LYCÉE LA MARTINIÈRE MONPLAISIR LYON
SCIENCES INDUSTRIELLES POUR L'INGÉNIEUR
CLASSE PRÉPARATOIRE M.P.S.I.
ANNÉE 2022 - 2023

C1 : MODÉLISATION DES SYSTÈMES PLURITECHNIQUES

C1-3 - Analyse comportemental des systèmes

20 Septembre 2022

Table des matières

Compétences

- **Analyser**
  - Identifier et décrire les chaines fonctionnelles du système.
  - Analyser un algorithme.
- **Modéliser**
  - Décrire le comportement d'un système séquentiel.
- **Communiquer**
  - Lire et décoder un document technique : SysML
  - Lire et décoder un document technique : Schéma Cinématique

I. 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.

II. Analyse comportementale globale et externe des systèmes

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** représente le système d'un **point de vue interne** (Cette deuxième partie sera plus détaillée durant le semestre 2).

1 Diagramme de séquence

Définition 1 : ***Diagramme de séquence (seq)***

Le **diagramme de séquence** est un *diagramme comportemental* appelé **Sequence Diagram** (**seq**) dans le langage **SysML**.
L'objectif de ce diagramme est de décrire les interactions existant entre plusieurs entités, celles-ci pouvant être des acteurs, le système ou ses sous-systèmes. Le diagramme ne montre donc que l'enchaînement séquentiel des différentes interactions.
Un diagramme de séquence est 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.

Propriété 1 : ***Représentation graphique***

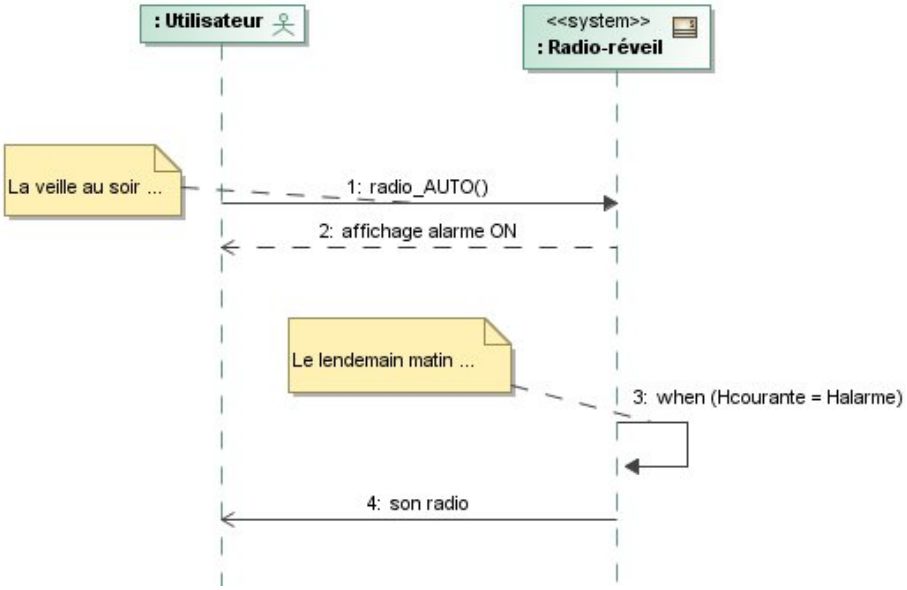
Les éléments graphiques utilisés dans ce diagramme sont principalement :
- Des *traits verticaux* en pointillés appelés **"lignes de vie"** avec l'indication des propriétaires (en général des acteurs, le système et tout ou partie de ses sous-systèmes) sur la partie supérieure. Le temps se déroule du 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.

Remarque 1 :
- 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 non visibles dans un diagramme de cas d'utilisation qui n'indique que l'association entre l'acteur et un cas d'utilisation.

Exemple 1 : *Radio réveil*

Pour le cas d'utilisation "Être réveillé à l'heure en musique".

- Le premier message est un message synchrone, donnant lieu à un retour : l'affichage d'un point à côté de l'heure indiquant que l'alarme est positionnée.
- Le fait que radio-réveil détecte que l'heure courante devient égale à l'heure d'alarme est représenté par un message réflexif avec le mot-clé when.
- Le dernier message est un signal asynchrone.



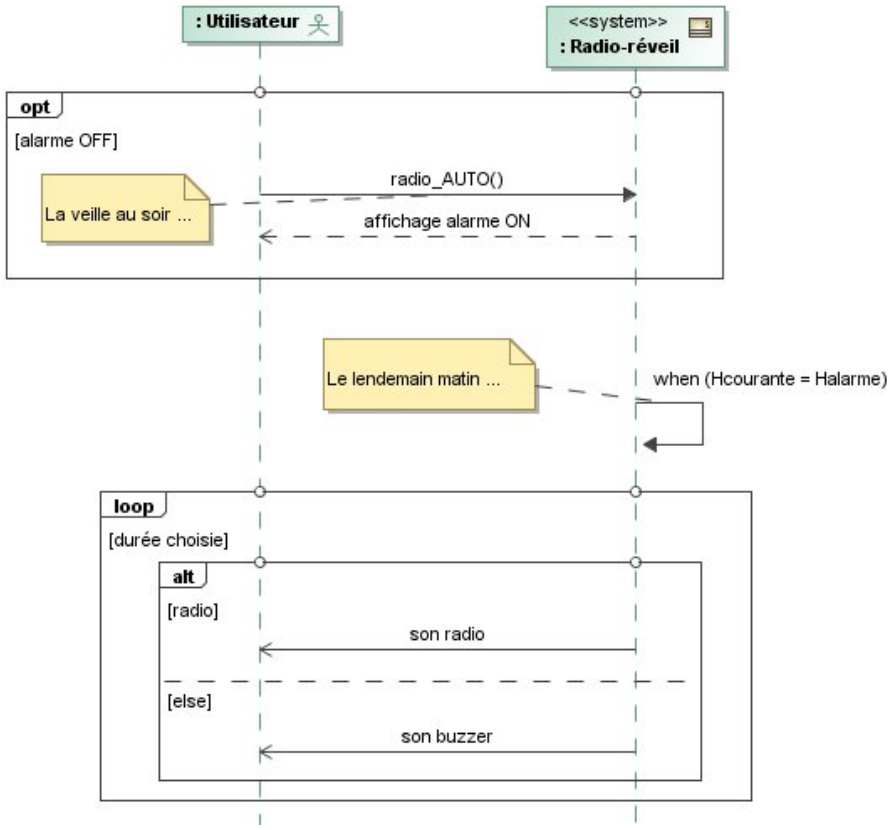
Définition 2 : *Fragments combinés*

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.

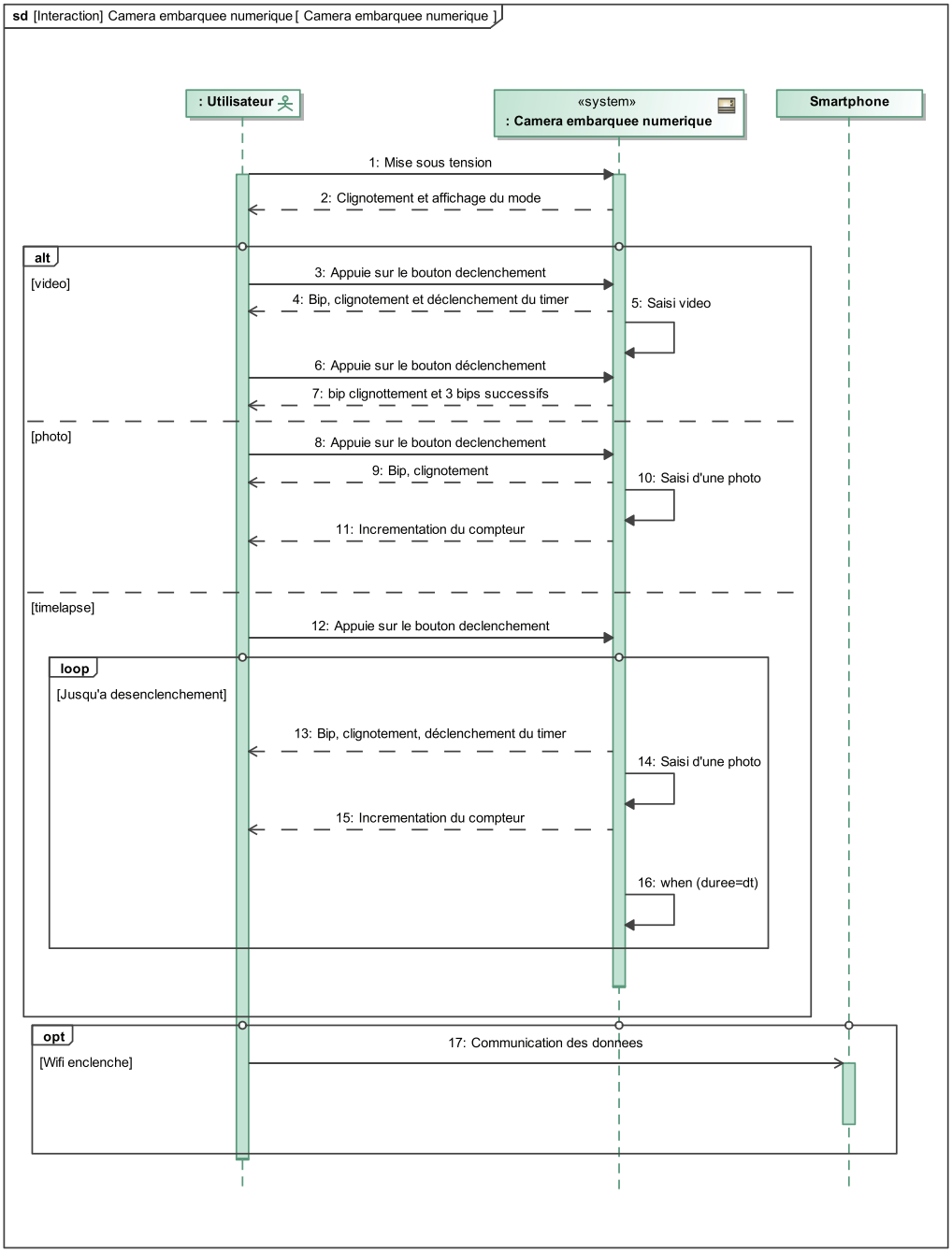
Exemple 2 : *Radio réveil*

- Le son qui sort de la radio est continu pendant plusieurs minutes, ce n'est pas un simple signal unitaire : pour le représenter, positionnons un fragment de boucle.
- L'utilisateur sera réveillé par la radio ou le buzzer suivant son choix. Nous pouvons donc ajouter un fragment ***alt*** avec deux opérandes.
- Enfin, le premier message n'est pas nécessaire si l'alarme était déjà positionnée la veille : il est donc optionnel.



Exemple 3 : *Diagramme de séquence de la caméra embarquée numérique*

On peut décrire le fonctionnement du système étudié plus complexe à l'aide du diagramme suivant entre l'utilisateur et la caméra embarquée numérique.



2 Diagramme d'états

Le diagramme d'états (ainsi que le diagramme d'activité) fera l'objet d'une étude plus poussée au semestre 2 mais nous pouvons déjà le définir. Ils permettent de modéliser le comportement d'un système à des fins de programmation.

III. Analyse et modélisation du comportement interne des systèmes

1 Algorigrammes ou diagramme d'activités.

Définition 3 : *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 orientés de l'état de départ à l'état cible pour les transitions,
- des occurrences attachées à la transition spécifient les conditions de franchissement.

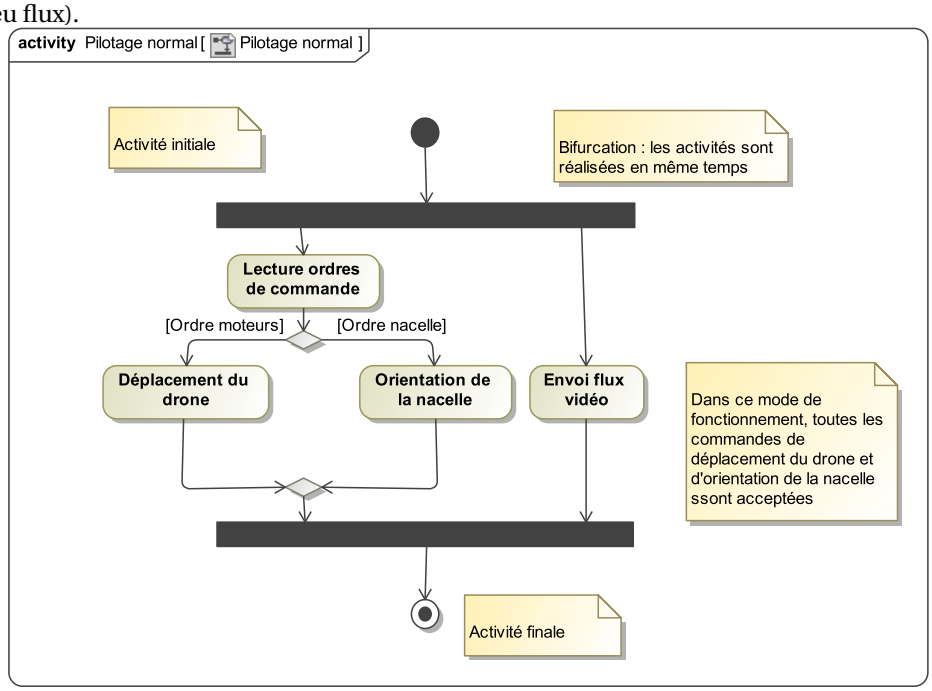
Remarque 2 :

- 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.

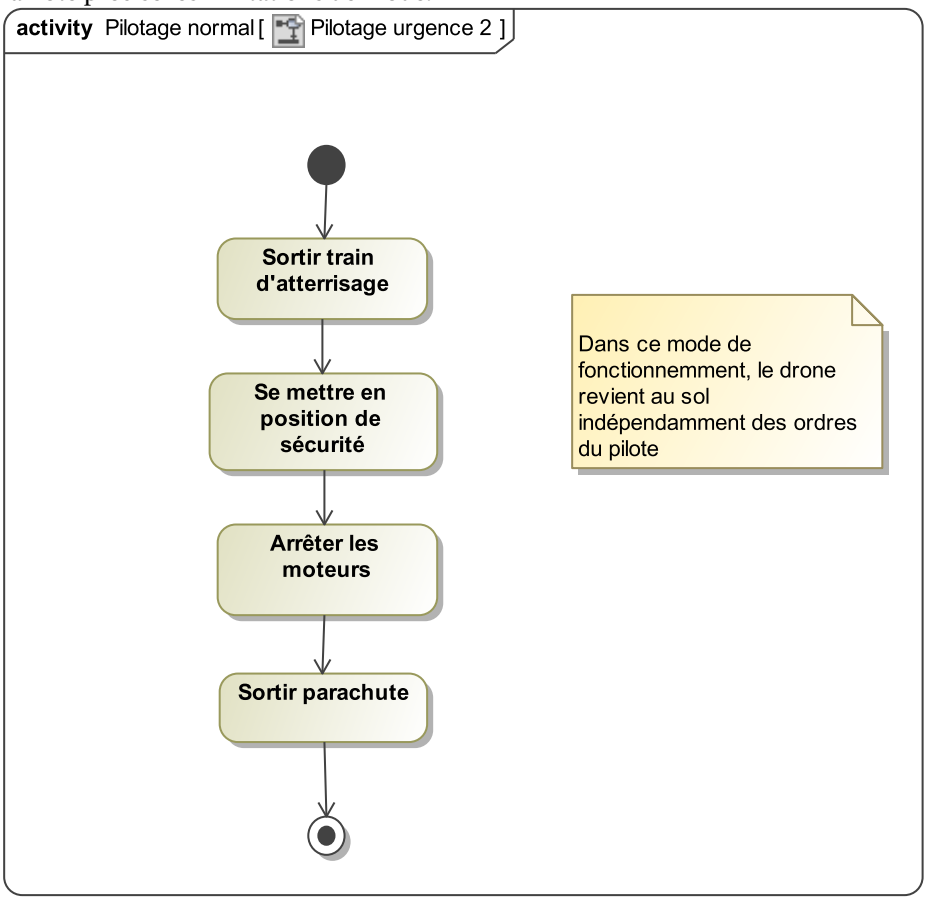
A l'aide de ce diagramme, on peut décrire plus en détail ce qui se passe dans chaque mode .

Exemple 4 :

- **Diagramme d'activité du mode de pilotage normal :** il correspond à un pilotage complet du drone avec en parallèle l'envoi du flux vidéo (les deux barres noirs marquent le début et la fin de l'envoi des deu flux).



- **Diagramme d'activité du mode d'urgence 2 :** il correspond à la procédure d'atterrissage automatique. La note précise les limitations de mode.



2 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**.

a) Boucles définies

Définition 4 : *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.

Propriété 2 : *Syntaxe en Python*

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

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.

Exemple 5 : *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)
```

b) Instructions conditionnelles

Définition 5 : *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.

Propriété 3 : *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`.

Exemple 6 :

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
```

c) Boucles indéfinies ou conditionnelles

Définition 6 : *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.

Propriété 4 : *Syntaxe en Python*

```
while condition :
    instructions
```

Exemple 7 :

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

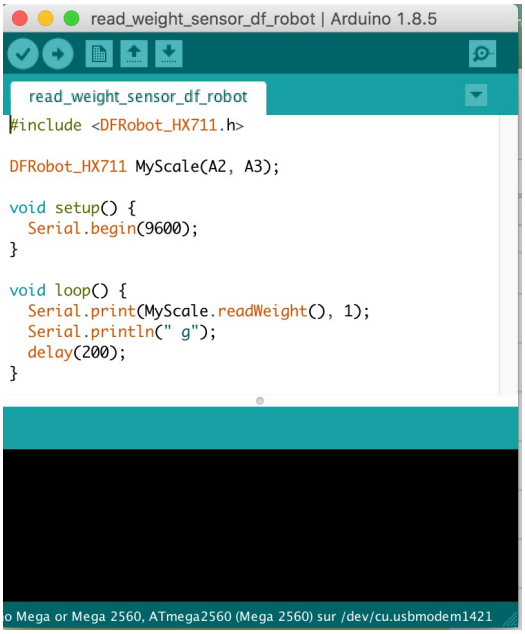
```
n = 1
s = 1
while s < 11 :
    n = n + 1
    s = s + n

n
```

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

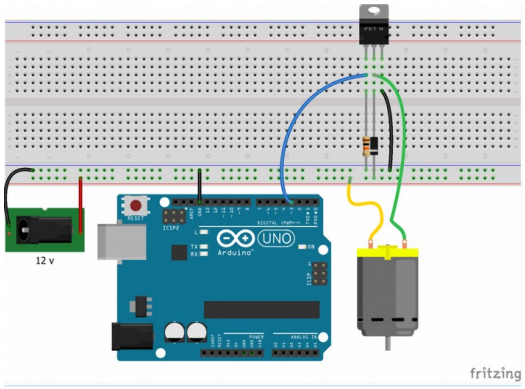
3 Vers l'implémentation sur un système pluritechinque complexe réel.

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...)

Exemple 8 : *Commande de moteur par micro-contôleur*



exemple de Code C++ pour piloter un moteur



exemple de montage de commande de moteur avec un micro-controleur arduino UNO https://transat.stephanecabee.net/

IV. Interaction entre les différents diagrammes SysML

Conclusion :

La modélisation SysML d'un système permet de décrire de manière ordonnée un système. La richesse et la polyvalence des diagrammes donne 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.

Le synoptique ci-dessous (figure 1) donne une possibilité de mise en place d'une modélisation SysML, depuis la définition des exigences et/ou des cas d'utilisation jusqu'à la mise en oeuvre du diagramme paramétrique en passant par les différents diagrammes et les simulations associées.

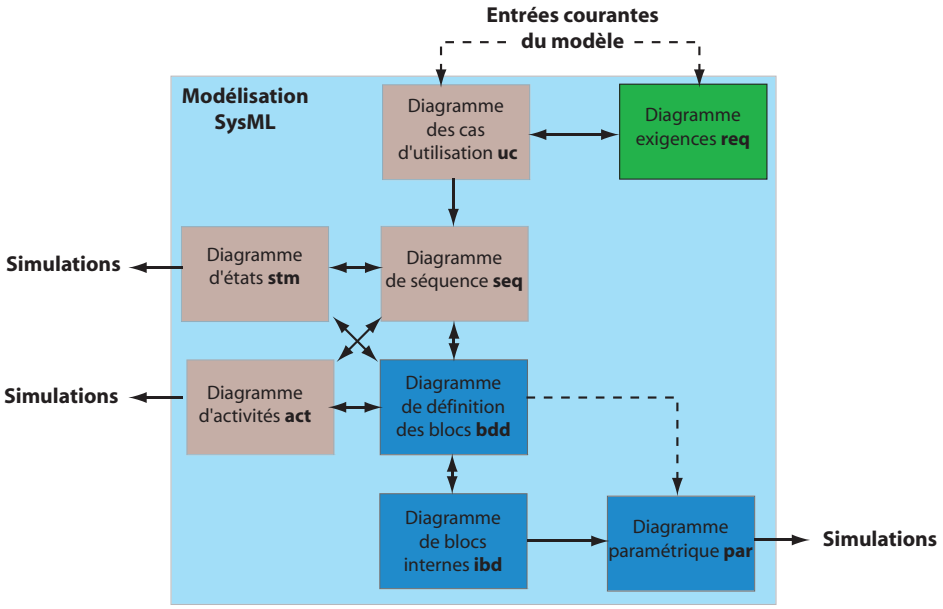


FIGURE 1 – Représentation synoptique des liens entre les différents diagrammes