# C1 : Modélisation des systèmes pluritechniques
# C1-3 : Analyse comportemental des systèmes

Émilien DURIF

Lycée La Martinière Monplaisir Lyon
Classe de MPSI
14 Septembre 2021

# Plan

# Plan

# Présentation du langage SysML

## intérêts et objectifs

Ce chapitre est dans la continuité du précédent et permet de définir les base de la modélisation comportemental des système.
Nous verrons alors plusieurs aspect de modélisation :

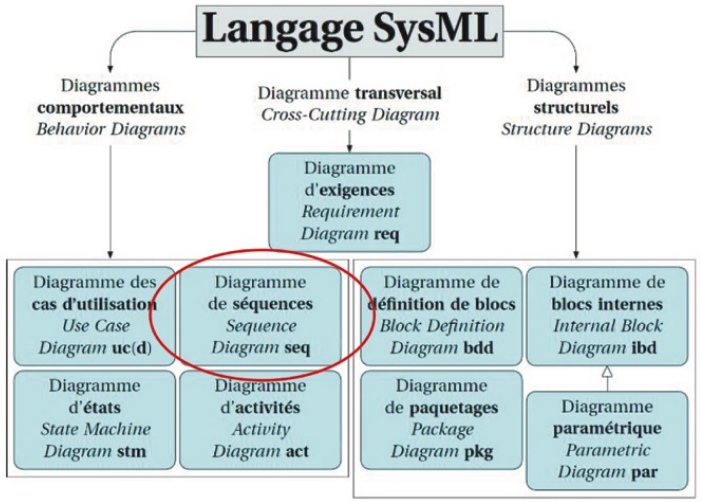
- Une approche globale à l'aide d'outil de représentation graphique utile pour la conception de la commande de système ou pour leur analyse externe.
- Une approche plus spécifique en donnant quelques aspect de modélisation à l'aide de langage de programmation.

# Plan

# Analyse comportementale du système

L'analyse comportemental d'un système peut se présenter suivant deux approches.

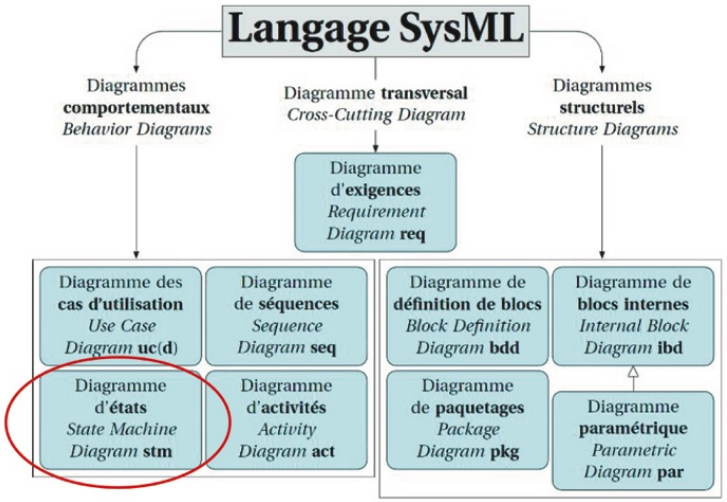
- La première à l'aide du **diagramme de séquence** permet une **analyse globale** du système.
- La deuxième avec les **diagrammes d'état et d'activité** qui représentent le système d'un **point de vue interne** (Cette deuxième partie sera plus détaillée durant le semestre 2).



# Analyse comportementale du système

L'analyse comportemental d'un système peut se présenter suivant deux approches.

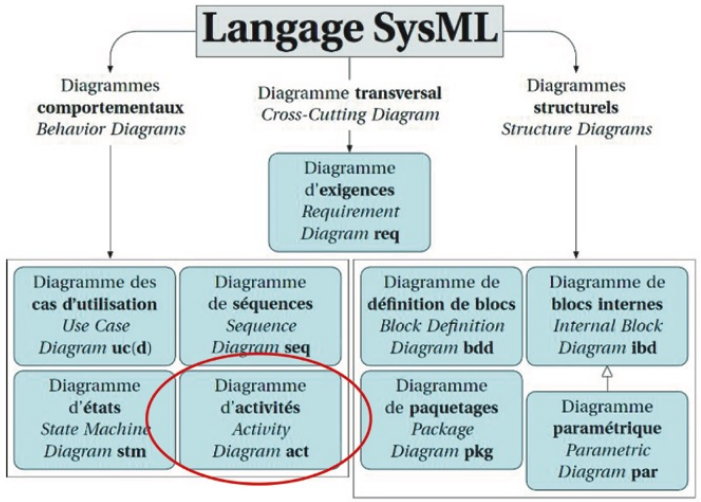
- La première à l'aide du **diagramme de séquence** permet une **analyse globale** du système.
- La deuxième avec les **diagrammes d'état et d'activité** qui représentent le système d'un **point de vue interne** (Cette deuxième partie sera plus détaillée durant le semestre 2).



# Analyse comportementale du système

L'analyse comportemental d'un système peut se présenter suivant deux approches.

- La première à l'aide du **diagramme de séquence** permet une **analyse globale** du système.
- La deuxième avec les **diagrammes d'état et d'activité** qui représentent le système d'un **point de vue interne** (Cette deuxième partie sera plus détaillée durant le semestre 2).



# Diagramme de séquence

## Diagramme de séquence (**seq**)

- *Diagramme comportemental* appelé **Sequence Diagram** (**seq**).
- Description des interactions existant entre plusieurs entités, celles-ci pouvant être des acteurs, le système ou ses sous-systèmes.
- Rattaché à un cas d'utilisation et décrit ce dernier en entier ou en partie, ce qui correspond à un scénario de fonctionnement possible, défini dans un cadre précis : il peut donc aboutir tout aussi bien à des évolutions positives (fonctionnement normal) ou négatives (gestion des problèmes), en particulier dans la phase de démarrage avant le fonctionnement normal.

# Diagramme de séquence

## Diagramme de séquence (**seq**)

- *Diagramme comportemental* appelé **Sequence Diagram** (**seq**).
- Description des interactions existant entre plusieurs entités, celles-ci pouvant être des acteurs, le système ou ses sous-systèmes.
- Rattaché à un cas d'utilisation et décrit ce dernier en entier ou en partie, ce qui correspond à un scénario de fonctionnement possible, défini dans un cadre précis : il peut donc aboutir tout aussi bien à des évolutions positives (fonctionnement normal) ou négatives (gestion des problèmes), en particulier dans la phase de démarrage avant le fonctionnement normal.

# Diagramme de séquence

## Diagramme de séquence (**seq**)

- *Diagramme comportemental* appelé **Sequence Diagram** (**seq**).
- Description des interactions existant entre plusieurs entités, celles-ci pouvant être des acteurs, le système ou ses sous-systèmes.
- Rattaché à un cas d'utilisation et décrit ce dernier en entier ou en partie, ce qui correspond à un scénario de fonctionnement possible, défini dans un cadre précis : il peut donc aboutir tout aussi bien à des évolutions positives (fonctionnement normal) ou négatives (gestion des problèmes), en particulier dans la phase de démarrage avant le fonctionnement normal.

# Diagramme de séquence

## Représentation graphique

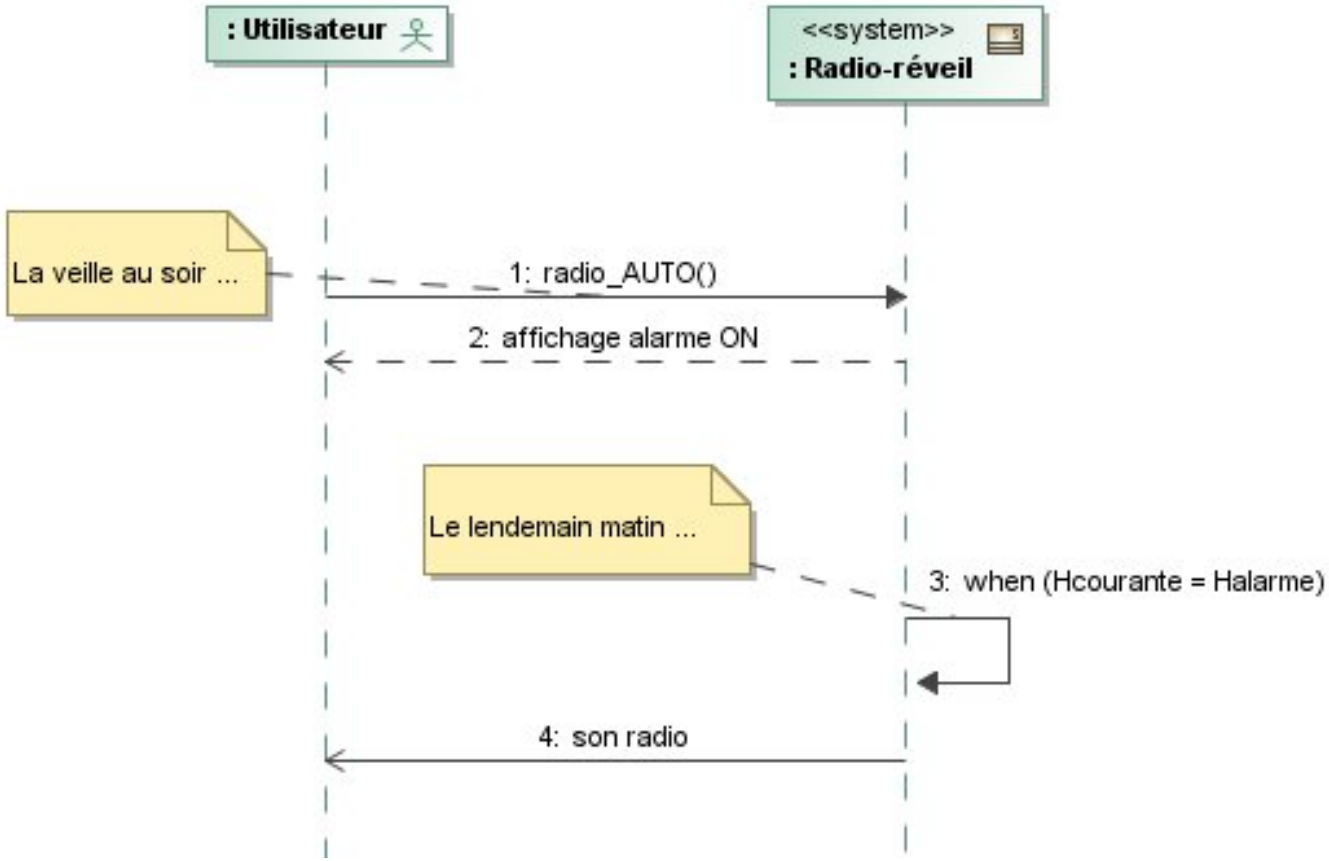
- Traits verticaux en pointillés (**“lignes de vie”**) avec l’indication des propriétaires (en général des acteurs, le système et tout ou partie de ses sous-systèmes) sur la partie supérieure. Le temps se déroule du haut vers le bas, sans échelle particulière.
- Les **messages** sont les entités qui peuvent transités d’une ligne de vie à l’autre (*traits horizontaux*). La réception d’un **message** provoque un événement chez le récepteur.
  - La flèche pointillée représente un retour. Cela signifie que le message en question est le résultat direct du message précédent.
  - Un message synchrone (émetteur bloqué en attente de réponse) est représenté par une flèche pleine ;
  - un message asynchrone est représenté par une flèche évidée.
  - La flèche qui boucle (message réflexif) permet de représenter un comportement interne.

# Diagramme de séquence

## Représentation graphique

- Traits verticaux en pointillés (**"lignes de vie"**) avec l'indication des propriétaires (en général des acteurs, le système et tout ou partie de ses sous-systèmes) sur la partie supérieure. Le temps se déroule du haut vers le bas, sans échelle particulière.
- Les **messages** sont les entités qui peuvent transités d'une ligne de vie à l'autre (*traits horizontaux*). La réception d'un **message** provoque un événement chez le récepteur.
  - La flèche pointillée représente un retour. Cela signifie que le message en question est le résultat direct du message précédent.
  - Un message synchrone (émetteur bloqué en attente de réponse) est représenté par une flèche pleine ;
  - un message asynchrone est représenté par une flèche évidée.
  - La flèche qui boucle (message réflexif) permet de représenter un comportement interne.

# Diagramme de séquence



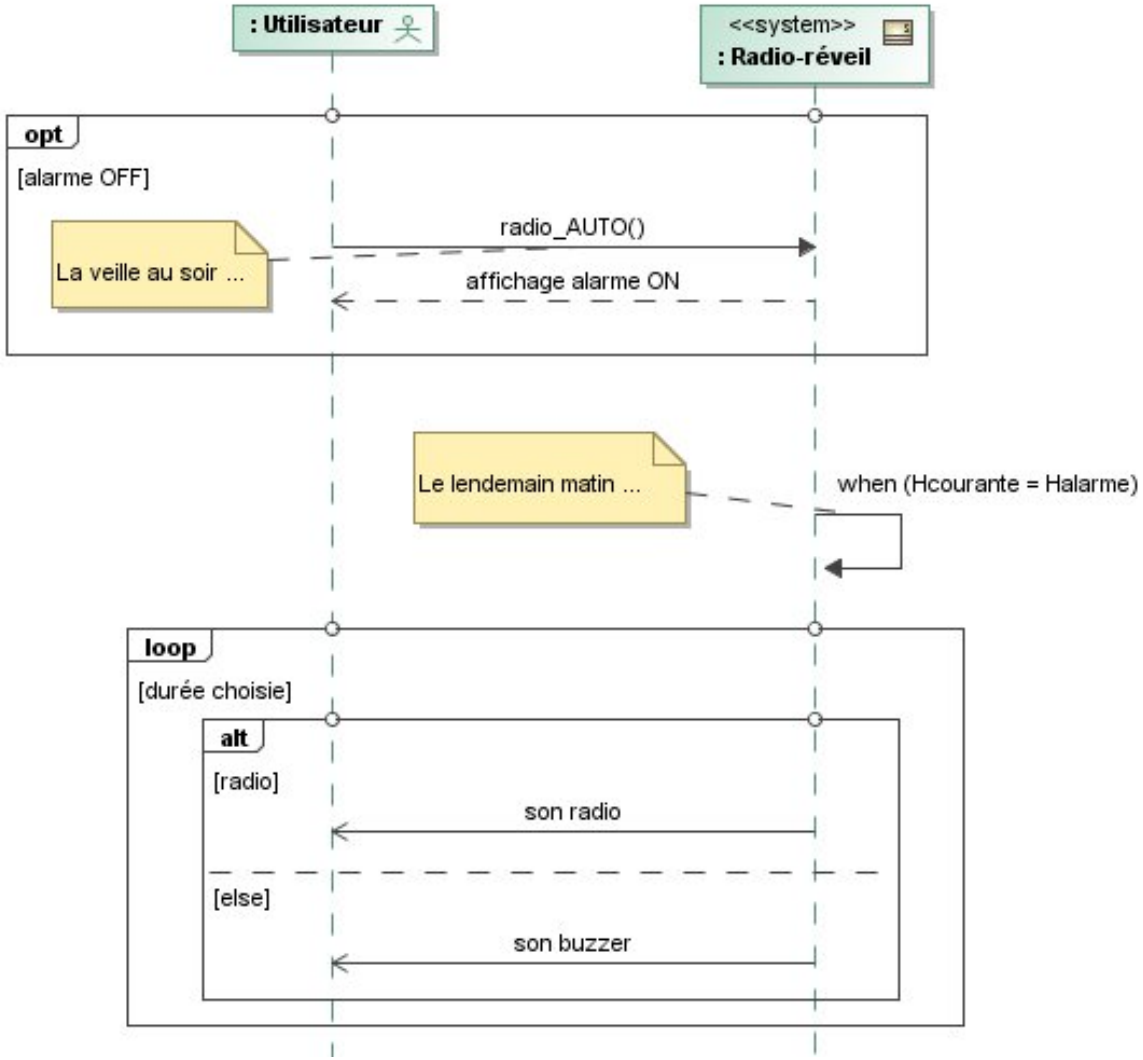
# Fragments combinés

## Fragments combinés

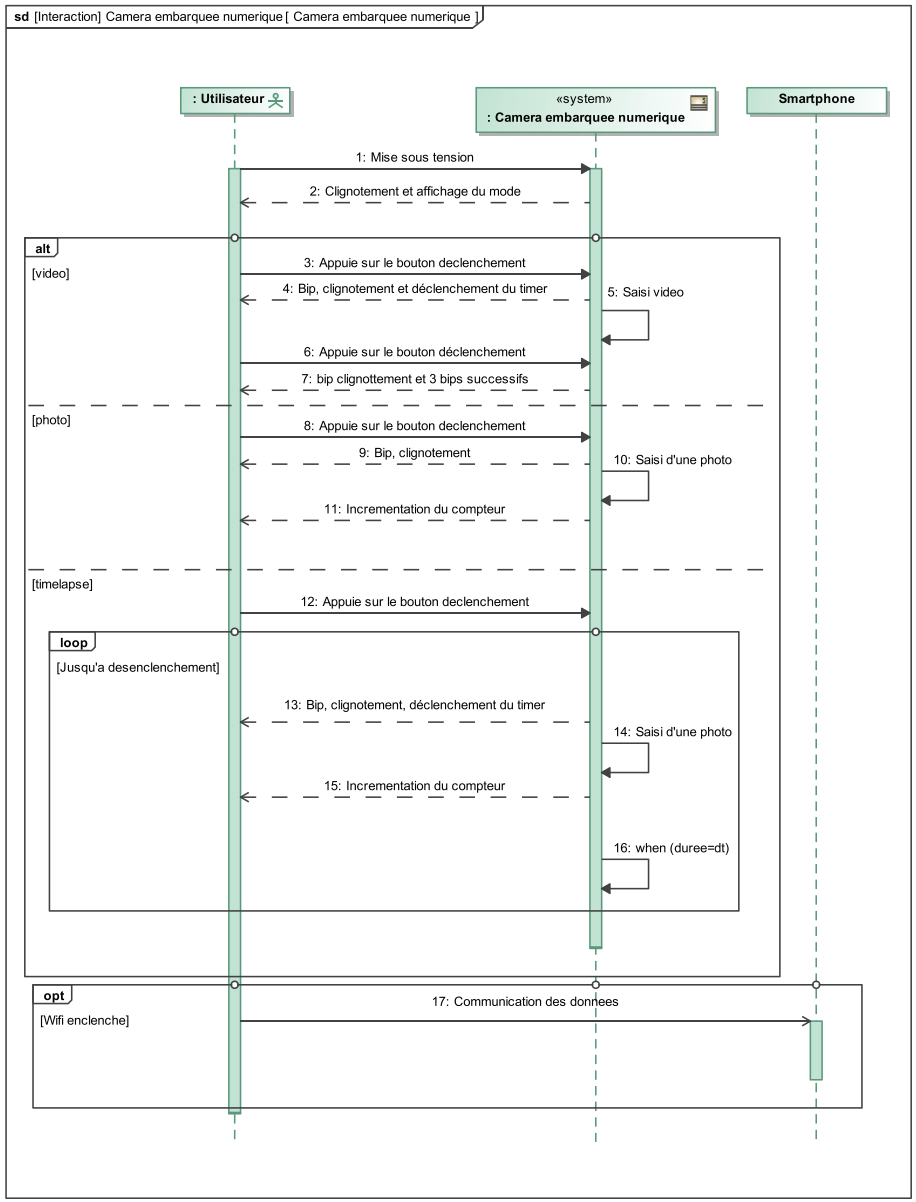
Chaque fragment possède un opérateur et peut être divisé en opérandes. Les principaux opérateurs sont :

- **loop** - boucle. Le fragment peut s'exécuter plusieurs fois, et la condition de garde explicite l'itération ;
- **opt** - optionnel : le fragment ne s'exécute que si la condition fournie est vraie ;
- **alt** - fragments alternatifs : seul le fragment possédant la condition vraie s'exécutera.

# Fragments combinés



# Fragments combinés



# Diagramme de séquence

**Remarque**

- Ce diagramme comportemental est en forte interaction avec le diagramme de cas d'utilisation.
- On construit généralement un diagramme de séquence par scénario.
- Ce diagramme permet de montrer les interactions entre les différentes parties et non visibles dans un diagramme de cas d'utilisation qui n'indique que l'association entre l'acteur et un cas d'utilisation.

# Diagramme d'états

## Diagramme d'états **stm**

Le diagramme d'états (ainsi que le diagramme d'activité) fera l'objet d'une étude plus poussée au semestre 4 mais nous pouvons déjà le définir. Ils permettent de modéliser le comportement d'un système à des fins de programmation.

# Plan

# Diagramme d'activités : définition

## Diagramme d'activité

Le **diagramme d'activité** est un diagramme comportemental appelé *Activity Diagram* (**act**) dans le langage SysML. Il permet de modéliser le déroulement d'un processus sous la forme d'une activité correspondant à la décomposition séquentielle d'actions aussi appelées tâches. Les éléments graphiques utilisés dans ce diagramme sont principalement :

- des rectangles aux coins arrondis pour les états,
- des flèches orientées de l'état de départ à l'état cible pour les transitions,
- des occurrences attachées à la transition spécifient les conditions de franchissement.

## Remarque

- Ce diagramme ne possède aucun événement associé aux transitions entre actions : la fin d'une action implique automatiquement le passage à la suivante, donc dans un ordre déterminé d'actions menant à un résultat. Lorsque le processus est enclenché il va à son terme selon un ordre précis.
- Ce diagramme permet aussi de représenter des **algorigrammes**, c'est à dire un flux de contrôle.
- Ce diagramme ne figure pas explicitement dans le programme.

# Diagramme d'activités : définition

## Diagramme d'activité

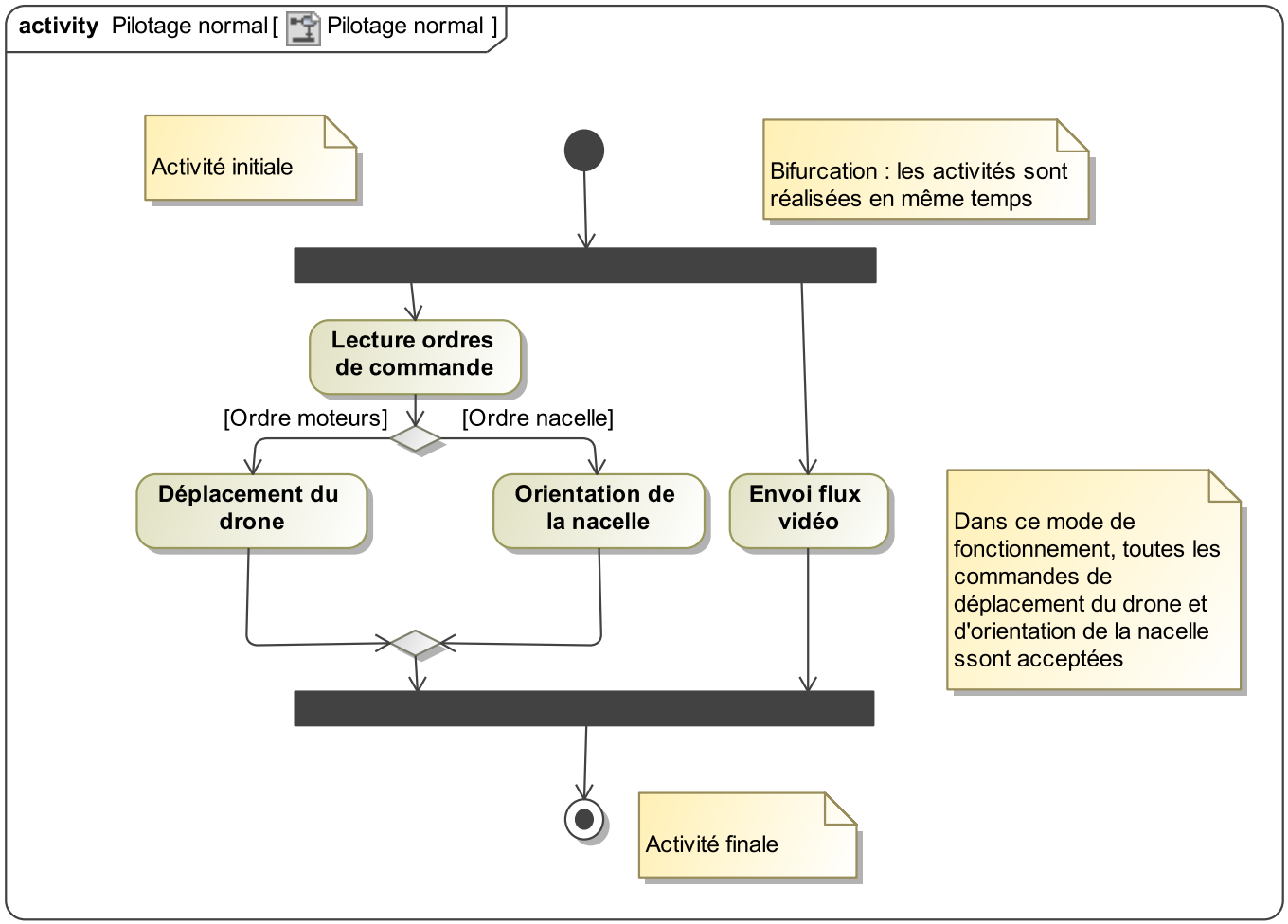
Le **diagramme d'activité** est un diagramme comportemental appelé *Activity Diagram* (**act**) dans le langage SysML. Il permet de modéliser le déroulement d'un processus sous la forme d'une activité correspondant à la décomposition séquentielle d'actions aussi appelées tâches. Les éléments graphiques utilisés dans ce diagramme sont principalement :

- des rectangles aux coins arrondis pour les états,
- des flèches orientées de l'état de départ à l'état cible pour les transitions,
- des occurrences attachées à la transition spécifient les conditions de franchissement.

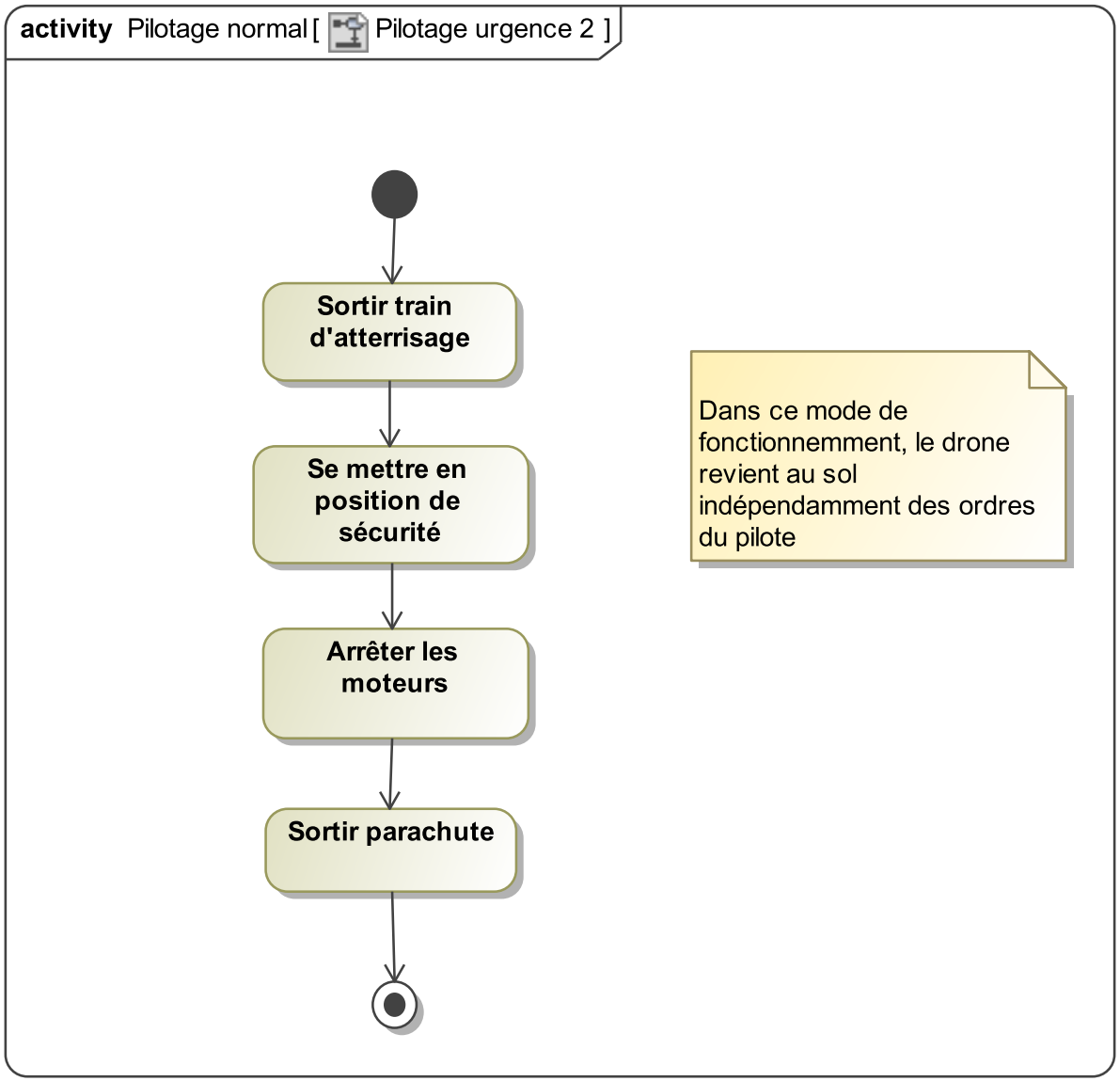
## Remarque

- Ce diagramme ne possède aucun événement associé aux transitions entre actions : la fin d'une action implique automatiquement le passage à la suivante, donc dans un ordre déterminé d'actions menant à un résultat. Lorsque le processus est enclenché il va à son terme selon un ordre précis.
- Ce diagramme permet aussi de représenter des **algorigrammes**, c'est à dire un flux de contrôle.
- Ce diagramme ne figure pas explicitement dans le programme.

# Diagramme d'activités : exemples



# Diagramme d'activités : exemples



# Utilisation d'algorithmes.

- Une autre approche pour modéliser le comportement des systèmes est d'utiliser des algorithmes. Cette approche permet en outre de générer la commande du système via l'organe de traitement de l'information (micro-contrôleur, carte de traitement) par exemple.
- Cette partie présente les trois structures algorithmiques de base et présente leur syntaxe en langage **Python**.

# Utilisation d'algorithmes.

- Une autre approche pour modéliser le comportement des systèmes est d'utiliser des algorithmes. Cette approche permet en outre de générer la commande du système via l'organe de traitement de l'information (micro-contrôleur, carte de traitement) par exemple.
- Cette partie présente les trois structures algorithmiques de base et présente leur syntaxe en langage **Python**.

# Algorithmes : Boucles définies

## Boucles définies

Une **boucle itérative définie**, permet de **répéter** l'application d'une même séquences d'instructions sur une liste **définie à l'avance**.

> **Pour** variable **dans** liste **répeter**
> bloc d'instructions b
> **Fin-de-la-boucle**

signifie que

**pour** chaque élément de la liste `liste`,
le programme exécute les instructions du bloc b.

# Algorithmes : Boucles définies

```
for variable in liste :
    instructions
```

## Syntaxe en Python

Ici encore, la ligne contenant le mot-clé **for** doit se finir par un « **:** » et les instructions du bloc doivent être indentées. La fin de la boucle est marquée par un retour à la ligne non indenté.

# Algorithmes : Boucles définies

**Exemple : afficher successivement tous les prénoms provenant d'une liste**

prenoms = [ 'Baptiste', 'Lisa', 'Pierrick', 'Louise-Eugénie', 'Qâsim', 'Lorenzo', 'Arthur', 'Ylies' ]

```
for x in prenoms:
    print('Bonjour ' + x)
```

# Algorithmes : Instructions conditionnelle

## Instructions conditionnelles

Quand on veut écrire un programme, on souhaite établir des connections logiques entre les instructions. Ainsi, l'instruction conditionnelle a pour objet d'intervenir dans le choix de l'instruction suivante en fonction d'une expression booléenne qu'on désignera par **condition** :

> **Si** condition
> **alors** bloc d'instructions 1
> **sinon** bloc d'instructions 2
> **Fin-du-Si**

signifie que

- **Si** la condition est vérifiée (expression booléenne=`True`) **alors** le programme exécute les instructions du bloc 1 ;
- **si** la condition n'est pas vérifiée (expression booléenne=`False`) **alors** le programme exécute les instructions du bloc 2.

# Algorithmes : Instructions conditionnelle

## Syntaxe en Python

```
if condition 1 :
      bloc d instructions 1
elif condition 2 :
      bloc d instructions 2
elif condition 3 :
      bloc d instructions 3
.
.
.
else :
      bloc final
```

- **Sinon si** se traduit par `elif`.

# Algorithmes : Instructions conditionnelle

## Exemple

On veut tester si un nombre $x$ est proche de 3 à $10^{-4}$ près. On peut alors écrire la fonction suivante.

```
def est_proche(x):
    """x est proche de 3 à10**-4 près ?"""
    distance = abs(x-3)
    if distance <= 10**(-4) :
        return True
    else :
        return False
```

# Algorithmes : Boucles indéfinies ou conditionnelles

## Boucles indéfinies ou conditionnelles

On peut aussi être amené à répéter un bloc d'instructions sans savoir combien de fois on devra le répéter.
Dans ce cas, on utilise la boucle **Tant que** qui permet de répéter le bloc d'instructions tant qu'une certaine condition est vérifiée.

> **Tant que** condition
> **faire** bloc d'instructions
> **Fin-du-Tant-que**

signifie que

Tant que la condition est vérifiée (expression booléenne=*True*)
Faire le bloc d'instructions.

# Algorithmes : Boucles indéfinies ou conditionnelles

## Syntaxe en Python

```
while condition :
    instructions
```

## Exemple

Rechercher le premier entier $n$ tel que la somme des entiers de 1 à n dépasse 11.

```
n = 1
s = 1
while s < 11 :
    n = n + 1
    s = s + n

n
```

REPONSE : $n = 5$ (dans ce cas $s = 15$)

# implémentation

Une fois la modélisation effectuée, on peut implémenter le programme qui va gérer la commande du système sur le mircro-controleur. Différentes méthodes et interfaces existent. Certaines interfaces utilisent directement représentation graphique (similaire aux diagrammes d'activité) ou d'autres utilisent un langage de programme (Python, c++, etc...)

# implémentation

## Exemple : commande de moteur par micro-contôleur

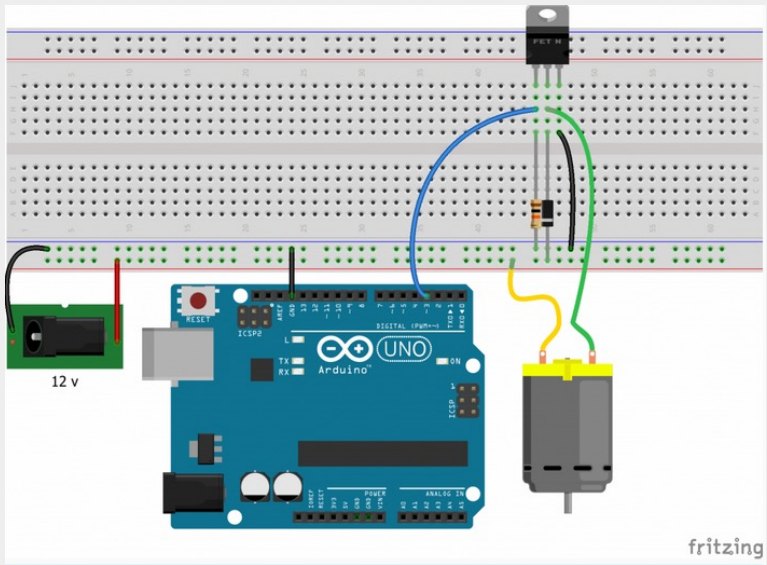
read_weight_sensor_df_robot | Arduino 1.8.5

read_weight_sensor_df_robot

```
#include <DFRobot_HX711.h>

DFRobot_HX711 MyScale(A2, A3);

void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.print(MyScale.readWeight(), 1);
  Serial.println(" g");
  delay(200);
}
```

o Mega or Mega 2560, ATmega2560 (Mega 2560) sur /dev/cu.usbmodem1421

exemple de Code C++ pour piloter un moteur



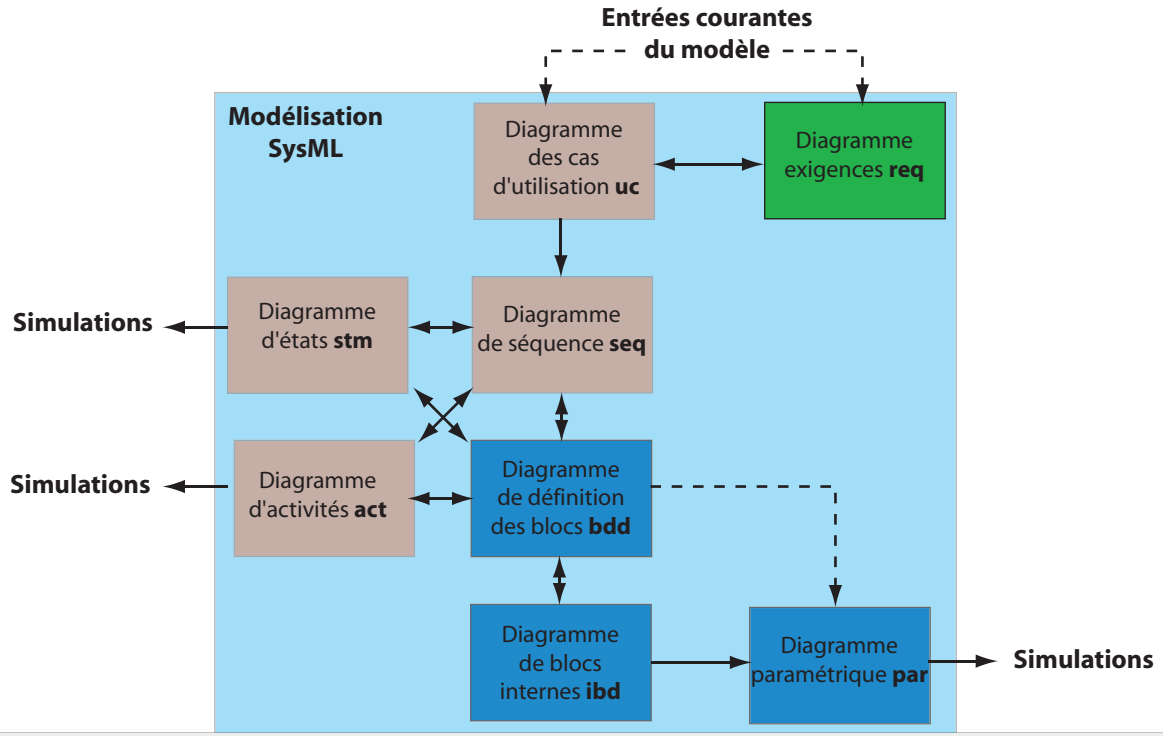

exemple de montage de commande de moteur avec un micro-controleur arduino UNO https:

# Plan

# Diagramme d'états

## Conclusion

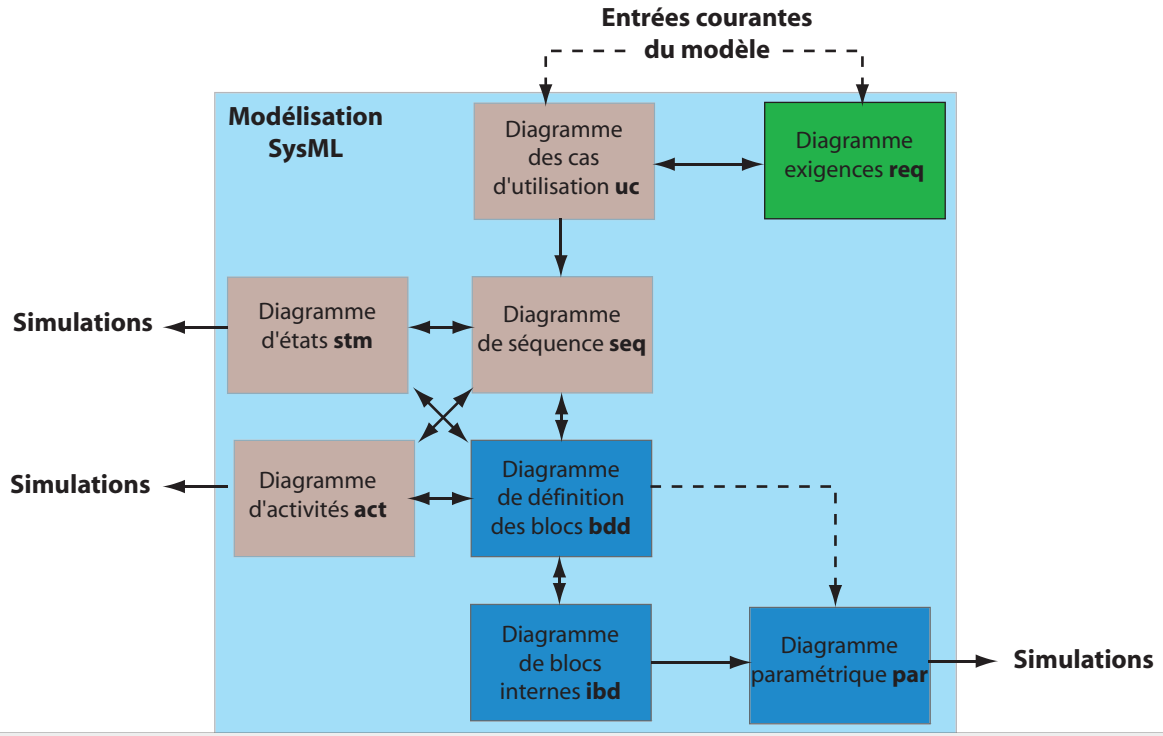
- La modélisation SysML d'un système permet de décrire de manière ordonnée un système.
- La richesse et la polyvalence des diagrammes permet d'avoir une bonne vision d'ensemble du système.
- De plus cette modélisation permet d'aller jusqu'à la simulation des systèmes et permet ainsi d'aider les ingénieurs dans leur démarche de conception.



# Diagramme d'états

## Conclusion

- La modélisation SysML d'un système permet de décrire de manière ordonnée un système.
- La richesse et la polyvalence des diagrammes permet d'avoir une bonne vision d'ensemble du système.
- De plus cette modélisation permet d'aller jusqu'à la simulation des systèmes et permet ainsi d'aider les ingénieurs dans leur démarche de conception.



# Diagramme d'états

## Conclusion

- La modélisation SysML d'un système permet de décrire de manière ordonnée un système.
- La richesse et la polyvalence des diagrammes permet d'avoir une bonne vision d'ensemble du système.
- De plus cette modélisation permet d'aller jusqu'à la simulation des systèmes et permet ainsi d'aider les ingénieurs dans leur démarche de conception.