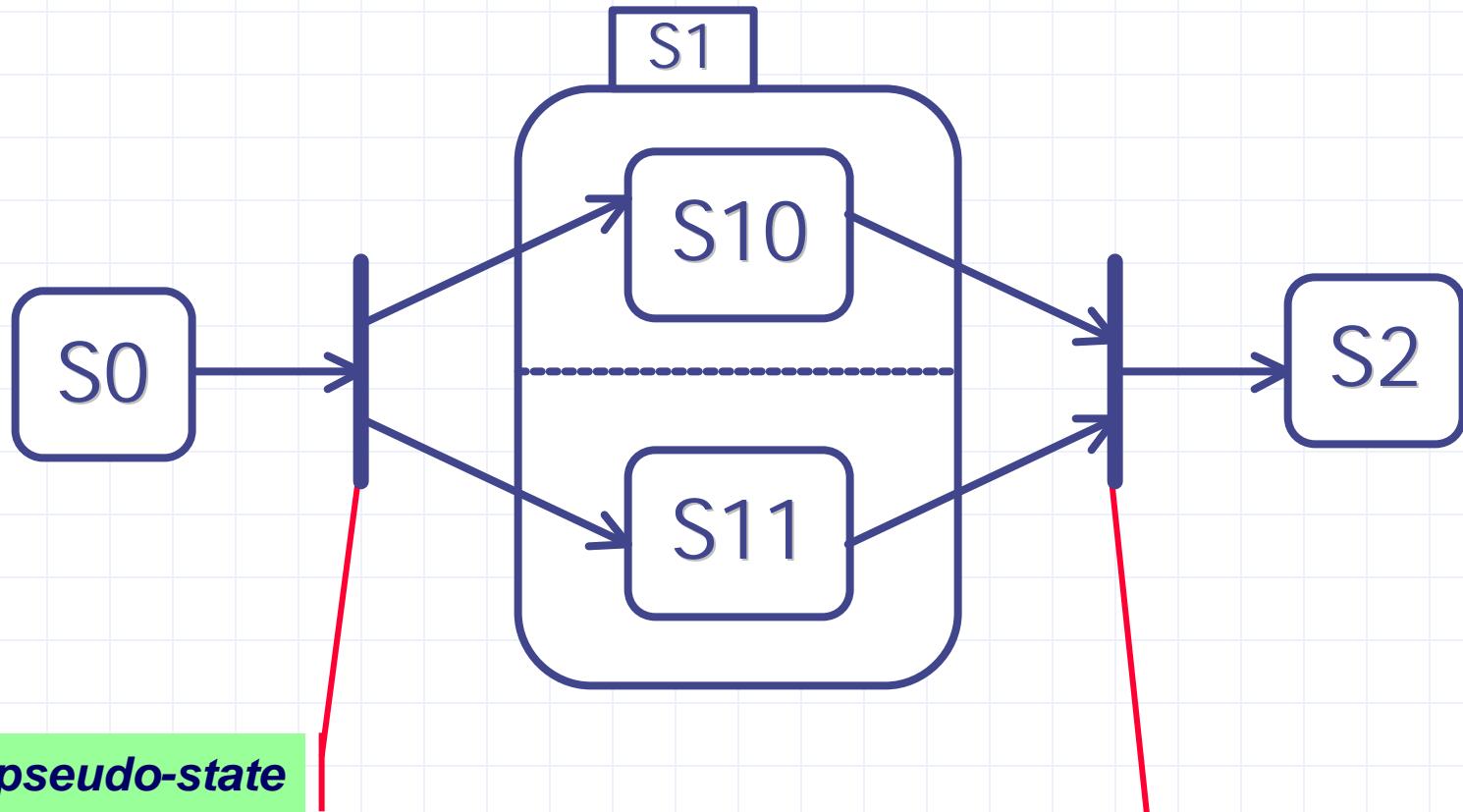
Etat composite



Etats concurrents



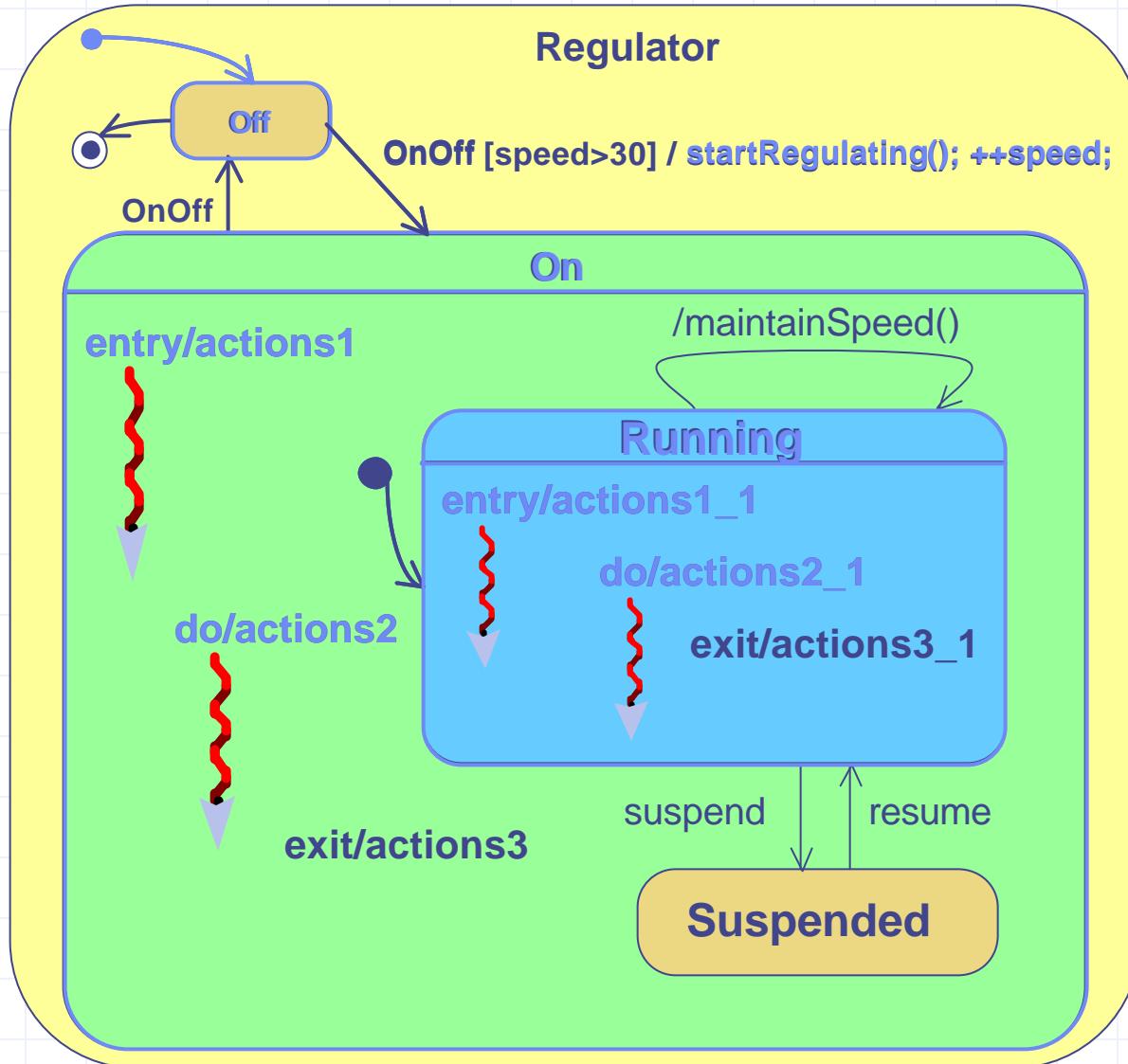
Transitions d'états concurrents



Fork pseudo-state

Join pseudo-state

Actions



Ordre des actions :

2- transition...

3- entry

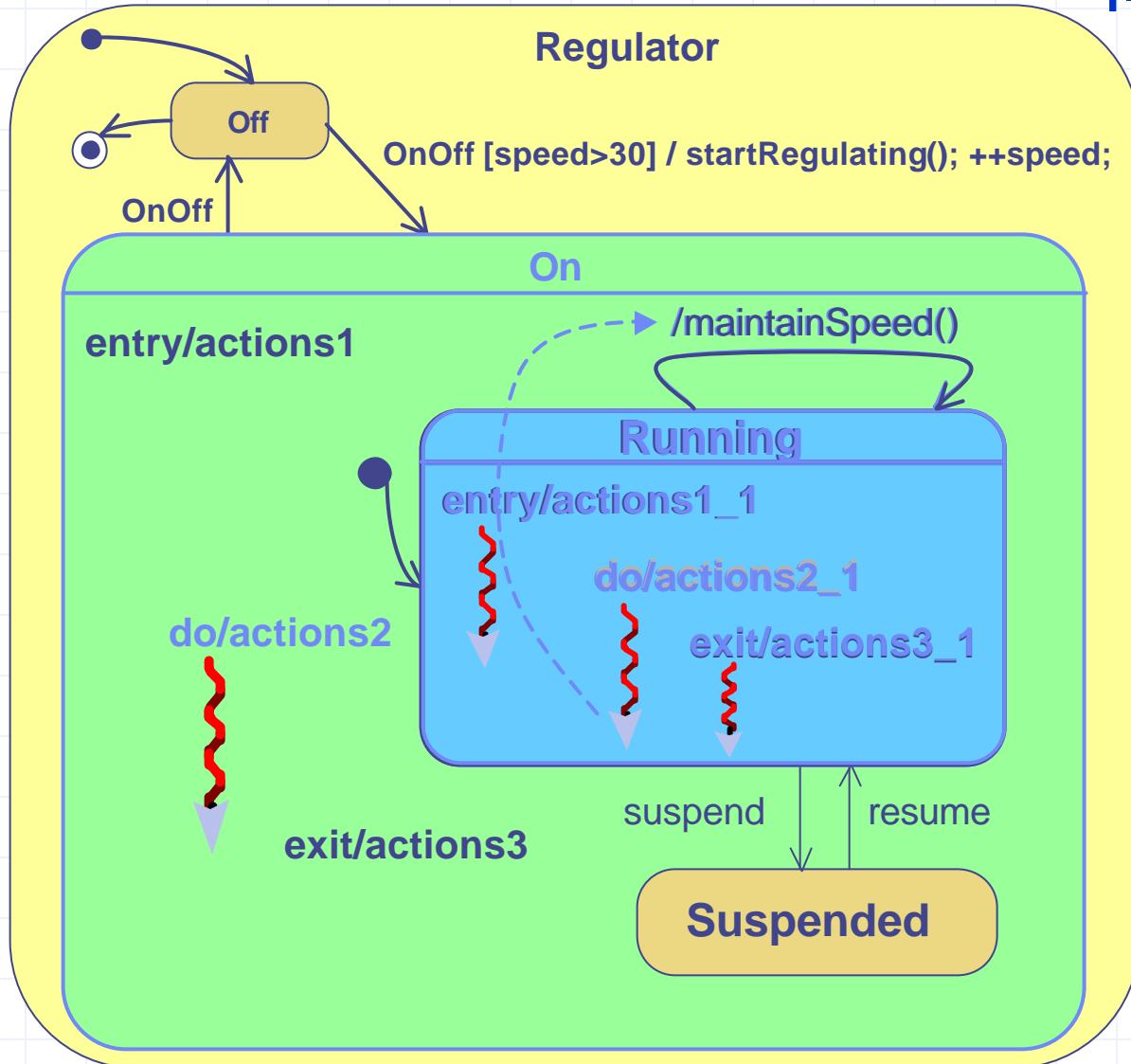
(récursion depuis « top »)

4- do...

☞ **Etat indéfini entre le 1er exit et le dernier entry**

➤ **Récursion des actions « do » dans les états imbriqués**
P possibilità de // ...

Actions



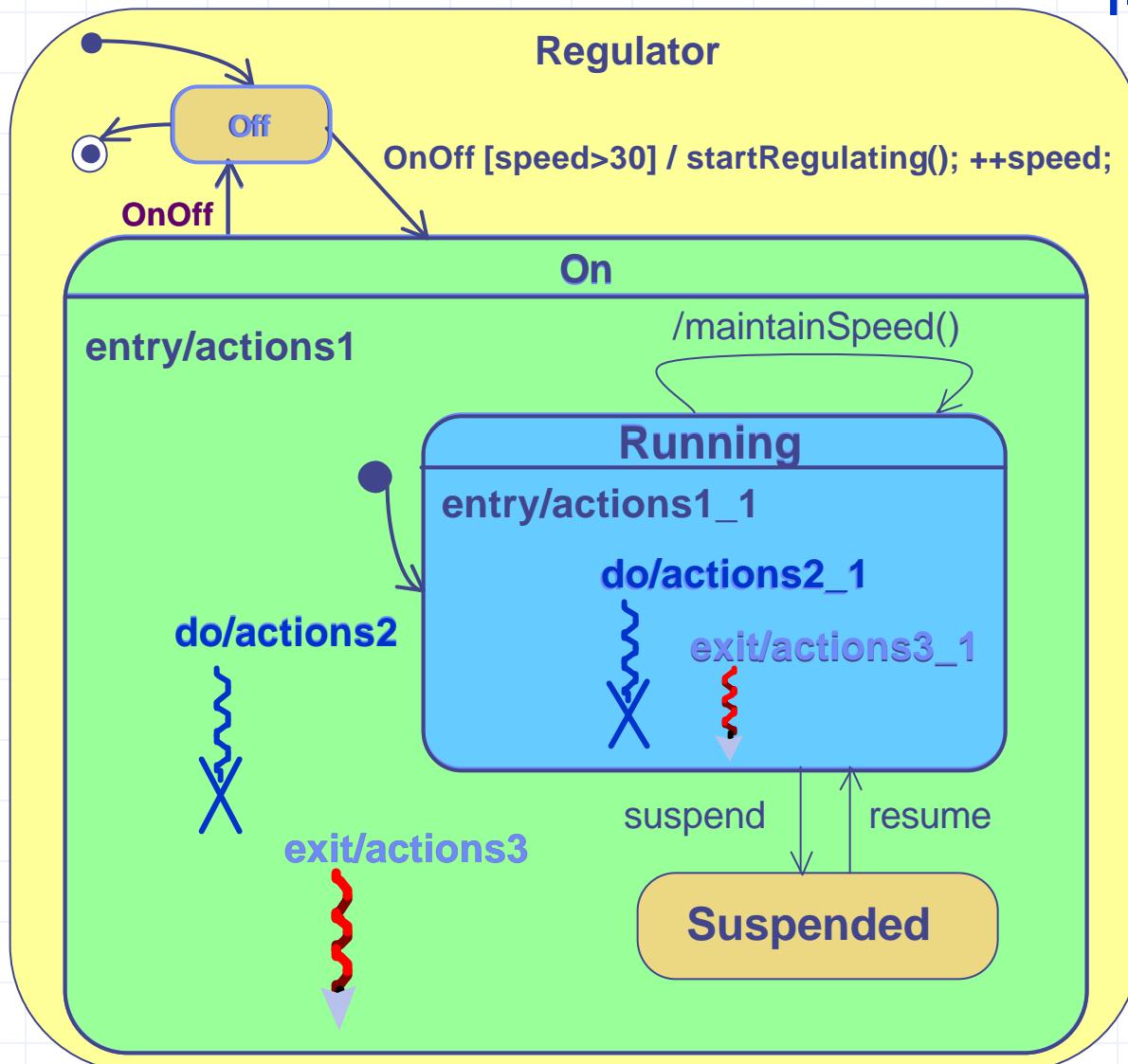
Ordre des actions :

- 1- **exit** (*récursion depuis down*)
- 2- **transition...**
- 3- **entry**
(*récursion depuis « top »*)
- 4- **do...**

☞ **Etat indéfini entre le 1er exit et le dernier entry**

- **Récursion des actions**
« do » dans les états imbriqués
P possibilité de // ...
- **Fin d'action « do » génère:**
completion event
P *completion transitions*

Actions



Ordre des actions :

- 1- **exit** (*récursion depuis down*)
- 2- **transition...**
- 3- **entry**
(*récursion depuis « top »*)
- 4- **do...**

☞ *Etat indéfini entre le 1er exit et le dernier entry*

- *Récursion des actions « do » dans les états imbriqués*
P *possibilité de // ...*
- *Fin d'action « do » génère: completion event*
P *completion transitions*
- *Tout événement interrompt les actions « do » en cours*

Variabilité des Machines d'états UML

- ◆ Choix de la (des) politique de livraison des événement
 - Le plus répandu = FIFO
- ◆ Une tâche par défaut est définie (celle qui lit implante la politique de livraison des événements)
 - Mais il est possible d'en avoir plusieurs, par exemple: une tâche par région concurrente
- ◆ Héritage de machine d'état: non défini
- ◆ ...

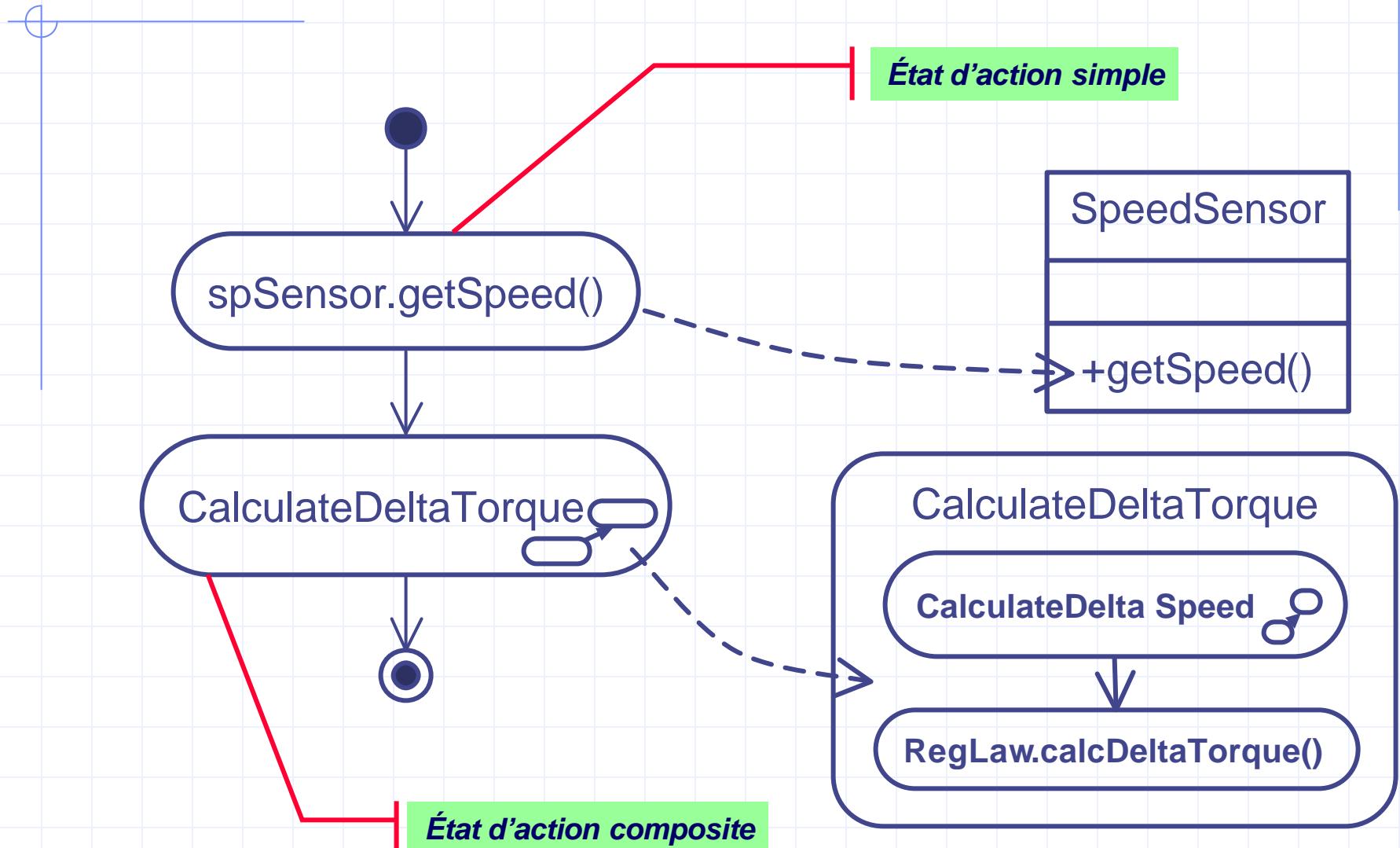
Points de vues de modélisation

- ◆ Spécifier le système
- ◆ Modéliser des objets communicants
- ◆ Modéliser la structure de l'application
- ◆ Modéliser le comportement de l'objet
- ◆ Modéliser les traitements
 - Diagramme d'activité
- ◆ Modéliser l'instanciation de l'application

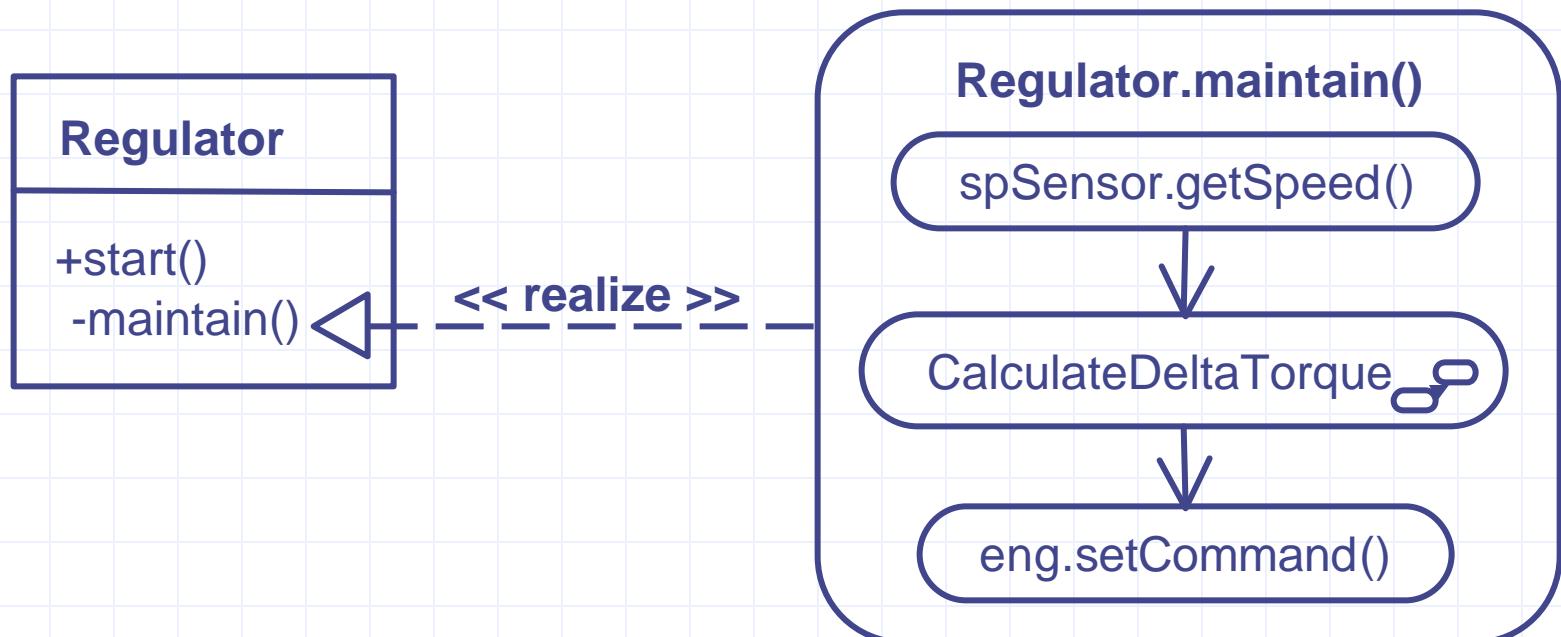
Activity diagrams

- ◆ Pour spécifier les flux de contrôle, de données ou d'objets
- ◆ Graphe constitué d'étapes (états d'action)
(ressemblances avec statechart)
 - Une étape se termine → étape suivante
 - Supporte la concurrence
- ◆ Insiste sur l'ordre des étapes
- ◆ Sous-activités = décomposition fonctionnelle

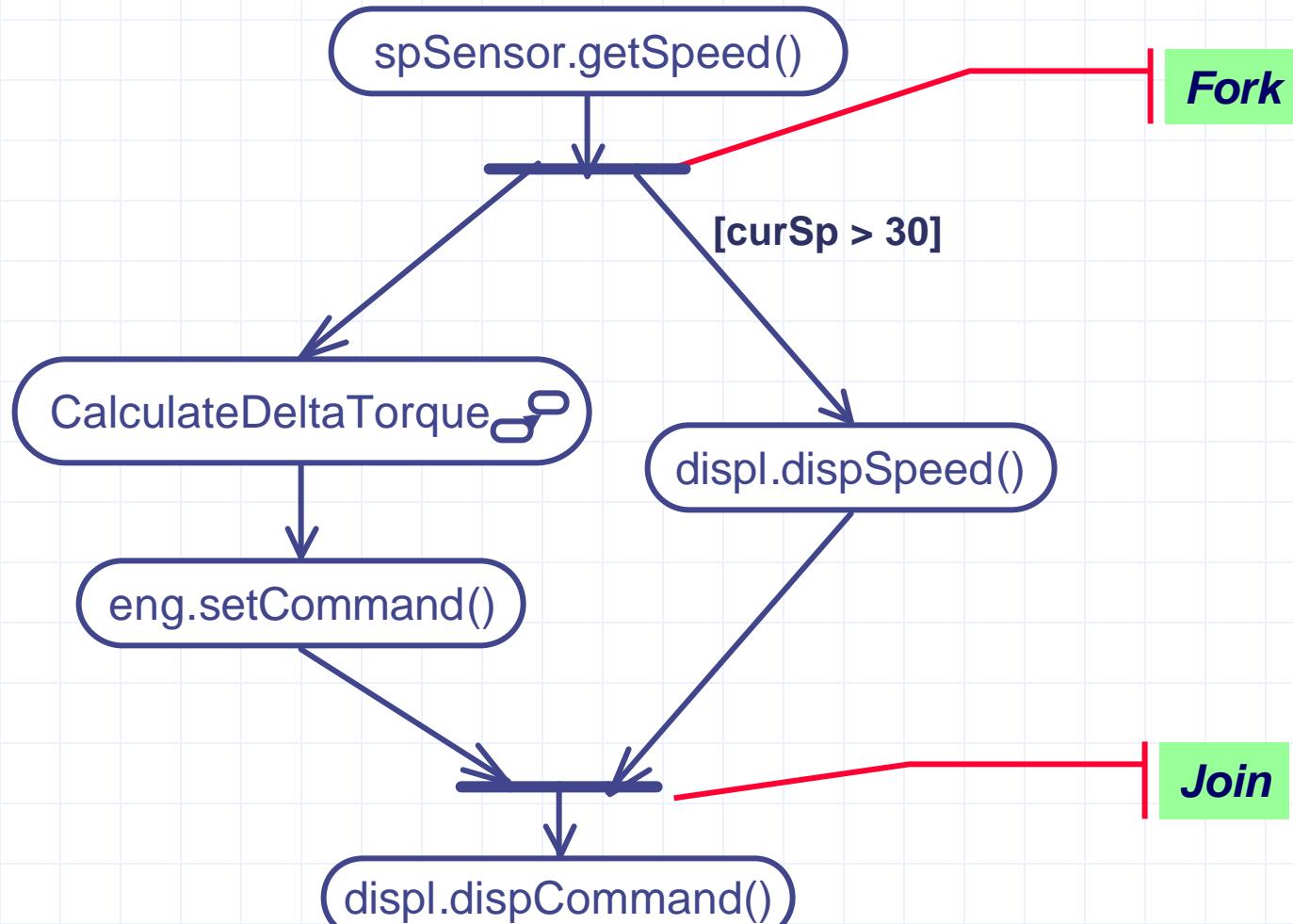
Eléments de base



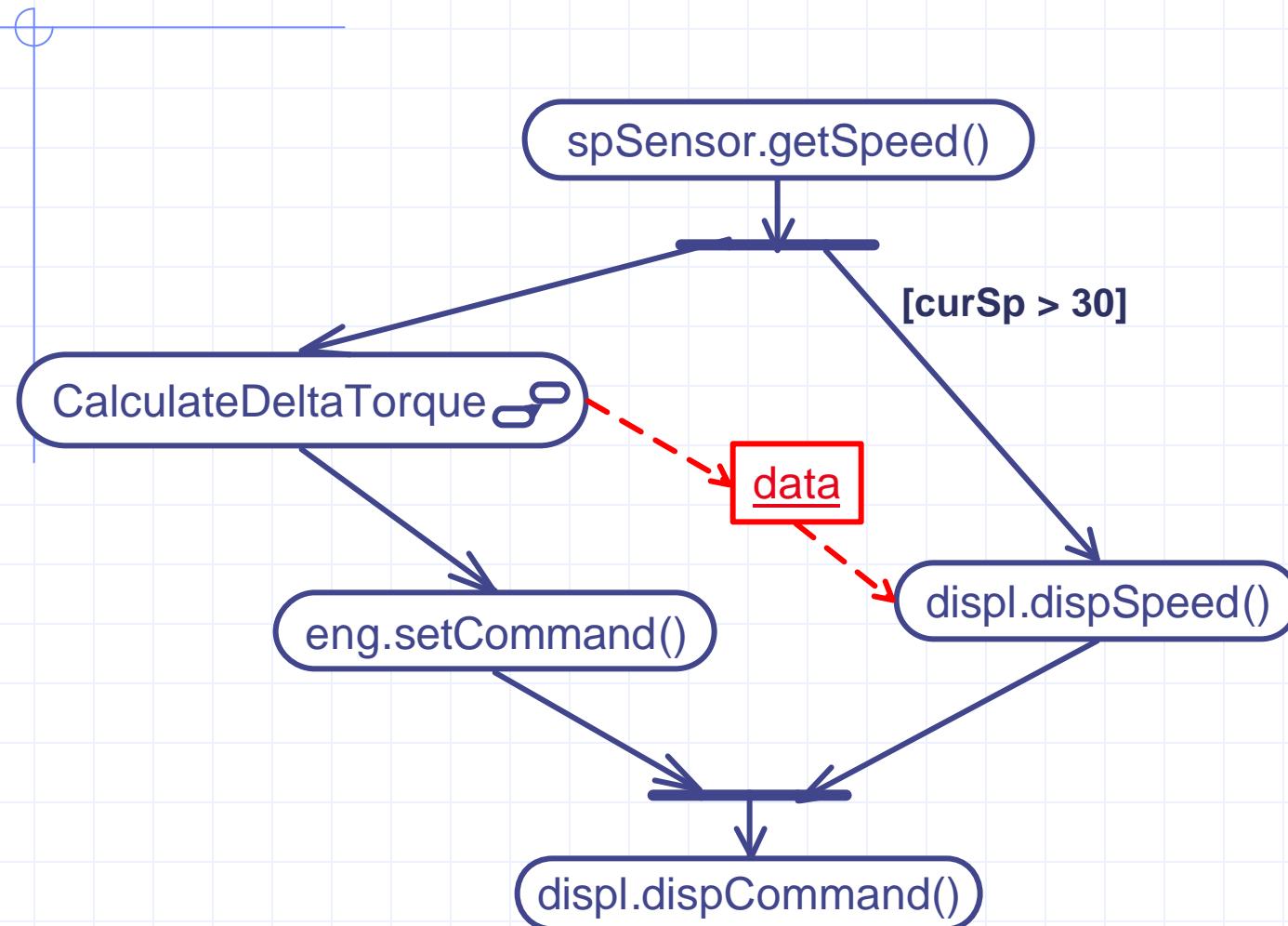
Graphe d'activité et spécification de méthode



Concurrency



Flux d'objet: représente la disponibilité d'un objet



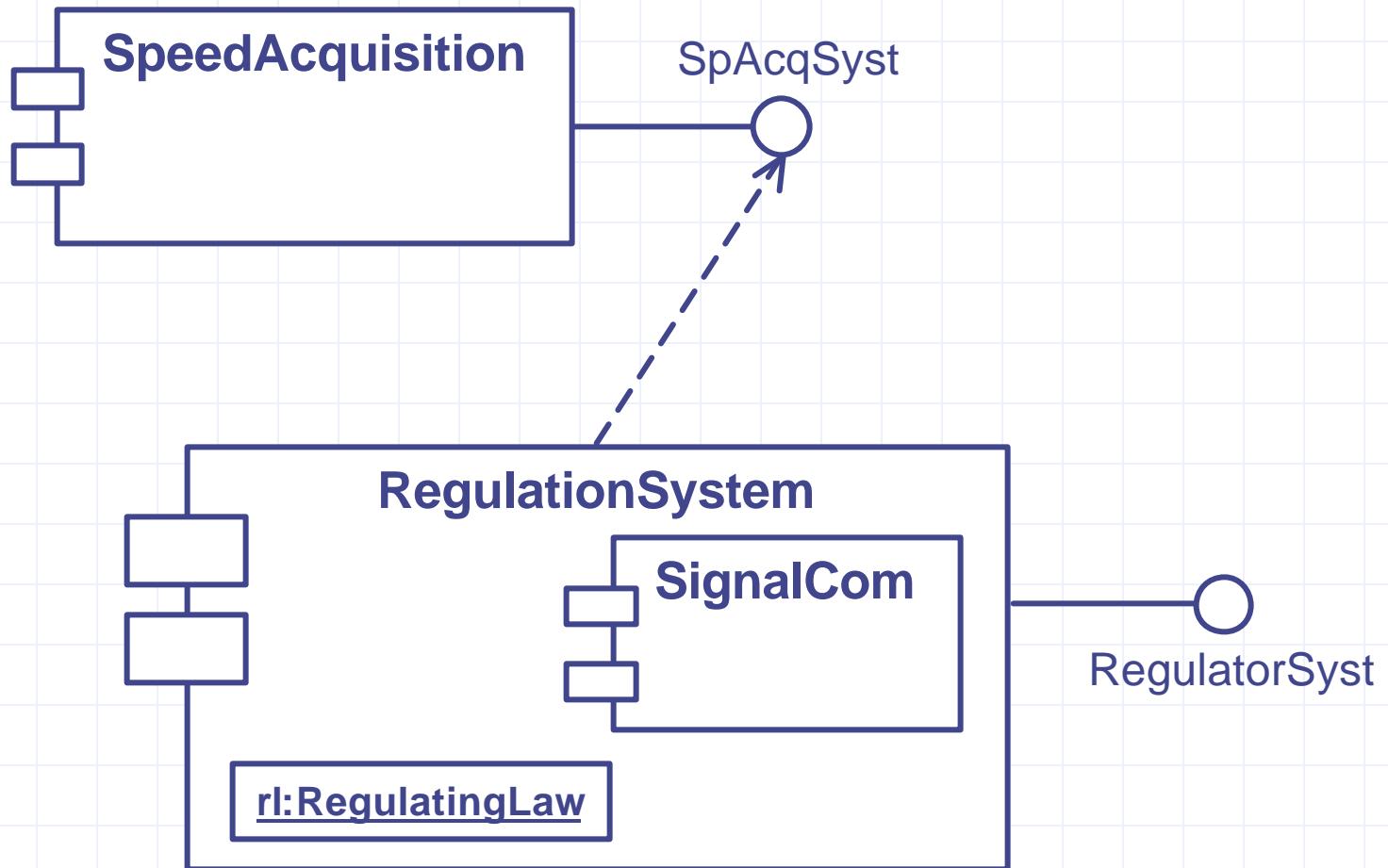
Points de vues de modélisation

- ◆ Spécifier le système
- ◆ Modéliser des objets communicants
- ◆ Modéliser la structure de l'application
- ◆ Modéliser le comportement de l'objet
- ◆ Modéliser les traitements
- ◆ Modéliser l'instanciation de l'application

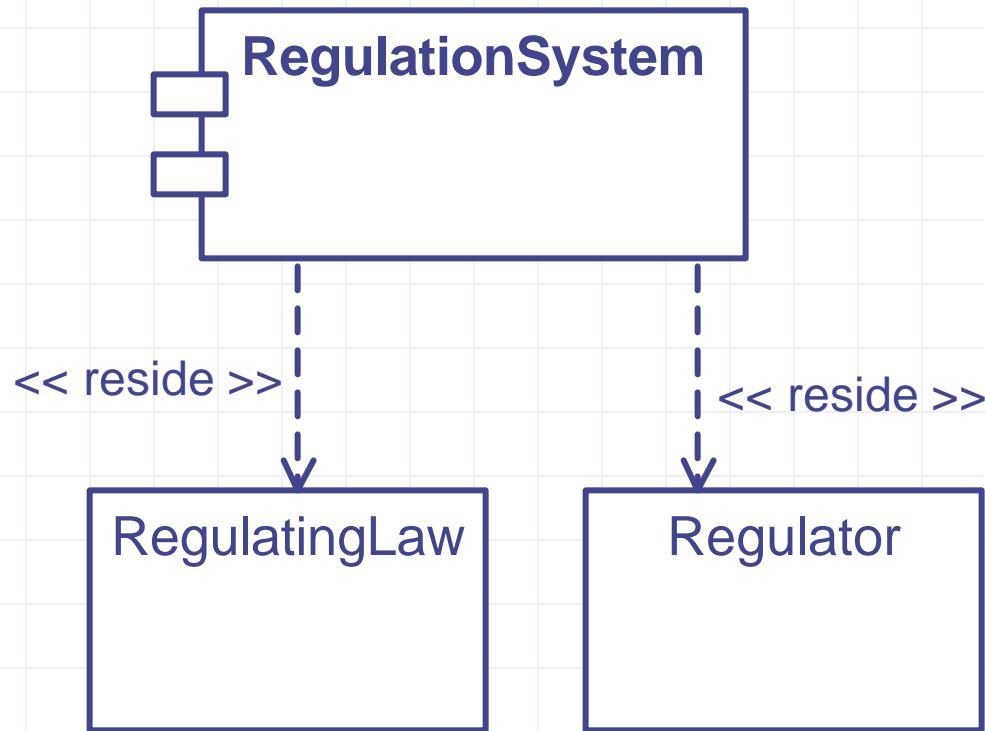
Diagrammes d'implantation

- ◆ Diagrammes de composants: organisation et dépendances des composants de l'application
 - code source, binaires, exécutables, procédures, documents, etc.
- ◆ Diagrammes de déploiement: configuration de distribution des composants

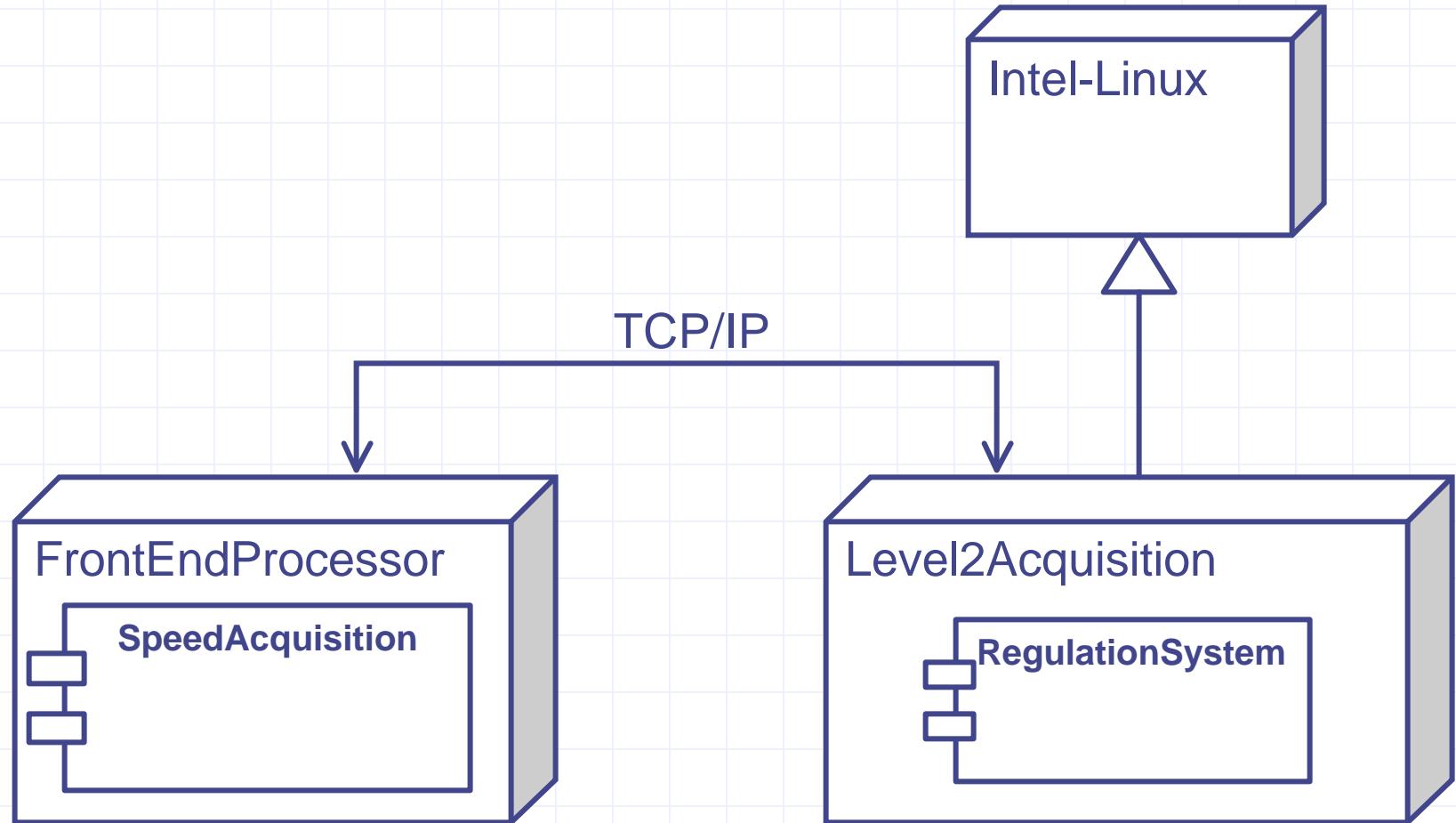
Component Diagram



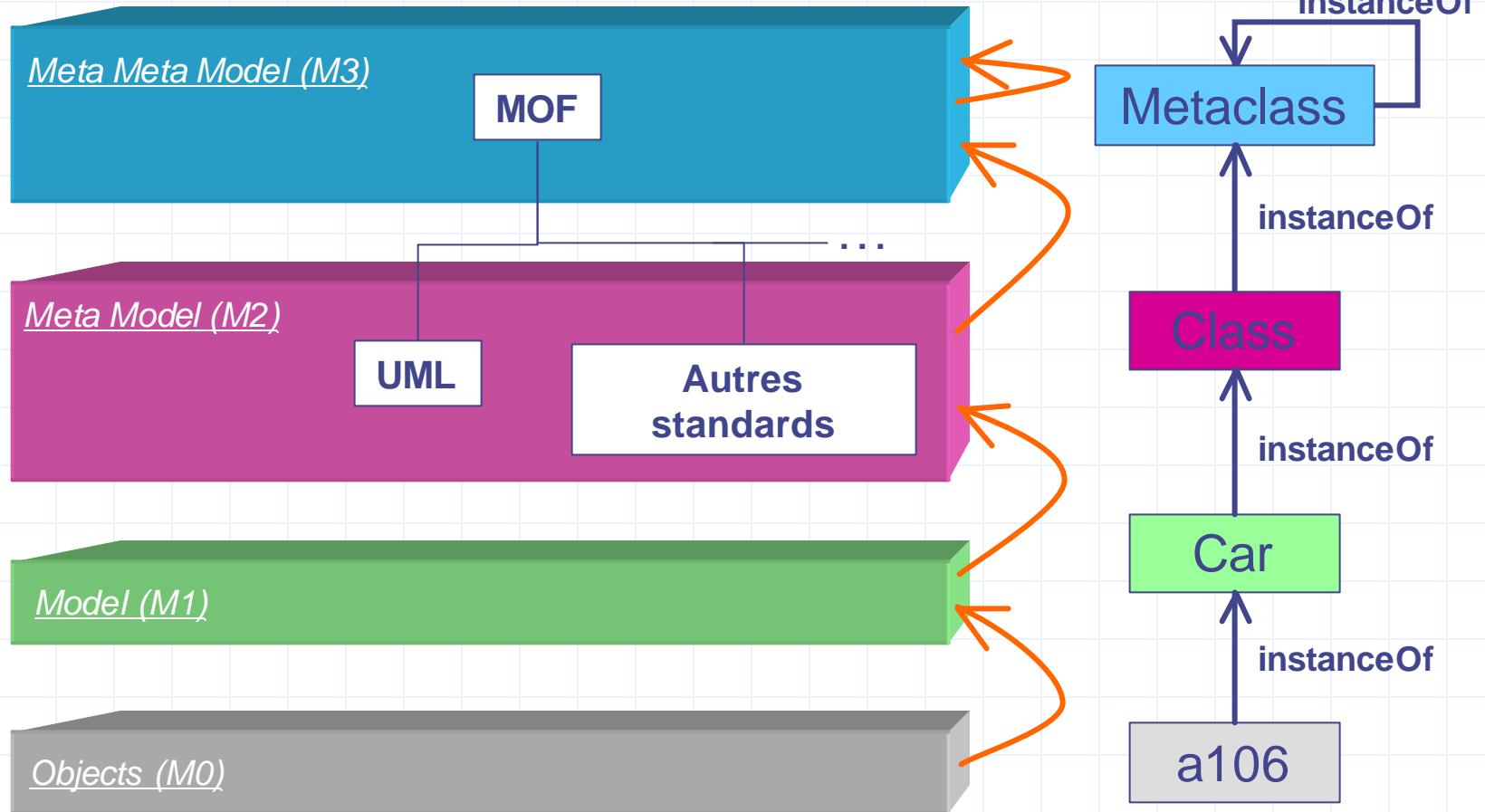
Composants (classes résidentes)



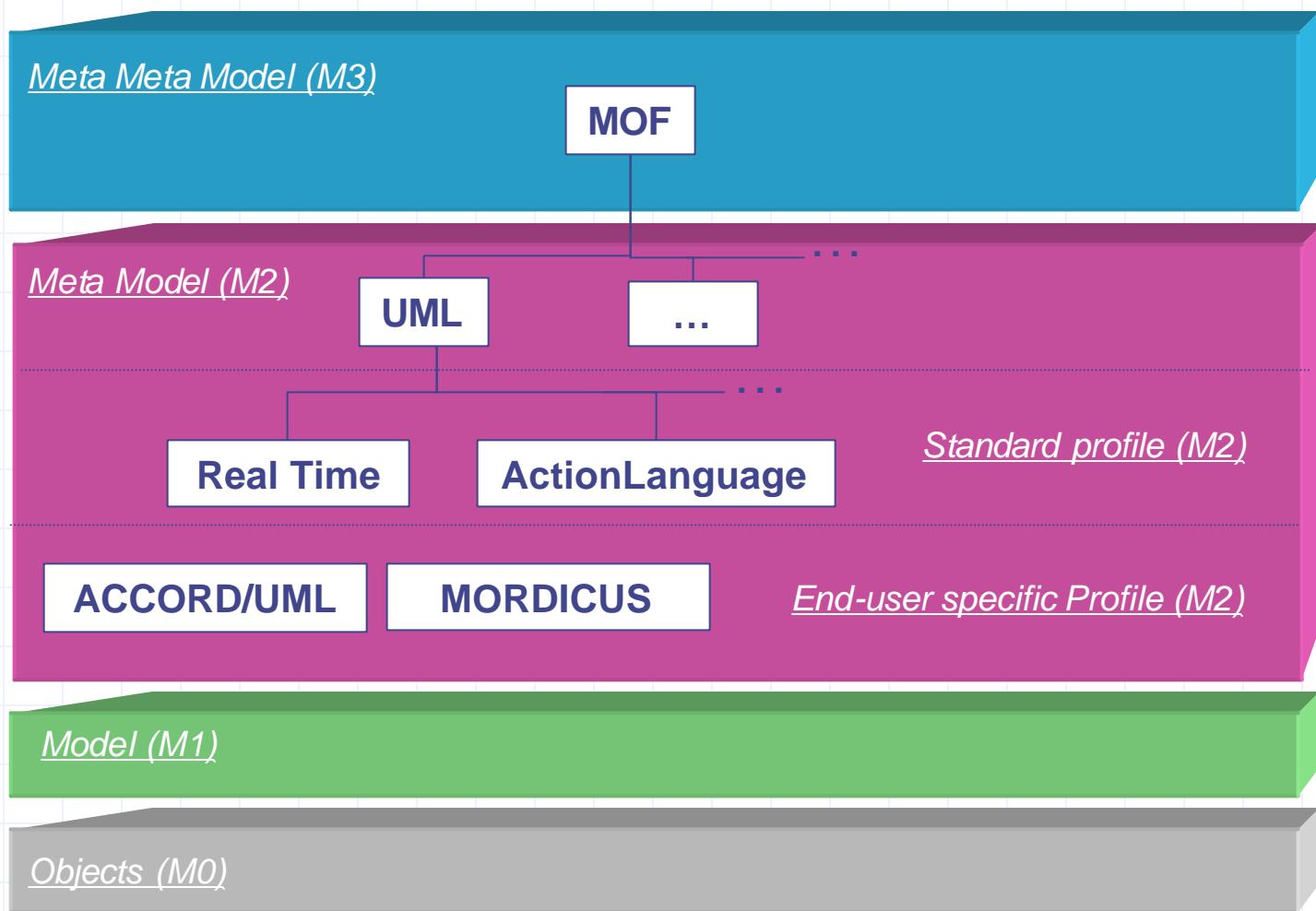
Deployment Diagram



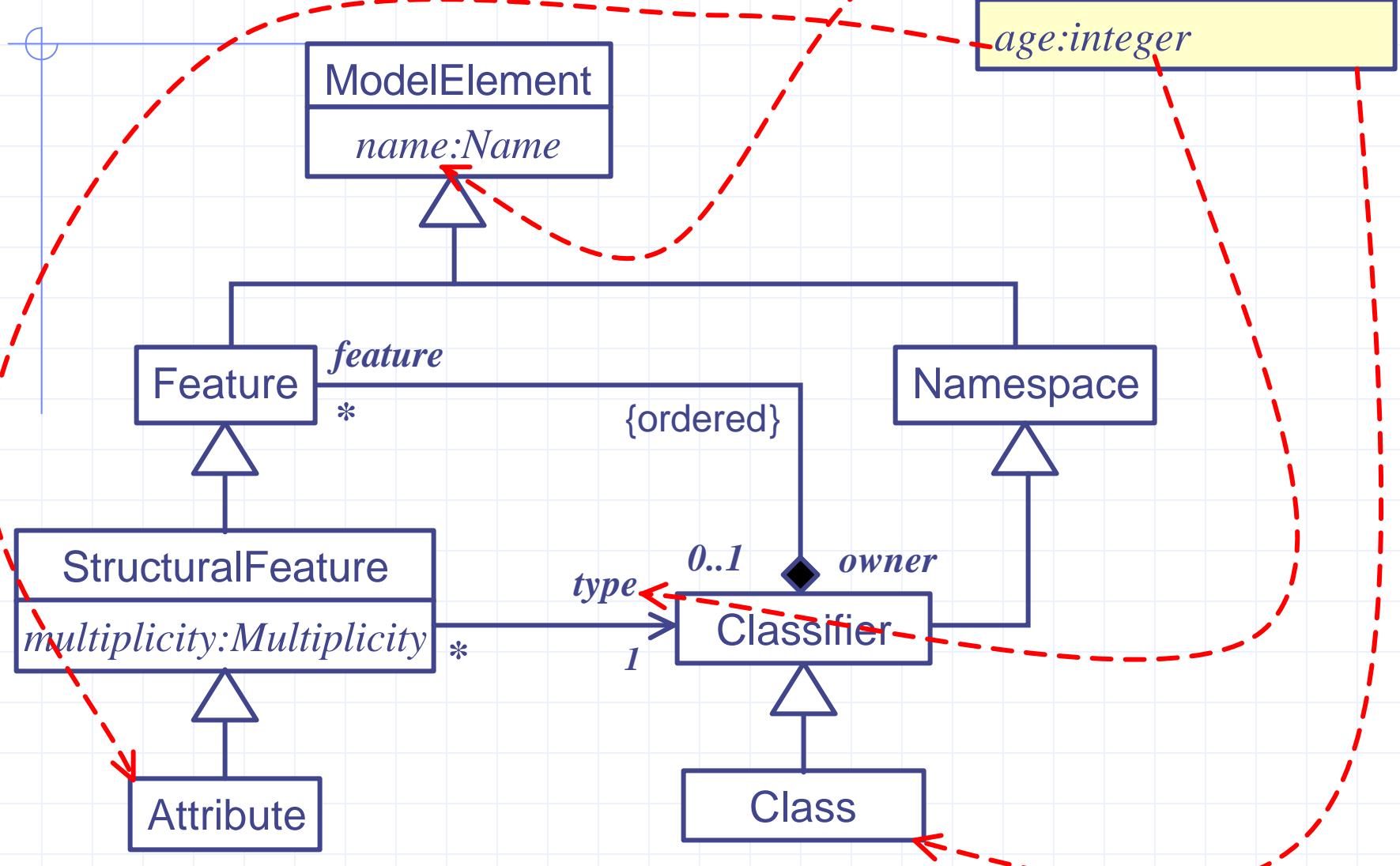
Modèle en 4 couches



Extensions Métier d'UML



Exemples de Métaclasses



Mécanismes d'extension UML

◆ Stereotype

- Méta-classe spécialisée (ex: « real-time »)
- Ajout de nouveaux stéréotypes → extension

◆ Tagged value

- méta-attribut (ex: {abstract})
- Ajout d'un nouveau méta-attribut → extension

◆ Constraint

- Règle de formation d'expression (ex: {ordered})
- Nouvelles contraintes sur le méta-modèle → extension

Notion de profil UML

(Cf. http://www.objecteering.com/us/smot_uml_white.htm)

- ◆ Standardisation d'un méta-modèle étendu d'UML
- ◆ Adapté à un domaine métier ou middleware
- ◆ Un profil UML peut contenir
 - Les éléments sélectionnés dans la méta-modèle de référence
 - Des extensions utilisant les différents mécanismes d'extension
 - Descriptions sémantiques des extensions
 - Notations supplémentaires
 - Règles de validation, présentation, transformation