# Systèmes logiques et numériques

**Objectifs du cours :**
Après avoir *étudié* l'ensemble de la séquence cours-TD-TP, vous devez être capable :

- Coder une information
- Exprimer un fonctionnement attendu par des équations logiques
- Représenter tout ou partie de l'évolution temporelle d'un SED
- Décrire et compléter un algorithme représenté sous forme graphique
- Interpréter tout ou partie de l'évolution temporelle d'un système (diagrammes SysML de séquences et d'états)
- Modifier un programme pour faire évoluer le comportement du système

## Table des matières

# 1 Introduction aux systèmes logiques

## 1.1 Système logique

Un système *logique* est un système dont les entrées et les sorties sont de type *vrai* ou *faux*, 0 ou 1, ou encore *tout* ou *rien*.

Il s'oppose aux systèmes *analogiques* (grandeurs d'entrées/sorties variables continuement) et aux systèmes *numériques* (grandeurs échantillonnées -discrétisation temporelle- et quantifiées -discrétisation spatiale-). Remarquons que les systèmes numériques peuvent être réalisés par des systèmes logiques (un nombre pouvant être codé en binaire).

*Exemple* :

- Portail de métro : le ticket est valide ou non valide, le portail est ouvert ou fermé, le voyant est allumé ou éteint ;
- Ordinateur : l'information traitée par le processeur est codée sous forme binaire (0 et 1, soit des niveaux de tension bas -0V- ou haut -quelques, en général 3 ou 5 V-) ;
- CD, DVD : ces supports contiennent des signaux numérisés ; la qualité sonore ou visuelle du rendu dépend de la numérisation effectuée (et bien sûr de la qualité de la partie de restitution !).

## 1.2 Présentation du cours

Nombreux sont les systèmes pluritechnologiques actuels qui comportent un contrôle logique par l'intermédiaire d'un automate programme industriel (API), d'un microprocesseur ou d'un microcontrôleur.

Dans ces systèmes, les signaux à contrôler sont exclusivement des signaux logiques. Le système obéit à un processus pré-établi qui envisage toutes les possibilités d'évolution.

On distingue deux types de systèmes à contrôle logique :

- Les systèmes à **logique combinatoire** : un signal d'entrée logique (ou une combinaison de signaux d'entrée) conduit invariablement au même signal de sortie. La même cause produit toujours le même effet, l'effet disparaît dès que la cause disparaît. On utilise alors les propriétés de l'algèbre de Boole (voir partie 3).
- Les **systèmes à évènements discrets** ou **automatismes séquentiels** : le signal de sortie est élaboré à partir d'un signal d'entrée logique (ou d'une combinaison de signaux d'entrée) et prend en compte une chronologie pré-établie qui porte sur un nombre fini d'opérations. Le diagramme SysML d'états et celui d'activités seront utilisés pour analyser et prévoir leur comportement (voir partie 4).

En partie 6, nous présenterons quelques éléments clés concernant l'étude et la programmation de systèmes numériques embarqués (microcontrôleurs).

## 1.3 Illustration : robot humanoïde DARwIn-OP (Robotis)

Le robot humanoïde DARwIn-OP (Dynamic Anthropomorphic Robot with Intelligence – Open Platform) est le dernier né des robots humanoïdes. Il est capable de marcher, se relever après une chute en avant ou en arrière, suivre une balle et jouer au football, parler et reconnaître des documents.

Il est un exemple type de système complexe actuel à composants mécatroniques intégrés.

FIGURE 1 – Robot humanoïde DARwIn-OP (Robotis).

## Caractéristiques principales

Avec son apparence futuriste (figure 1), il intègre toutes les dernières technologies et des fonctionnalités très avancées (voir le diagramme de définition de blocs en figure 2) :

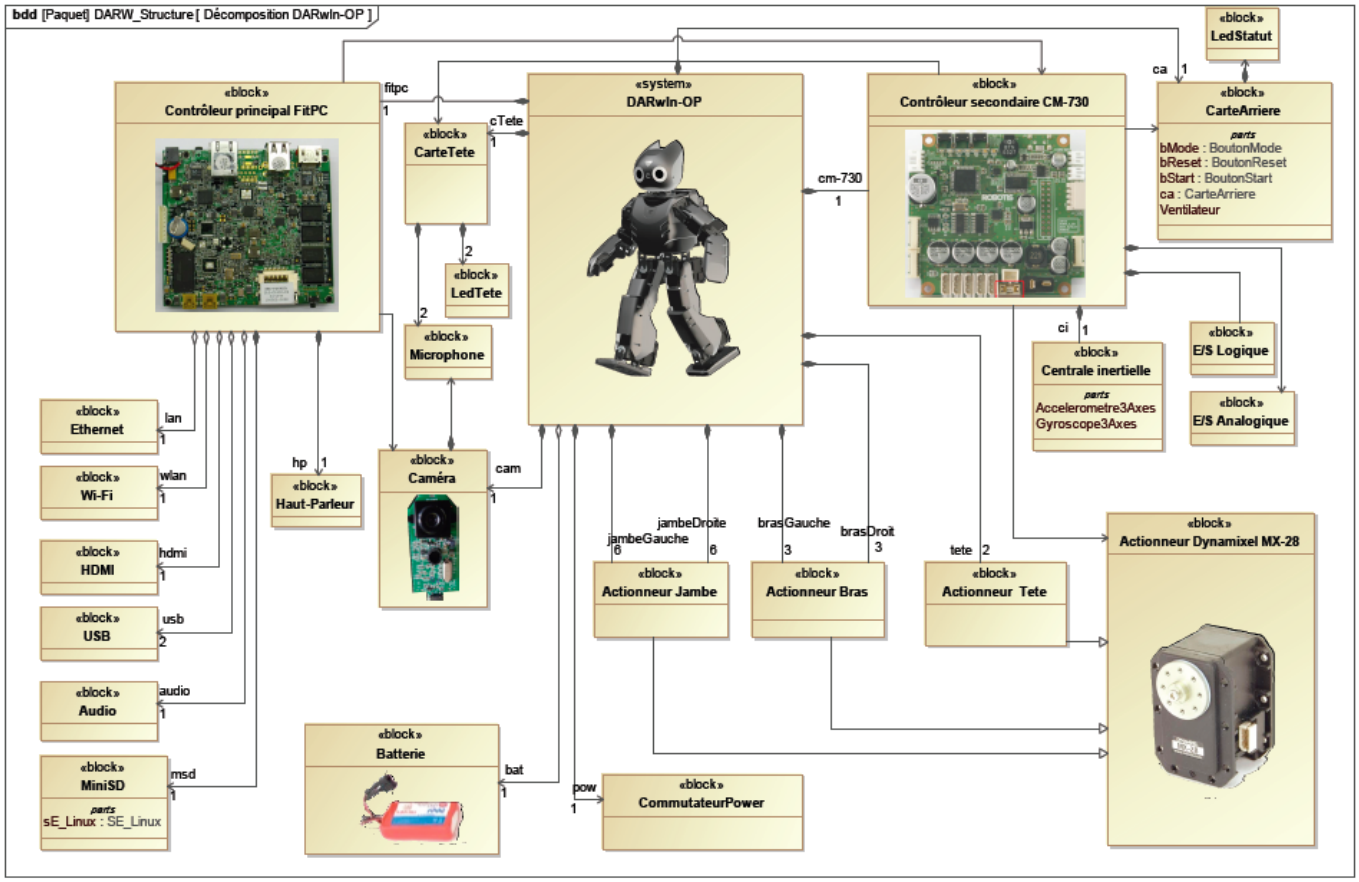



FIGURE 2 – Diagramme de définition de blocs de DARwIn-OP.

- Électronique de commande :
  - Processeur principal Intel Atom Z530 à 1,6 GHz avec 4GO mémoire flash SSD.

  - Carte contrôleur CM-730 avec ARM Cortex M3 à 72MHz.
- Capteurs :
  - Un gyromètre 3 axes
  - Un accéléromètre 3 axes
  - Deux microphones
  - Une caméra USB, 2 MPixels
  - Des yeux avec LED
- Connectique :
  - Un port HDMI
  - Deux ports USB
  - Un port Ethernet
  - Un câble batterie
- Motorisation :
  - Le robot humanoïde est composé de 20 servomoteurs Dynamixel MX-28
  - 6 degrés de liberté pour chaque jambe
  - 3 degrés de liberté pour chaque bras
  - 2 degrés de liberté pour le cou

**Communication avec les servomoteurs**

Le servomoteur MX28 Dynamixel est un actionneur à haute performance incluant toutes les fonctionnalités dont un roboticien pourrait avoir besoin pour contrôler des articulations. Il comporte un microcontrôleur intégré (CORTEX-M3 de 32 bits), un moto-réducteur à courant continu, un codeur de position à effet Hall sur 12 bits ainsi qu'un correcteur PID.

Le protocole de communication entre les éléments est un protocole "propriétaire" dans lequel la transmission s'effectue par paquets de données (avec relation maître/esclave). La vitesse de communication varie de $8000\ bps$ à $3\ Mbps$.

La communication s'effectue physiquement par un bus TTL, suivant une transmission asynchrone série ($8$ bits de données, $1$ bit de stop, pas de parité).

On trouvera en Annexe des détails de ce mode de communication (étudié en Travaux Pratiques).

**Communication entre les deux micro-contrôleurs**

Un bus USB réalise la liaison série haut débit permettant les échanges de données entre le contrôleur principal et secondaire, et bien sûr le flux vidéo de la caméra USB de DARwIn-OP.

# 2 Systèmes de numération et codage de l'information

## 2.1 Expression des nombres dans une base

### Définition

Une base $B_N$ est un système libre de $N$ éléments (chiffre ou caractère) tel que : $B_N = C_{i\,i=1}^{i=N}$ :

$$\forall a \in \mathbb{R}^* \qquad a = \sum_{j=1}^{j=N} C_j N_j$$

### Bases usuelles

| base 10 | base 2 | base 16 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 10 | 2 |
| 3 | 11 | 3 |
| 4 | 100 | 4 |
| 5 | 101 | 5 |
| 6 | 110 | 6 |
| 7 | 111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | $A$ |
| 11 | 1011 | $B$ |
| 12 | 1100 | $C$ |
| 13 | 1101 | $D$ |
| 14 | 1110 | $E$ |
| 15 | 1111 | $F$ |

Tout nombre entier positif peut s'exprimer à l'aide des dix chiffres de 0 à 9 en base 10. Par exemple :

$$23 = 2 \times 10^1 + 3 \times 10^0$$

De même, tout nombre entier positif peut s'exprimer à l'aide des deux chiffres 0 et 1 en base 2 (binaire) ou de 16 chiffres en base 16 (hexadécimal).

Les 3 bases les plus utilisées dans les systèmes industriels sont ainsi :
• Base décimale : $B_{10} = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$
• Base binaire : $B_2 = 0, 1$
• Base hexadécimale :
$B_{16} = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$

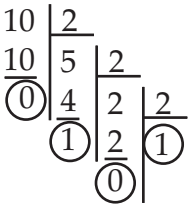
Pour distinguer les bases, elles sont indiquées en indice. Par exemples :
– $8_{10} = 1000_2$
– $16_{10} = 10_{16}$

La notation en langage C utilise un préfixe ($0b$ pour binaire et $0x$ pour hexadécimal) pour préciser la base. Par exemple : $0b01011010 = 0x5A = 90$. La base décimale n'a pas de notation particulière.

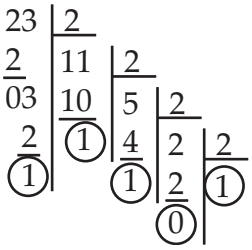
### Passage de la base binaire à la base décimale

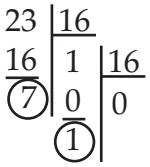
$$1010_2 = (1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0)_{10} = 10_{10}$$

### Passage de la base décimale à la base binaire




$$10_{10} = 1010_2$$





$$23_{10} = 10111_2 = 17_{16}$$

## 2.2 Opérations sur les nombres binaires et hexadécimaux

### 2.2.1 Addition

| Décimal | Binaire | Hexadécimal |
|---|---|---|
| $23$ | $1^1 0^1 1^1 11$ | $1^1 7$ |
| $+\ 10$ | $+\quad 1\ 0\ 10$ | $+\ 0\ A$ |
| $33$ | $10\ 0\ 0\ 01$ | $2\ 1$ |

### 2.2.2 Soustraction

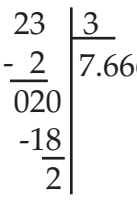
| Décimal | Binaire | Hexadécimal |
|---|---|---|
| $23$ | $1_1 0111$ | $1_1 7$ |
| $-\ 10$ | $-\ {}_1\ 1010$ | $-\ {}_1\ A$ |
| $13$ | $0\ 1101$ | $0D$ |

### 2.2.3 Multiplication

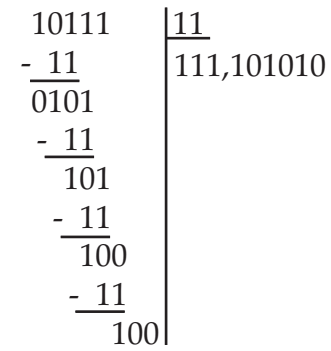
| Décimal | Binaire |
|---|---|
| | $1\ 0\ 1\ 11$ |
| $23$ | $\times\qquad 11$ |
| $\times\ 3$ | $^1 1^1 0^1 1^1 11$ |
| $69$ | $+\ 10\ 1\ 1\ 10$ |
| | $100\ 0\ 1\ 01$ |

### 2.2.4 Division

Décimal



Binaire



## 2.3 Codage de l'information

Coder une information revient à fixer une convention permettant de lire et écrire cette informaion sans ambiguité.

Les chiffres arabes sont notre code habituel pour les données numériques. Ce code est peu approprié aux machines.

Les systèmes automatiques exploitent des codes basés sur le binaire pour des raisons d'architecture. Les qualités requises pour un code sont principalement :

- la taille du codage (nombre de bits (binary digit) nécessaires) ;
- la fiabilité de lecture ;
- la simplicité de manipulation.

### 2.3.1 Le code binaire naturel

Il permet de coder les nombres. C'est le seul code qui permette de réaliser les opérations. La taille de codage est optimale (minimum de bits pour coder un nombre).

Remarque : il faut 4 bits pour coder un chiffre entre 0 et 9.

### 2.3.2 Le code binaire réfléchi (ou code Gray)

| Hexadécimal | Gray |
|---|---|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0011 |
| 3 | 0010 |
| 4 | 0110 |
| 5 | 0111 |
| 6 | 0101 |
| 7 | 0100 |
| 8 | 1100 |
| 9 | 1101 |
| $A$ | 1111 |
| $B$ | 1110 |
| $C$ | 1010 |
| $D$ | 1011 |
| $E$ | 1001 |
| $F$ | 1000 |

Il permet de coder les nombres. Le passage d'un nombre au suivant se fait en changeant la valeur d'un seul bit.
Il est utilisé dans les capteurs de position absolu et évite les sources d'erreur dans les positions intermédiaires.
La taille de codage est optimale.

### 2.3.3 Le code 3 parmi 5

Ce code consiste à choisir 3 bits à 1 parmi 5 bits. Le nombre de combinaisons vaut : $C_5^3 = \frac{5!}{3!(5-3)!} = 10$.

Il permet donc de coder les dix chiffres décimaux.

| Décimal | 3 parmi 5 |
|---|---|
| 0 | 00111 |
| 1 | 01011 |
| 2 | 01101 |
| 3 | 01110 |
| 4 | 10011 |
| 5 | 10101 |
| 6 | 10110 |
| 7 | 11001 |
| 8 | 11010 |
| 9 | 11100 |

Avantages :
- auto-détection de certaines erreurs de lecture,
- Codage personnalisable ($10!$ possibilités).

Inconvénients :
- la taille du codage n'est pas optimale (5 bits pour un chiffre décimal),
- toute opération nécessite un transcodage préalable dans le système de numération adéquat.

Ce code ci-dessous est celui utilisé à la poste pour la lecture du code postal. Le code indiqué a été copié sur une lettre arrivée au lycée. Vous devriez retrouver 84000, qui nécessite 17 bits de codage.

### 2.3.4 Code décimal codé binaire (DCB)

Chaque chiffre décimal est codé en binaire sur 4 bits. Ce code est utilisé sur les afficheurs 7 segments. Chaque afficheur reçoit le chiffre codé en binaire.

| Décimal | 2 | 0 | 1 | 4 |
|---|---|---|---|---|
| DCB | 0010 | 0000 | 0001 | 0100 |

Avantages :
- le code est plus proche de la base 10 ;
- les opérations peuvent être adaptées.

Inconvénients :
- la taille du codage n'est pas optimale.

### 2.3.5 Code ASCII (American Standard Code for Information Interchange)

Le code ASCII permet de coder sur 8 bits les 256 caractères classiques du clavier :
- l'alphabet majuscule et minuscule ;
- les chiffres ;
- les lettres accentuées ou spéciales des langues européennes ;
- la ponctuation.

Le mail est transféré par code ASCII.

### 2.3.6 Code barres

Le code à barres est similaire à celui de la poste mais permet de coder plus d'information sur un espace réduit.

Le code EAN (très utilisé dans la grande distribution) permet de coder les articles avec un protocole de vérification de la lecture.

### 2.3.7 Code 2D : Datamatrix et QR-Code

De nouveaux codes, plus denses, apparaissent. Ces codes permettent de transmettre des adresse internet, des cartes de visites, etc.

Ils peuvent être utilisés à partir de la caméra d'un smartphone.

## 2.4 Transmission série de données

Il existe bon nombre de protocoles de communication série. Sur les systèmes embarquées, on trouve généralement les protocoles RS-232, I2C et CAN. Le protocole RS-232 est parmi les plus simples et permet de communiquer avec un ordinateur.

Le plus souvent, on utilise une communication par trames comportant :

- 1 bit de départ, toujours à 0, servant à la synchronisation du récepteur ;
- les données (souvent 5 à 9 bit) ;
- 1 bit de parité optionnel, permettant de détecter d'éventuelles erreurs de transmission ;
- 1 ou plusieurs bits d'arrêt, toujours à 1.

# 3 Systèmes logiques combinatoires

## 3.1 Illustration intuitive : commande d'essuie-glaces automobile

Afin de garder une certaine visibilité au conducteur d'une automobile sous la pluie, les véhicules sont équipés d'essuie-glaces, sur le pare-brise avant, et sur la lunette arrière. Le commodo est un levier à coté du volant qui par rotation et déplacement vertical permet d'obtenir différentes positions pour commander les essuie-glaces.

On s'intéresse ici au cablage de commande à réaliser pour répondre au Cahier des Charges partiel énoncé ci-dessous, afin d'automatiser la commande et de la réguler sur la vitesse du véhicule.


FIGURE 3 – Commodo d'essuie-glaces.

### 3.1.1 Extrait du Cahier des Charges

Une voiture possède 3 vitesses d'essuie-glace, $V_1$, $V_2$ et $V_3$, croissantes en fonction de l'indice ($V_1$ étant souvent un mouvement intermittent) associées au réglage de la position du commodo. Lorsque le véhicule roule, la vitesse de balayage $V_1$ correspond au réglage dans la position $p_1$, $V_2$ à $p_2$ et $V_3$ à $p_3$. Lorsque le véhicule s'arrête, la vitesse de balayage diminue d'un indice, sauf dans le cas de $V_1$, qui reste maintenue. Le tachymètre (compteur de vitesse) donne une information $r$ lorsque la vitesse est différente de 0.

L'essuie vitre arrière fonctionne de manière permanente (vitesse $V_{ar}$) par rotation du commodo en position $p_{ar}$. Il fonctionne pendant une marche arrière si l'essuie-glace avant est actionné (quelle que soit la vitesse choisie). Un capteur au niveau de la boîte de vitesse délivre un signal $a_r$ lorsque la marche arrière est enclenchée.

### 3.1.2 Étude proposée

- Définir la frontière du système étudié. Quelles en sont les entrées et les sorties ?
- Écrire les équations logiques de $V_1$, $V_2$ et $V_3$ en fonction des variables d'entrée.
- Écrire l'équation logique de $V_{ar}$ en fonction des variables d'entrée.
- Proposer un logigramme des variables de sortie.

## 3.2 Introduction aux systèmes logiques combinatoires

**Un système logique *combinatoire* est un système dont l'état des sorties à l'instant $t$ ne dépend que de l'état des entrées au même instant $t$ (figure 4).** L'histoire de l'évolution des entrées et l'état du système n'interviennent pas. La sortie ne dépend que de l'état des variables d'entrée.

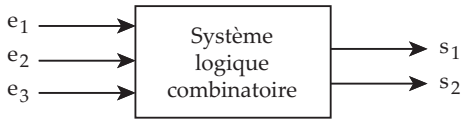



FIGURE 4 – représentation d'un système combinatoire.

*Rappel : une variable logique est une variable dont la valeur vaut 0 ou 1 (faux ou vrai) au cours du temps.*

## 3.3 Algèbre de Boole

### Définition

Soit l'ensemble $\mathcal{B}$ de deux éléments : $\mathcal{B} = \{0, 1\}$. 0 est l'élément nul et 1 est l'élément unité.

Cet ensemble présente une structure d'algèbre avec les lois suivantes :

– Loi unaire : fonction NON (ou complément) :

$$\mathcal{B} \longrightarrow \mathcal{B}$$
$$a \longrightarrow NON(a) = \bar{a}$$

| $a$ | $\bar{a}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

– Loi binaire : fonction OU (ou somme) :

$$\mathcal{B} \times \mathcal{B} \longrightarrow \mathcal{B}$$
$$(a, b) \longrightarrow a\ OU\ b = a + b$$

| $a$ | $b$ | $a + b$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

– Loi binaire : fonction ET (ou produit) :

$$\mathcal{B} \times \mathcal{B} \longrightarrow \mathcal{B}$$
$$(a, b) \longrightarrow a\ ET\ b = a.b$$

| $a$ | $b$ | $a.b$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

– Relation d'équivalence : $=$ (égale).

On précise ici la fonction OU EXCLUSIF :

$$\mathcal{B} \times \mathcal{B} \longrightarrow \mathcal{B}$$
$$(a, b) \longrightarrow a\ OUEXCLUSIF\ b = a \oplus b$$

| $a$ | $b$ | $a \oplus b$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Propriété : $a \oplus b = \bar{a}.b + a.\bar{b}$
c'est le complémenté de l'opérateur identité.

**Propriétés**

Les propriété de l'algèbre de Boole sont :
- commutativité pour $+$ et . : $a+b=b+a$ et $a.b$=$b.a$ ;
- associativité : $a+(b+c)=(a+b)+c$ et $a.(b.c)=(a.b).c$ ;
- distributivité par rapport à $+$ et . : $a.(b+c)=a.b+a.c$ et $a+(b.c)=(a+b).(a+c)$ ;
- éléments neutres : 0 est l'élément neutre de + ($a+0=a$) et 1 est l'élément neutre de . ($a.1=a$) ;
- compléments : $a+\bar{a}=1$ et $a.\bar{a}=0$ ;
- théorèmes de De Morgan : $\overline{a+b}=\bar{a}.\bar{b}$ et $\overline{a.b}=\bar{a}+\bar{b}$ ;
- identités remarquables : $a+\bar{a}.b=a+b$ et $a+a.b=a$.

## 3.4 Manipulation des fonctions logiques

Les sorties d'un système logique combinatoire sont des fonctions booléennes des entrées (figure 4).

Ces fonctions sont en général décrites en français sous forme de "comportement attendu du système". On peut alors en déduire une *table de vérité* pour chaque fonction, c'est à dire présenter sous forme de tableau les valeurs prises par une (ou plusieurs) sortie(s) selon les combinaisons d'entrées possibles (voir exemple ci-dessous).

| $a$ | $b$ | $c$ | $S_1$ | $S_2$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |

L'expression analytique se déduit alors de la lecture de la table de vérité : ici, $S_1(a,b,c)=\bar{a}.\bar{b}.\bar{c}+\bar{a}.\bar{b}.c+a.\bar{b}.\bar{c}+a.b.\bar{c}+a.\bar{b}.c$.

Cette expression peut souvent être simplifiée. La simplification algébrique vise à simplifier une expression en utilisant les propriétés de l'algèbre de Boole :

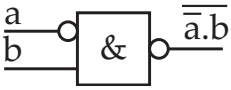
$$\begin{aligned} S_1(a,b,c) &= \bar{a}.\bar{b}.(\bar{c}+c)+a.\bar{b}.(\bar{c}+c)+a.b.\bar{c} \\ &= \bar{a}.\bar{b}+a.\bar{b}+a.b.\bar{c} \\ &= \bar{b}+a.b.\bar{c} \\ &= \bar{b}+a.\bar{c} \end{aligned}$$

Pour une fonction dépendant de beaucoup de variables, la simplification algébrique peut devenir difficile. D'autres méthodes *(pas au programme)* peuvent alors être utilisées.

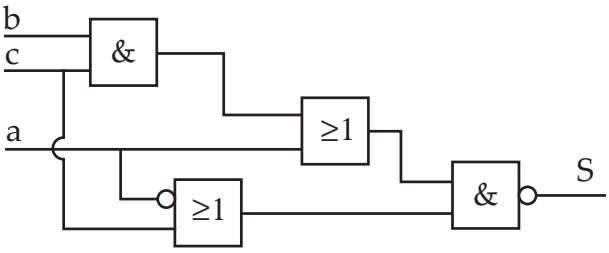
## 3.5 Réalisation de fonctions logiques

### Logigramme

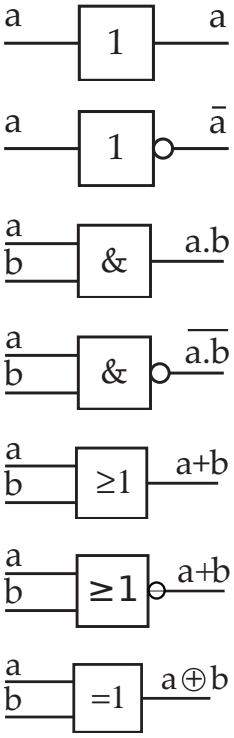
La construction d'un logigramme est une première étape vers la réalisation de fonctions logiques sous forme électronique, pneumatique ou hydraulique.
Les fonctions élémentaires sont représentées par des cellules et l'information d'entrée (à gauche) est traitée pour fournir la sortie (à droite). Le temps de propagation de l'information dans les portes est supposé infiniment court.
Le symbole $\circ$ designe la complémentation. On peut l'utiliser en entrée ou sortie des cellules :




Exemple : logigramme de la fonction $S = \overline{(a + b.c).(c + \bar{a})}$.



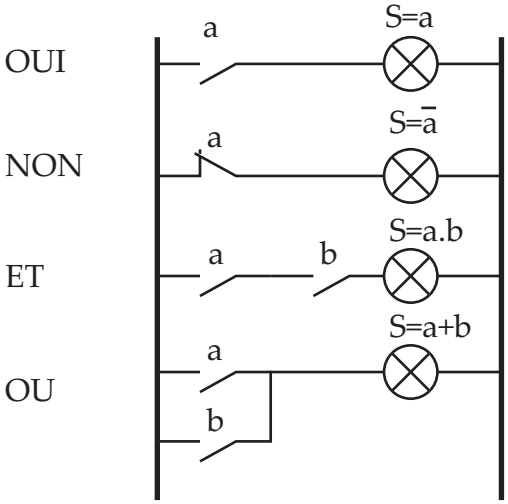

Cellules élémentaires :



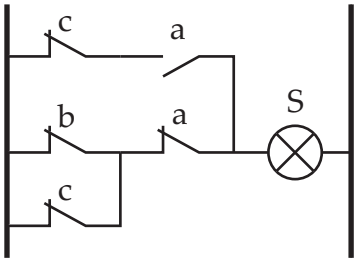

### Schéma à contacts

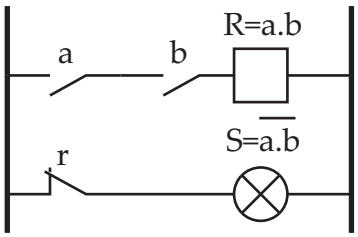
Les schémas à contact sont une première étape vers la réalisation électrique de fonctions logiques.
Les entrées sont représentées par des interrupteurs et les sorties par des ampoules, sur un câblage entre une ligne à gauche représentant une source de tension et une ligne à droite représentant la masse.
L'opération ET (resp. OU) est réalisée par la mise en série (resp. parallèle) des interrupteurs.

La représentation pour chaque fonction élémentaire est donnée ci-contre.




Exemple (voir ci-dessous) : construisons le câblage de la fonction $S = (a + b.c).(c + \bar{a}) = (\bar{a}.(\bar{b} + \bar{c})) + (\bar{c}.a)$.




Un *relais* est, dans un schéma à contact, une sortie complémentaire utlisée comme nouvelle entrée. Dans l'exemple ci-dessus, $S = \overline{a.b}$.

**Réorganisation de fonctions : fonction universelle**

Une fonction est dite *universelle* si elle permet de réaliser les fonctions ET, OU, et NON. Il est alors possible de réaliser toutes les fonctions logiques à l'aide de cette seule fonction.

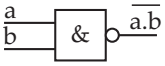
Exemple – fonction NAND (NON ET) :

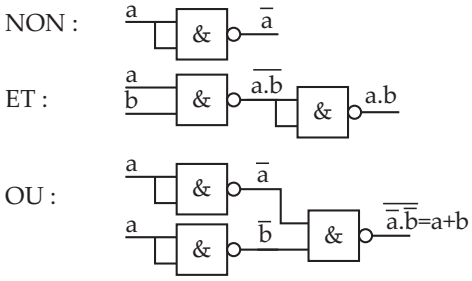
$$a \text{ NAND } b = \overline{a.b}$$

Cette fonction permet de réaliser les fonctions ET, OU et NON :
- NON$(a) = \bar{a} = \overline{a.a}$,
- $a$ ET $b = a.b = \overline{\overline{a.b}} = \overline{(\overline{a.b}).(\overline{a.b})}$,
- $a$ OU $b = a+b = \overline{\overline{a+b}} = \bar{a}.\bar{b} = \overline{(\overline{a.a}).(\overline{b.b})}$.

Logigrammes des fonctions NON, ET et OU réalisées avec la porte NAND :

Porte NAND :



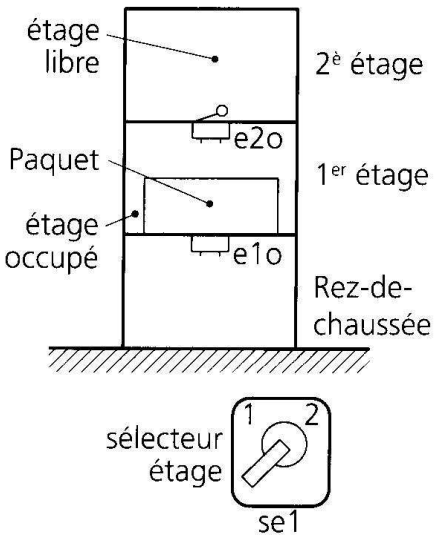




## 3.6 Méthodologie de conception d'un système logique combinatoire

Nous allons illustrer la démarche de conception sur un exemple, un système de rangement [1], dont le cahier des charges est donné ci-dessous.

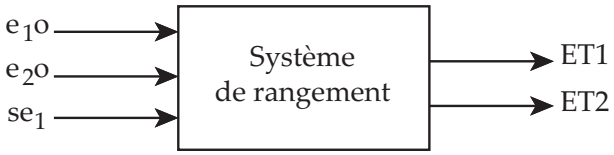
**Système de rangement : Cahier des charges**

Soit un système de rangement de paquets comprenant deux étages. Les paquets arrivent au rez-de-chaussée. Un mécanisme non représenté et hors étude permet de placer les paquets arrivant au rez-de-chaussée soit au premier étage, soit au deuxième étage. L'occupation de chaque étage est détectée par un capteur "étage occupé" ($e_1o$ et $e_2o$). Un sélecteur permet de définir un premier niveau de priorité entre les deux étages, la modélisation de ce sélecteur correspondant à une information : sélection de l'étage 1 $se_1$, la sélection de l'étage 2 correspond à $\overline{se_1}$.
La fonction logique recherchée doit élaborer l'information : étage choisi à tout moment, c'est-à-dire quelle que soit la configuration du système (nombre d'étages occupés et position du sélecteur).




1. Exemple provenant du livre Sciences Industrielles, G. Colombari, J. Giraud.

1. **Réaliser le bilan des entrées – sorties du système**



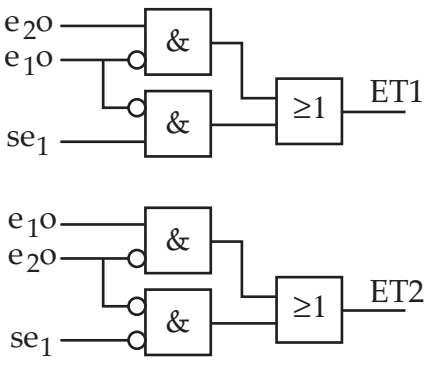

2. **Vérifier que le système est bien combinatoire**
   Les sorties à l'instant $t$ dépendent-elles uniquement des entrées à l'instant $t$ ?
3. **Écrire la table de vérité pour chaque sortie**
   Table de vérité :

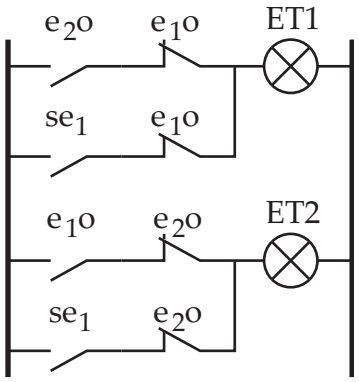
| $e_1o$ | $e_2o$ | $se_1$ | $ET1$ | $ET2$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 |

4. **Déterminer les fonctions logiques pour chaque sortie**
   Les fonctions logiques simplifiées sont :

$$\begin{cases} ET1 = e_2o.\overline{e_1o} + se_1.\overline{e_1o} \\ ET2 = e_1o.\overline{e_2o} + \overline{se_1}.\overline{e_2o} \end{cases}$$

5. **Construire les schémas de réalisation sous forme de logigramme ou de schéma à contacts**




   *Remarque : la solution n'est pas unique.*
6. **Vérifier le bon respect du cahier des charges sur le câblage obtenu**

# 4 Systèmes à évènements discrets *(SED)*

## 4.1 Introduction aux systèmes séquentiels

Un système *séquentiel* est un système dont les sorties $S_i$ dépendent de l'histoire des entrées $E_i(t)$.

*Exemple* : commande d'un moteur électrique par un système séquentiel (figure 5). On remarque sur le chronogramme de droite que la sortie $S$ peut présenter une valeur différente (0 ou 1) pour une configuration identique des entrées $m$ et $a$.

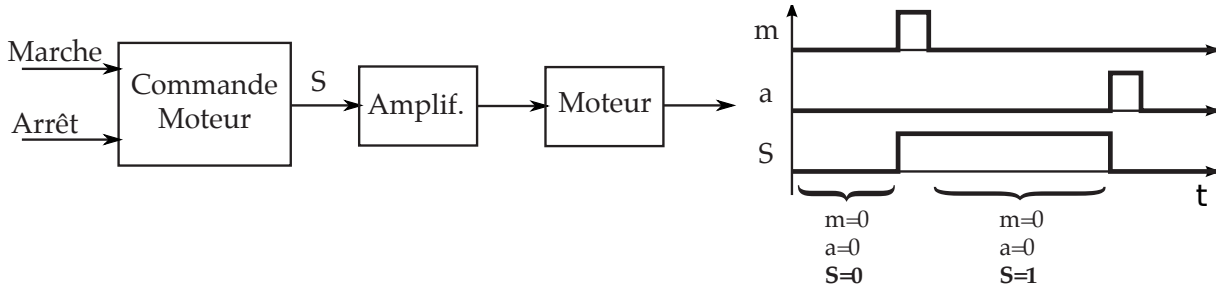



FIGURE 5 – Commande d'un moteur électrique.

Le système est capable de *mémoriser* de l'information. Cette information mémorisée est *l'état du système*, qui peut être représenté par un vecteur d'état booléen : $\vec{X} = (x_1, x_2, ..., x_n)$.

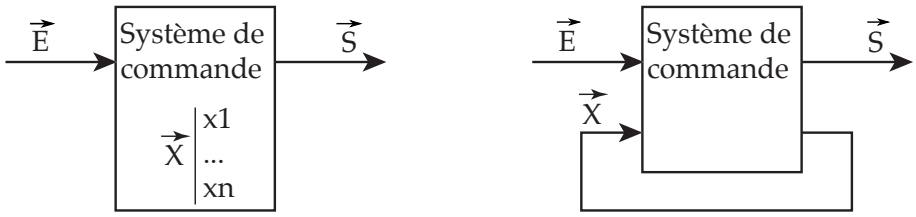



FIGURE 6 – Représentation d'un système séquentiel par un système combinatoire muni d'une variable d'état.

Connaissant $\vec{X}$ et $\vec{E}$, la sortie $\vec{S}$ peut être déterminée comme une fonction booléenne (figure 6) de $\vec{X}$ et de $\vec{E}$ : $\vec{S} = f(\vec{X}, \vec{E})$.

L'état interne $\vec{X}$ à l'instant $t$ dépend de $E(t)$ et de l'état interne immédiatement précédent : $\vec{X}(t) = g(\vec{E}(t), \vec{X}(t - \Delta t))$.

Les tables de vérité ne permettent plus de décrire l'évolution de $S$ en fonction de $E$. On utilise alors les *chronogrammes* (aussi appelés *diagrammes de Gantt*, figure 7) : on y représente l'évolution chronologique des entrées et sorties avec changements d'états instantanés et simultanés.

Les fronts (montant et descendant) permettent de caractériser le changement d'état d'une variable et sont utilies pour synchroniser des processus. Le **front montant** d'une variable ($\uparrow a$) est vrai lors du passage de la variable de l'état logique 0 à l'état logique 1 ; le **front descendant** ($\downarrow a$) est vrai lors du passage de l'état 0 à l'état 1. Un front est représenté par un Dirac sur un chronogramme.

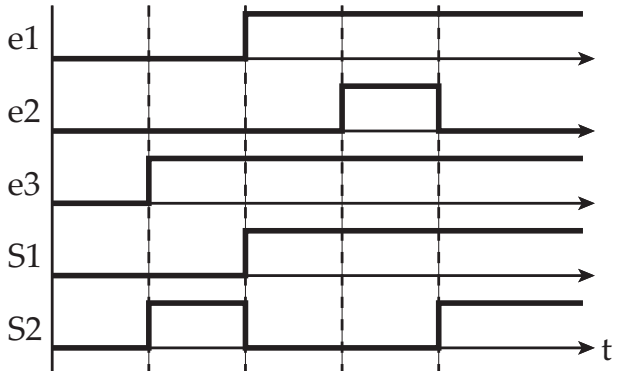



FIGURE 7 – Exemple de chronogramme.

## 4.2 Illustration : fonctionnement du robot DARwIn-OP

On trouvera en Annexes A, B et C les diagrammes comportementaux SysML du robot DARwIn-OP. La partie suivante fournit un guide de lecture de ces diagrammes.

# 5 Outils de modélisation des systèmes logiques

## 5.1 Diagramme de séquences (*SD – SysML Sequence Diagram*)

Le diagramme de séquences est un diagramme comportemental permettant de décrire le(s) scénario(s) d'un cas d'utilisation (voir figures 21, 23 et 22 pour l'exemple du robot DARwIn-OP).

Il répond à la question : *"Comment est réalisé ce cas d'utilisation ?"*.

Il représente les différentes **interactions** du système (ou de sous-systèmes le cas échéant) avec les différents acteurs (ou système ou sous-systèmes) au moyen de messages, dans le contexte d'un scénario donné. Il ne montre donc que l'enchaînement séquentiel des différentes interactions :

- dans l'ordre chronologique (le temps s'écoule vers le bas),
- par des *messages* (appel d'un comportement chez le destinataire),
- sans détailler les comportements individuels de composants du système (utiliser alors les diagrammes d'états ou ceux d'activités).

On trouvera en figure 8 un récapitulatif des syntaxes utilisées dans les diagrammes de séquences. En particulier, on trouve 4 types de messages :

- synchrones : l'expéditeur attend une réponse ;
- asynchrones : l'expéditeur n'attend pas de réponse ;
- des réponses ;
- des messages réflexifs (interactions internes).

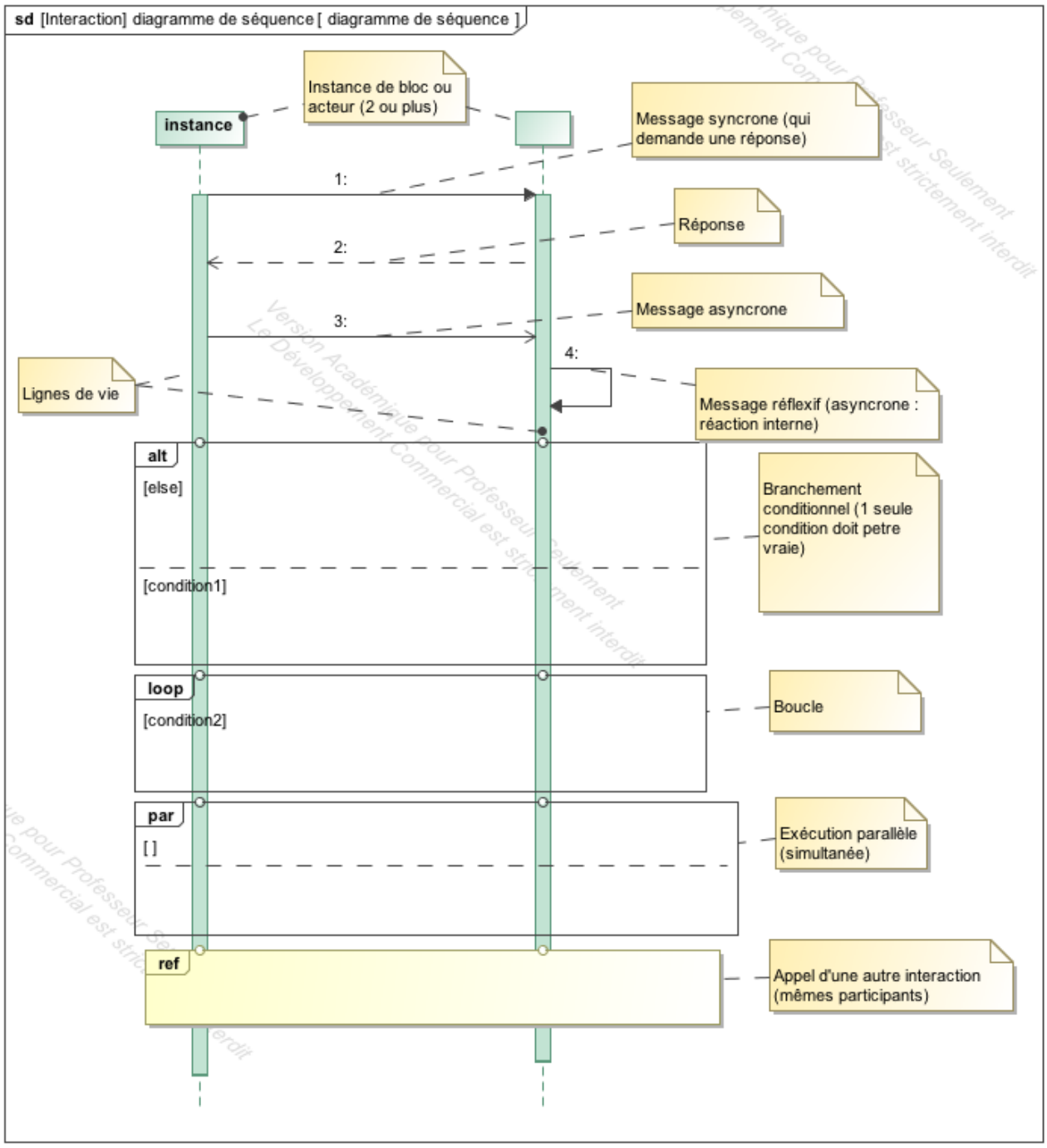



FIGURE 8 – Récapitulatif des syntaxes usuellement utilisées dans les diagrammes de séquences.

## 5.2 Diagramme d'états (*STM – SysML State Machine Diagram*)

### 5.2.1 Description

Le diagramme d'états/transitions permet de décrire les différents états pris par un bloc (le système, un sous-système ou un composant défini dans un diagramme BDD de définition de blocs) en fonction des événements qui lui arrivent (voir figures 24, 25 et 26 pour

l'exemple du robot DARwIn-OP).

Ce diagramme ne permet pas de décrire un algorigramme. On l'utilisera essentiellement pour décrire par exemple les modes de marche et d'arrêt d'un système (ou sous-ensemble) : mode normal, mode dégradé, arrêt normal, arrêt d'urgence.

Un tel modèle de **machine à nombre fini d'états**, aussi appelé automate fini (*FSM : finite state machine*) permet l'implémentation logicielle générant les différents diagrammes de séquence.

On trouvera en figure 9 un récapitulatif des syntaxes utilisées dans les diagrammes d'états - transitions.

### 5.2.2 État

Un **état** représente une situation d'une durée finie durant laquelle un système exécute une activité (une ou plusieurs actions), satisfait à une certaine condition ou bien est en attente d'un événement. Le passage d'un état à un autre se fait en franchissant une **transition**, activée par une condition logique ou un évènement.

Tout diagramme d'états comporte **un état initial** (unique), qui correspond à la création de l'instance du bloc pour lequel le diagramme d'état est spécifié. Un **état final** (il peut y en avoir un ou plusieurs) correspond à la destruction de cette instance de bloc.

### 5.2.3 Activité, action

Il est possible de rajouter des événements internes afin de montrer la réponse à un événement sans changer d'état. Les événements *entry*, *do* et *exit* peuvent indiquer ce qu'il se passe à l'entrée dans l'état (mot clé *entry*), pendant l'état (*do*) et à la sortie de l'état (*exit*).

### 5.2.4 Évènements, transitions

Les transitions régissent les changements d'état. Elles sont franchies lorsque :
- un **événement déclencheur** se produit ;
- qu'une **condition de garde** est vraie (expression booléenne) ;
- le franchissement déclenche alors une **action**.

La notation est alors la suivante : "événement[condition]/action".

Il existe quatre types d'évènements associés à une transition :
- le **message** (signal event) : un message asynchrone est arrivé ;
- l'**évènement temporel** (time event) : un intervalle de temps s'est écoulé depuis l'entrée dans un état (mot clé « after ») ou un temps absolu a été atteint (mot clé « at ») ;
- l'**évènement de changement** (change event) : une valeur a changé de telle sorte que la transition est franchie (mot clé « when ») ;
- l'**évènement d'appel** (call event) : une requête de fonction (operation) du bloc a été effectuée. Un retour est attendu. Des arguments (paramètres) de fonction peuvent être nécessaires.

### 5.2.5 État composé *(aussi traduit par : état composite)*

Un **état composé** contient un autre automate, avec un état initial (activé à l'entrée dans l'état) et sans état final. Cela permet d'introduire une hiérarchisation d'états.

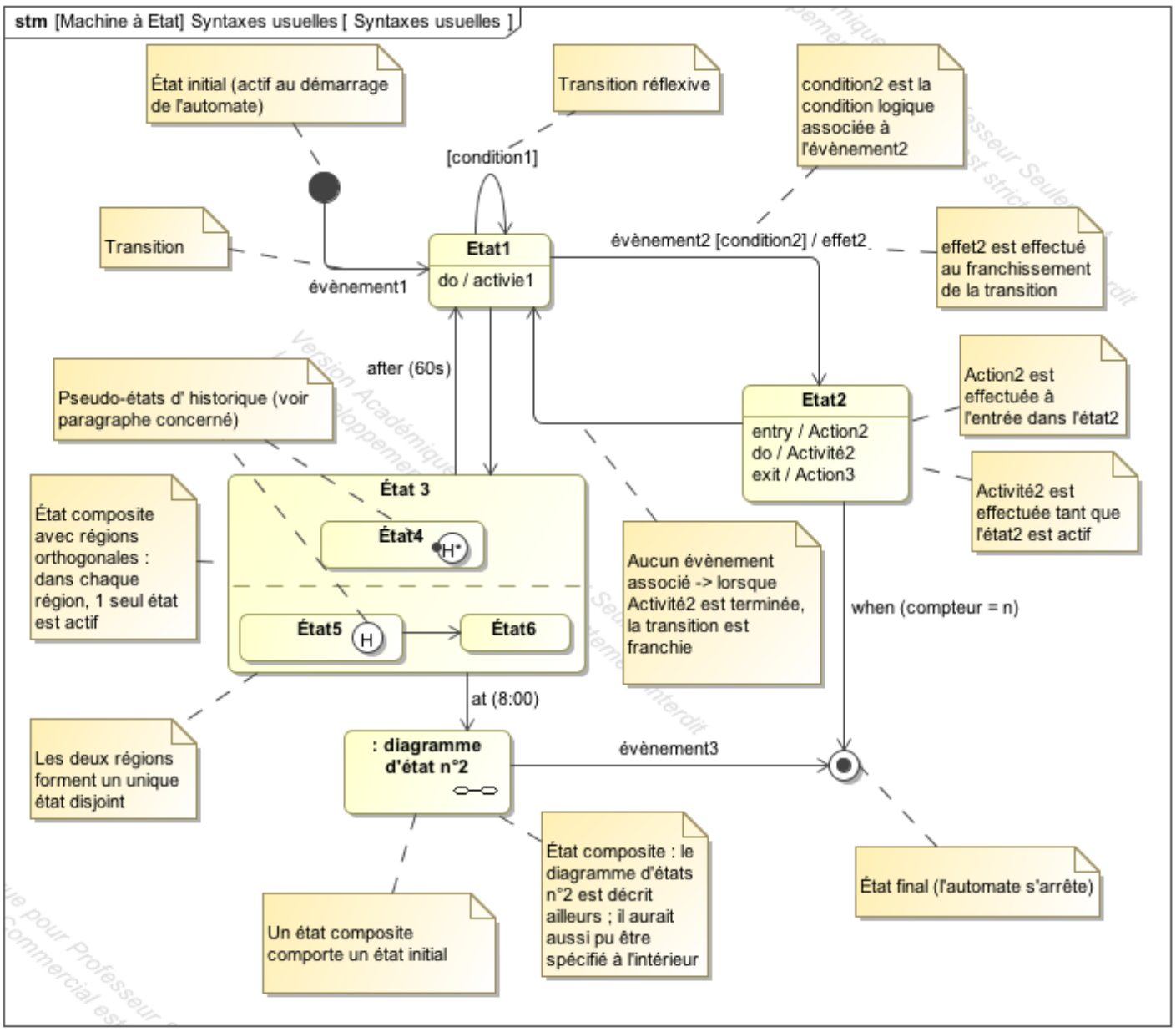



FIGURE 9 – Récapitulatif des syntaxes usuellement utilisées dans les diagrammes d'états - transitions.

## 5.3 Diagramme d'activités (*ACT – SysML Activity Diagram*)

### 5.3.1 Description

Le diagramme d'activités (voir figure 27 pour l'exemple du robot DARwIn-OP) décrit :

- L'enchaînement des **actions** (transformation de flux) pour une activité du système.
- Les **flux** transformés par ces actions.
- L'ordre et les **conditions d'exécution** de ces actions.

Il permet de représenter le déroulement d'un processus sous la forme d'une activité correspondant à une décomposition séquentielle d'actions, appelées **tâches**.

Il est par exemple fort utile pour détailler ce qu'il se passe dans un état d'un diagramme d'états.

Dans sa forme la plus restreinte, ce diagramme représente un *algorigramme*, c'est-à-dire un flux de contrôle (attention : ce flux n'a rien à voir avec ceux présents dans le diagramme de blocs internes : il ne faut donc pas les confondre).

Contrairement au diagramme d'états, il n'est pas rattaché à un seul bloc de l'ibd.

### 5.3.2 Tâches

Les tâches peuvent être reliées aux composants (définis dans un diagramme de définition de blocs) qui les exécutent. Une tâche :

- Transforme un flux d'entrée (matière, énergie, information) en un flux de sortie.
- Démarre lorsque les entrées nécessaires sont présentes.
- S'arrête une fois qu'elle a produit les sorties demandées.

Contrairement au diagramme d'états, il n'existe aucun évènement associé aux transitions entre actions (la fin d'une action implique automatiquement le passage à la suivante).

*Rappel : ce diagramme n'est pas explicitement au programme.*

### 5.3.3 Éléments de syntaxe

On trouvera en figure 10 un récapitulatif des syntaxes les plus courantes utilisées dans les diagrammes d'activités. En particulier :

- Une macro-action (représentée par un "rateau" dans le coin inférieur droit, voir "Action2") représente une autre activité, définie dans un autre diagramme.
- Il peut exister un **flux de contrôle**, qui impose des conditions supplémentaires de démarrage des actions, en plus du flux d'objet (matière, énergie, information). Les sorties de contrôles activent alors une fois l'action terminée. Les flux de contrôle permettent d'imposer un ordre d'exécution des actions.
- Pour coupler des comportements, on peut utiliser des **signaux**, attendus ou envoyés.
- Les possibilités de routage de flux se font par des pseudo-états (voir partie 5.4) : la **bifurcation**, la **décision**, la **jonction** et le **mélange**.

Pour affecter les comportements (ou flux fonctionnels) aux composants, on peut utiliser des **partitions** (*swimlanes*) : voir le diagramme d'activités du robot DARwIN-OP en figure 27.

## 5.4 Remarque sur les pseudo-états

Les diagrammes SysML d'états et d'activités peuvent comporter des pseudo-états, qui sont des éléments de commande qui influencent le comportement d'une machine d'états.

Les principaux pseudo-états utilisés dans le cadre du programme de CPGE sont (voir figures 10) :

- Bifurcation *("fork")* et jonction *("join")* : divergence et convergence de séquences parallèles,
- Décision *("choice")* : sélection et convergence de séquences exclusives. Il est nécessaire qu'**une** (seule) condition située en aval soit vraie pour que l'évolution du système se poursuive. Les conditions de gardes doivent être exclusives : «else» peut alors être utilisé pour englober tout ce qui n'est pas décrit dans une ou des expressions booléennes.

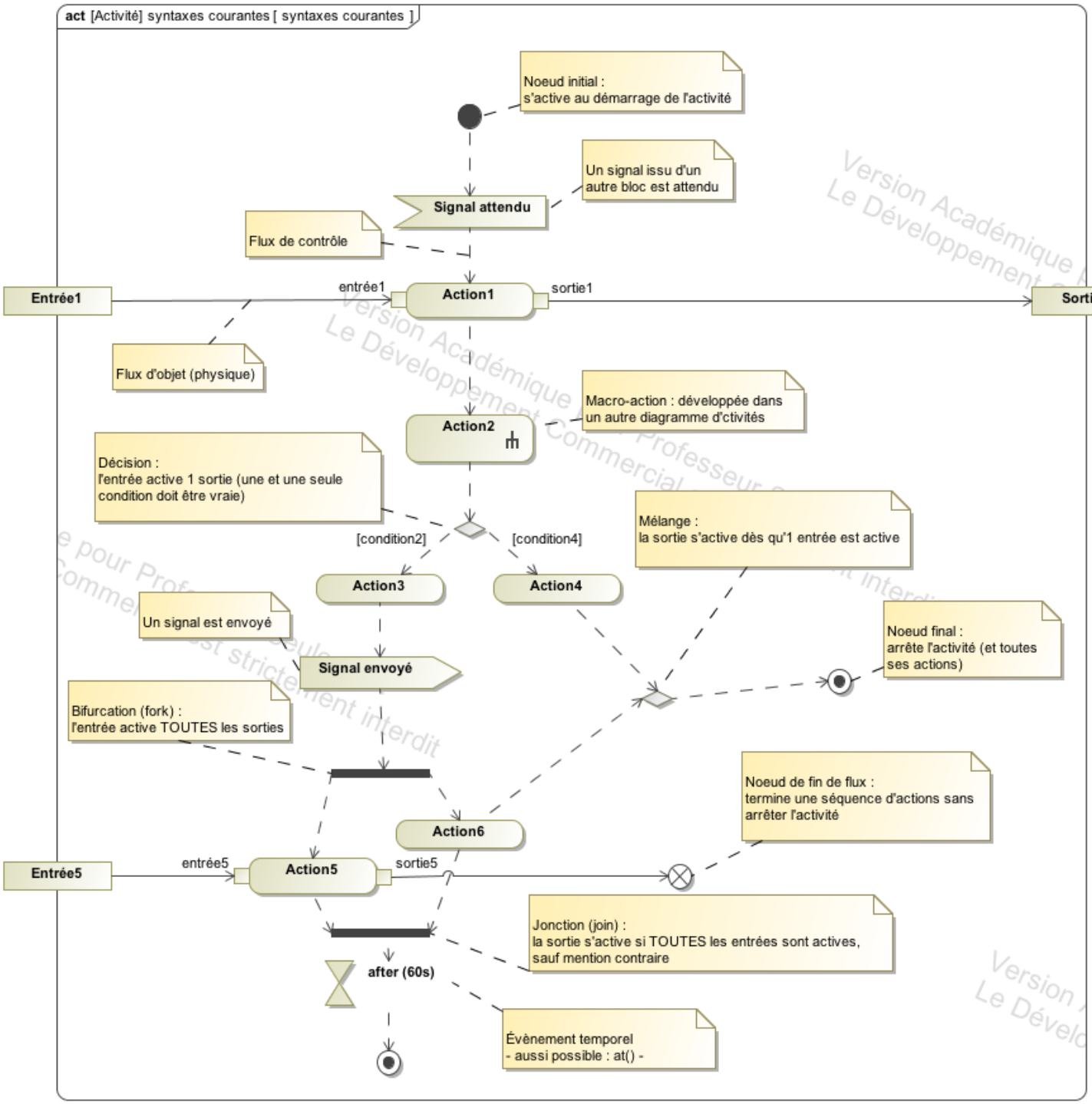



FIGURE 10 – Récapitulatif des syntaxes utilisées dans les diagrammes d'activités.

- Mélange *("junction")* : pour qu'un chemin soit emprunté, **toutes** les conditions de garde situées en aval et en amont, doivent être vraies.
- *H «shallow history»*) : permet à un état de niveau hiérarchique supérieur (état composite) de se souvenir du dernier sous-état, avant qu'il n'évolue vers un autre état.
- *H** (*«deep history»*) : idem mais avec la propagation de l'historique à tous les sous-états composites de niveaux hiérarchiques inférieurs.

## 5.5 Structures algorithmiques

### 5.5.1 Définitions

#### Algorithme et algorigramme

Parfois la description du comportement d'un système à évènement discret s'effectue facilement à l'aide d'un **algorigramme**, qui consiste en une représentation graphique d'un **algorithme** (i.e. de l'enchaînement des opérations et des décisions d'un système ou programme).

Les principales structures algorithmiques sont définies ci-après. *Se reporter au cours d'informatique pour plus de précisions.*

#### Affectation

L'affectation consiste à donner une valeur à une variable.

#### Fonctions, procédures

La décomposition d'un algorithme en fonctions et procédures permet :
- de scinder une problématique générale (ou un programme) en plusieurs problématiques élémentaires (ou sous-programmes) ;
- de pouvoir utiliser (ou appeler) plusieurs fois ces sous-programmes réalisant des tâches élémentaires (sans avoir à les ré)écrire ou recopier : gain d'espace mémoire) ;

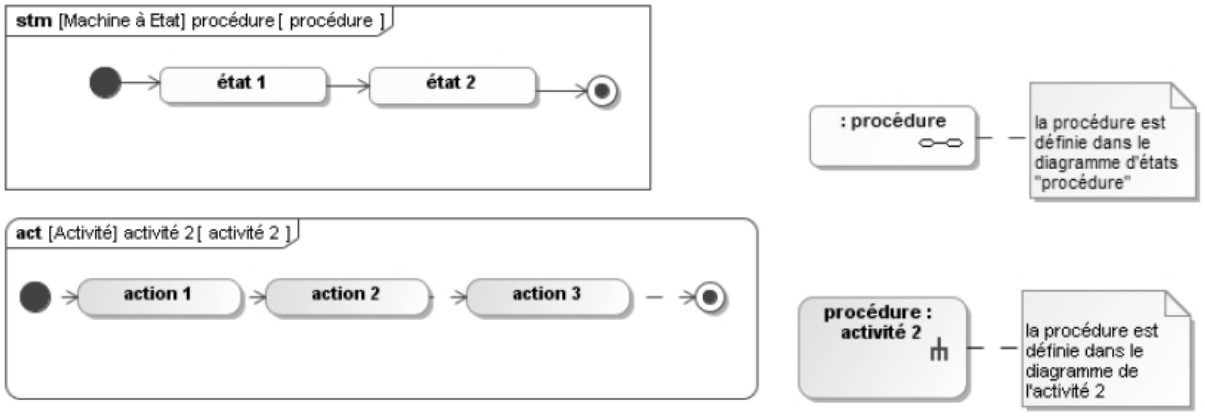



FIGURE 11 – Utilisations de diagrammes SysML d'activités ou d'états-transitions pour décrire une procédure.

Une procédure comporte une succession d'instructions mais ne renvoie rien. Il est nécessaire de l'appeler pour se faire dérouler le programme qu'elle décrit.

Une fonction retourne le contenu d'un objet (valeur, liste, ...).

*On peut par exemple décrire une procédure ou une fonction à l'aide d'un diagramme SysML d'activités ou à l'aide d'états composites d'un diagramme d'états-transitions (voir figures 11 et 12).*

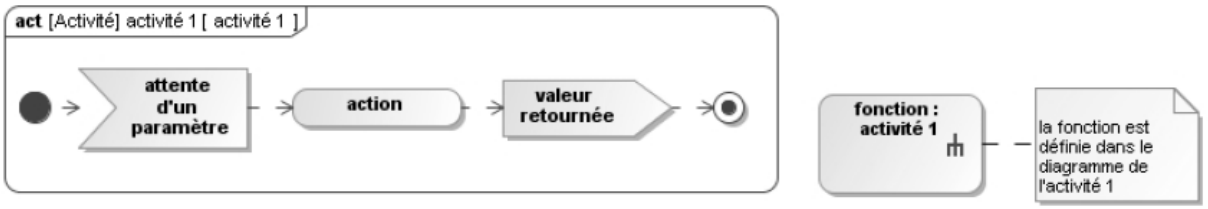



FIGURE 12 – Utilisations de diagrammes SysML d'activités ou d'états pour décrire une fonction.

### 5.5.2 Structure alternative

On trouvera en figure 13 une architecture alternative d'algorigramme.

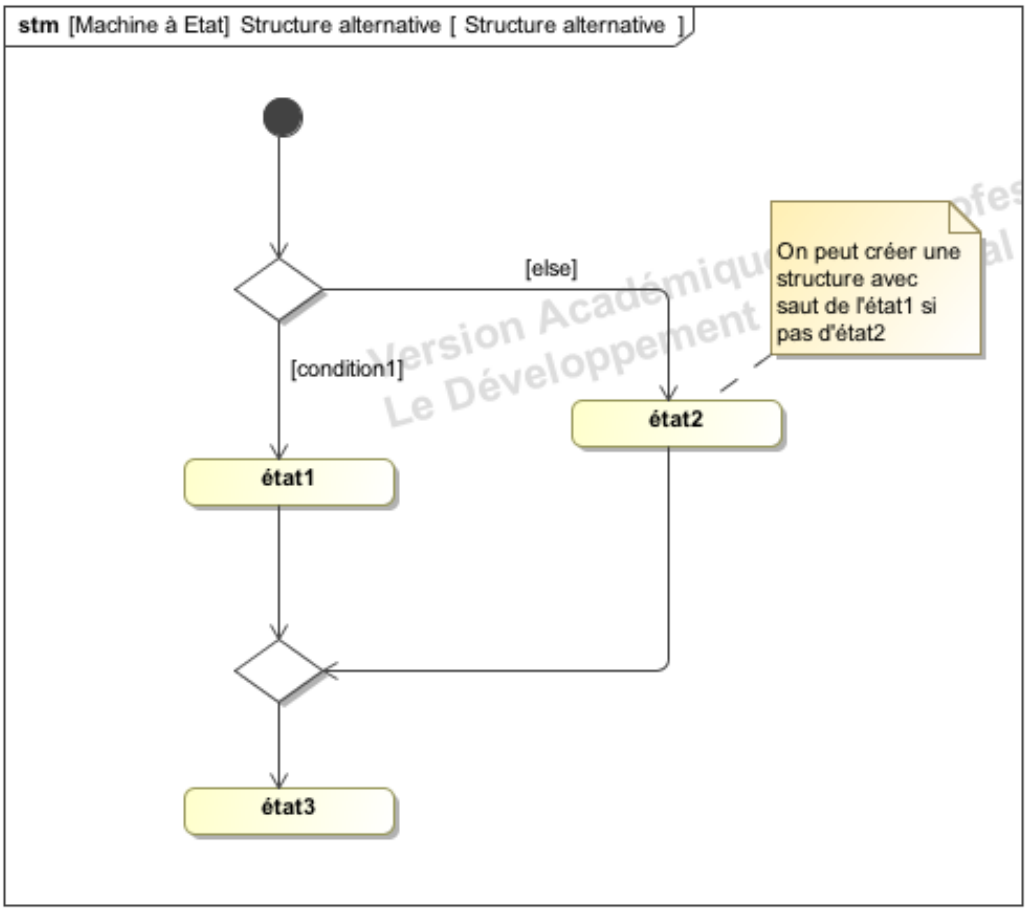



FIGURE 13 – Structure alternative d'algorigrammes.

### 5.5.3 Structures itératives

On trouvera en figure 14 différentes architectures itératives d'algorigrammes.

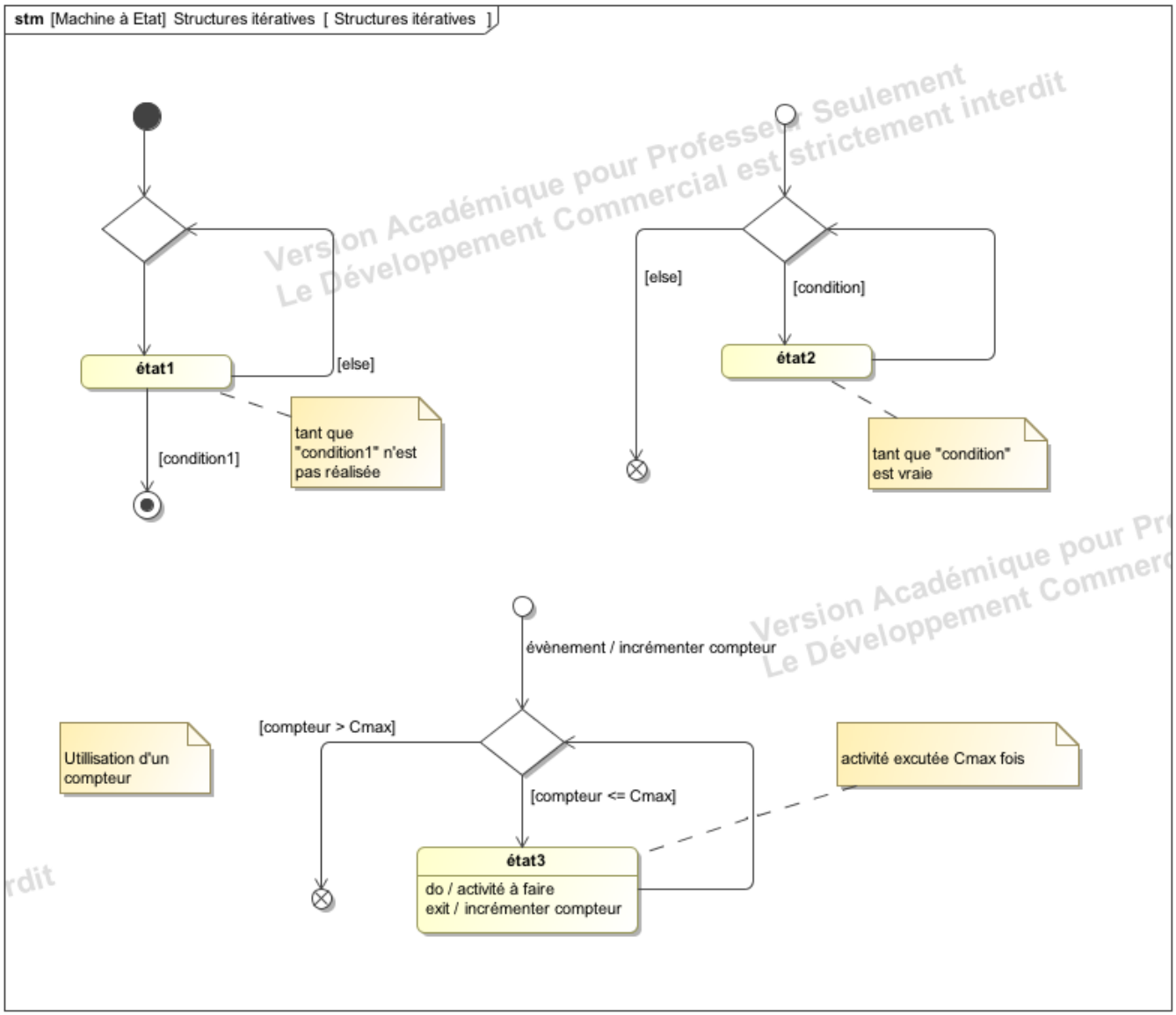



FIGURE 14 – Structures itératives d'algorigrammes.

# 6 Systèmes numériques de commande

## 6.1 Introduction

### 6.1.1 Liens entre automatique et informatique

Suite au développement des cartes comportant des micro-contrôleurs et à la numérisation des systèmes de commande, les domaines de l'automatique et de l'informatique connaissent actuellement de forts liens et rapprochements.

En effet, les algorithmes de commande étant de plus en plus implantés sur microprocesseur, leur efficacité nécessite des connaissances complémentaires issues du monde de l'automatique (par la connaissance du système logique et l'adaptation de son système de commande) et de l'informatique (par la qualité du code informatique traduisant le fonctionnement du système logique)

### 6.1.2 Numérisation : discrétisation du modèle temporel

L'implantation sur une cible numérique de la commande d'un système contrôlant l'évolution de grandeurs physiques nécessite une **numérisation** (valeurs discrètes et finies) et un **échantillonnage** (temps discret régulier) de ces grandeurs.

De plus les éventuelles équations différentielles du modèle à implanter sont **discrétisées** en temps (équations aux différences) et en valeur (paramètres quantifiés).

*Remarque : à notre niveau il s'agira essentiellement sur ce point de savoir avoir un esprit critique sur des paramètres de calcul, et le plus souvent savoir choisir un pas de temps de calcul (échantillonnage).*

## 6.2 Solutions technologiques actuelles de systèmes de commande

On présente ici quelques architectures matérielles de systèmes de commande. Les technologies évoluent et s'améliorent tous les jours.

*Remarque : certaines solutions techniques détaillées ici sont des produits souvent très accessibles voire grand public, qui pourront servir dans le cadre du TIPE.*

### 6.2.1 Réseau logique programmable

Un réseau logique programmable (*FPGA pour field-programmable gate array*, soit : réseau de portes programmables) est un circuit intégré logique qui peut être reprogrammé (plus précisément reconfiguré) après sa fabrication. Il est composé de nombreuses (plusieurs centaines de milliers) cellules logiques élémentaires et bascules logiques librement connectables (voir figure 15 [2]).

Les principaux intérêts à notre niveau sont :
- une exécution parallèle (donc très rapide),
- des possibilités multiples de programmation (même en cours de fonctionnement),
- la présence de périphériques analogiques.

### 6.2.2 Micro-contrôleur

**Présentation et intérêt**

Un micro-contrôleur est un circuit intégré comportant les éléments essentiels d'un ordinateur : processeur, mémoires (morte pour le programme, vive pour les données), unités périphériques et interfaces d'entrées-sorties (voir figure 16). À la différence d'un FPGA, il ne peut être reconfiguré. Un bus de données permet de faire circuler les informations entre les éléments le composant.

Les micro-contrôleurs se caractérisent par un plus haut degré d'intégration, une plus faible consommation électrique, une vitesse de fonctionnement plus faible (de quelques MHz à quelques GHz) et un coût réduit par rapport aux micro-processeurs présents dans les ordinateurs de bureau. C'est leur développement qui a permis de démocratiser l'utilisation de l'informatique (applications industrielles ou en domotique et robotique). On en trouve souvent plusieurs dans de nombreux objets de la vie courante : jouets, boite internet, appareil photo, téléphone ...

Les principaux intérêts à notre niveau sont :

2. Extrait du livre S.I.I. MPSI, PCSI, PTSI ; Caignot A., Crespel V., Dérumaux M., Garreau C., Kaszynski P., Martin B., Roux S. ; Éditions Vuibert Prépas. Un grand Merci aux auteurs.

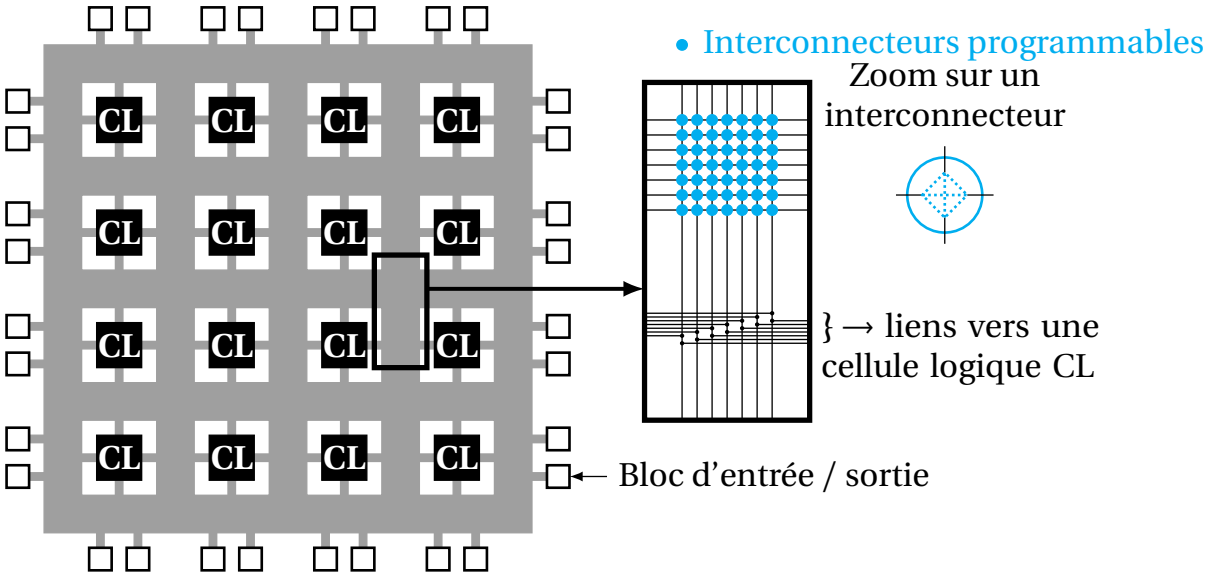



FIGURE 15 – Schématisation d'un réseau de cellules logiques (CL) à interconnections programmables .

- une structure logique organisée pour traiter l'information séquentiellement,
- un MCU est équivalent à une carte mère complète,
- la présence de nombreux périphériques matériels,
- l'autonomie : programme contenu dans la mémoire morte peut être exécuté dès la mise sous tension.

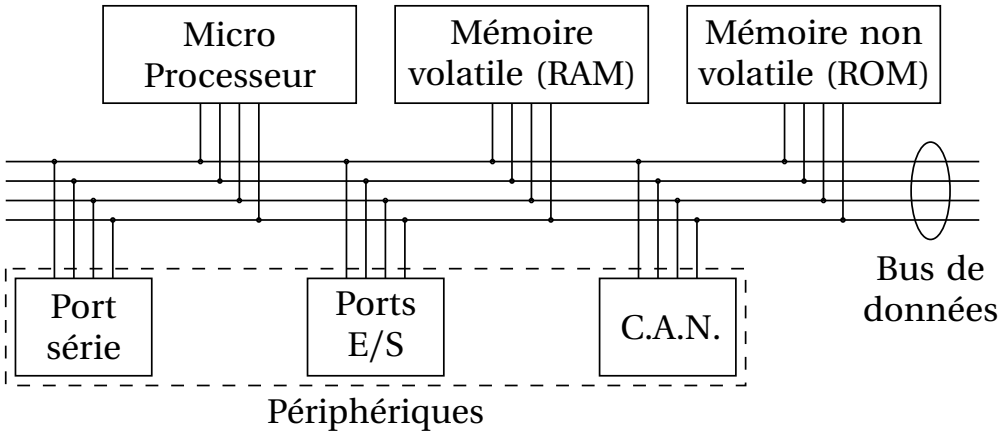



FIGURE 16 – Schématisation d'un micro-contrôleur.

L'utilisation d'un micro-contrôleur s'impose par rapport à une solution logique câblée (voir partie 3) dès lors que la complexité ou le nombre de fonctions à réaliser sont importants.

**Cartes types Arduino et Raspberry**

Les cartes à micro-contrôleur du type des Arduino Uno ou Raspberry Pi sont des produits prêt à l'emploi faciles à prendre en mains (voir les travaux pratiques du Centre d'Intérêt CI6) et peu onéreux. Ces cartes sont organisées autour d'un micro-contrôleur et possèdent l'électronique de commande nécessaire, ainsi qu'un certain nombre (variable selon

les cartes) de broches d'entrées sorties (numériques, analogiques, PWM). Elles sont programmables (souvent en C ou C++, ou par interface graphique) par l'intermédiaire d'une liaison série.

**Cas du robot DARwIn-OP**

La carte contrôleur secondaire CM-730 (voir figure 17) a pour fonction de contrôler les capteurs et actionneurs du robot.

Cette carte embarque une unité de traitement équipé d'un micro-contrôleur ARM Cortex M3 à architecture RISC (*Reduced Instruction Set Computer*, architecture qui offre peu d'instructions mais un temps d'exécution défini et constant) et l'électronique spécifique du robot. Elle intègre le gyromètre, l'accéléromètre, les entrées/sorties standards, les interfaces (Led, haut-parleur, micro), sans oublier la gestion des 20 servomoteurs Dynamixel MX-28.

En étroite collaboration avec la carte PC embarquée (voir partie 6.2.4) par l'intermédiaire d'un bus USB à 2 Mbps (Méga bits par seconde), elle contrôle les capteurs et actionneurs du robot.

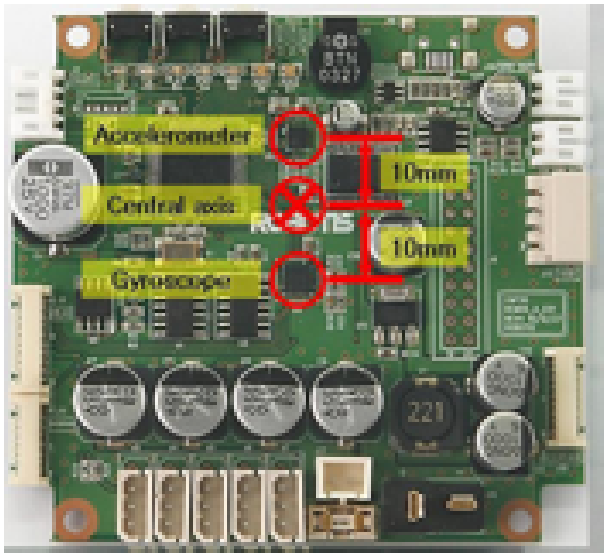



FIGURE 17 – Photo de la carte contrôleur secondaire CM-730 du robot DARwIn-OP.

### 6.2.3 Processeur de signal numérique (DSP pour *Digital Signal Processor*)

C'est un micro-processeur optimisé et spécialisé dans certaines taches de traitement du signal, de calcul ou de contrôle-commande le plus rapidement possible. On le trouve fréquemment dans les modem, téléphones mobiles, lecteurs MP3, GPS. Du fait de sa spécialisation, un DSP est plus rapide qu'un micro-contrôleur classique mais moins généraliste.

### 6.2.4 Ordinateur embarqué et architecture distribuée

**Présentation et intérêt**

Un ordinateur embarqué est une carte constituée d'un micro-contrôleur puissant (32 ou 64 bits, beaucoup de mémoire, périphériques de haut niveaux) permettant d'installer un OS ainsi que de nombreuses interfaces, parmi lesquelles :

– Ethernet, USB, UART, I2C, SPI ;

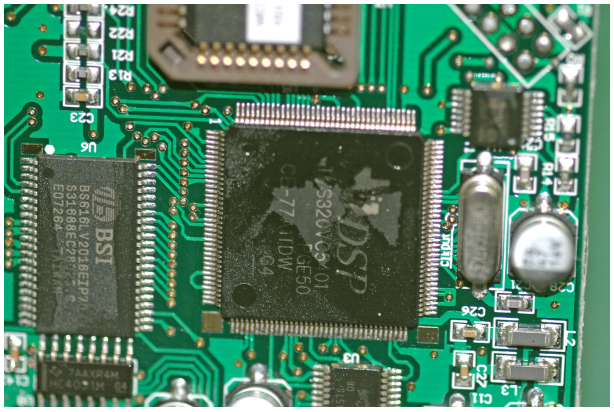



FIGURE 18 – Photo d'un processeur de signal numérique (DSP).

– HDMI (vidéo), son, mémoire flash ;
– la possibilité de brancher un écran et un clavier.

Ces cartes sont en général dédiées à la commande globale du système, et associés à des micro-contrôleurs pour les taches rapides ou précises (pilotage d'actionneur, traitement de données issues de capteurs...). Le tout est alors relié par des bus de données. On parle alors d'*architecture distribuée*.

Un tel type de carte répond au exigences des systèmes embarqués, à savoir des fonctionnalités très différentes de nos PC standards, parmi les plus importantes :

– des dimensions et un poids réduits ;
– une consommation très faible pour permettre un fonctionnement sur batterie et une autonomie optimisée ;
– l'absence de pièces mécaniques en mouvement (disque dur, lecteur de DVD) ;
– la nécessité de fonctionner sous un système d'exploitation (Linux, Windows) et de pouvoir embarquer des logiciels standard du marché (traitement d'image, serveur Web...) ;
– la nécessité d'intégrer des interfaces pour communiquer avec le monde extérieur.

**Cas du robot DARwIn-OP**

Le contrôleur principal (SBC-FITPC2i, voir figure 19) du robot humanoïde DARwIn-OP est un ordinateur mono carte spécialement conçu pour les systèmes embarqués. C'est la carte principale (présence d'une carte micro-contrôleur secondaire) qui gère le fonctionnement global du robot. On trouvera en Annexe D les diagrammes SysML de définition de blocs (figure 28) et de blocs internes (figure 29) de ce contrôleur principal.

Le choix de cette carte n'est pas innocent : spécialement étudiée pour optimiser les ressources sur les systèmes embarqués (poids, consommation, coût), elle répond aux besoins particuliers du robot, supporte un système d'exploitation Linux (open source) permettant d'intégrer des logiciels standard du marché pour la vision de Darwin, et un serveur Web pour communiquer avec le monde extérieur. Elle permet par ce choix judicieux d'éviter l'étude et la fabrication d'une carte par la société et de réduire le coût de développement et par voie de conséquence celui du produit fini.

### 6.2.5 Automate programmable industriel (API)

Un automate programmable industriel est un dispositif électronique programmable destiné à automatiser des processus (commande de machines ou robots).

FIGURE 19 – Photo de l'ordinateur embarqué du robot DARwIn-OP (carte SBC-FITPC2i).

Il gère selon une suite logique le déroulement ordonné des opérations à réaliser. Il reçoit les consignes et les informations en provenance des capteurs, le tout est ensuite traitées par un programme défini afin de commander les pré-actionneurs et actionneurs. Le cycle de traitement est toujours le même et nécessite la maîtrise de langages spécifiques (comme le grafcet).

Ce dispositif a l'avantage d'être composé d'éléments particulièrement robustes et possède d'énormes capacités d'exploitation. En contrepartie, il est beaucoup plus cher que des solutions informatiques classiques à base de micro-contrôleurs, qui tendent à se généraliser.

On en trouve dans les armoires électriques de certains systèmes du laboratoire : ouvre portail, tireuse de pellicules, pompe doseuse.

## 6.3 Bases de programmation pour les systèmes de commande à base de micro-contrôleur

### 6.3.1 Programmation, compilation, transfert

Un micro-processeur ne comprend que la langage machine, c'est à dire du code en binaire, car il n'exécute que les instructions cablées dans son UAL (unité arithmétique et logique). Programmer dans ce langage (en *assembleur*) est possible mais fastidieux. On programme plus souvent en langage de plus haut niveau, comme Python ou, plus souvent, C, ce qui nécessite une étape (la *compilation*) de traduction en langage machine avant de pouvoir le transférer de l'ordinateur à la mémoire du micro-contrôleur.

### 6.3.2 Exécution

L'exécution du programme est cadencée par le signal d'horloge du micro-processeur (quelques dizaine de MHz). Les calculs sont effectués par l'*UAL*, qui communique avec des zones de mémoire : les *registres* (voir figure 20). Ce sont ces registres qui forment l'interface entre le logiciel et les matériels ; on peut les manipuler directement en programmant en C,

ou, plus simplement, en utilisant des bibliothèques *(voir les TP du CI6 sur cartes Arduino)*.

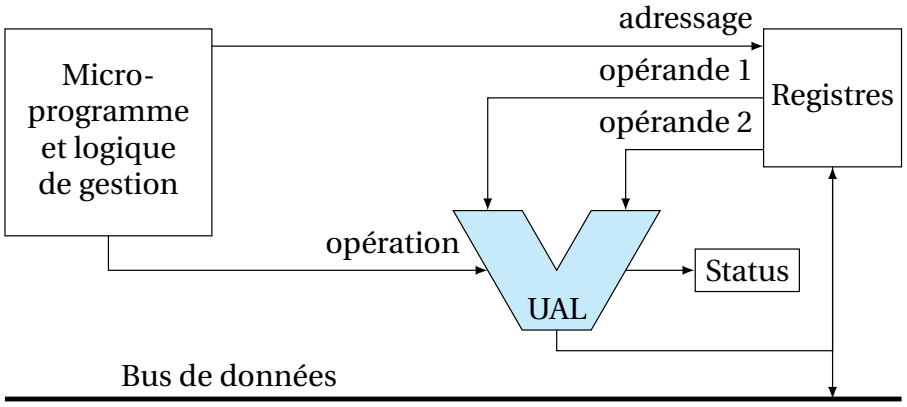



FIGURE 20 – Schématisation du fonctionnement d'un micro-processeur.

### 6.3.3 Démarche de conception d'un système de contrôle ou commande

La démarche de conception d'un système de commande peut se décomposer en différentes étapes :
- Définition des exigences par rédaction du cahier des charges ;
- Formalisation : décliner les exigences du cahier des charges en diagrammes SysML d'activités ou d'états ;
- Implémentation : traduire les diagrammes SysML d'activités ou d'états en programme (souvent en code C ou par interface graphique) ;
- Compilation et chargement ;
- Validation par mise en œuvre du programme proposé, mesure des performances réelles et comparaison aux performances attendues.

### 6.3.4 Compétences attendues aux concours

*Les compétences attendues aux concours, spécifiques au systèmes numérique de contrôle ou commande, consistent simplement à :*
- *Traduire un cahier des charges simple en un programme (ou une modification d'un programme donné) à charger dans un micro-contrôleur ;*
- *Valider que le comportement réel programmé satisfait les exigences.*

### 6.3.5 Correspondances Scilab – Python – C

On trouvera ci-dessous un tableau de correspondance des opérations logiques de bases dans les langages C, Python et Scilab.

| Opération | Langage C | Langage Python | Langage Scilab |
|---|---|---|---|
| NON | ! | not | ~ |
| OU | \|\| | or | \| |
| ET | && | and | & |
| ÉGALITÉ | == | == | == |

Remarque : l'opérateur "différent" s'écrit en Scilab : ~=.

# A Diagrammes SysML de séquence du robot DARwIn-OP

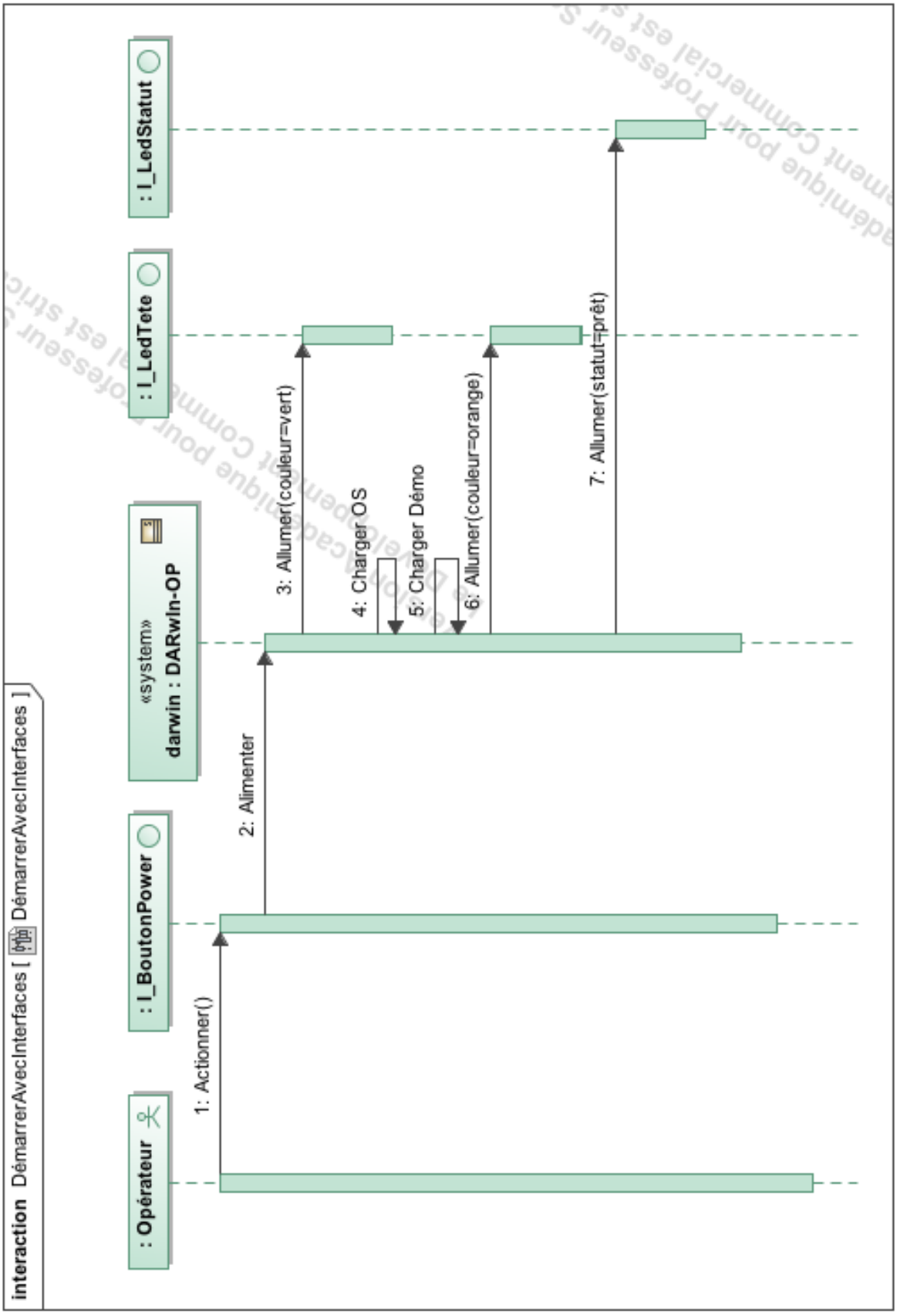



FIGURE 21 – Diagramme SysML de séquence du robot DARwIn-OP pour le cas d'utilisation "démarrer").

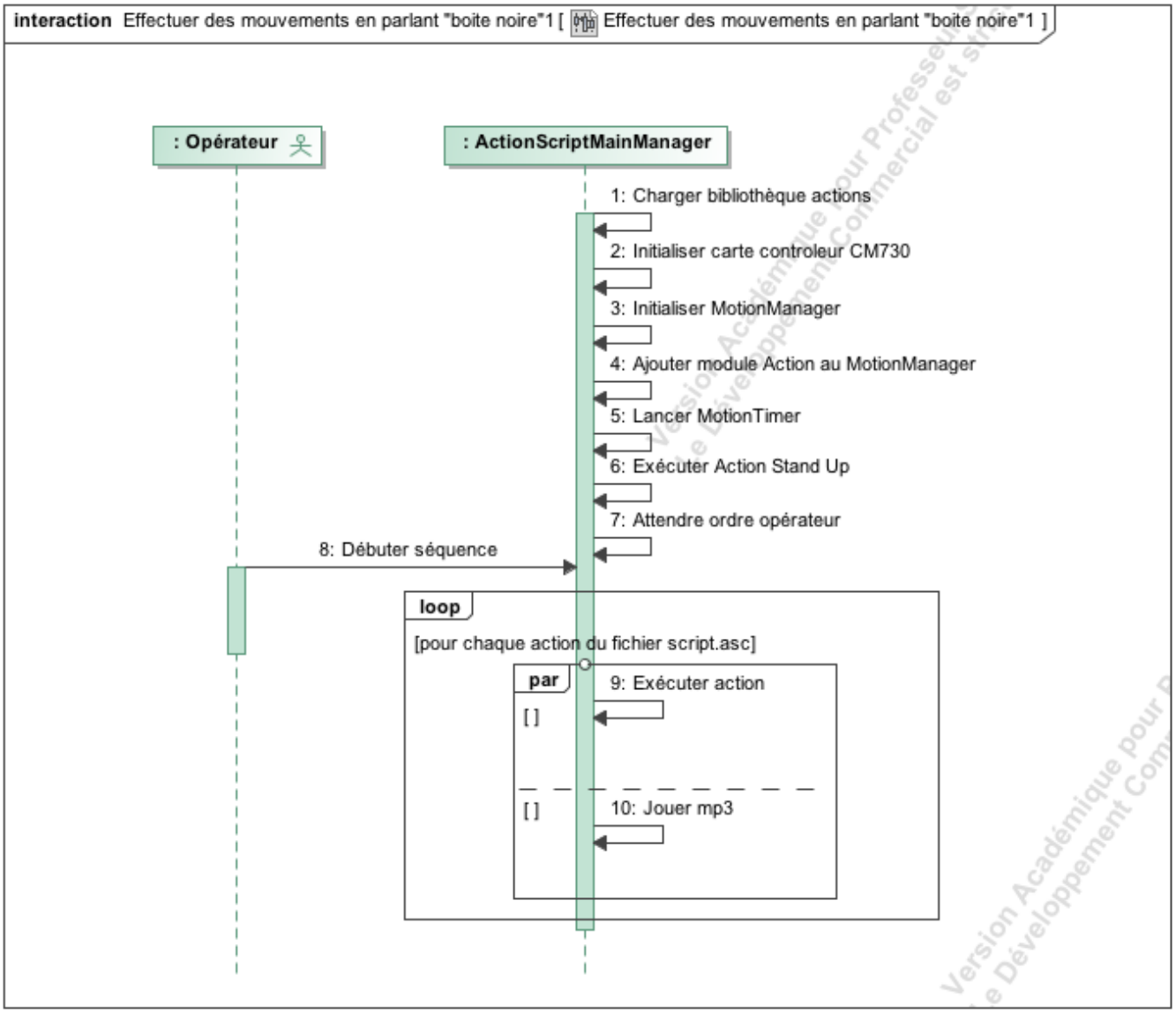



FIGURE 22 – Diagramme SysML de séquence du robot DARwIn-OP pour le cas d'utilisation "effectuer des mouvements en parlant").

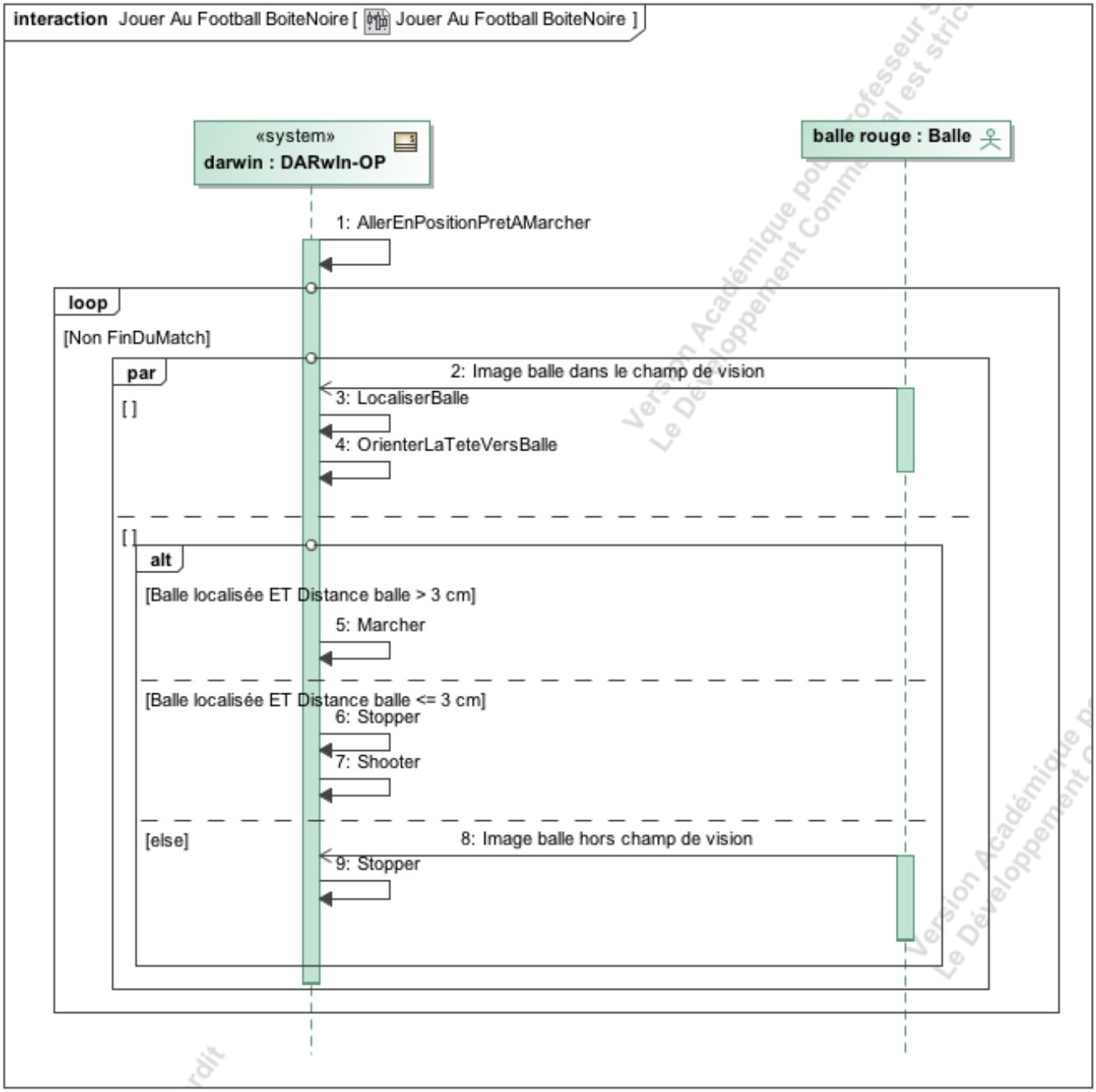



FIGURE 23 – Diagramme SysML de séquence du robot DARwIn-OP pour le cas d'utilisation "jouer au football").

# B Diagrammes SysML d'états du robot DARwIn-OP

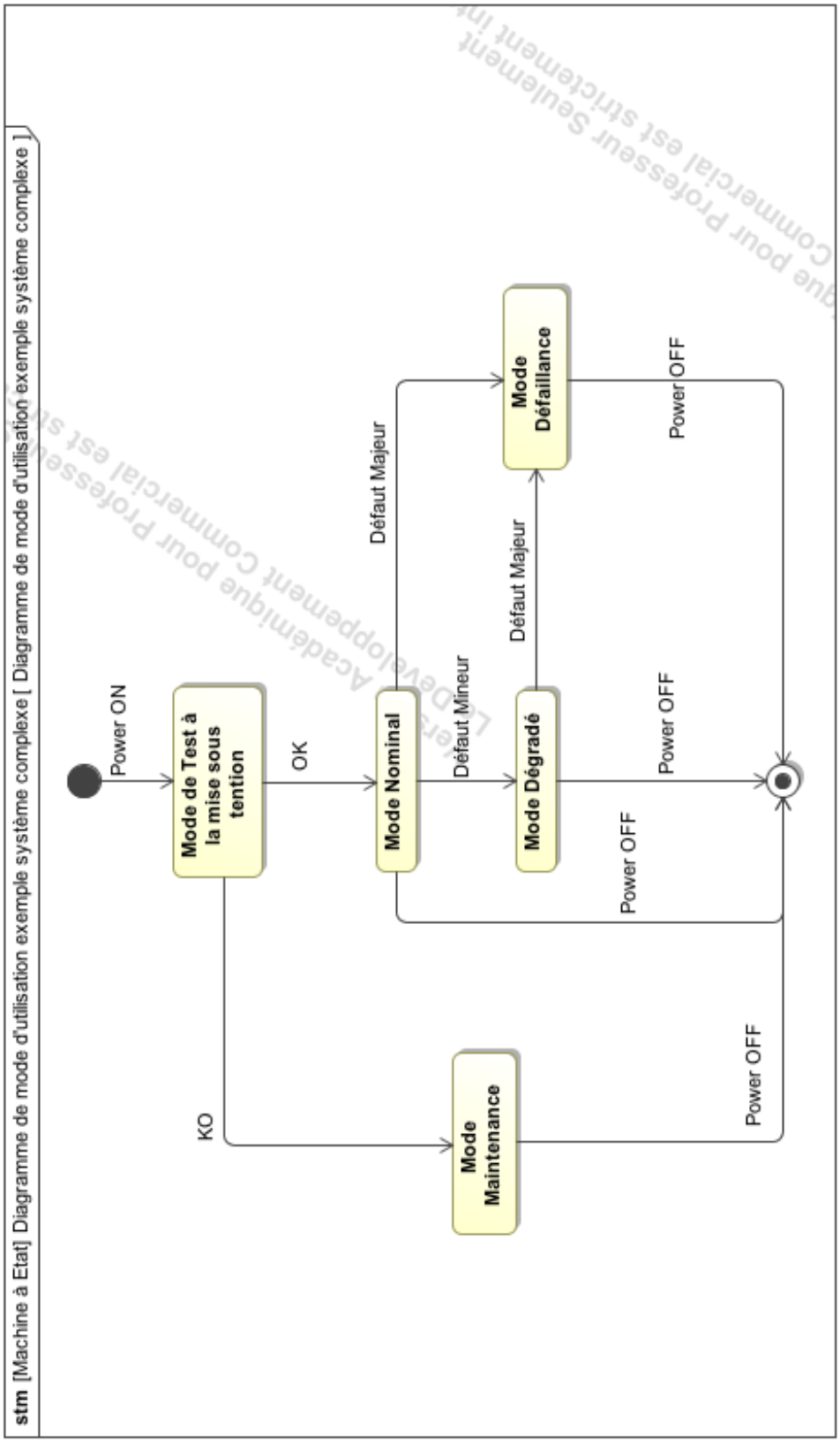



FIGURE 24 – Diagramme SysML d'états des modes d'utilisation du robot DARwIn-OP.

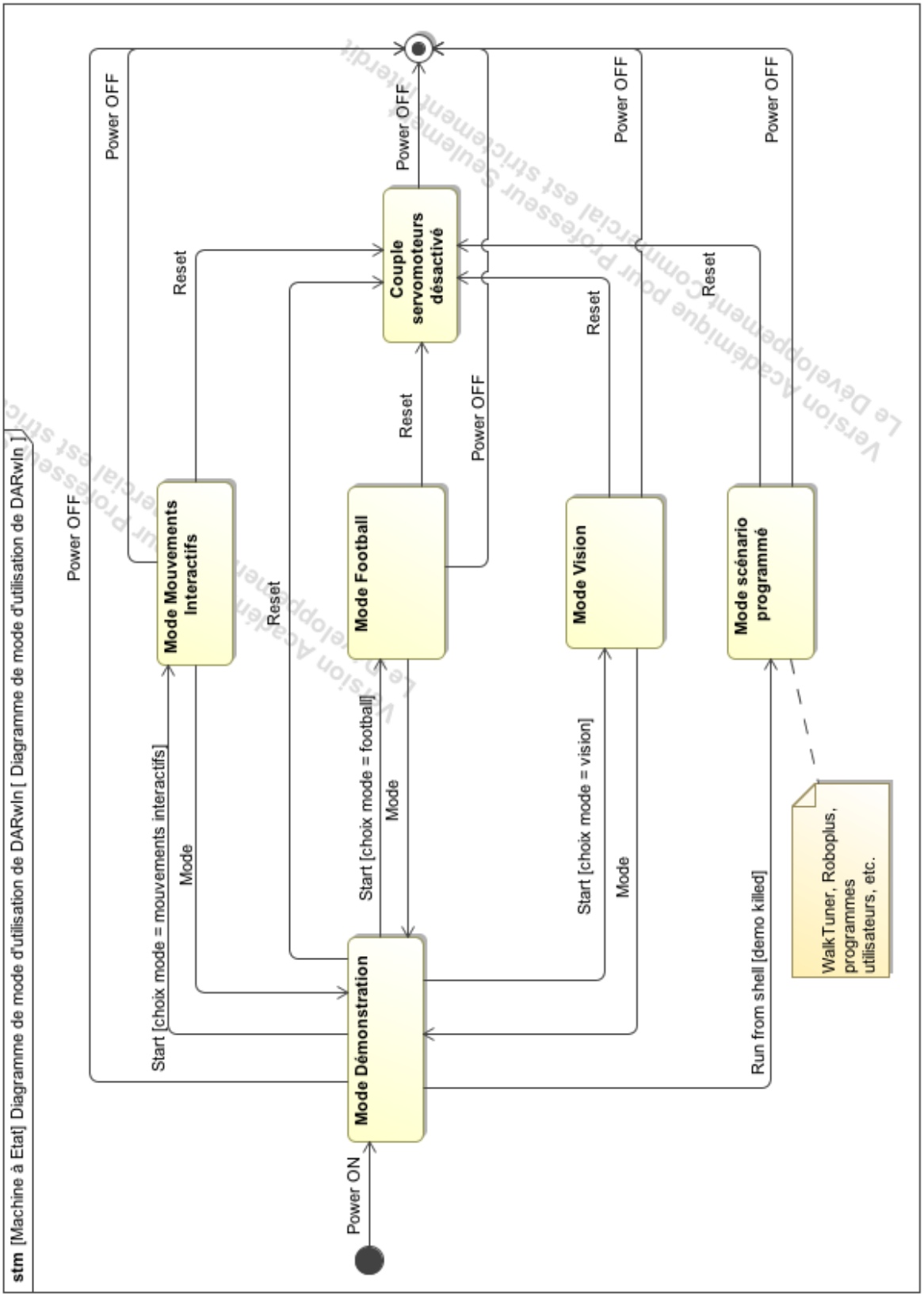



FIGURE 25 – Diagramme SysML d'états des modes de démonstration du robot DARwIn-OP.

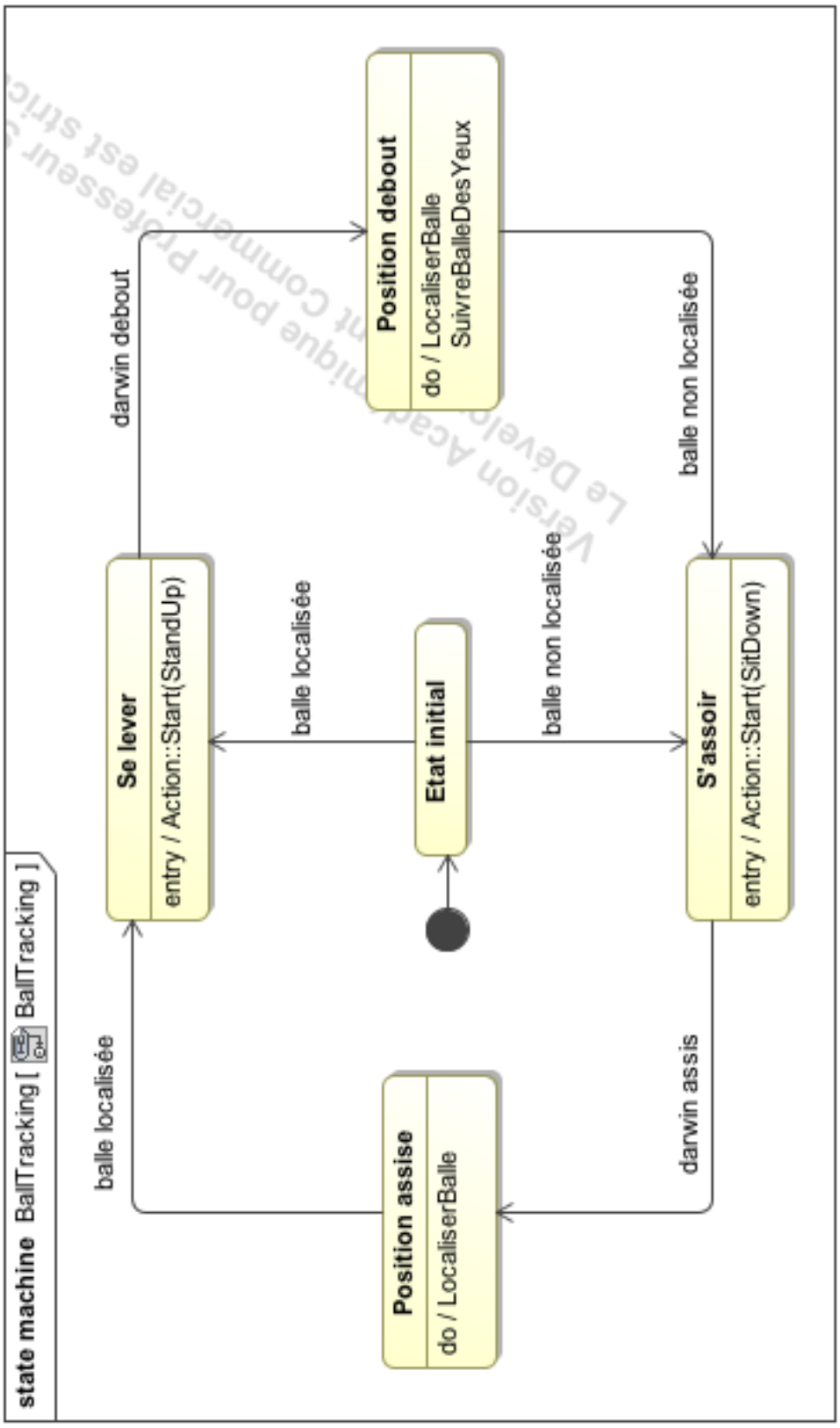



FIGURE 26 – Diagramme SysML d'états de la recherche de balle du robot DARwIn-OP.

# C Diagramme SysML d'activités du robot DARwIn-OP

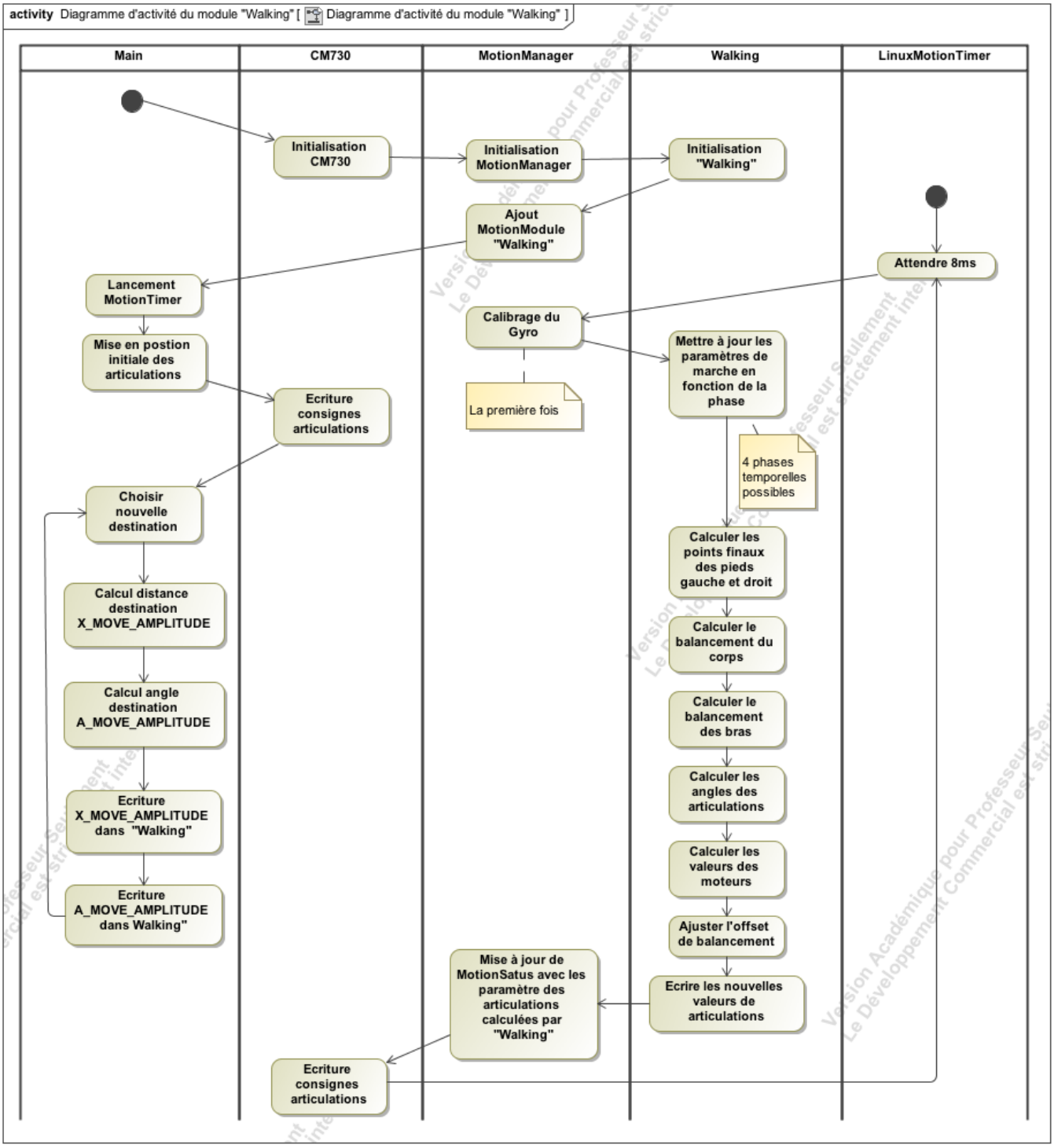



FIGURE 27 – Diagramme SysML d'activités du robot DARwIn-OP.

# D Diagrammes SysML du contrôleur principal du DARwIn-OP

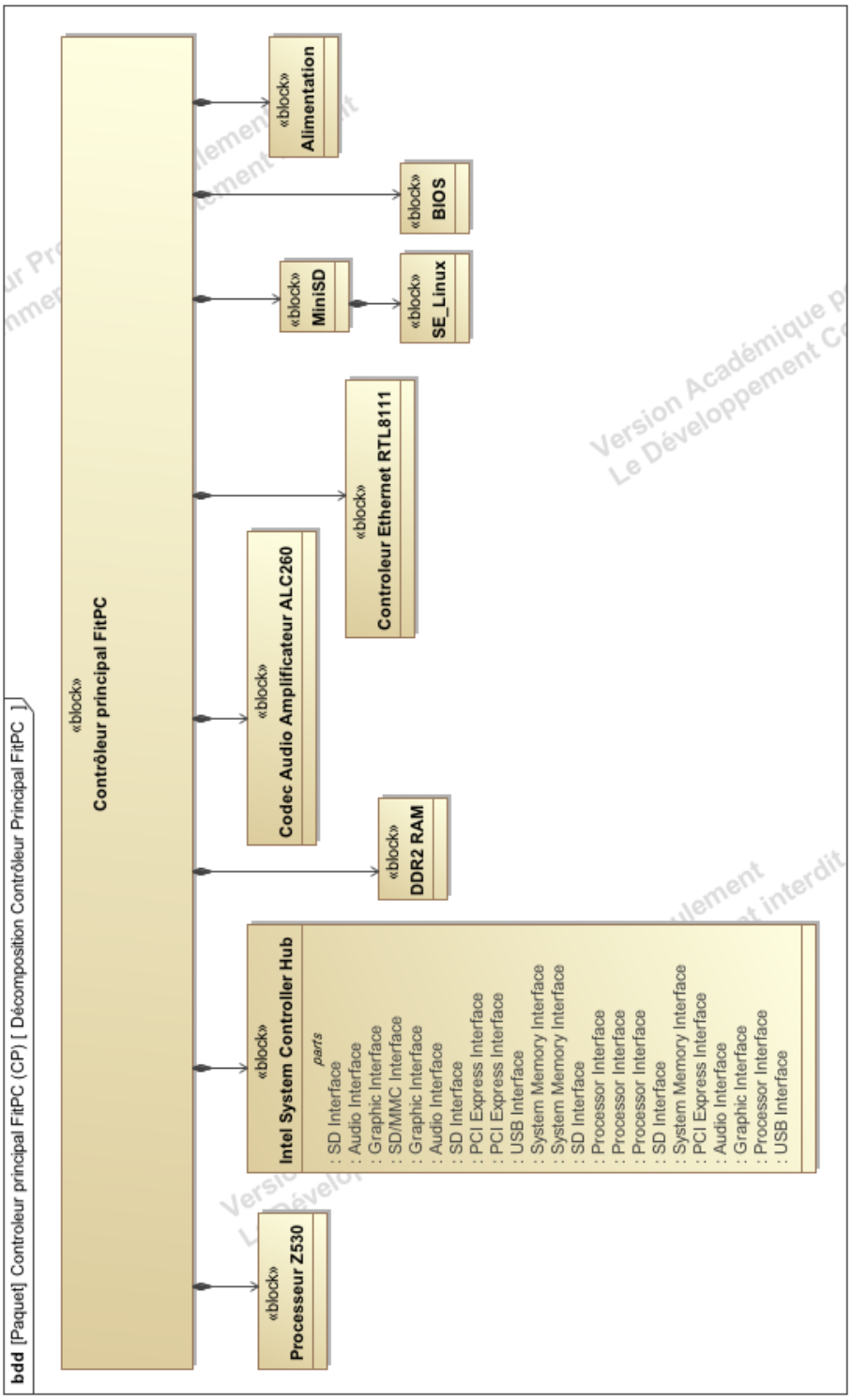



FIGURE 28 – Diagramme SysML de définition de blocs du contrôleur principal du robot DARwIn-OP.

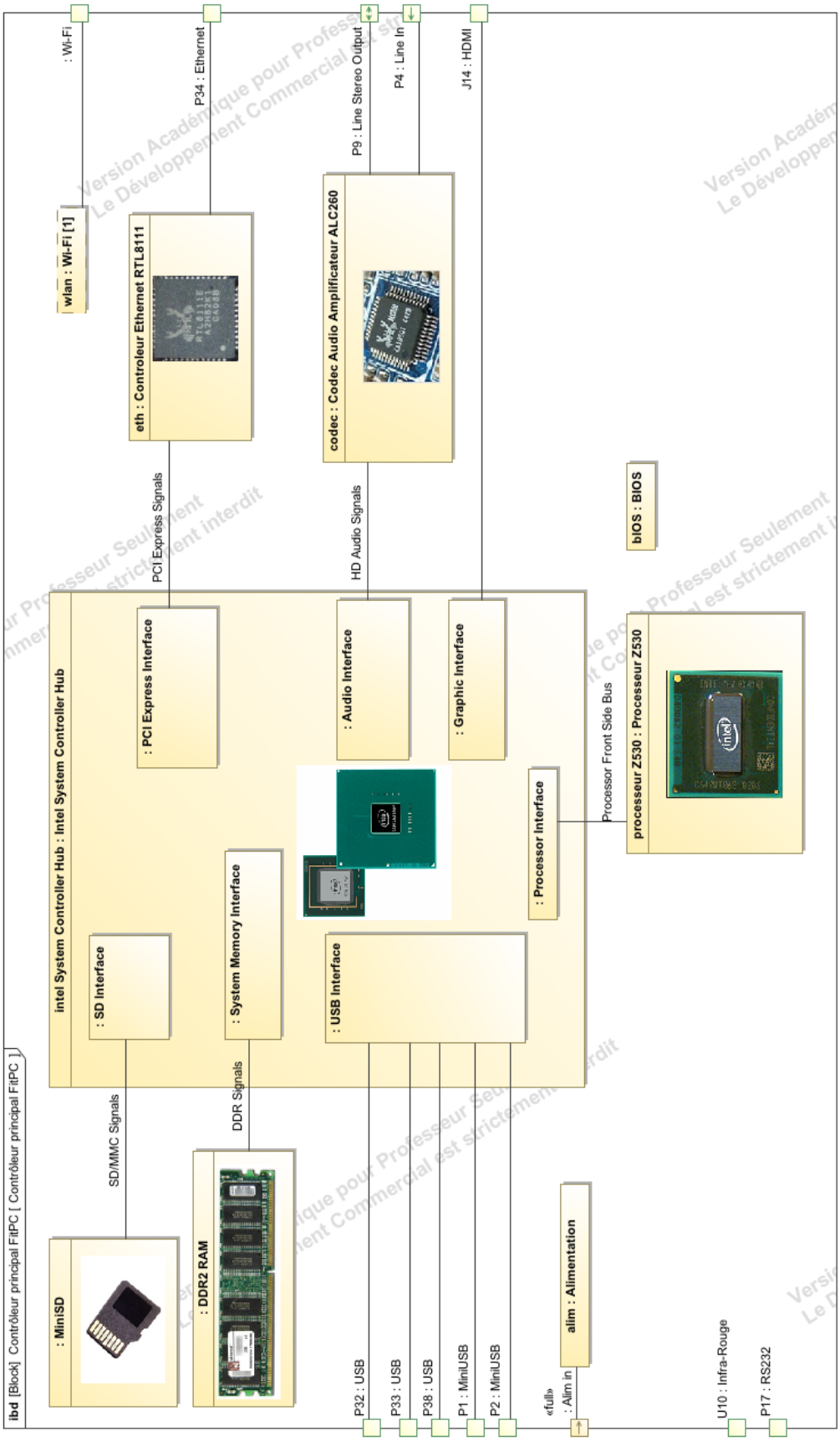



FIGURE 29 – Diagramme SysML de blocs internes du contrôleur principal du robot DARwIn-OP.