31/03/2017

AppliConso : Dossier de Maintenance

LOPEZ Clément – THIEBAUD Charles – TOCABENS Emmanuel
IUT BLAGNAC

Table des matières

I. Introduction

L'application « AppliConso » est un outil permettant à l'utilisateur de pouvoir consulter la consommation en temps réel de sa machine ainsi que la consommation d'une application spécifique ou de toutes les applications ouvertes au lancement de « AppliConso ».

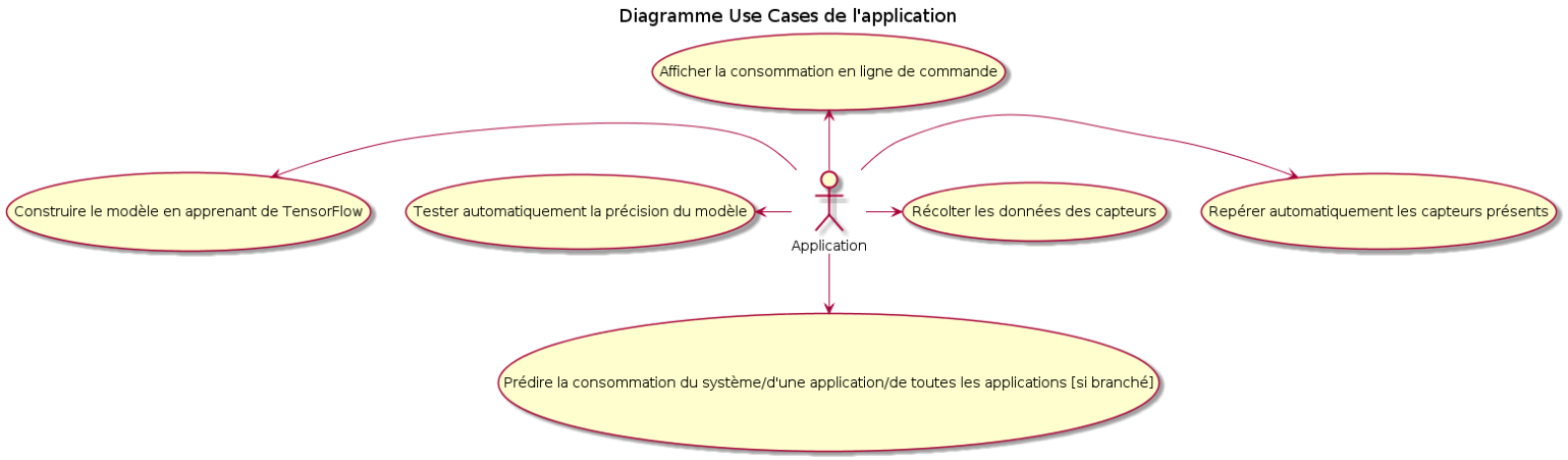
Elle est le fruit d'une demande de l'Institut de Recherche en Informatique de Toulouse (IRIT) et s'inscrit notamment dans un objectif pédagogique (permettre à de jeunes programmeurs d'avoir un aperçu direct de la consommation des applications qu'ils ont créés) ainsi que dans un but écologique (permettre aux utilisateurs de se rendre compte de la consommation de leurs machines).

« AppliConso » se veut très simple d'utilisation, aussi, le lancement de cette application et l'affichage de la consommation se font via un terminal de commande.

Ce Guide a par ailleurs été rédigé afin de permettre une meilleure compréhension des aspects technique de l'application et de sa conception afin d'en permettre la maintenance et l'amélioration.

II. Traitements implémentés

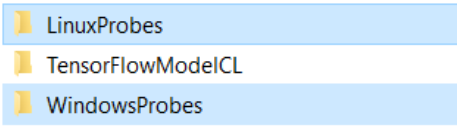
Nous avons implémenté plusieurs fonctionnalités dans AppliConso :



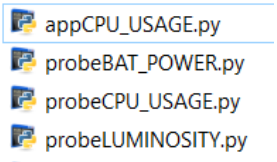
A. Gestion des capteurs

AppliConso détecte automatiquement les capteurs qui sont à sa disposition et si un modèle a déjà été construit avec ces capteurs. Si tel est le cas, AppliConso poursuit avec ce modèle, dans le cas contraire l'application crée un tout nouveau modèle pour cette configuration de capteurs.

Tout cela se fait par une méthode de référencement des différents capteurs et modèles existants. En effet, les capteurs étant, pour la plupart, différents en fonction du système d'exploitation, ils sont triés dans des sous-dossiers en fonction du système d'exploitation sur lequel ils agissent (nom des sous-dossiers : **<Nom du système d'exploitation>Probes**)



A l'intérieur de ces sous-dossiers, une autre méthode de référencement est utilisée. En effet, les capteurs sont repérés par AppliConso grâce à leur nom (**probe<Nom Capteur>.py**). Cas particulier, dans le cas où l'utilisateur demande la consommation d'une application en particulier, on regarde un capteur en plus, celui de la consommation de CPU de l'application (**appCPU_USAGE.py**) ce qui permet de déduire la consommation de l'application de la consommation totale du système.



A noter : les données récoltées par les capteurs sont également enregistrées dans un fichier log.txt que l'utilisateur peut consulter.

Un dernier capteur est utilisé, il ne suit pas le modèle des autres capteurs et ne figure pas dans un répertoire de capteurs mais est directement présent dans le répertoire de « AppliConso », c'est **ChargeListener.py**

Ce capteur bien particulier va récupérer une information de la batterie très importante pour le bon fonctionnement de l'application : c'est ce capteur qui va déterminer si la batterie est en charge ou en décharge.

Il est recommandé de ne toucher à aucun capteur présent par défaut, mais les capteurs à ne surtout pas enlever sont : **ChargeListener.py**, **probeBAT_POWER**, **appCPU_USAGE**

Pour rajouter des capteurs :

Pour rajouter des capteurs, il faut créer un fichier python sur le même modèle que ceux existants (**probe<Nom Capteur>.py** placé dans le répertoire du système d'exploitation sur lequel il est actif). Il est également impératif que dans le fichier, il y ait une méthode « getData() » renvoyant un nombre. C'est cette méthode qui sera appelé par le programme principal pour récupérer les données du capteur.

Exemple avec le code de probeCPU_USAGE.py :

```
import psutil

def getData():
    return psutil.cpu_percent(interval=0.1)
```

ensorFlow

détecte automatiquement les capteurs disponibles (cf II.A) et sélectionne le modèle TensorFlow correspondant (ou le construit s'il ne l'est pas encore). La présence ou non d'un modèle est facilement vérifiable puisque les données d'un modèle sont dans un dossier nommé : **TensorFlowModel<Désignation>** où Désignation correspond à la compilation de tous les premiers caractères des capteurs utilisés. Si le dossier n'existe pas, il est généré automatiquement.

Pour exemple, un modèle utilisant les capteurs « probeCPU_USAGE.py » et « probeLUMINOSITY.py » verra son dossier nommé « TensorFlowModelCL ».

C. Prédiction de la consommation et test de précision

En s'appuyant sur le modèle TensorFlow actif, AppliConso est capable d'estimer la consommation du système.

Lorsque la machine est débranchée, la consommation du système est directement accessible et il n'y a alors pas besoin d'afficher la prédiction (on affiche la consommation réelle et on récolte les données pour construire le modèle) mais l'application va tester sa prédiction à intervalle régulier en comparant la consommation réelle et prédite. Si la valeur prédite est dans un intervalle déterminé de la valeur réelle, alors le programme arrête de continuer à construire le modèle TensorFlow (jugé assez précis) ce qui permet de diminuer la consommation de l'application.

Lorsque la machine est branchée, les données sur l'état de la batterie sont faussés et donc inutilisables, AppliConso va alors prédire la consommation en s'appuyant sur les données fournies par les capteurs ainsi qu'à la correspondance dans le modèle TensorFlow.

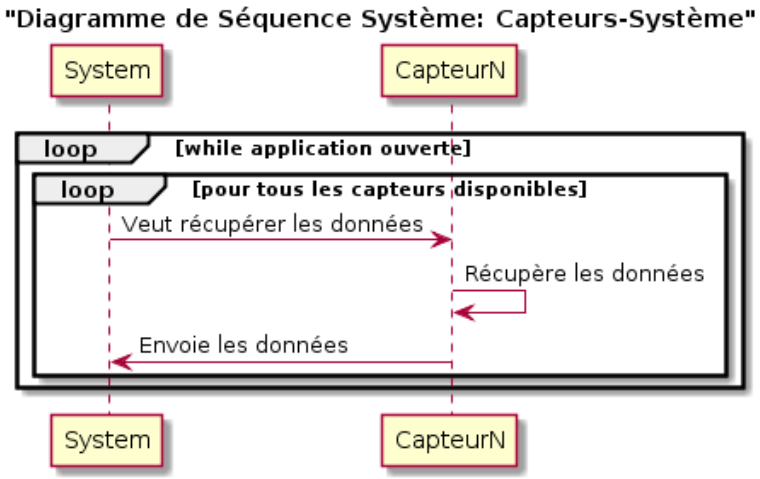
Si l'utilisateur veut la consommation d'une application, la consommation est calculée avec la formule suivante : **consommation(système) - consommation(système-application)**.

D. Affichage de la consommation

L'affichage de la consommation dans l'interface en ligne de commande est implémenté. On affiche la consommation réelle lorsque la machine est débranchée et la consommation prédite lorsque la machine est branchée.

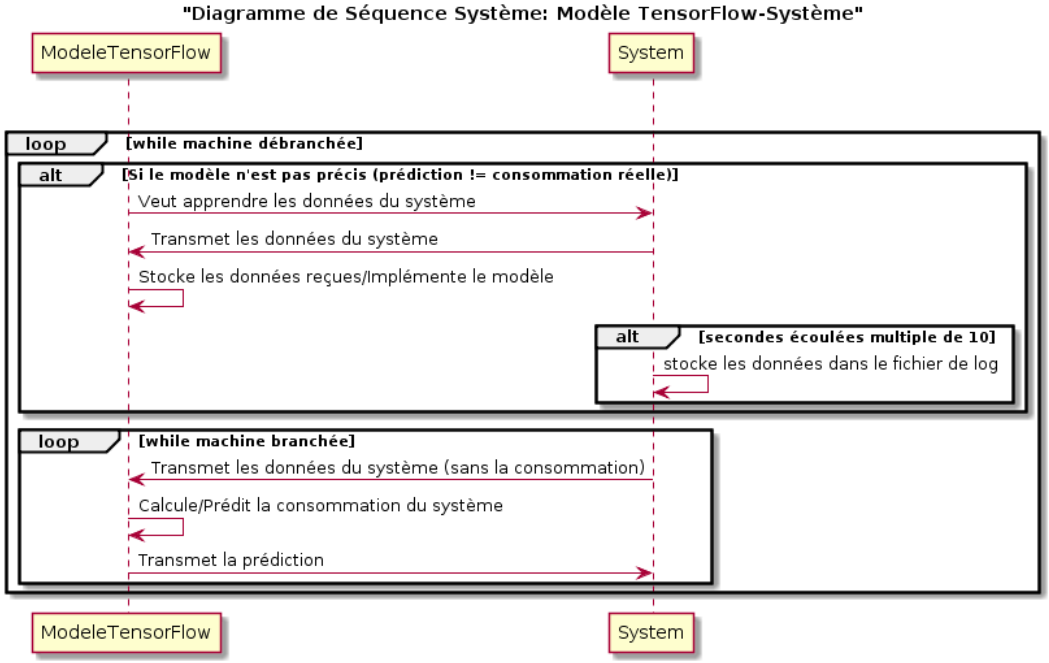
III. Diagrammes dynamiques

A. Diagramme de Séquence Système : Capteurs – Système



On récupère continuellement et à intervalles réguliers les données des différents capteurs disponibles.

B. Diagramme de Séquence Système : Modèle TensorFlow – Système



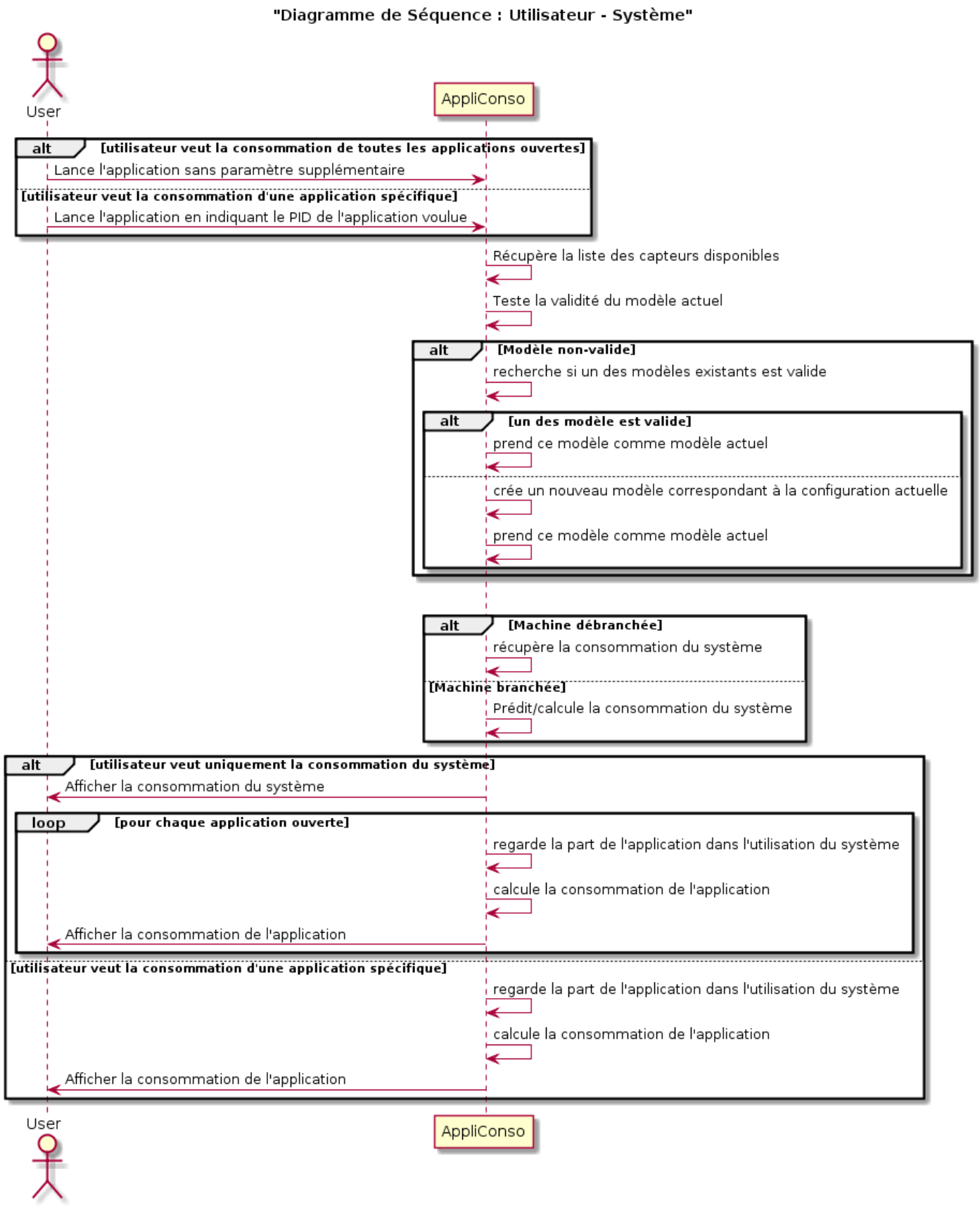
Le modèle TensorFlow d'AppliConso peut se résumer en deux étapes :

Si la machine est débranchée, on récupère un maximum de données utiles à l'évaluation de la consommation (cf DSS Capteurs-Système). Cela, tant que le modèle n'est pas précis, c'est-à-dire tant que la prédiction que l'on peut réaliser sur la consommation à partir des données collectées n'est pas assez proche de la consommation réelle (connue en récupérant les données de la batterie). De plus, toutes les 10 secondes on va enregistrer les données du système dans un fichier log.txt que l'utilisateur pourra consulter.

Si la machine est branchée en revanche, les données que l'on peut récupérer sur la consommation réelle sont faussées du fait de la recharge de la batterie (on récupère donc les données de tous les capteurs disponibles sauf celui sur la batterie), on va donc chercher à prédire la consommation à partir

des données stockées et des données de la configuration actuelle de la machine récupérées sur les capteurs (ex : luminosité, ect…).

C. Diagramme de séquence : Utilisateur – AppliConso



L'interaction entre l'utilisateur et l'application se fait totalement en parallèle des deux autres boucles des diagrammes de séquence système précédents (Capteurs-Système et TensorFlow-Système).

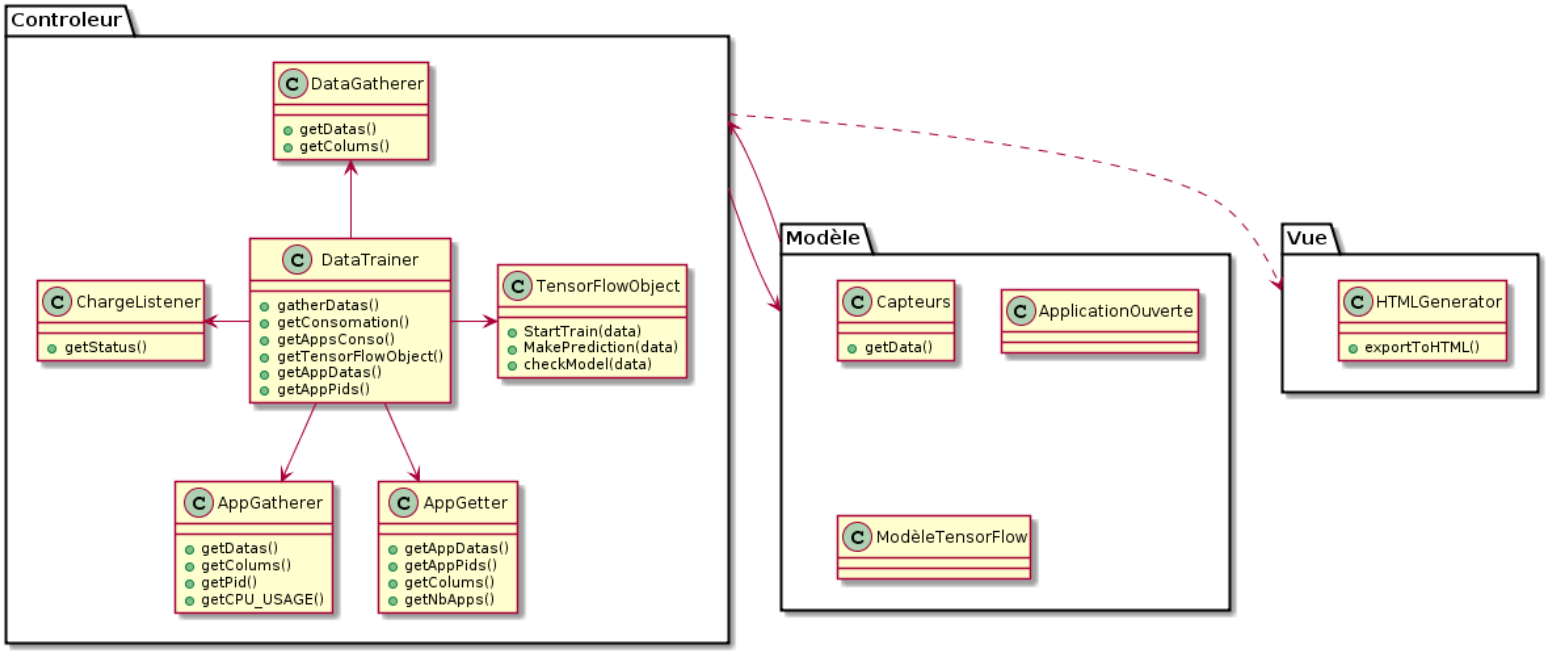
Lorsque l'utilisateur lance l'application, AppliConso regarde les capteurs à sa disposition et récupère ou crée le modèle TensorFlow correspondant. Les boucles Capteurs-Système (permettant de récupérer les données des capteurs en continu) et TensorFlow-Système (permettant d'implémenter le modèle jusqu'à ce qu'il soit assez précis et de prédire la consommation du système) se lancent alors en parallèle et fournissent des données utilisées dans la boucle Utilisateur-AppliConso.

En fonction des options demandées par l'utilisateur, AppliConso affiche les données correspondantes (consommation du système, de toutes les applications, d'une application).

IV. Diagrammes Statiques

A. Diagramme MVC

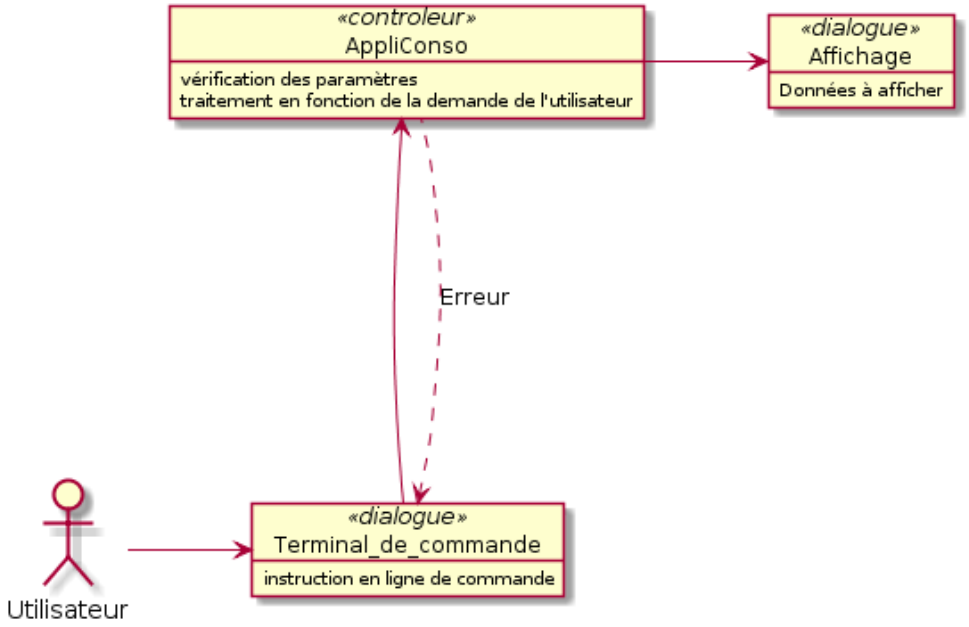
Diagramme MVC



Lorsqu'AppliConso se lance, il lance DataTrainer.py qui est comme le centre de contrôle de l'application. Il permet d'utiliser TensorFlowObject (le contrôleur permettant à l'application d'utiliser un modèle TensorFlow), ChargeListener (contrôlant si la machine est branchée ou débranchée), DataGatherer (qui récupère les différentes données des capteurs) ainsi que AppGetter et AppGatherer (permettant de capter les différentes applications ouvertes ainsi que leurs données correspondantes).

Avec toutes ces informations, DataTrainer peut prédire la consommation du système (ou d'une application) et l'affiche en ligne de commande. Un affichage Web est en cours d'implémentation (class HTMLGenerator).

B. Diagramme DCP



Le diagramme DCP est très basique pour AppliConso, il n'y a peu de dialogues entre l'utilisateur et l'application.

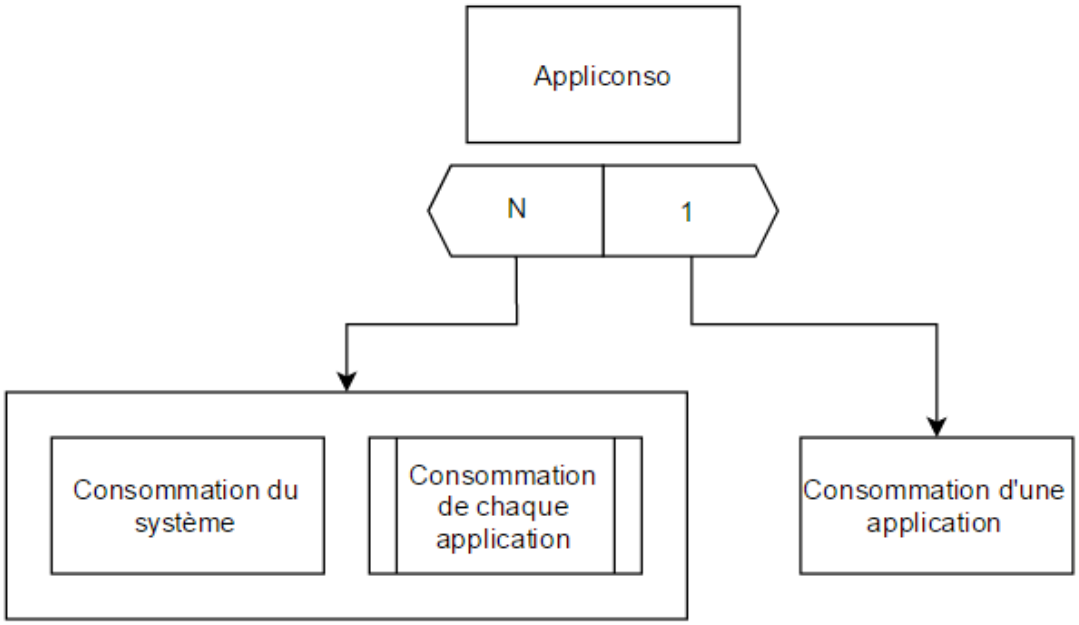
Il est à noter que lorsque l'utilisateur lance AppliConso (avec une instruction en ligne de commande), l'application passe d'abord par une phase où elle vérifie s'il s'agit des bons paramètres de lancement et dans le cas contraire renvoi un message d'erreur à l'utilisateur lui indiquant la bonne méthode d'appel de l'application.

```
if len(sys.argv) > 3:
    print("usage : python AppliConso.py [-w] [Application to monitor]")
    sys.exit()
if len(sys.argv) == 2 and sys.argv[1] == "-w":
elif len(sys.argv) == 3 and sys.argv[2] == "-w":

elif len(sys.argv) > 2:

else:
    print("lancement de l'interface console")
    dataTrainer = DataTrainer()
```

V. Diagrammes d'IHM

A. Diagramme SNI

Le SNI d'AppliConso est très basique, l'utilisateur formule sa demande lors du lancement de l'application (consommation système ou d'une application en particulier) :



B. Diagrammes SEF et SEP

L'affichage de l'application se faisant en ligne de commande, il n'y a pas de diagramme de schéma d'enchaînement des fenêtres.

De la même manière, il n'y a pas de diagramme de schéma d'enchaînement des pages