Ploemeur

Quimper

Stage

BTS Service Informatique aux Organisations 2e année

Liberasys – Husson Consulting SAS

Période du 10 au 21 Décembre 2018 et du 7 Janvier au 1 Février 2019

Vincent CABILLIC

2018-2019

Remerciements

- Je tiens à remercier Gautier HUSSON pour m'avoir accueilli pendant ce stage, ainsi que pour l'accueil et la formation qu'il m'a apporté tout au long du stage.

- Je remercie Ronan, employé, pour les connaissances qu'il m'a apportées.

- Je remercie l'ensemble de la pépinière d'entreprises, plus particulièrement, Laurent, Graziella, Emmanuel, Armand pour leur accueil.

Table des matières

Table des figures

1. La société Liberasys

Liberasys est une entreprise de service informatique. Elle propose des solutions logicielles clés en mains, la mise en place ou l'évolution d'infrastructure réseau et VoIP[1], ainsi que de la maintenance, tout en privilégiant au maximum l'utilisation de logiciel libre.

Un logiciel libre est un logiciel dont l'utilisation, l'étude du code source, la modification et la duplication en vue de sa diffusion sont permises, techniquement et légalement. Le logiciel libre se finance par le service et non pas via les licences.

Liberasys a été créée en 2015, et intervient dans la région de Vannes à Quimper et plus loin grâce au télétravail. Elle se compose de M. Gautier HUSSON Président et d'un salarié arrivé en Janvier 2019. Depuis Mars 2019, elle s'est installée au Pôle d'activité de Technellys à Lanester (56600).

Mon stage quant à lui s'est déroulé à l'Espace Teknica à Ploemeur (56270) qui est une pépinière d'entreprise.

Figure 1: Localisation de stage Liberasys

[1] VoIP est une technique de communication permettant la transmission d'appels audio, vidéo et messagerie instantanée sur des réseaux IP.

2. Introduction

Avec l'augmentation considérable des appareils électriques au 21e siècle, la consommation d'électricité mondiale a fait un bond de 61% entre 2000 et 2016[2]. La transition énergétique a ainsi été engagée, elle vise à modifier notre mode de production et de consommation pour réduire l'impact du secteur de l'énergie sur l'environnement.

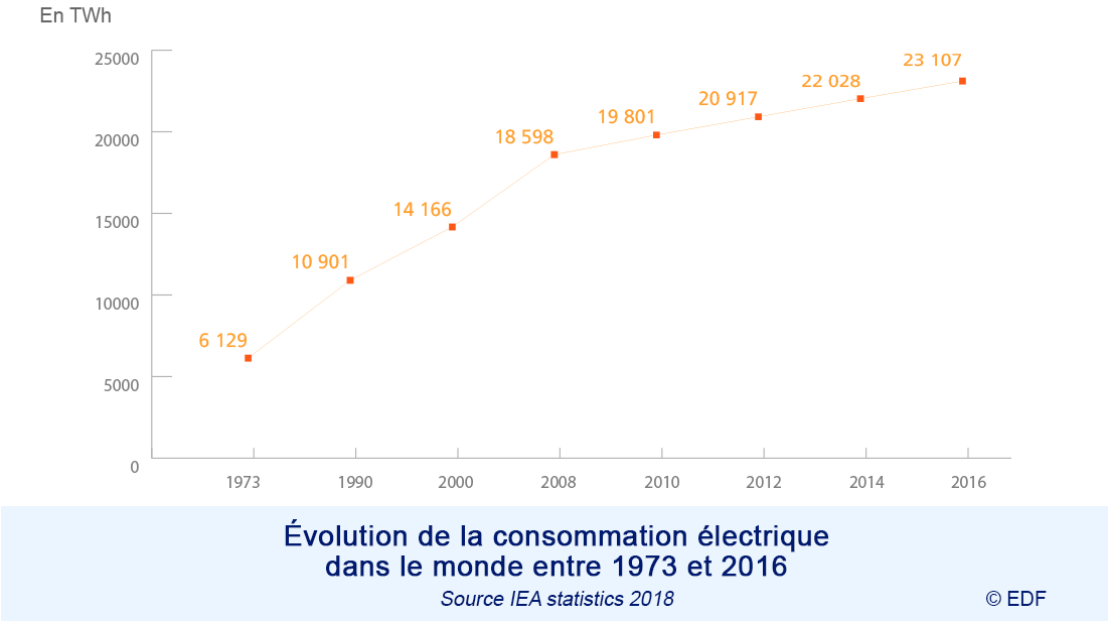


Figure 2 : Consommation électrique mondiale entre 1973 et 2016

A Lorient, le collectif des *Consometers* a été créé en 2017, Liberasys en fait partie. Son objectif est de *rendre visible les consommations d'énergie pour les comprendre et in fine les réduire*[3]. Le collectif souhaite aussi mettre en avant les logiciels libres, et faciliter l'accès aux données énergétiques, le tout dans un but économique, pédagogique et écologique, au niveau de la région Bretagne.

Pour pouvoir modifier notre consommation, il faut d'abord pouvoir l'analyser. Ces informations de consommations peuvent être fournies par les compteurs électriques. Avec ces informations, il est mis en place une surveillance (ou monitoring) de l'infrastructure électrique. Elle inclut la consommation et/ou la production de l'infrastructure. Pour cela Liberasys a développé une application en python traitant les données fournies par un compteur électrique. Ces données sont accessibles via une API[4] qui permet d'alimenter une Interface Graphique. Zabbix est le logiciel utilisé pour l'affichage de ces données, il permet entre autre d'afficher des graphiques en temps réels de consommations. Toutefois le logiciel python n'a été développé que pour les compteurs de type PME-PMI.

[2] Source edf.fr et image : tinyurl.com/yy3sa9bj
[3] Citation Consometers : tinyurl.com/y4t7oygw
[4] API ou interface de programmation applicative est un ensemble de code informatique offrant des services à d'autres logiciels.

3. Problématique

Afin d'améliorer la surveillance des réseaux électriques, il faut que le logiciel puisse gérer plusieurs types de compteurs numériques électriques. Pour faciliter son paramétrage, un fichier de configuration est nécessaire.

La création de ce logiciel a été engagée à la suite d'une initiative de la ville de Lorient (56), où les Consometers ont répondu présents. Cette initiative en faveur de la transition énergétique vise à surveiller la consommation des bâtiments publics et administratifs, certaines installations possèdent aussi un potentiel de production grâce à des panneaux solaires.

En effet depuis 2015, ENEDIS déploit de nouveaux compteurs. Le compteur LINKY est un nouveau compteur numérique, intégrant diverses fonctionnalités. Pour les distributeurs, transporteurs et producteurs d'énergies, son utilité est de mieux répartir la charge du réseau, grâce aux données collectées. Lorsqu'il y a un déséquilibre entre la production et la consommation, les données servent à mieux cibler les zones et les utilisateurs devant moins consommer, via des dispositifs comme EcoWatt[5]. Certaines installations ont une priorité d'alimentation tels les hôpitaux, les infrastructures de transport, les grands industriel, etc (Annexe 1). Ces données servent aussi au client qui peut surveiller sa consommation et sa production avec une interface graphique fourni par ENEDIS.

Il devient nécessaire de faire évoluer le logiciel, la surveillance se limitant au compteur PME-PMI et d'ajouter la compatibilité LINKY. De plus, le nombre de nouveaux compteurs installés est passé à 25 000 par jour fin 2017[6].

4. Objectifs

Le but de ce stage a été d'intégrer et de développer de nouvelles fonctionnalités afin d'ajouter la compatibilité avec les compteurs LINKY. A ceci s'ajoute la création de documentations au programme existant, ainsi qu'aux autres composants développés. On ne s'occupera pas de l'interface graphique et de l'affichage en temps réel.

4.1. Eléments existants

Le logiciel est développé en python, pour fonctionner sur une carte programmable Raspberry PI. Cette carte intègre Raspbian, une distribution du système d'exploitation Debian (Linux). Une documentation est fournie pour l'installation et la configuration du logiciel. Pour comprendre son fonctionnement, le code est commenté. Il y a aussi un schéma de l'automate à état, gérant les trames que le logiciel reçoit. Les fichiers sont versionnés et stockés sur une forge[7].

[5] Initiative en région Bretagne et Provences Alpes Cotes d'Azur prévenant les inscrits qu'il faut réduire sa consommation lors des période d'instabilité.
[6] Source LaCroix.com : https://tinyurl.com/y3gh3qoo
[7] Source Wikipédia : Une forge est un système de gestion de développement collaboratif de logiciel.

4.2. Principe de fonctionnement

Le fonctionnement est le suivant, le PI est lié par USB à un module télé-information puis au borne TIC du compteur. Le module sert à transformer le signal généré par le compteur vers un protocole COM, les informations reçues sur le PI sont des octets. Ensuite, le programme traite les informations qu'il reçoit du compteur et les fournit de façon structurée via l'API web. La TIC, c'est la Télé-Information Client, un protocole particulier utilisé par les compteurs numériques pour transmettre des informations. Ces informations sont structurées et transmises à intervalle régulier. Les octets reçus sont encodés au format ASCII.

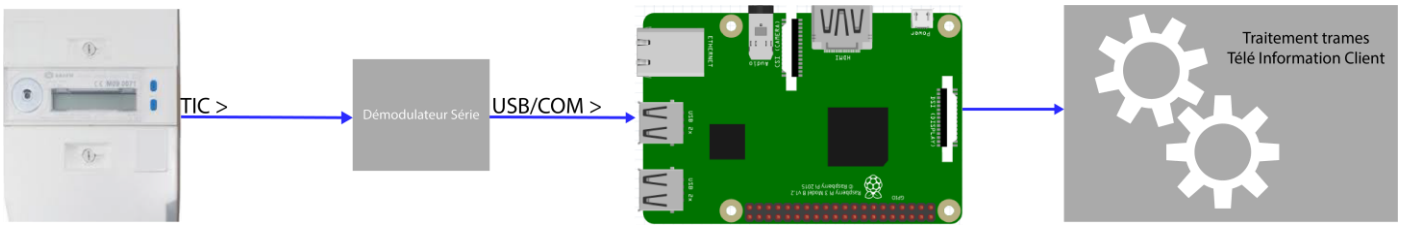


Figure 3 : Schéma du lien physique vers le traitement informatique

Les informations structurées, un automate à état gère les octets reçus et les assemble en paquets cohérents, ce sont des trames (Annexe 2). Ces trames contiennent des groupes composés d'une étiquette qui est le nom du champ et de sa valeur associée. En fonction du compteur, il peut y avoir un horodatage. On retrouve en information par exemple le numéro du compteur, l'option tarifaire ou l'intensité utilisée.

Structure des trames		
Début de trame «Start of TeXt» «STX» 0x02	Corps de la trame composée de groupes d'informations	Fin de trame «End of TeXt» «ETX» 0x03

Structure d'un groupe								
Début groupe «LF» 0x0A	Etiquette	Séparateur *	Horodate **	Séparateur **	Données	Séparateur	Controle/ Checksum	Fin Groupe «CR» 0x0D
	Zone contrôlée par la checksum ***							

La structure change en fonction du type (PMEPMI, LINKY...) et du mode du compteur (standard, historique)
* Espace «SP» (0x20) ou Horizontal Tab «HT» (0x09)
** Présent sur Linky, si le groupe comprend un horodatage voir «description des trame de télé-information
*** Zone de contrôle général, peut changer
Pour de plus amples informations, voir documentation techniques de compteurs

Figure 4 : Structure des trames et d'un groupe

4.3. Décomposition du traitement pour les compteurs de type PME-PMI

Le traitement a le fonctionnement suivant, un processus parallèle nommé ‘lecture_série’ reçoit les octets qu’envoie le compteur. A chaque nouvel octet reçu, l’objet ‘decodeur_trames’ le fait passer dans la machine à état déterminant s’il est à inclure dans un groupe de caractère ou si la trame est finie. Si la trame n’est pas complète ou a des erreurs, elle ne sera pas interprétée. Une fois la trame terminée, elle est interprétée par ‘interpreteur_trames’ qui va analyser les groupes de caractères de la trame et les attribuer dans un dictionnaire. Ce dictionnaire est structuré et représente les informations possibles que le compteur peut envoyer, il est créé à partir de la documentation technique du compteur type PME-PMI. Enfin, cette structure de données est mise à disposition via l’API Flask. Flask est un paquet python permettant la création d’un serveur web. En fonction des paramètres passé en URL, on obtient le dictionnaire entier, les données demandées en format JSON (Annexe 4) ou une donnée précise (Annexe 3).

4.3.1. Création de documentation

La documentation a deux buts. Le premier a été d’apprendre le langage python et les concepts utilisés dans le programme. De plus, ceci m’a permis de comprendre le fonctionnement des rappels de fonctions (callback), de la machine à état et des processus parallèles (threads). Le second est de fournir des éléments pour le développement futur du programme et de faciliter la prise en main du code, pour les prochains développeurs.

4.3.2. Compatibilité Linky

L’évolution comprend l’adaptation et la création de nouveaux paquets. Pour le décodage des trames, il a été modifié l’automate à état et le calcul du checksum. Pour l’interpréteur de trame il a fallu adapter le dictionnaire avec la description des trames. Ensuite, il faut charger les paquets correspondants au compteur et ce grâce au fichier de configuration. Il y aura donc un paquet principal appelé ‘api_compteur_tic’ et les paquets s’occupant des types de compteurs ‘decode_pmepmi’ et ‘decode_linky’. Toutefois, certaines classes sont redondantes.

4.4. Choix des outils

Liberasys mettant en avant le logiciel libre, tous les logiciels de développement sont eux aussi libre. Le code source l'est lui aussi. Pour la documentation, le format markdown '.md' est utilisé. Il permet de structurer la documentation et est utilisé sur plusieurs plateformes.

Figure 5 : Exemple documentation markdown

Le code a été créé avec les éditeurs geany (Raspbian) et atom (Windows). Je travaillais en parallèle sur le rapberry PI mis en place pour développer en tant que poste individuel et mon pc portable. Les données étaient synchronisées et historisées grâce à Owncloud, une solution de stockage en nuage. Les diagrammes et schémas ont été réalisés avec LibreOffice Draw ou Dia, un éditeur de diagramme libre. Les sources de ces dessins sont inclues dans le projet pour laisser la possibilité de modifier la documentation. La gestion de version était gérée avec Git.

5. Première partie : Documentation

La documentation a commencé par l'apprentissage du langage python et ces formalismes. Pour cela, j'ai utilisé trois sources : le mooc Python du site OpenClassrooms, les notes de M. HUSSON sur la documentation python et les règles de programmation PEP 8. En parallèle, j'ai analysé le programme de gestion PME-PMI pour comprendre son fonctionnement. Je me suis aussi informé sur les notions suivantes : processus, fork(processus), pointeur, pointeur de fonctions et rappel de fonction (callback).

5.1. Automate à état

La partie la plus compliquée à comprendre a été l'automate à état. Un automate à état est un modèle de calcul mathématique représenté par un nombre fini d'états, mais possédant un seul état à la fois. Un état, à un moment donné, est appelé 'état courant'. La 'transition' est le passage d'un état à un autre, activé par un évènement ou une condition. Un exemple simple est l'ascenseur, il y a 3 états : arrêter, monter et descendre. Les transitions sont déclenchées à l'aide du panneau de contrôle interne ou externe à l'ascenseur. M. HUSSON a créé l'automate suivant pour le compteur de type PME-PMI.

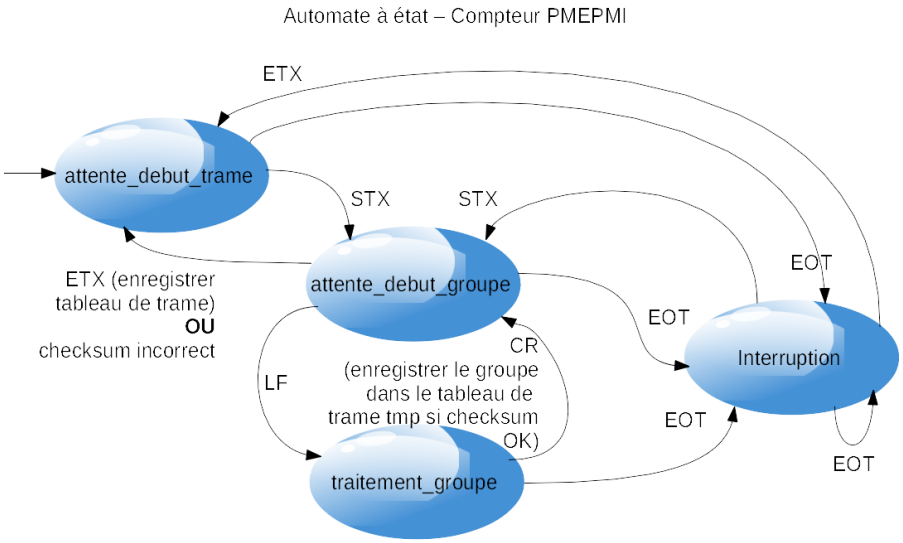


Figure 6 : Automate à état - Compteur PMEPMI (LibreOffice Draw)

On retrouve comme expliqué dans le chapitre 4.3 le traitement de trame. Les flèches représentent les transitions, et les libellés le caractère qui les déclenchent comme expliqué dans la Figure 4 : Structure des trames et d'un groupe (p.8). Le compteur PME-PMI a une particularité représentée par l'état 'interruption' et déclenché par le caractère ASCII « EOT ». Cela arrive lorsque la liaison série est interrompue, par exemple quand un agent vient relever les informations du compteur. Le comportement de la liaison série de ce type de compteur est expliqué dans la documentation technique ENEDIS.

Après avoir compris son fonctionnement, j'ai adapté l'automate pour fonctionner avec LINKY. J'ai décidé de séparer le traitement de chaque type de compteur dans une classe unique. Le schéma est le suivant.

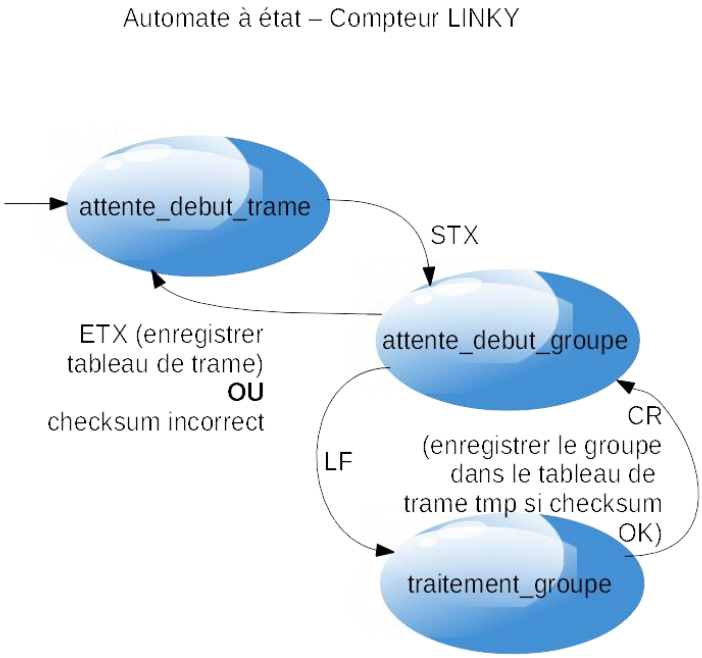


Figure 7 : Automate à état - Compteur LINKY (LibreOffice Draw)

L'automate gérant LINKY n'inclut pas d'interruption, en effet sa documentation n'en parle pas. Cela a été confirmé, quand j'ai appelé un spécialiste chez ENEDIS, puis le LinkyLab, le laboratoire de Recherche et Développement du LINKY. Le bus série du LINKY normalement envoie des données en continue et ne subit pas d'arrêt. Et une batterie permet de garder en mémoire les données.

5.2. Modélisation du programme

Avant de développer, j'ai modélisé les relations entre les objets héritant de classe. Cette modélisation est utile afin de mieux comprendre les liens entre les fonctions, quelles fonctions fait appel à l'une ou l'autre. Une fonction peut en faire appel à plusieurs.

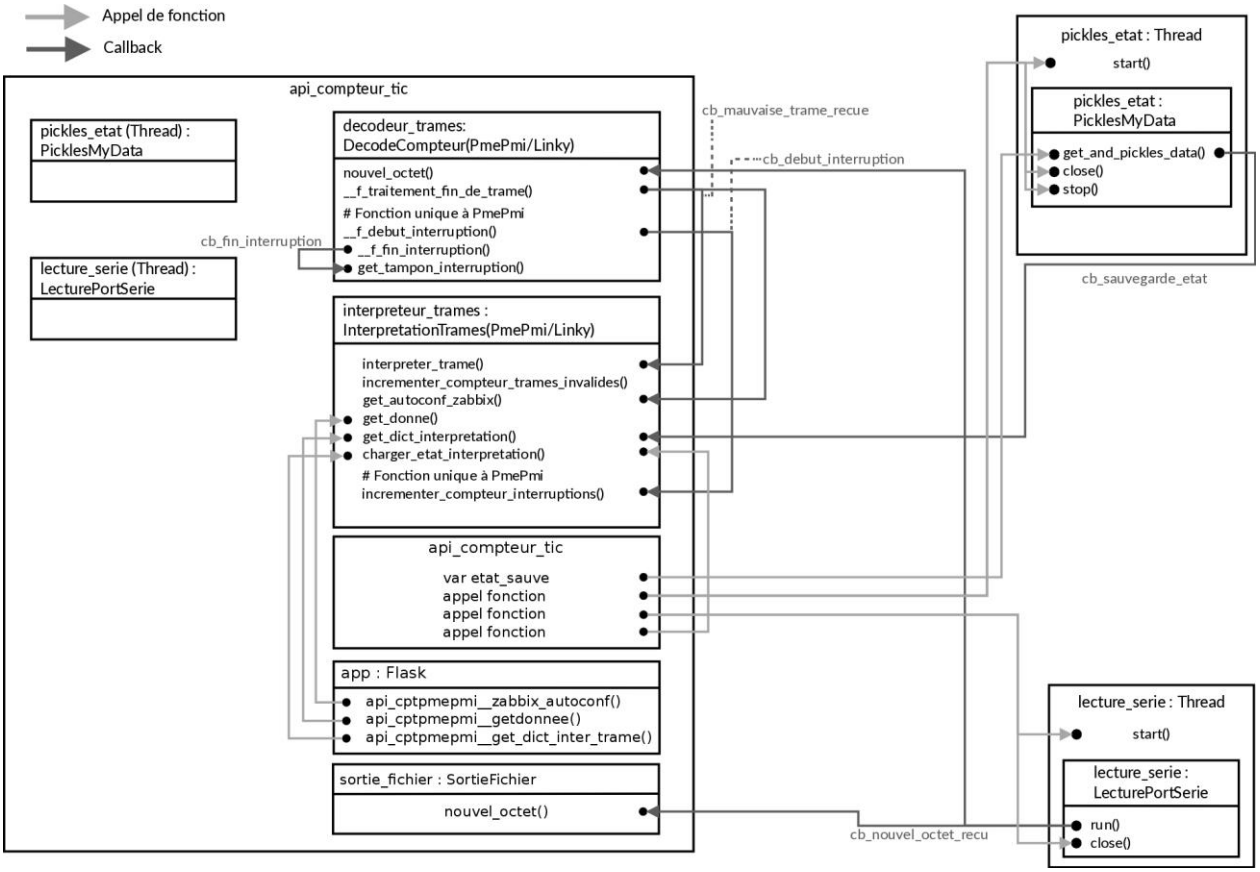


Figure 8 : Schéma lien objet api_compteur_tic (Dia)

Ce diagramme représente les Threads avec et le ou les objets qu'ils comportent après instanciation.
Les flèches en gris clairs sont des appels de fonctions, qui lancent des traitements ou affectent des variables.
Les flèches en gris foncés sont des rappels de fonctions (callback).
Quand un callback est affecté, il est possible d'appeler la fonction d'une autre classe (avec des paramètres) indépendamment du Thread.
Exemple : Le Thread contenant l'objet de **lecture série** à un callback qui est appelé à chaque nouvel octet reçu. Ce callback est paramétré pour appeler la fonction **nouvel_octet()** de l'objet **decodeur_trames**.

6. Deuxième partie : Développement

Lors du développement après la création d'un nouveau repository[8], j'ai d'abord déterminé les paquets à créer et à modifier :

- L'automate à état étant différent pour les deux compteurs ainsi que le contenu des trames, j'ai décidé d'ajouter un paquet gérant le type LINKY 'decode_linky' en plus du paquet 'decode_pmepmi' existant. Ces paquets ont les mêmes classes, toutefois ils sont adaptés à l'automate et à la description des trames. Il a fallu modifier les classes 'decodeur_trames' et 'interpreteur_trames' du paquet 'decode_linky'.
- Le programme comportant plusieurs types de compteurs, le fichier de configuration 'api_compteur_tic.conf' a nécessité des modifications. Une première partie est le choix du type de compteur et la seconde est la configuration de la liaison série.
- Enfin la modification du fichier de configuration et l'ajout de paquet a demandé de modifier le paquet principal 'api_compteur_tic'. Les changements effectués sont : la lecture du fichier de configuration où on a ajouté le type de compteur, et le chargement du paquet correspondant au compteur sélectionné.

6.1. Automate à état et Desctription de trames

La classe 'decodeur_trames' contient l'algorithme s'apparentant à l'automate a état. Comme pour LINKY, il n'y a plus d'interruption de trames, j'ai supprimé les transitions d'états inutiles. Ces transitions sont stockées dans un dictionnaire et en fonction de la valeur de transition reçue, l'algorithme associe un nouvel état et déclenche des fonctions.

Exemple : Quand le programme reçoit le caractère « Start of TeXt » (0x02), l'état courant est affecté en attente de début de groupe et les données de trames stockées dans un tableau sont remise à zéro.

```
self._dict_transitions[self._ID_ETAT_ATTENTE_DEBUT_TRAME]  [self._CHAR_STX] =   \
    (self._ID_ETAT_ATTENTE_DEBUT_GROUPE,
     self.__f_noop,
     self.__f_noop,
     self.__f_raz_nouvelle_trame)
```

Figure 9 : Exemple dictionnaire de transition d'état

[8] Repository, c'est une archive de code ici stockée sur une forge: https://github.com/Liberasys/api_compteur_tic

En plus du dictionnaire, j'ai modifié le calcul du checksum. Cette valeur, présente en fin de groupe de trame permet de la valider. Si le checksum est différent du calcul alors le groupe est erroné.

La structure d'une trame peut parfois avoir un horodatage. Au préalable, la trame a été séparé en valeur unique dans un tableau, sans la valeur de checksum. Si le calcul du checksum est bon, on ajoute les valeurs d'étiquette et de données, si le tableau contient 3 items, il y a l'horodatage en plus.

A chaque fois que le caractère « End of TeXt » est émis, la trame est envoyée pour interprétation à la fonction 'interpreter_trames' de la classe 'interpreteur_trames' via un callback.

Dans la classe 'interpréteur_trames', j'ai modifié la variable 'config_champs' qui est un tableau répertoriant les éléments qu'émet la TIC, et des informations facilitant le traitement comme son type (numérique ou non).

```
# <etiquette> : (<est numerique>, <est horodate>, <unite donnee>, <description>)
self._config_champs = {"ADSC":      (False, False, self._unite_donnee_linky['none'], "Identifiant du compteur"),
                       "VTIC":      (False, False, self._unite_donnee_linky['none'], "Version de la TIC"),
                       "DATE":      (False, False,  self._unite_donnee_linky['none'], "Date et heure courante"),
                       "NGTF":      (False, False, self._unite_donnee_linky['none'], "Nom du calendrier tarifaire fournisseur"),
                       "LTARF":     (False, False, self._unite_donnee_linky['none'], "Libelle tarif fournisseur en cours"),
                       "EAST":      (True,  False, self._unite_donnee_linky['wh'],   "Energie active soutiree totale"),
```

Figure 10 : Exemple description de trames

Comme précédemment, j'ai supprimé le code inutilisé de cette classe et je l'ai modifié pour être cohérent avec les informations de la documentation. Ce code est celui qui créer le dictionnaire que fournit l'API. Il compare 'config_champs' et la trame qu'il a reçu. Ensuite, il prend chaque groupe et structure chaque donnée. Une mutex est validée avant la modification du dictionnaire, pour ne pas modifier une variable qui pourrait être lu en même temps. Ceci permet aux autres instances du programme d'accéder à cette donnée, lorsqu'elle n'est pas utilisée.

6.2. Fichier de configuration

Le fichier a des entrées pour la configuration de la liaison série, car celle du LINKY diffère du PMEPMI. Pour choisir l'un ou l'autre, on commente ou décommente les lignes. De même pour le choix du type de compteur.

```
# linky :
port = /dev/ttyUSB0
baudrate = 9600
bytesize = 7
parity = even
stopbits = 1
xonxoff = false
rtscts = false
dsrdtr = false
timeout = 1
```

Figure 12 : Configuration série LINKY

```
# Type de compteur possible : "linky" ou "pmepmi"
#type_compteur="pmepmi"
type_compteur="linky"
```

Figure 11 : Configuration type de compteur

6.3. Programme principal

Maintenant que le fichier de configuration comporte un nouveau champ, il faut que le programme le lise pour charger les bons paquets. Le chargeur de configuration a été modifié, on a accès à une variable contenant le type de compteur. Lorsque le programme instancie les objets de décodage et d'interprétation des trames, une condition vérifie au préalable le type de compteur et charge les bons paquets.

```
# Importation des paquets necessaire au traitement des trames en fonction du type de compteur
# Instanciation des objets avec la classe utilise par le type de compteur
if type_compteur == 'linky':
    from decode_linky import DecodeCompteurLinky
    from decode_linky import InterpretationTramesLinky
    decodeur_trames = DecodeCompteurLinky()
    interpreteur_trames = InterpretationTramesLinky()
elif type_compteur == 'pmepmi':
    from decode_pmepmi import DecodeCompteurPmePmi
    from decode_pmepmi import InterpretationTramesPmePmi
    decodeur_trames = DecodeCompteurPmePmi()
    interpreteur_trames = InterpretationTramesPmePmi()
```

Figure 13 : Chargement des classes en fonction du type de compteur

Pour finir, j'ai uniformisé le nom des variables pour qu'elle soit explicite, les précédentes faisant seulement référence au compteur PMEPMI. Maintenant, elles sont neutres et désignent le compteur en général.

7. Résultats et tests

Enfin, le programme a été mis en environnement de tests dans l'un des bâtiments administratifs de la ville de Lorient. Le PI est connecté au LINKY, et l'on recevait des trames. Toutefois, les premiers traitements n'ont pas fonctionné comme prévue. En effet, dans les commentaires, j'ai utilisé des caractères spéciaux tel que les lettres accentuées, mais Python en version 2.7 ne les compilent pas. Il a fallu corriger tous ces caractères.

Après avoir réglé les problèmes de caractères, pour vérifier le fonctionnement l'API a été liée à Zabbix. Résultat, on a bien obtenu une courbe de consommation.

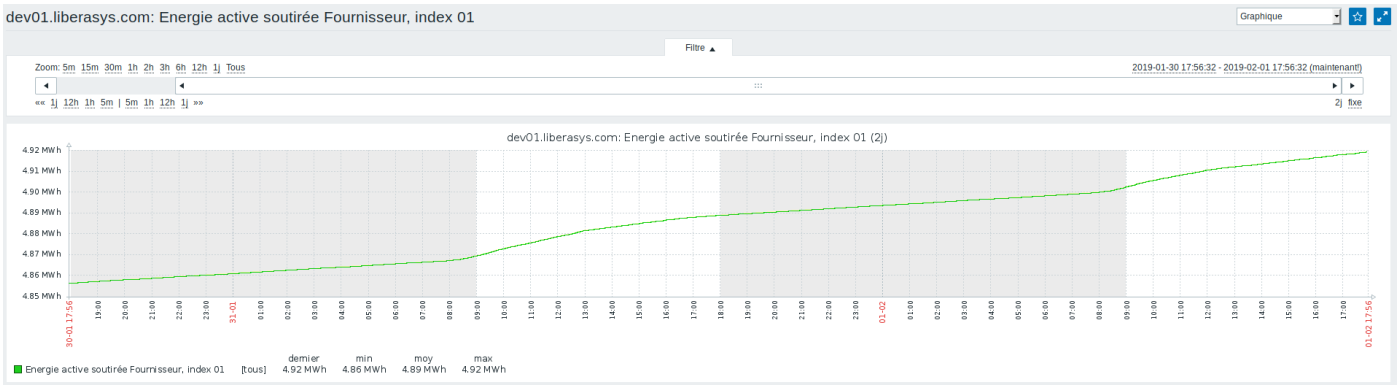


Figure 14 : Courbe de consommation LINKY

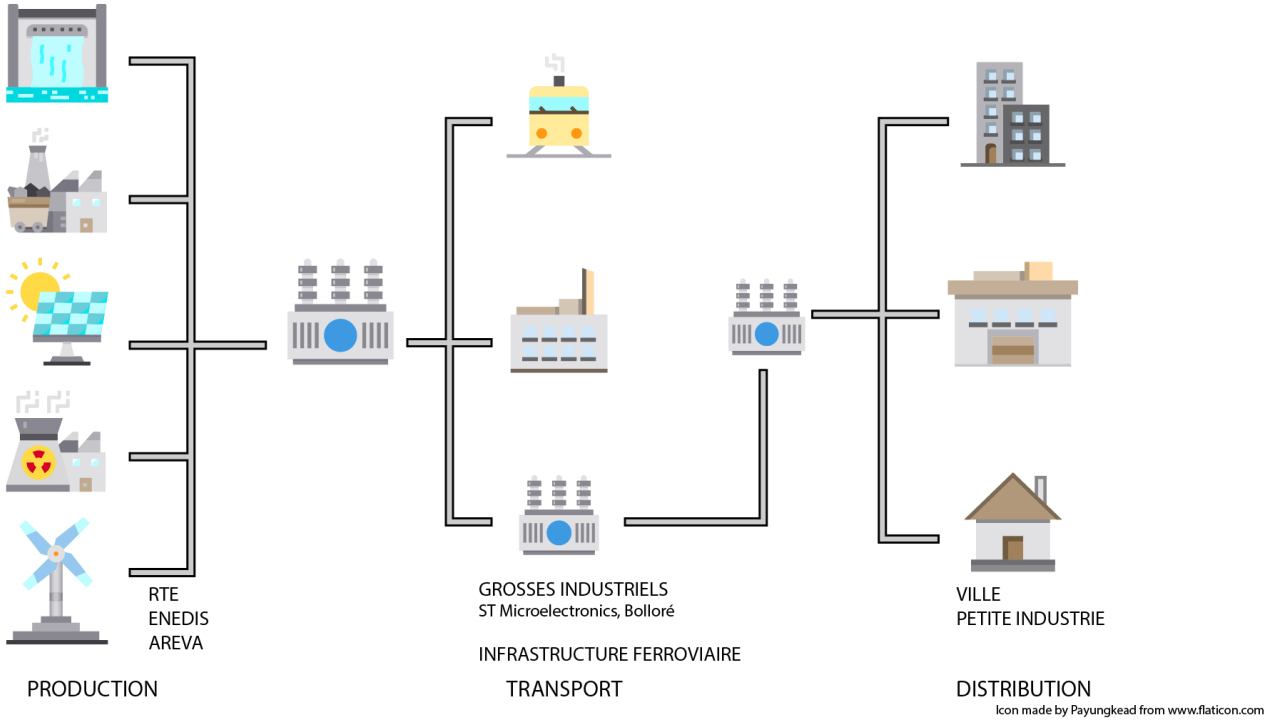
8. Conclusion

Le logiciel nécessite encore quelques améliorations, que j'ai envisagé avant le développement : l'automatisation du chargement des configurations via un paquet dédié et non plus du code unique. Et comme, il y a une redondance des classes dans les paquets de gestion 'decode_pmepmi' et 'decode_linky', on peut envisager une structure en héritage de classe et éviter la duplication de code.

Ce stage m'aura apporté de nouvelles connaissances techniques : j'ai appris un nouveau langage de programmation, le Python. J'ai par la même occasion utilisé un système de sauvegarde en nuage qui m'a permis de récupérer des travaux supprimés par inadvertance. Cette expérience aura été avant tout humaine, et je tiens à remercier toutes les personnes qui ont rendu ces six semaines aussi intéressantes et enrichissantes.

9. Annexes

9.1. Annexe 1 : Acteurs du réseau électrique Français



9.2. Annexe 2 : Exemple d'une trame transmise par la TIC

```
ADSC    031864527974    E
VTIC    02      J
DATE    H181212140256           B
NGTF            BASE            <
LTARF           BASE            F
EAST    003818598       9
EASF01  003818598       L
EASF02  000000000       #
EASF03  000000000       $
EASF04  000000000       %
EASF05  000000000       &
EASF06  000000000       '
EASF07  000000000       (
EASF08  000000000       )
EASF09  000000000       *
EASF10  000000000       "
EASD01  003818598       J
EASD02  000000000       !
EASD03  000000000       "
EASD04  000000000       #
IRMS1   006     4
URMS1   237     F
PREF    12      B
PCOUP   12      \
SINSTS  01465   V
```

9.3. Exemples de résultats de requête API

9.3.1. Annexe 3 : Consommation totale d'un compteur

Requête API : http://127.0.0.1:5000/get_donnee?tarif=INDEP_TARIF&etiquette=CONSO_TOTALE_s

336490

9.3.2. Annexe 4 : Interprétation d'une trame en format JSON

Lien d'accès à l'API du PI : http://127.0.0.1:5000/get_interpretation


```
{
  "HCE": {
    "EAP_s": ["59971", "kWh"],
    "ER+P-1_s": ["23105", "kvarh"],
    "ER+P_s": ["23222", "kvarh"],
    "ER-P-1_s": ["282", "kvarh"],
    "ER-P_s": ["282", "kvarh"],
    "PMAX_s": ["10", "kVA"]},
  "HCH": {
    "EAP_s": ["36580", "kWh"],
    "ER+P-1_s": ["9825", "kvarh"],
    "ER+P_s": ["9835", "kvarh"],
    "ER-P-1_s": ["707", "kvarh"],
    "ER-P_s": ["707", "kvarh"],
    "PMAX_s": ["17", "kVA"]},
  "HPE": {
    "EAP_s": ["147907", "kWh"],
    "ER+P-1_s": ["51402", "kvarh"],
    "ER+P_s": ["51638", "kvarh"],
    "ER-P-1_s": ["1670", "kvarh"],
    "ER-P_s": ["1683", "kvarh"],
    "PMAX_s": ["10", "kVA"]},
  "HPH": {
    "EAP_s": ["92033", "kWh"],
    "ER+P-1_s": ["19085", "kvarh"],
    "ER+P_s": ["19096", "kvarh"],
    "ER-P-1_s": ["3371", "kvarh"],
    "ER-P_s": ["3376", "kvarh"],
    "PMAX_s": ["19", "kVA"]},
  "INDEP_TARIF": {
    "CONSO_TOTALE_i": ["0", null],
    "CONSO_TOTALE_s": ["336491", null],
    "CONTRAT": ["BT 4 SUP36", null],
    "CPT_INTERRUPTIONS": ["0", null],
    "CPT_PREAVIS": ["0", null],
    "CPT_TRAMES_INVALIDES": ["0", null],
    "DATE": ["15/01/19 11:56:59", null],
    "DebP": ["14/01/19 02:15:00", null],
    "DebP-1": ["14/12/18 06:25:00", null],
    "EAPP_s": ["568", "VAh"],
    "EAPP_s_delta": ["38", "VAh"],
    "EA_s": ["560", "Wh"],
    "EA_s_delta": ["37", "Wh"],
    "ER+_s": ["24", "varh"],
    "ER+_s_delta": ["2", "varh"],
    "ER-_s": ["6", "varh"],
    "ER-_s_delta": ["0", "varh"],
    "FinP-1": ["14/01/19 02:15:00", null],
    "ID_COMPTEUR": ["031436188395", null],
    "PA1MN": ["16", "kW"],
    "PS": ["48", "kVA"],
    "PTCOUR1": ["HPH", null],
    "TGPHI_s": ["0.02", ""]}
}
```