

# « Model Driven Engineering » pour la robotique mobile

Tewfik Ziadi

[Tewfik.ziadi@lip6.fr](mailto:Tewfik.ziadi@lip6.fr)

# Motivations

- Comment programmer un robot mobile pour réaliser une mission?
  - Exemple de mission :
    - « ...*avancer, retourner si un obstacle est rencontré..* »
  - Exemple le robot **wifibot**

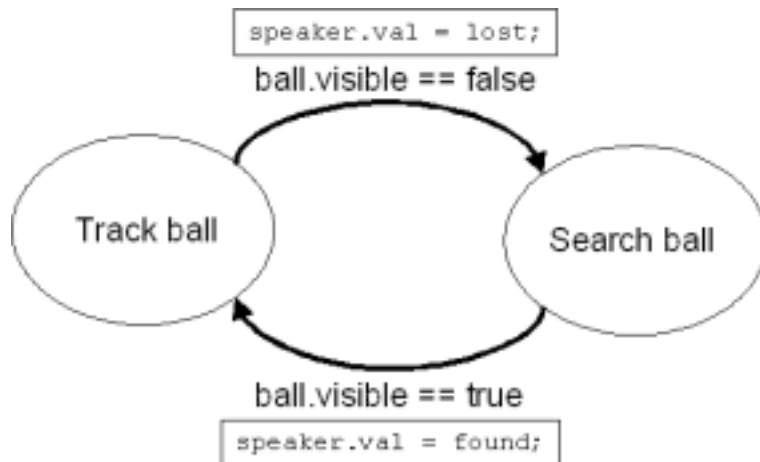
# Langages de Programmation

- Des langages classiques (généralistes)
  - Assembleur, C, C++, Java
- Des langages de robotiques plus spécifiques
  - RobotC.
  - Urbiscript (la plateforme URBI)
  - Orocos
  - ..

# URBI (Universal Real-time Behavior Interface)

- URBI [Baillie 05].
  - Une plateforme de développement expérimentale pour les robots.
  - Un langage de script et un interpréteur (**urbiscript**).
  - Structures de contrôles et gestion des événements.

# urbiscript



```
// Etat de suivi
function suis() {
  whenever (ball.visible) {
    headPan = headPan + ball.a * camera.xfov * ball.x &
    headTilt = headTilt+ ball.a * camera.yfov * ball.y;
  }
};

// Etat de recherche
function cherche() {
  period = 10s;
  {
    headPan'n = 0.5 smooth:1s &
    headTilt'n = 1 smooth:1s
  } |
  {
    headPan'n = 0.5 sin:period ampli:0.5 &
    headTilt'n = 0.5 cos:period ampli:0.5
  }
};
```

# urbiscript

```
// Transitions
at (ball.visible == true) {
  stop recherche;
  speaker = trouvee;
  suivi: suis();
};

at (ball.visible ==false ) {
  stop suivi;
  speaker = perdue;
  recherche: cherche();
};
```

# Démo

- Utilisation d'urbiscript pour contrôler un robot sous le simulateur webots.

# Constats

## **Absence d'abstraction et de capitalisation**

- Le code de la mission est spécifique à une simulation sous webots + e-puck
- Même mission pour un autre type de robot mobile (e.g.; AIBO)
- Même mission pour un robot réel comme wifobot.

## **Le roboticien n'est pas un informaticien**

- Il faut pas qu'il s'occupe des détails liés au langages de programmation.
- Il doit seulement trouver ses concepts métier.



# Utilisation de MDE

- Utilisation de modèles pour la spécification des missions
- Implémentation des générateurs de code
- **Dans ce cours** : deux solutions
  - Une solution utilisant les modèles comportementaux d'UML.
  - Une solution plus complète proposant un DSML pour la robotique mobile.

# UML pour la robotique (cf. article ISORC2009)

- Utiliser les diagrammes de séquence et les machines à états pour spécifier la mission
- Génération de code vers URBI
- Application sur le robot AIBO

# Machines à états vs. diagrammes de séquence

- Les machines à états:
  - est un formalisme adapté pour la modélisation du comportement des robots.
    - Proche de code : ex. on peut les implémenter facilement avec urbiscript..
  - Mais modéliser directement une machine à états n'est pas intuitive.
- Les diagrammes de séquence
  - Ils sont intuitifs et proches à l'utilisateur
  - Mais surtout pour décrire les exigences et pas pour la génération de code.

# Modélisation du comportement

- Utiliser les diagrammes de séquence (DS) pour spécifier les mission d'un robot.
- Générer automatique des machines à états à partir de DS.
- Générer de code à partir des machines à états.

# L' approche

- Phase 1:  
Diagrammes de séquence d'UML → machines à états.
  - Phase 2  
Machines à états → Code
- 
- Application au robot **AIBO**

# AIBO



AIBO : un robot chien (sony)

Un système **réactif**.

Un système **programmable**.

URBI : primitives de base (stand(), walk(), turn(),.etc.)

(ces primitives sont fournies avec URBI installé sur le robot et pas sur le simulateur.

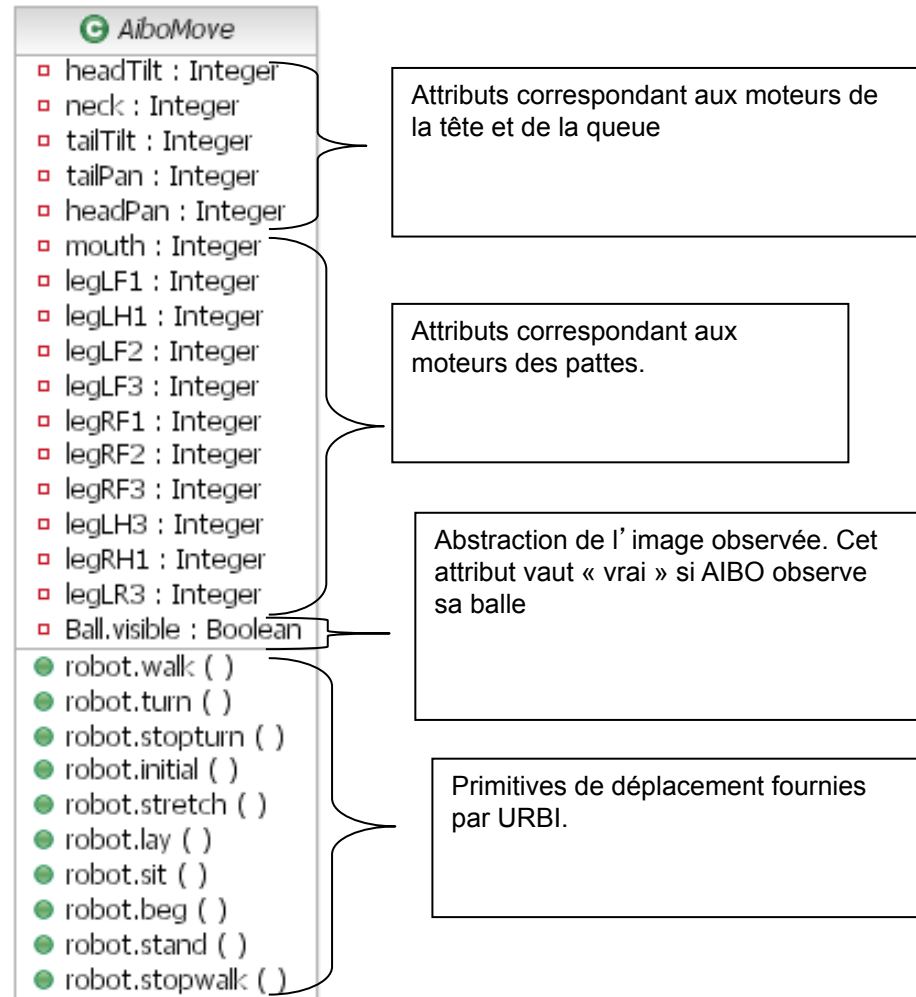
Un système **modélisable** en UML

Un diagramme de classes : la structure d' AIBO

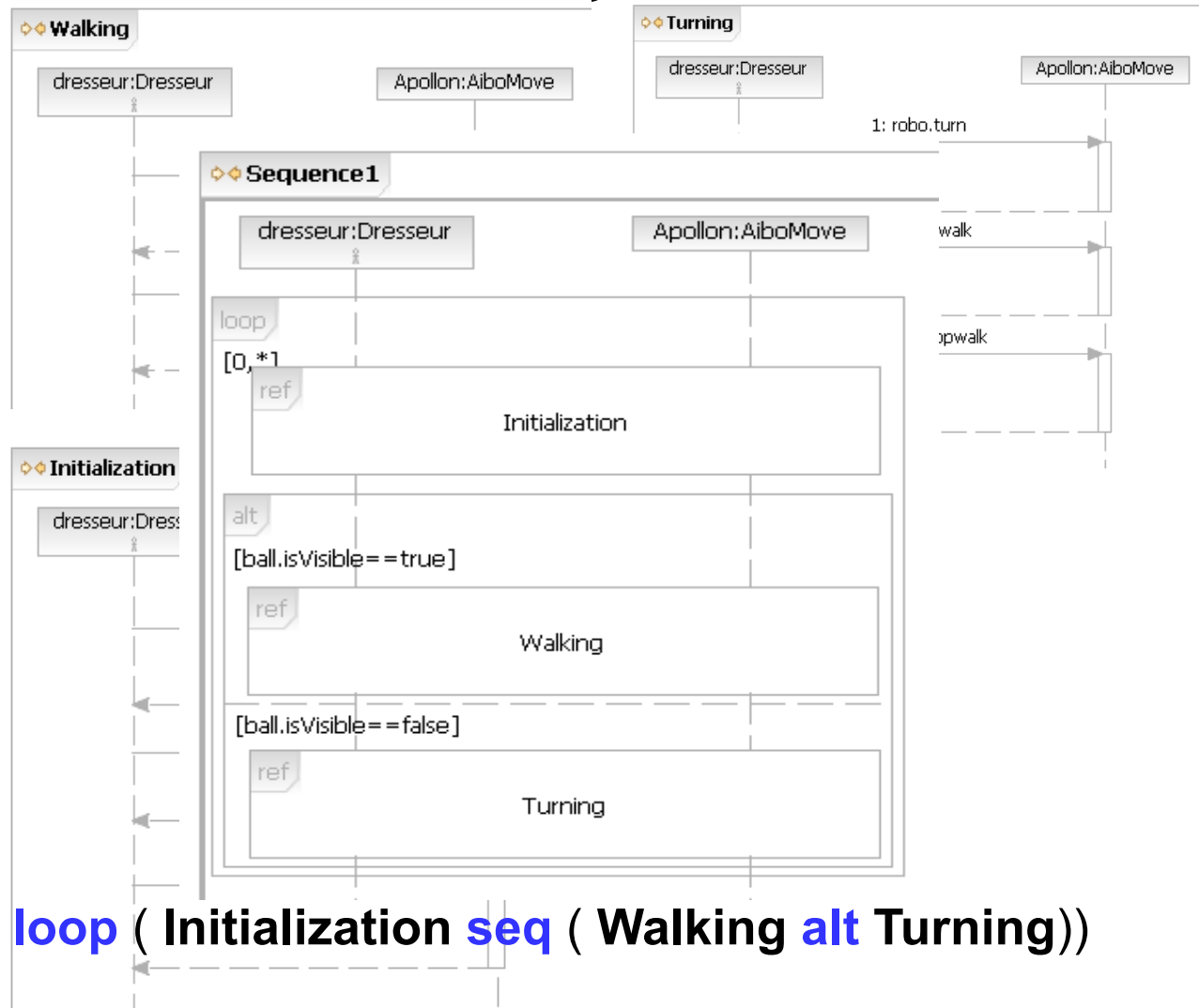
Des diagrammes de séquence : les interactions d' AIBO avec l' environnement.

# AIBO : diagramme de classe

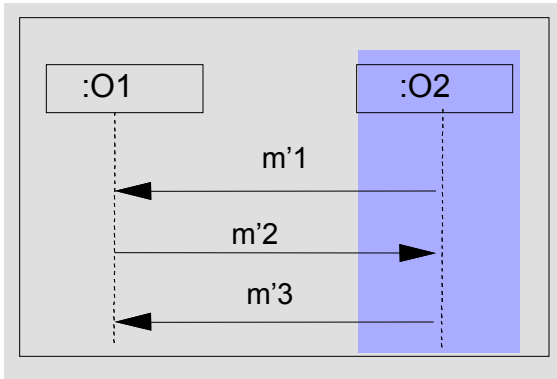
- Une seule classe principale
  - Raison : on s'intéresse aux réactions d'AIBO vs. l'environnement.



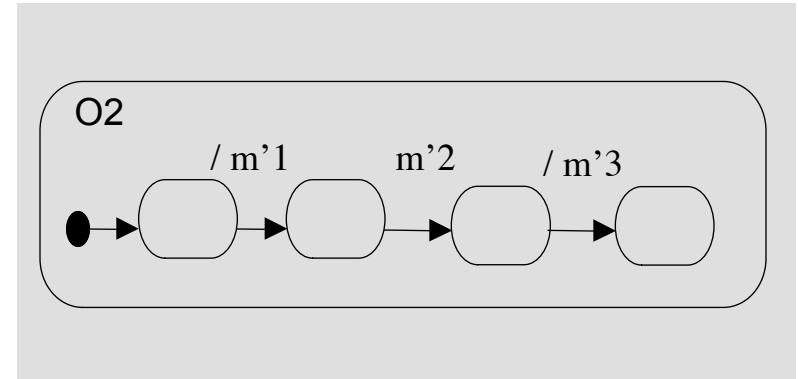
# AIBO · diagrammes de séquence







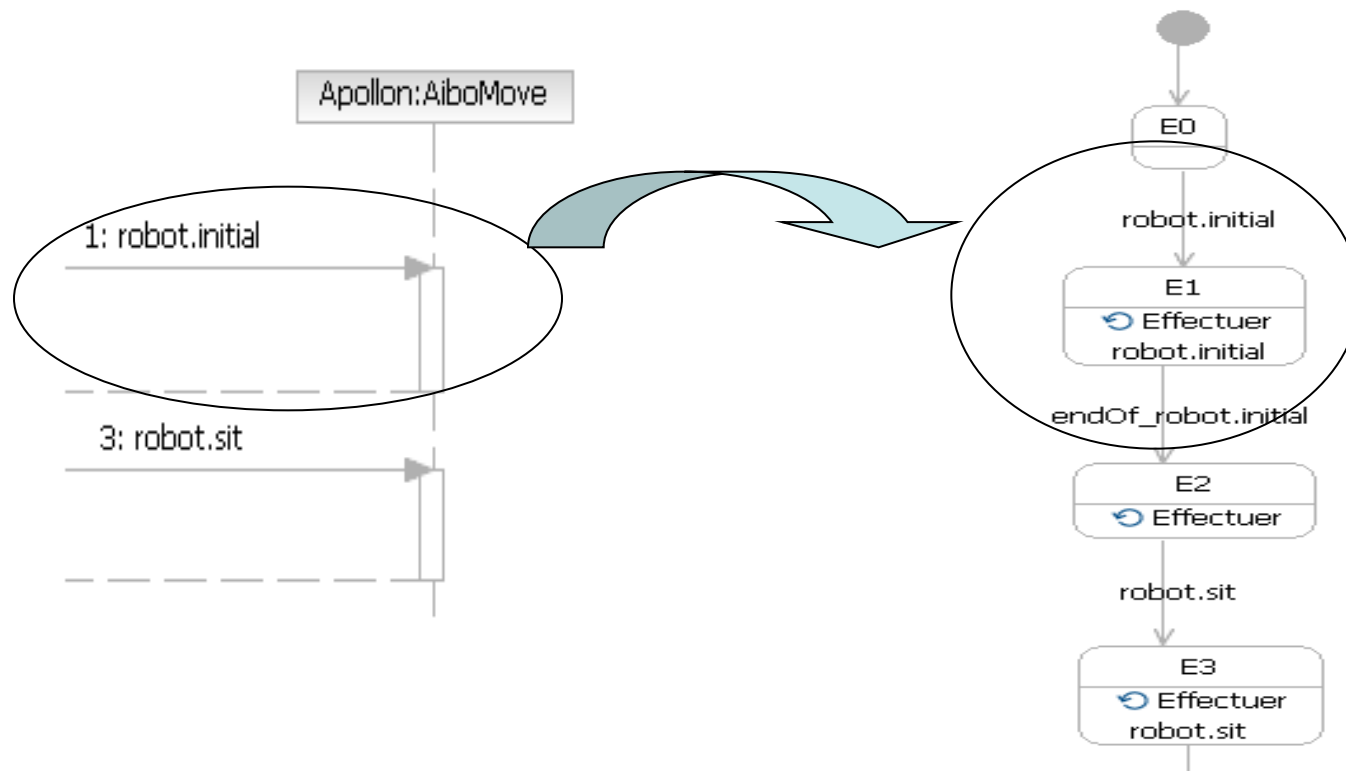
Synthèse  
automatique



**Vue Inter-objets :**  
Plusieurs objets, Un  
exemple.

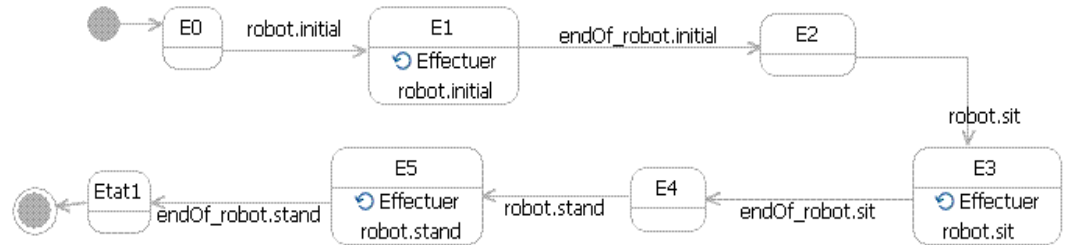
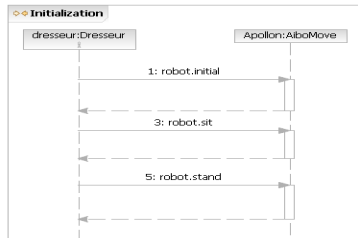
**Vue Intra-objet :** **Un**  
objet, un  
comportement  
complet.

# Phase 1 : diag. de séq. vers machine à états

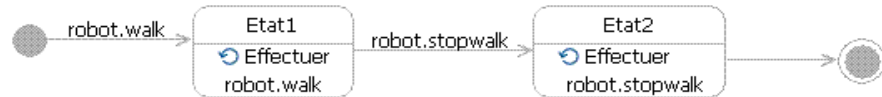
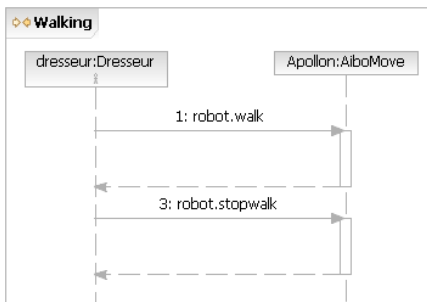


- **P ( SD, O )** : Un algorithme de synthèse basé sur la projection des événements

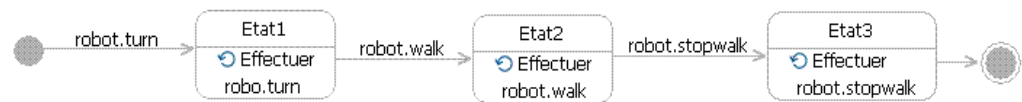
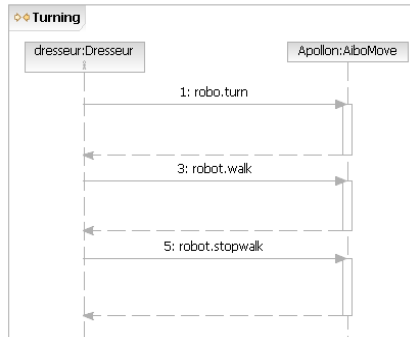
# Phase 1 : diag. de séq. vers machine à états



**P ( Initialization, Apollon)**



**P ( Walking, Apollon)**

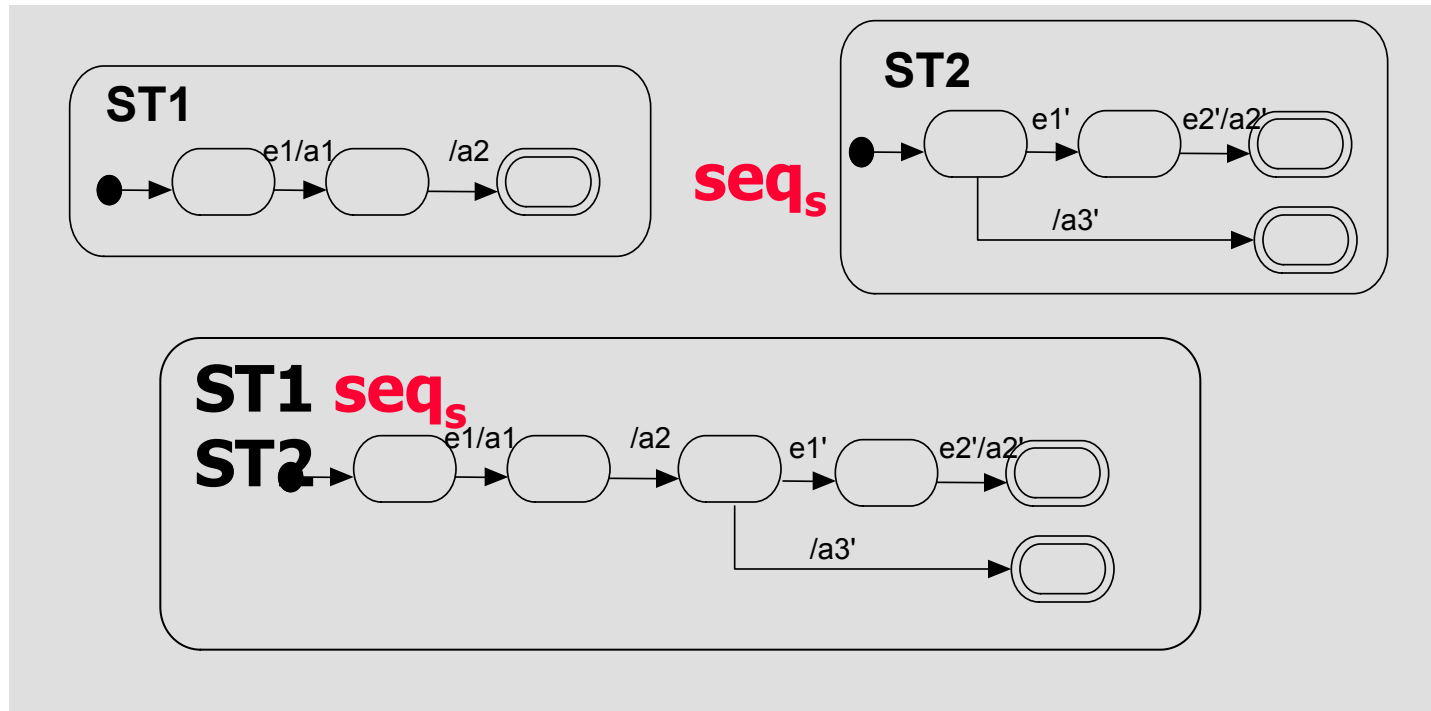


**P ( Turning, Apollon)**

(C) Tewfik ZIADI - UPMC 2013

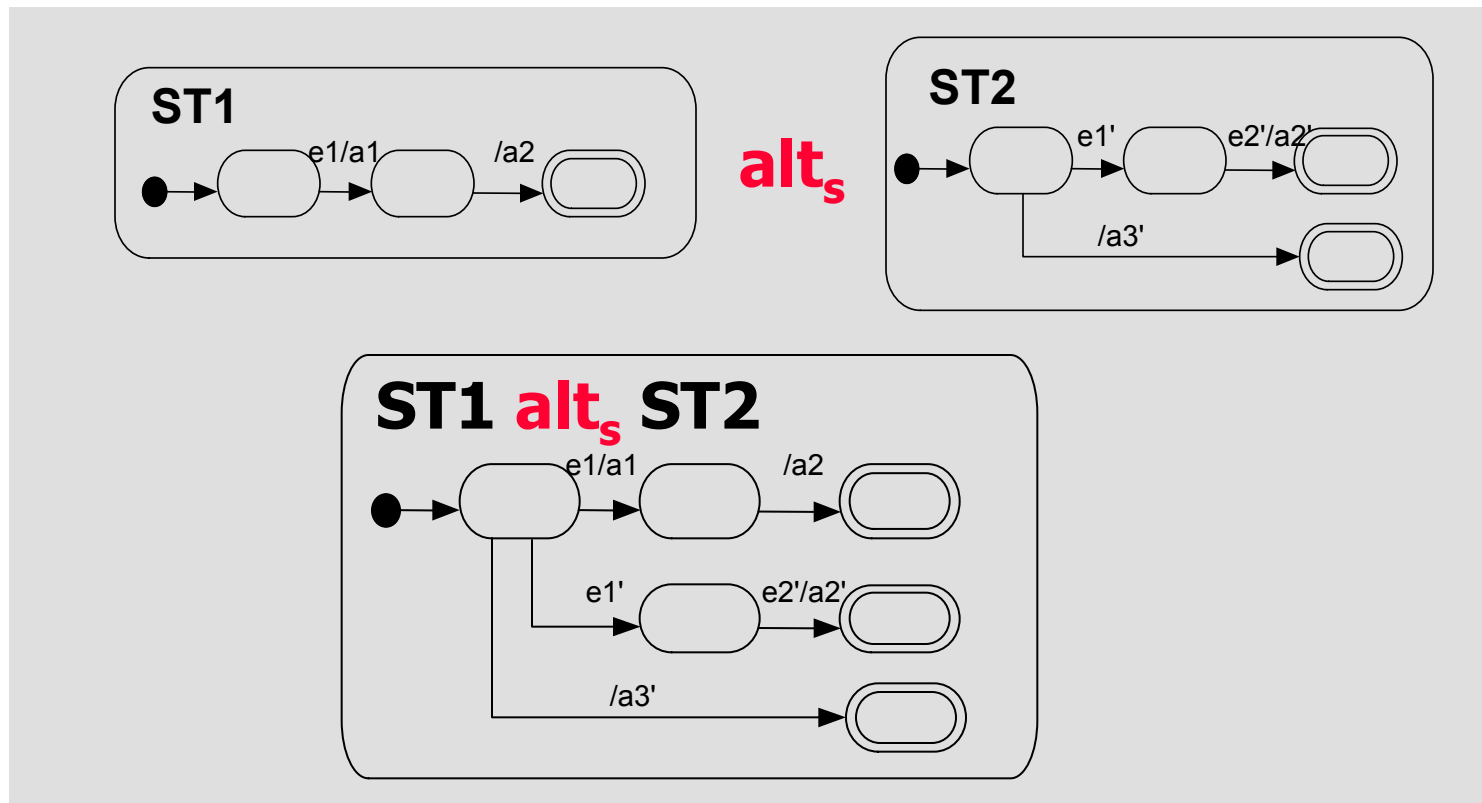
# Opérateurs de machines à états

- 1) Séquence ( $seq_s$ )



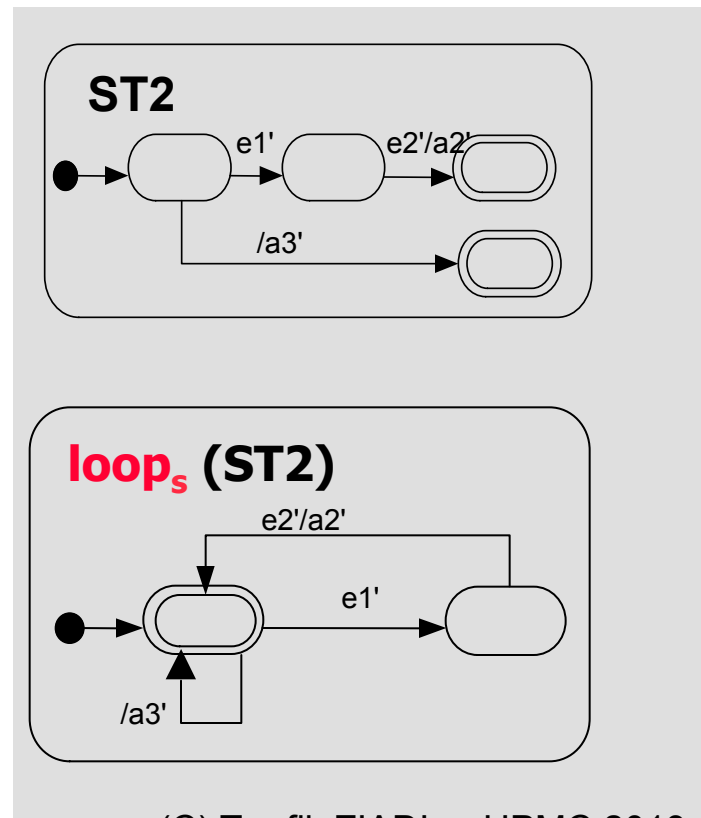
# Opérateurs de machines à états(suite)

## – 2) Alternative ( $\text{alt}_s$ )



# Opérateurs de machines à états(suite)

## – 3) Itération ( $\text{loop}_s$ )

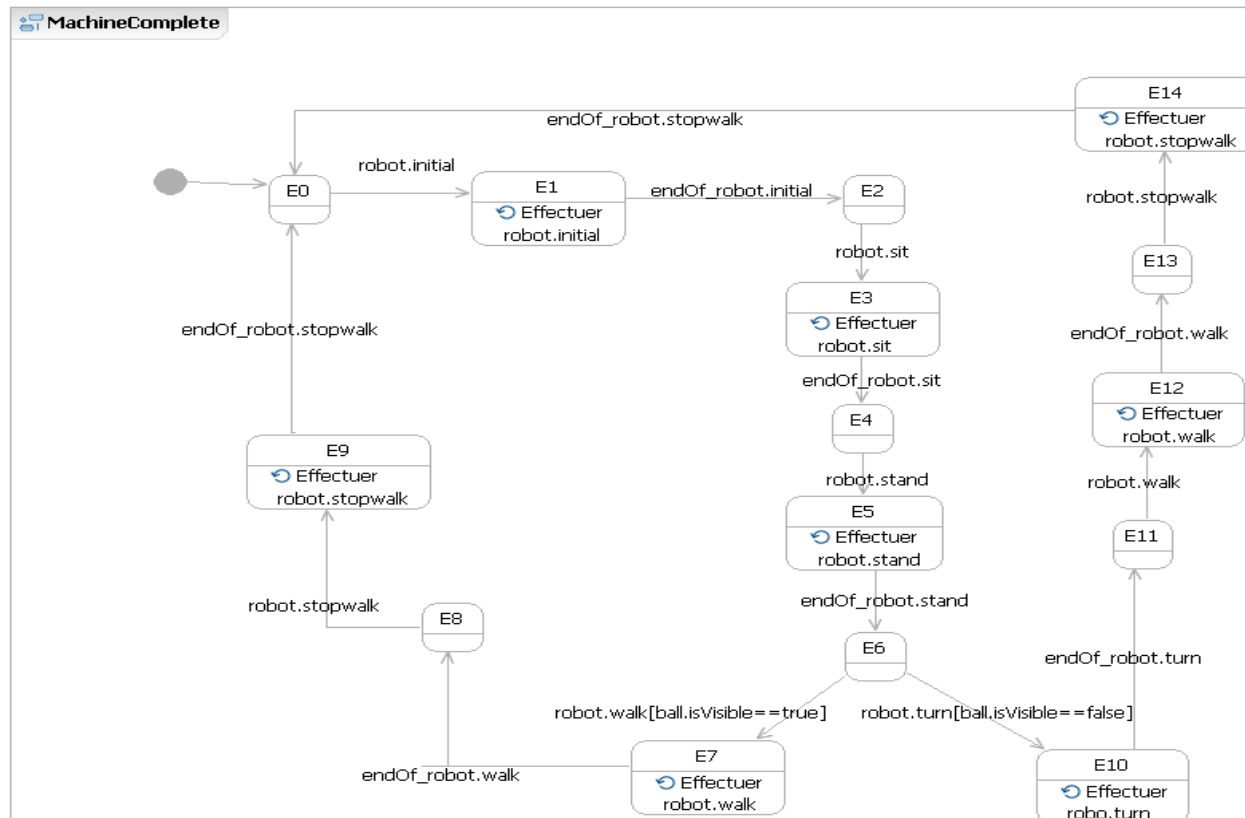


(C) Tewfik ZIADI - UPMC 2013

# Des diag. de séquence vers machines à états

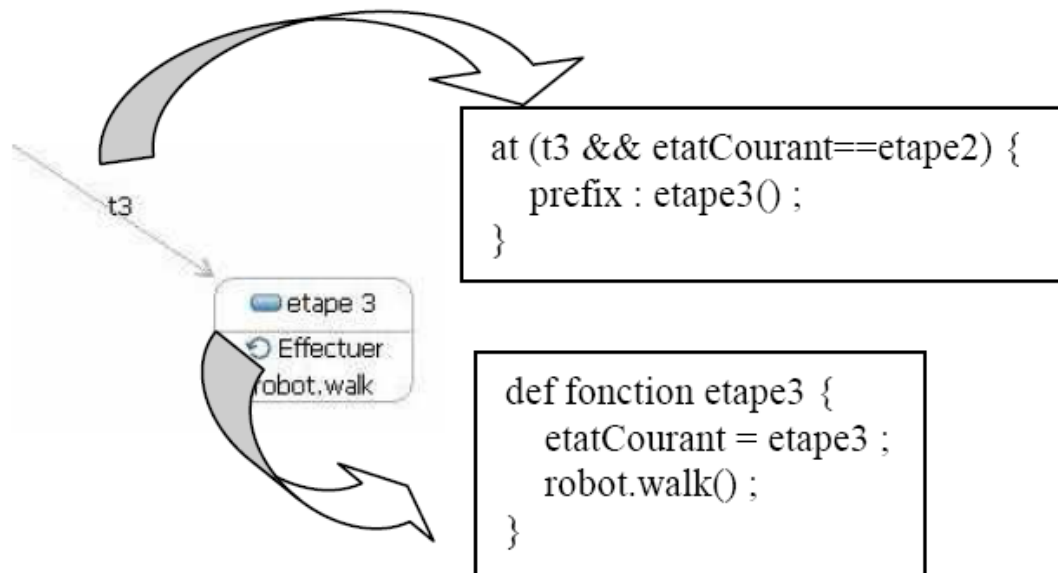
$loop_s$  (P (Initialization, Apollon)  $seq_s$

( P (Walking, Apollon)  $alt_s$  P (Turning, Apollon)))



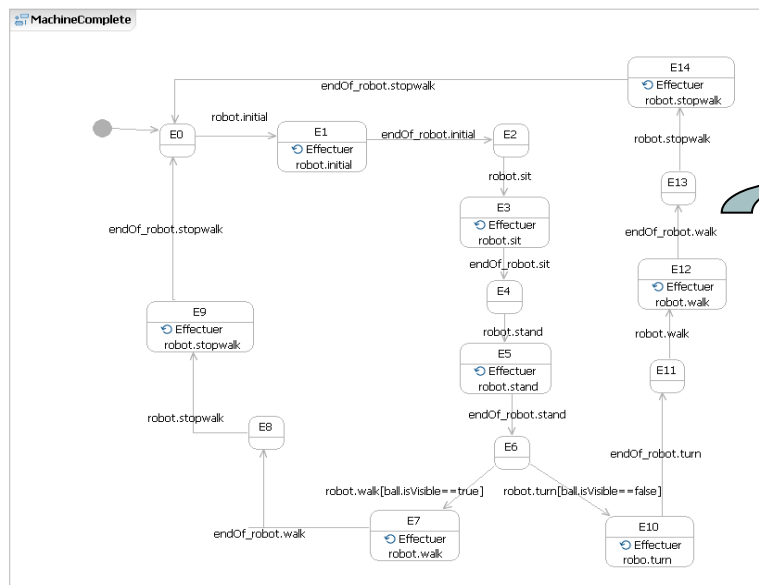
# Phase 2 : Machine à états vers code URBI

- Définir une fonction pour chaque état de la machine à états
- Mécanisme de gestion d'événements « at »





# Phase 2 : Machine à états vers code URBI



Code URBI pour la machine à états



# Critiques

- Seulement pour des missions simples
  - très difficile pour l'utiliser dans le contexte d'une mission complexe (ex. reconnaissance d'un bâtiment)
- Demande que l'utilisateur maîtrise UML
  - C'est pas le cas des roboticiens.
- La solution : proposer un DSML
  - Destiné aux roboticiens
  - Permettant de spécifier des missions complexes.

# Solution 2 : un DSML pour la la robotique

- Proposer un DSML permettant de spécifier d' une manière abstraite les missions.
- Implémenter des transformations vers :
  - les plateformes robotiques (e.g.; wifibot)
  - Les simulateurs (e.g.; webot)

# Exigences pour le DSML

- Comprendre comment les roboticiens définissent leurs missions
  - Analyse de domaine
- Pouvoir spécifier la mission d'une manière qui permettent de viser plusieurs plateformes
  - Des robots (ex. wifibot)
  - Des simulateurs (ex.; webots)

# Robotique mobile: concepts

- Capteur (*sensors*)
  - *Def. « Un capteur désigne un dispositif permettant d'acquérir des données provenant de l'environnement de robot »*
- Pour les robots mobiles :  
*infra-rouge,, sonars, laser, caméra, microphone*

# Robotique : terminologies

- Actionneurs (actuator)
  - Def. « *Un actionneur est le dispositif permettant au robot d'effectuer une action* »
  
- Pour les robots mobiles:
  - *Les moteurs des roues du robot.*

# Exemples

**ROBOT E-PUCK**

**ROBOT WIFIBOT S.C.**

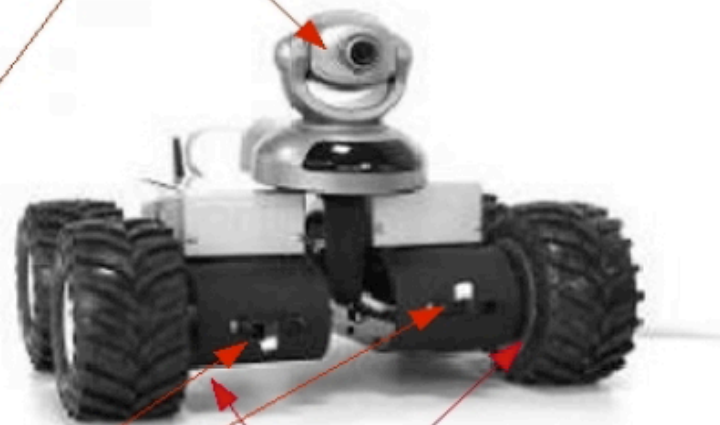
Accéléromètre

Microphones

Caméras

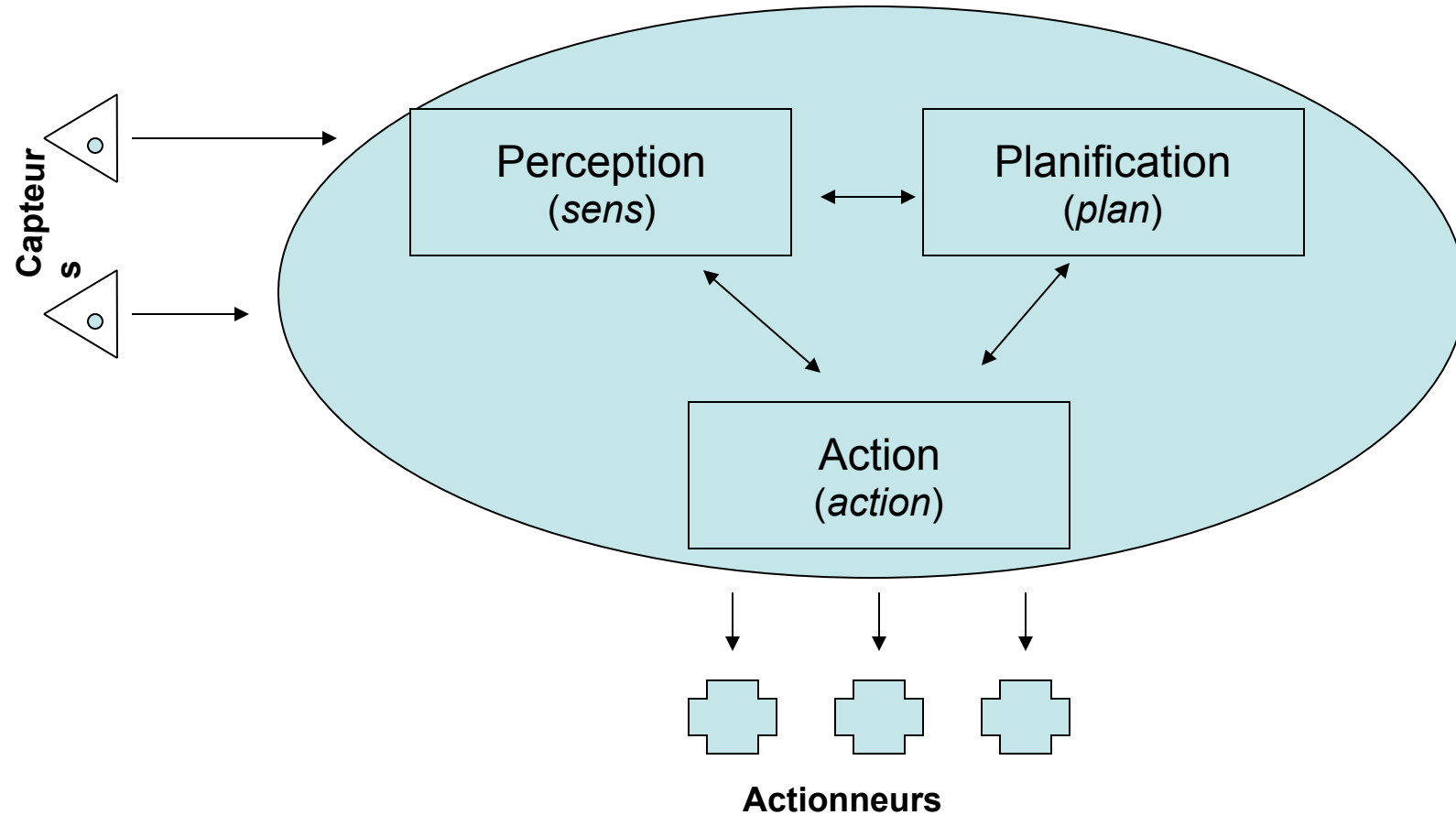


Capteurs infra-rouge



Encodeurs de roues

# Contrôle de robots

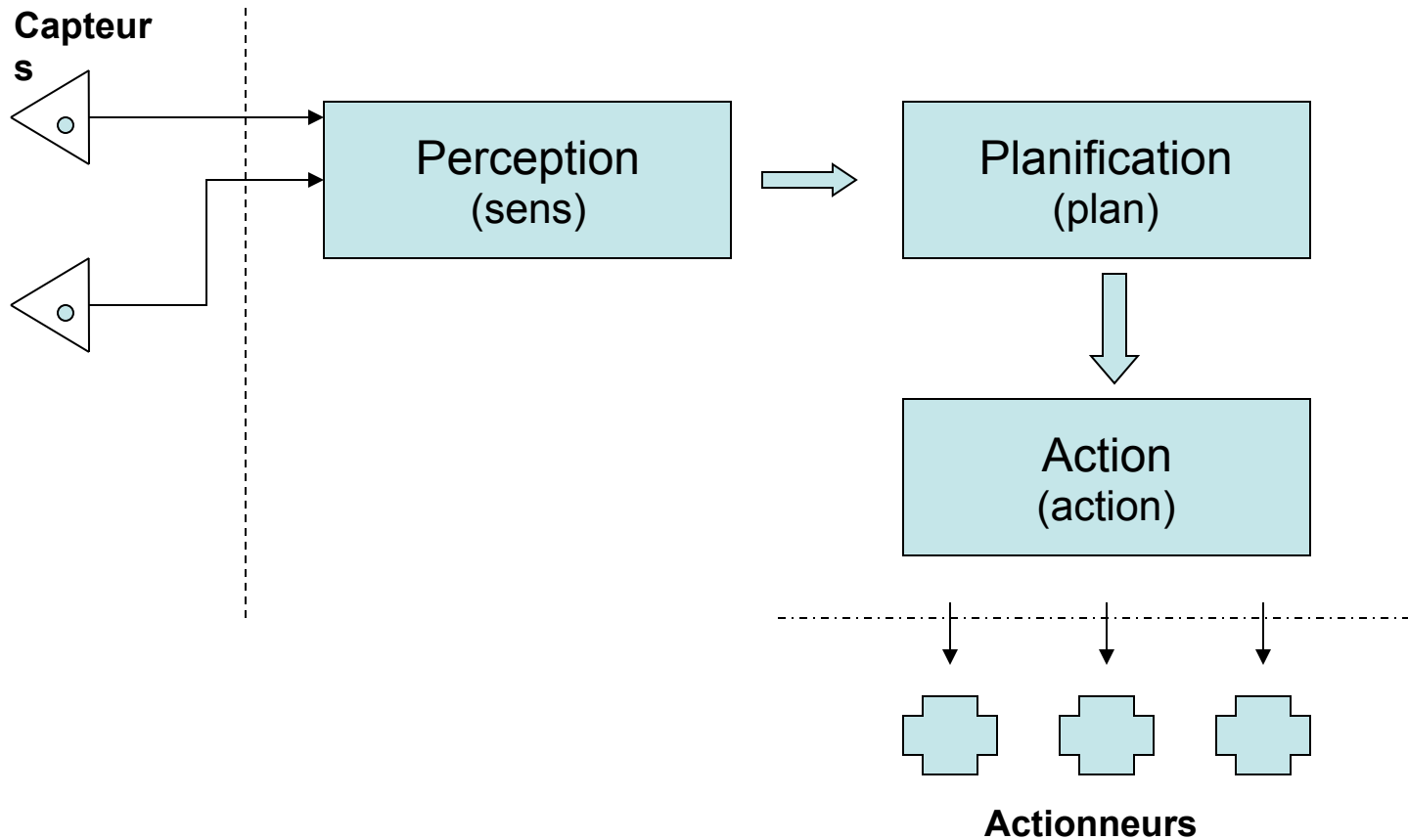




# Contrôle de robots

- Perception (*sens*)
  - La collection des informations depuis les capteurs
- Planification (*plan*)
  - Identifier les actions à réaliser en fonction des buts de la mission
- Action (*action*)
  - Réaliser les actions en émettant les commandes aux actionneurs

# Architecture « délibérative » « Think hard, act later »



# Boucle de contrôle SPA

```
while (true){
//PERCEPTION
    // 1. SENS : extraction des informations provenant des capteurs
    vectorInformation = extractAllInformation();
    //2. Modélisation d' une représentation du monde réel

//PLANIFICATION
    //génération d' un plan d' actions à effectuer à partir du but  et du monde
    plan = planification(goal, world) ;

//ACTION
    //exécution deu plan d' actions
    for step in plan {
        execute(step);
    }
}
```

# Architecture « délibérative » « Think hard, act later »

- Inconvénients
    - Le robot est arrêté pendant l' étape de planification!
    - Le monde réel change dans le temps et le plan devient obsolète !
- Architecture abandonnée depuis les années 80!!

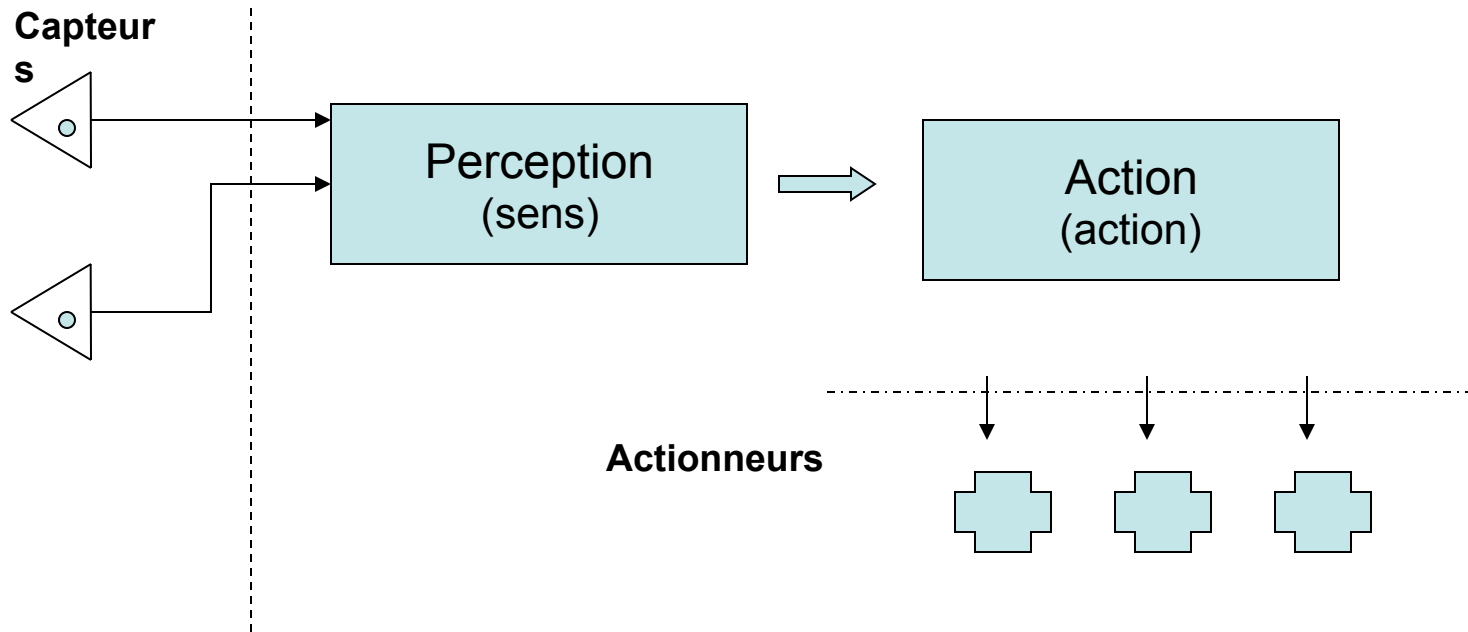
# Contrôle « réactif »

## Don't think (re) act

- Une collection de blocs : sense-act (if-then)
  - blocs qui s'exécutent en parallèle
  - très rapide et réactifs
- Inconvénients
  - Pas de mémoire (pas de sauvegarde du passé)
  - Trop de responsabilités aux capteurs qui ne sont pas fiables

# Contrôle « réactif »

## Don't think (re) act



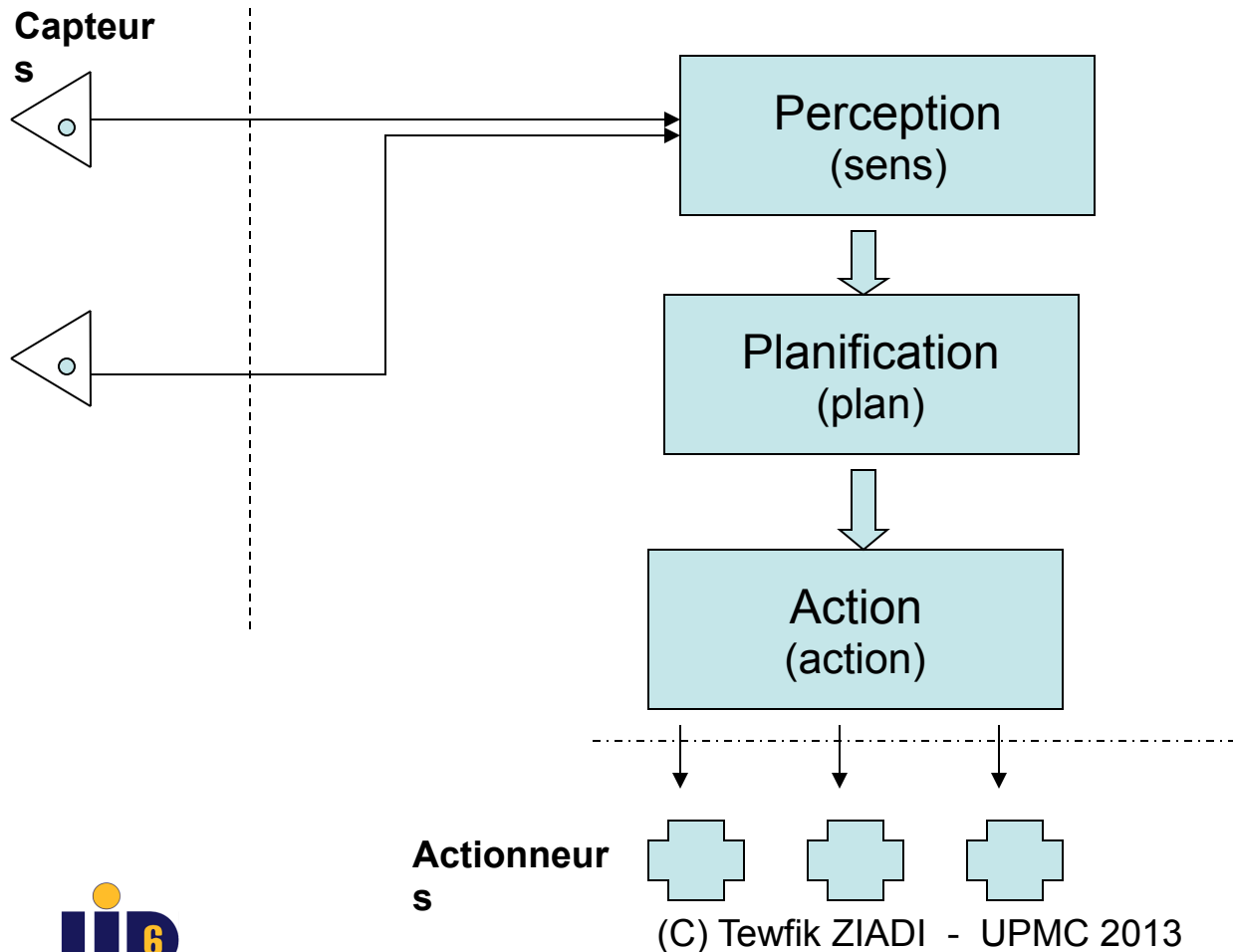
# L'architecture « hybride »

## Think and act independently,

- Combiner deux niveaux :
  - Contrôle réactive
  - Contrôle délibérative
  - Niveau intermédiaire pour lier les deux niveaux
- Souvent appelé architecture 3T

# L'architecture « hybride »

## Think and act independently,

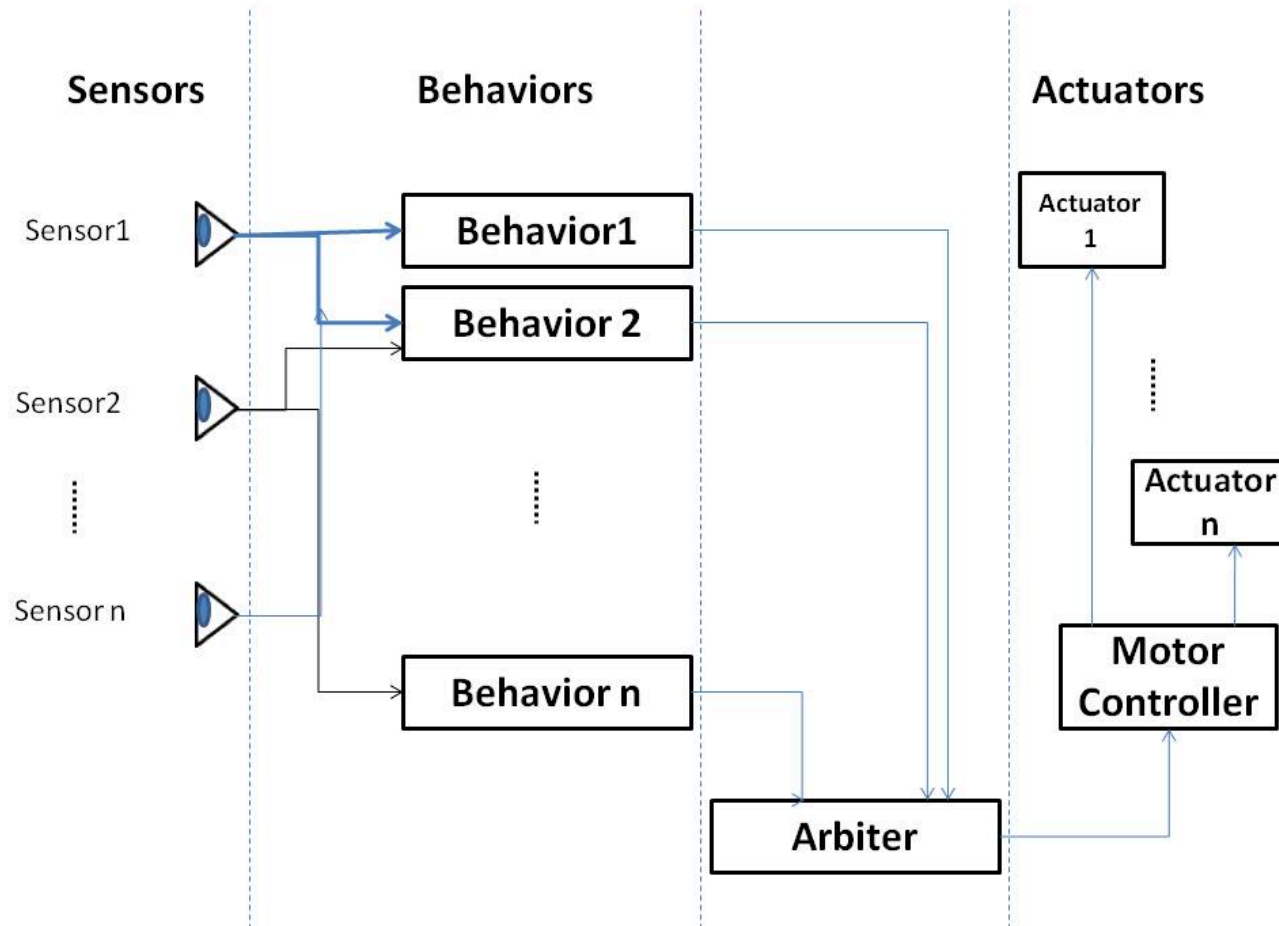




# L'architecture «Behavior-Based »

## Think the way that you act

- Une autre alternative des architectures hybrides
  - Elle combine « délibération » et « réaction »
- Une collection de « comportements » et non pas de blocs réactifs.
  - Comportement plus complexe qu'un « if-then »



# DSML : analyse du domaine

Architecture robotique  
hybride  
réactive  
délibérative  
Capteurs  
URBI  
Actionneurs  
Missions  
Comportement

# C'est quoi un DSML

- **Une syntaxe abstraite** (le méta-modèle)
  - Un ensemble de concepts et de relations entre ces concepts
  - Des règles/contraintes sur les concepts (well-formedness rules)
- **Une syntaxe concrète**
  - Représentation graphique et/ou textuelle

# Vers un DSML pour la robotique mobile

- Un exemple d'un DSML basée sur l'architecture « behavior-based »
  - Un méta-modèle (Ecore)
- Génération de code vers urbiscript.
  - Vers le simulateur (webots)
  - Vers le robot (wifibot)

# Le DSML

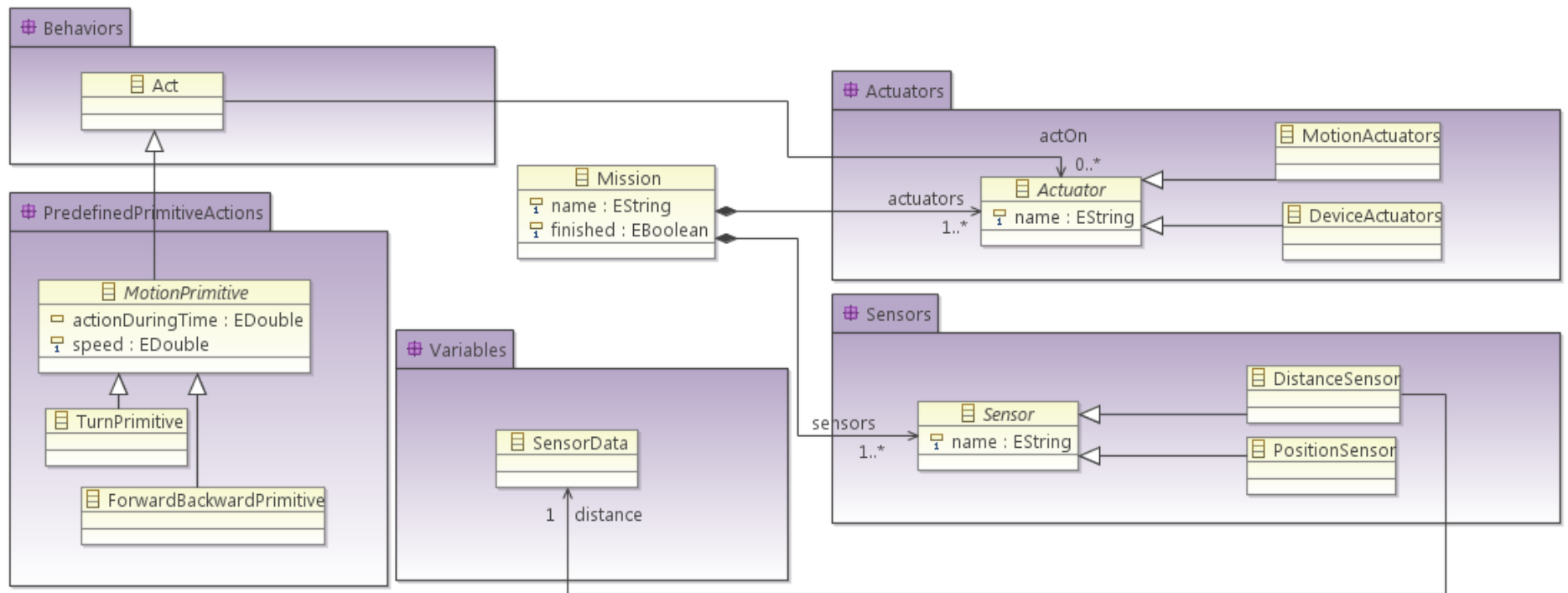
- Étape 1 : définition de la mission
  - Capteur et Actionneur « abstrait »
  - Identifier des comportements
    1. Comportements dits de « survie » (eg. Eviter les obstacles)
    2. D'autres comportements nécessaires pour réaliser la mission.

Chaque comportement est modélisé sous forme d'une machine à état.

# Le DSML

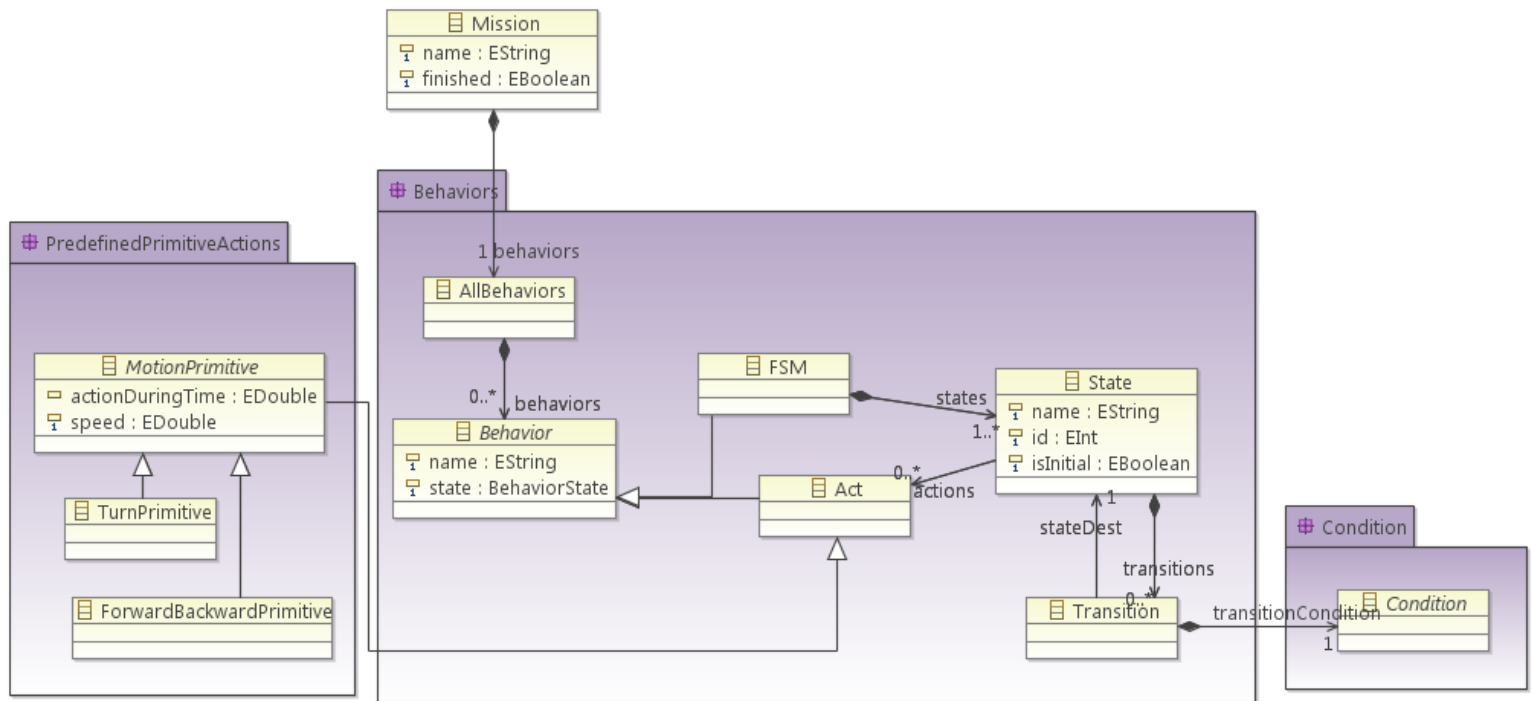
- Étape 2 : Configuration
  - Le choix de la plateforme (un robot et/ou simulateur)
  - Mapping entre capteurs et actionneurs abstraits et ceux de la plateforme (« concret »)
- Étape 3: Génération de code(urbiscript) et exécution

# Capteurs et actionneurs abstraits

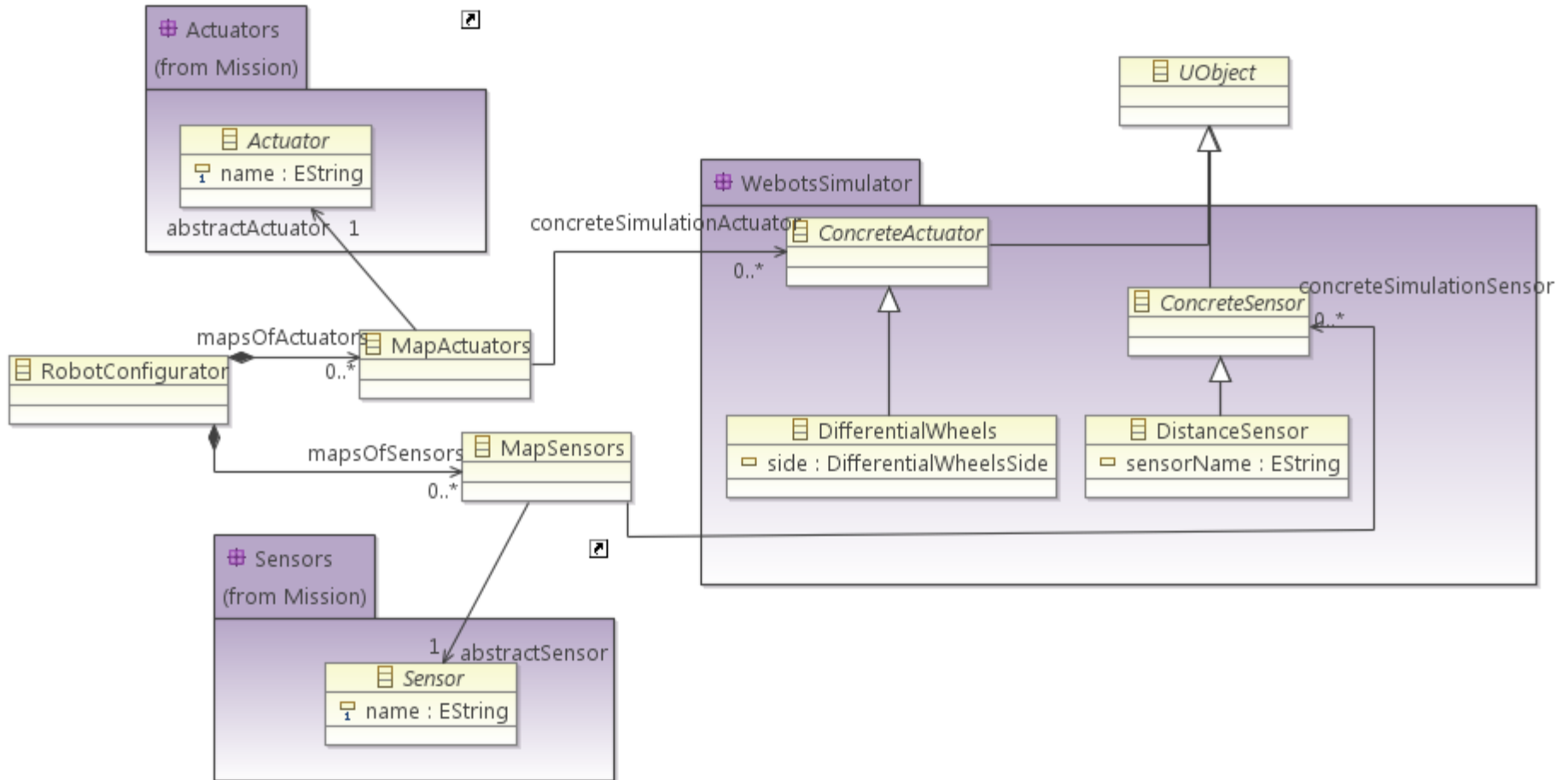




# Comportement



# Configuration pour webots



# Transformations vers scripts

- Un fichier par capteur abstrait de la mission
- Un fichier par actionneur abstrait
- Un fichier pour la mission:
  - Les variables de la mission
  - Les capteurs abstraits utilisés
  - Les actionneurs abstraits utilisés
  - Les FSM en utilisant les fonctions et l'opérateur « at »

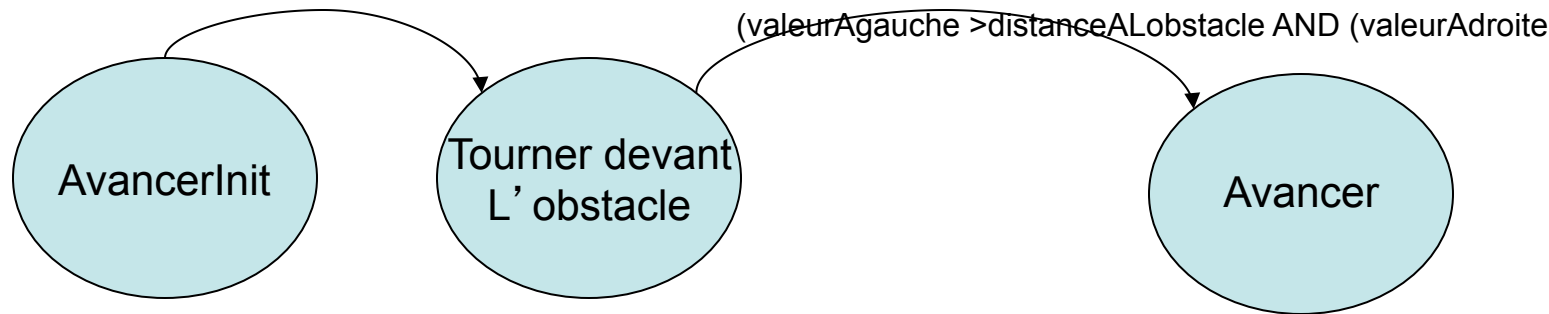
# Exemple mission demi-tour

- Capteurs abstraits:
  - distanceDroite
  - distanceGauche
- Actuators
  - Motion Actuators : déplacement

# Exemple mission demi-tour

- Comportement
  - Action Primitive : avancer
  - Action Primitive : tourner
  - FSM

(valeurAgauche < distanceALobstacle OR (valeurAdroite < distanceALobstacle))



# Exemple : mission-demi-tour

The screenshot shows a ROS resource set tree for the file 'scenario1ForWifibot.robotdsl'. The tree is organized into three main sections, each highlighted with a colored border:

- Configuration (black border):** Contains 'Robot Configurator' with sub-items 'Map Sensors' (two instances) and 'Map Actuators'.
- Mission (green border):** Contains 'Mission demi-tour' with sub-items: 'Simple Variable distanceAuMur', 'Sensor Data valeurAgauche', 'Sensor Data valeurAdroite', 'All Behaviors', 'Sequencer', 'Motion Actuators déplacements', 'Distance Sensor distanceDroite', and 'Distance Sensor distancegauche'.
- Robot concret (black border):** Contains 'Wifi Bot WifiBot' with sub-items: 'Infra Red irRight', 'Infra Red irLeft', and 'Motion Actuator speedAlpha'.

# Un DSML complet : RobotML (2009-2013)

Platform for *RO*botic modeling and *T*ransformations  
for *End-U*users and *S*cientific communities

<http://www.anr-proteus.fr>



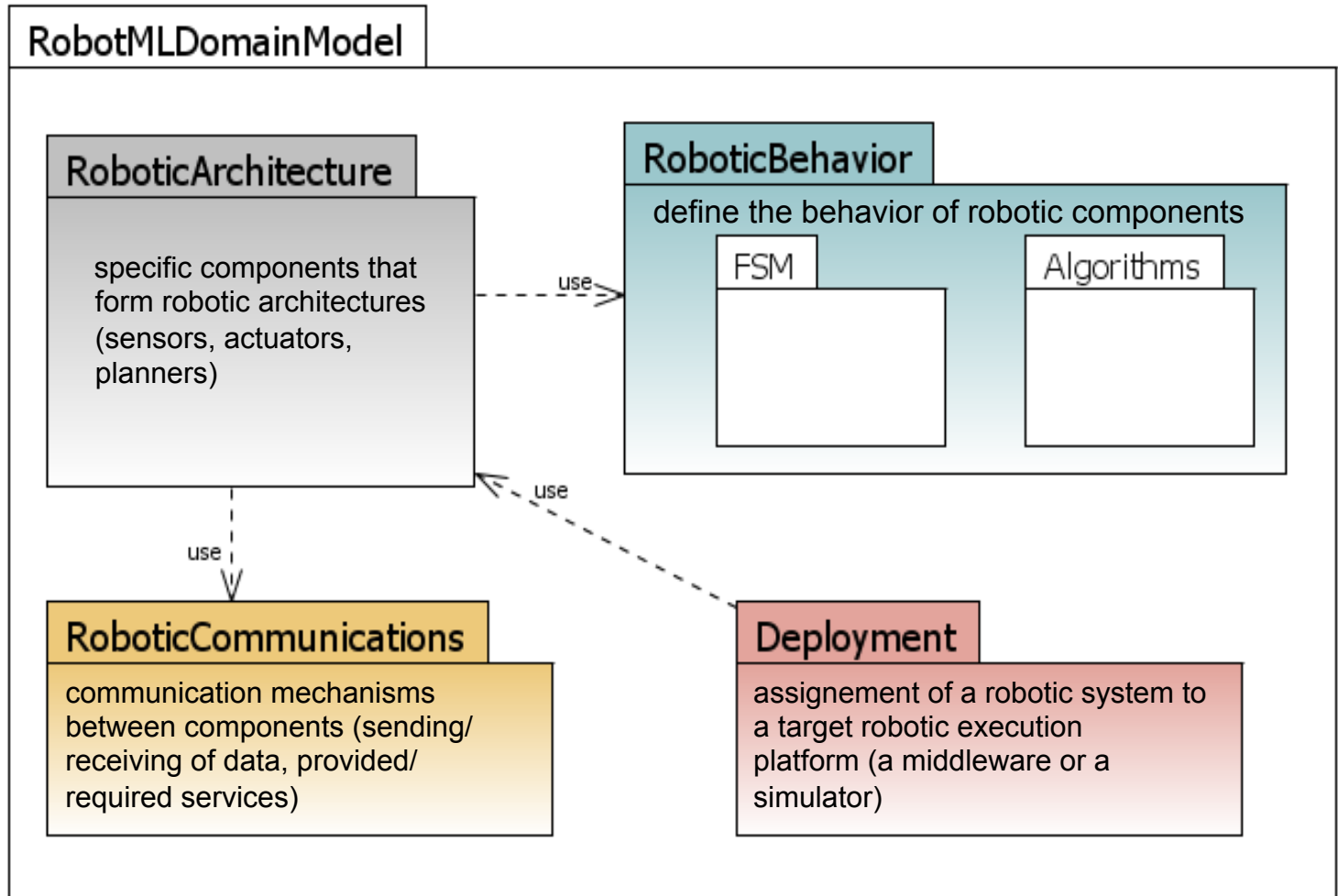
THALES



(C) Tewfik ZIADI - UPMC 2013



# RobotML Domain Model



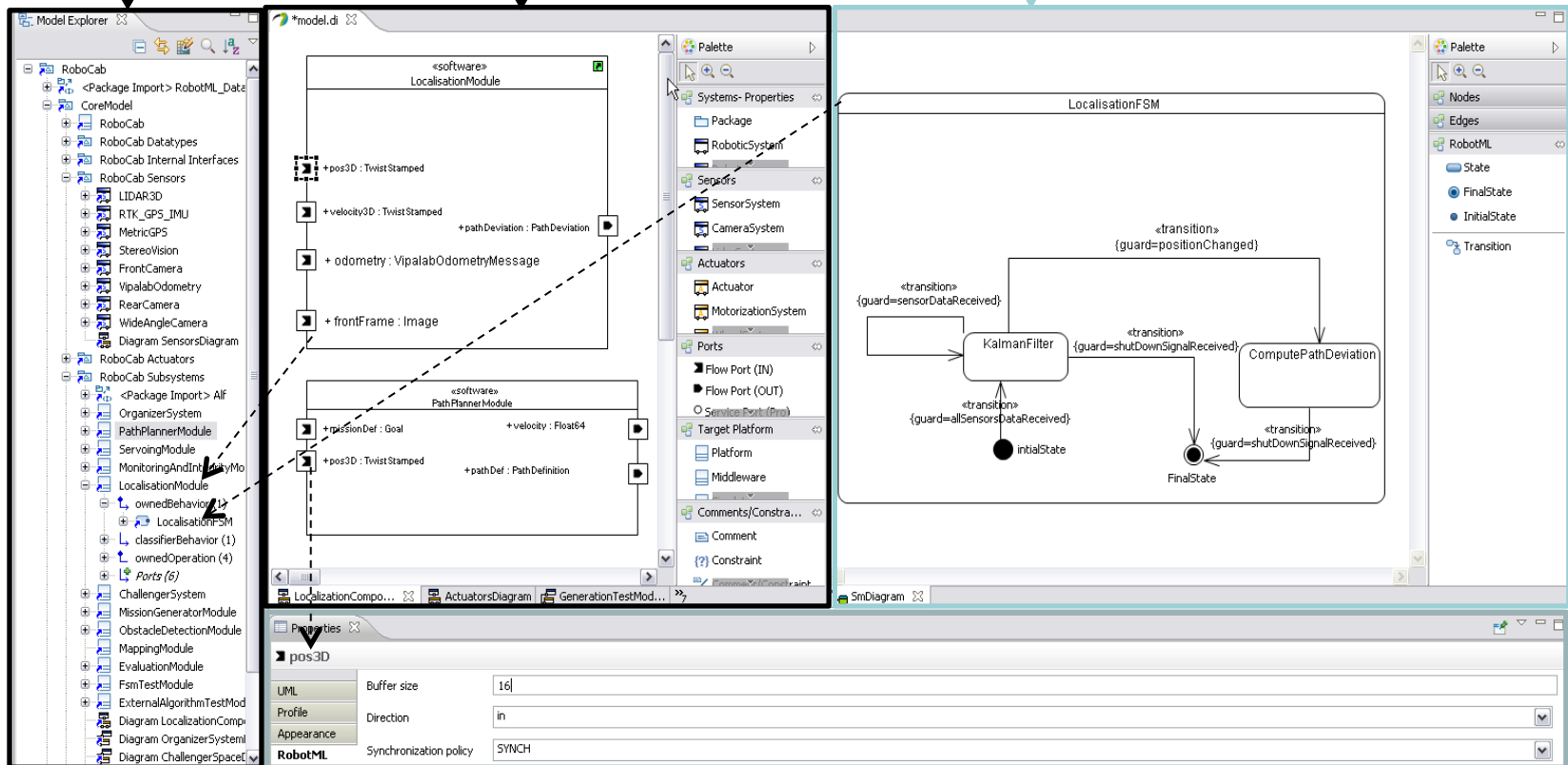


# RobotML Modelling Environment

Model Explorer

Components Definition Diagram

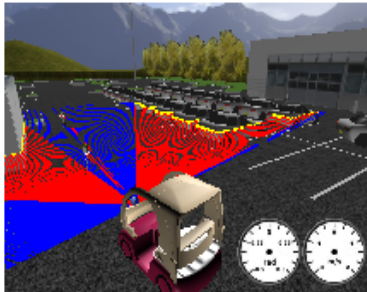
Finite State Machine Diagram



Properties View

# Code generation: Supported Targets

## Simulators

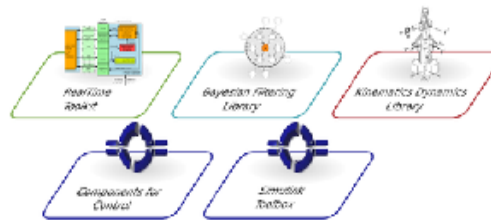


CycabTK



Blender/Morse

## Middleware platforms



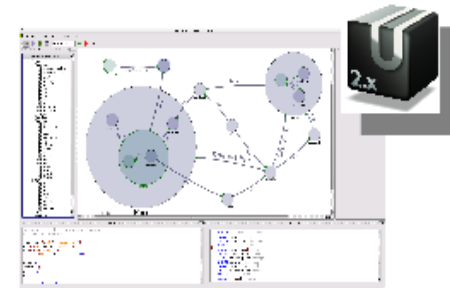
OROCOS



ARROCAM

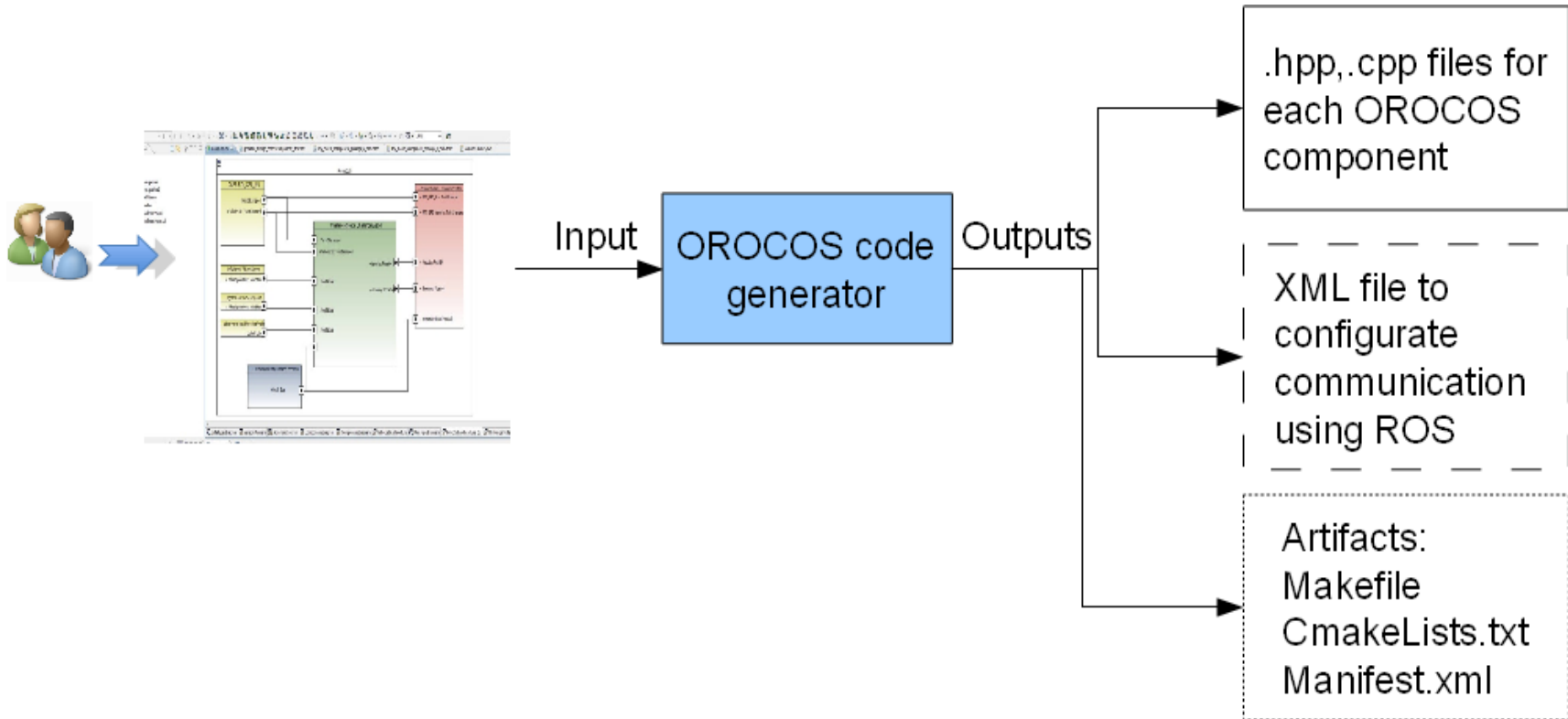


RTMaps



URBI

# Example: Code generation to OROCOS



# Conclusion

- La Robotique est un domaine qui illustre bien l'intérêt de l' IDM
  - Le besoin de l' abstraction
  - Le besoin de capitaliser les missions
- MAIS il faut travailler avec les roboticiens
  - Analyse de domaine
- Dans ce cours
  - UML pour la robotique
  - Un exemple d'un mini DSML.

- Vous êtes toujours des informaticiens :)