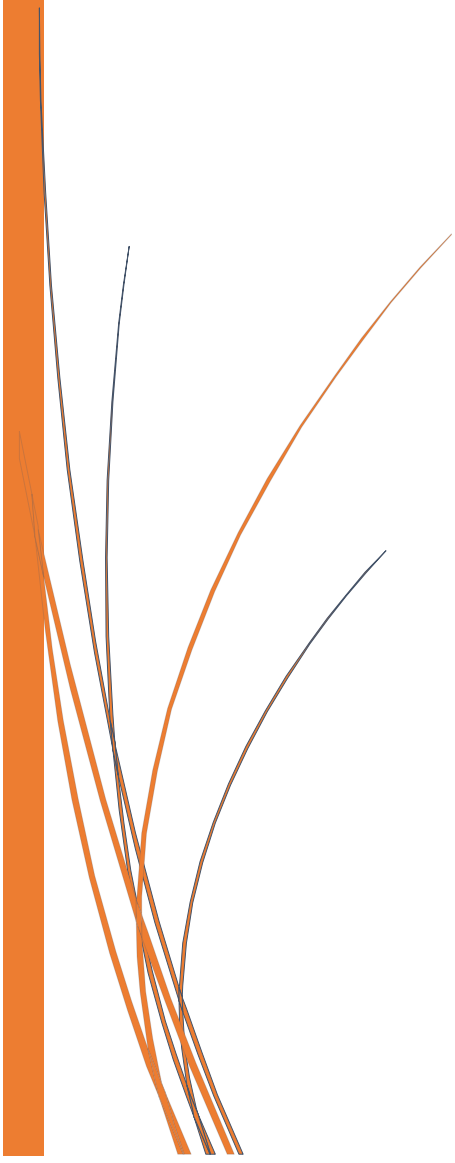
Rapport de stage

Dans la société : Ocean Data System

M1. Master en Systèmes Embarqués et Systèmes Intégrés
Projet Master 1
Étudiant : JAVIER LÓPEZ José Antonio
N° Etudiant : 21804920
Date de livraison : 9 juin 2019
Professeur : Johann Laurent



Faculté sciences & sciences de l'ingénieur
Université Bretagne Sud

Sommaire

1. Introduction

Du 1er Avril 2019 au 23 Août 2019, j'ai effectué un stage chez Ocean Data System à Lorient. La société est spécialisée dans l'électronique marine. J'ai donc eu l'occasion de me familiariser avec le monde marin et la programmation orientée objet pour les systèmes embarqués en utilisant l'IDE Visual Studio.

L'objectif de ce document est d'expliquer le travail accompli jusqu'à présent dans l'entreprise ainsi que les défis sur lesquels je vais travailler jusqu'à la fin du stage.

2. L'entreprise et son secteur d'activité

Ocean Data System est une société spécialisée en électronique et informatique marine créée en 1991 et basée à Lorient (France). L'entreprise a deux axes stratégiques : le développement de systèmes spécifiquement adaptées aux besoins des clients et la revente de produits électroniques proposés par des sociétés externes (ODS, 2019).

3. Les travaux effectués et les apports du stage

Le principal produit est la plateforme UpSideUp (USU) qui permet de réaliser diverses fonctionnalités comme l'anti-chavirage, la mesure et le contrôle des efforts dans le gréement, la gestion des situations critiques, la détection d'envahissement, la gestion d'alarme et d'autres fonctionnalités. Il existe 3 versions différentes : Easy, Master et SuperYacht. Afin d'afficher les données traitées par l'USU sur un ordinateur, l'entreprise a développé une Interface Homme Machine (IHM) personnalisée.

Toutes les versions de USU utilisent la carte de développement GHI 400 et toutes sont développées en utilisant la programmation C#. L'USU dispose d'une mémoire MicroSD pour stocker toutes les informations du système comme la configuration et les événements qui se sont produits.

Les projets qui me sont confiés ont pour but l'optimisation et l'ajout de fonctionnalités à l'USU. Les projets suivants ont donc été réalisés :

3.1. Création d'une bibliothèque pour la configuration de calcul complexe

Pour configurer une suite d'opérations, celles-ci devaient être gérées de façon consécutive et indépendante. Le développement de cette bibliothèque permet à l'utilisateur d'écrire une formule avec différentes opérations sur une seule ligne.

Il y a des bibliothèques spécifiques qui permettent de reconnaître des patrons ou même d'obtenir le résultat d'une formule écrite comme String. Cependant, en utilisant **.Net Micro Framework**, ce type de bibliothèques n'était pas compatible. Nous avons donc dû développer un algorithme complet pour la détection, la structuration et l'obtention de la valeur de la formule écrite comme une String.

Les six fichiers suivants ont été utilisés pour effectuer cette opération :

- "Formula Manager". Il se compose de deux méthodes. La première consiste à enregistrer dans une ArrayList la structure de la formule et. La seconde méthode est chargée d'exécuter toutes les opérations contenues dans la ArrayList.
- "Calcul". ". Il se compose de deux méthodes. La première consiste à obtenir la structure initiale

de la formule. La seconde méthode est chargée d'exécuter le calcul de la structure précédemment obtenue.

- "Parameter". Il se compose de trois méthodes. La première est chargée de créer un nouvel objet de calcul puisque le paramètre est une sous-formule. La seconde méthode permet de récupérer la valeur du paramètre. La troisième méthode nous permet de choisir l'exécution de la première ou de la deuxième méthode selon les conditions de la formule.
- "FormulaHandler". Cette classe est statique et nous permet d'analyser la formule afin de trouver sa structure. Cette méthode nous permet d'analyser si la formule contient la bonne quantité de parenthèses et si elle est bien écrite. Elle permet d'obtenir l'opérateur et les paramètres qui composent l'opération. Les paramètres obtenus sont analysés, s'il s'agit d'une sous-formule, un flag est levé pour que la méthode de la classe "Parameter" puisse extraire les paramètres de cette sous-formule.
- "Operation". Cette classe permet d'obtenir les valeurs de la formule et d'effectuer le calcul mathématique correspondant.
- "OperatorHandler". Cette classe permet d'avoir un meilleur contrôle sur la façon de structurer une formule puisque tout changement dans la définition de ses limites ou la déclaration d'une opération doit être fait dans ce fichier.

La calculatrice est exécutée en deux phases :

- La première est l'étape d'initialisation. Dans cette étape, toutes les formules déclarées par l'utilisateur sont récupérées. Les formules sont traitées pour obtenir une structure qui en facilite le calcul. Une fois obtenues, elles sont stockées dans une ArrayList. Il est à noter que si une formule n'est pas bien déclarée, elle ne sera pas sauvegardée dans l'Array.

  L'étape d'initialisation est un processus lent car la formule écrite est analysée afin d'obtenir un objet qui contient toute la structure de la formule (opérations et parameters). Toutefois, cela permet d'exécuter l'étape suivante plus rapidement.

- Ensuite, la deuxième étape, orientée vers l'obtention de la valeur de la formule d'une manière cyclique, est exécutée. Ceci est fait parce que les valeurs à calculer peuvent dépendre des valeurs de certains capteurs. Cependant, comme la structure a été obtenue précédemment, il suffit de récupérer la valeur du paramètre et d'exécuter l'opération.

Cette modification permet une compression plus claire du document de configuration de la plate-forme, gagnant ainsi 20% d'espace par rapport à l'ancienne méthode de calcul. La même performance a été obtenue au niveau de l'exécution des calculs.

Le schéma suivant est un algorithme simplifié qui vise à résumer l'ensemble du processus de calcul d'une formule complexe :

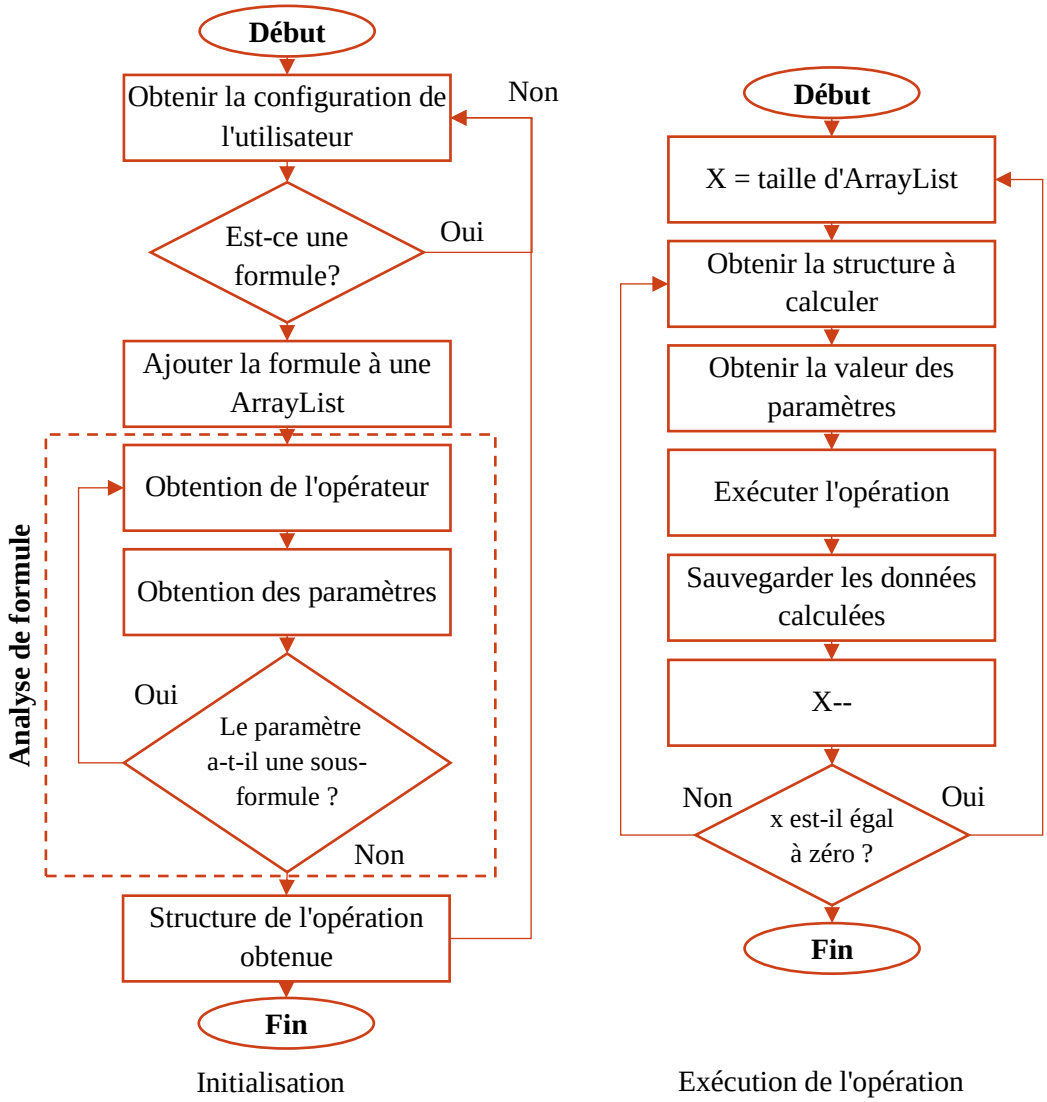


Diagramme 1. Algorithme simplifié de l'initialisation et l'exécution d'une formule complexe

a. Type de calculs

La calculatrice répond à 15 types d'opérations différents. La liste est la suivante :

Tableau 1. Calculs possibles à exécuter

Opération	Représentation	Nombre de paramètres par opération	Résultat
Addition	add	>1	Addition de FLOAT : A, B, C, etc.
Soustraction	sub	2	Soustraction de FLOAT : A - B
Division	div	2	Division de FLOAT : A / B
Multiplication	mul	>2	Multiplication de FLOAT : A, B, etc.
Racine carrée	sqrt	1	Racine carrée de FLOAT : $\sqrt{A}$
Surélevée	pow	2	FLOAT A Surélevée à FLOAT B: A^B

Opération	Représentation	Nombre de paramètres par opération	Résultat
Absolue	abs	>1	Valeur absolue d'un FLOAT : | A |
Maximum	max	>1	Maximum de n FLOATS : max(A, B, C, D, etc.)
Minimum	min	>1	Minimum de n FLOATS : min(A, B, C, D, etc.)
Cosinus	cos	1	Cosinus de float A
Sinus	sin	1	Sinus de float A
Tangent	tan	1	Tangent de float A
ArcCosinus	acos	1	Arc Cosinus de float A
ArcSinus	asin	1	ArcSinus de float A
ArcTangent	atan	1	Arctan de float A
ArcTangent2	atan2	2	Arctan de float A (côté opposé) sur float B (côté adjacent)

b. Forme d'utilisation

Il existe un fichier de configuration, appelé PlaysetDataset, qui définit tous les paramètres du système et son comportement. Dans la section PlaySet, toutes les conditions et la réponse du système à ces conditions sont déclarées. Sur Dataset se fait la déclaration des ressources, fonctions, variables, protocoles de communication qui seront utilisés dans le système.

La configuration PlaysetDataset est unique. Cependant, pour l'activation de certaines fonctions, il est nécessaire de respecter certaines restrictions. Pour réaliser un calcul d'une formule complexe, le fichier PlaysetDataset doit être configuré comme suit :

- Dans la section PlaySet, les colonnes doivent être remplies comme suit :

Tableau 2. Définition d'une formule sur le ficher PlaySet

Opération	Représentation
VarNb	Peu importe
Value	***formula(***Formule à calculer ***)***
Tmr	Laisser vide

- Dans la section DataSet, les colonnes doivent être remplies comme suit :

Tableau 3. Définition d'une formule sur le ficher DataSet

Opération	Représentation
Channel	Peu importe
Variable Name / Short name	Peu importe
Type	5 (float)
Source	39 (calculée)
Param	Formule à calculer
Format nombre	Laisser vide

- Exemples

Tableau 4. Exemples de déclaration d'une formule

Opération	Représentation
5+5+3+#1053+#1052 #1052/ #1051	add(5 ;5 ;3 ;#1053 ;#1052) div(#1052 ;#1051)
Cos(#1051)	cos(#1051)
Max(4 ,2,6,5,8,10)	max(4 ;2 ;6 ;5 ;8 ;10)
5+Cos(#1051)-#1054	sub(add(5 ;cos(#1051));1054)

c. Contraintes

- La formule effectue des opérations seulement avec des nombres flottants. Dans le cas contraire, un message d'erreur sera enregistré dans le document logexepcion.txt
- Le nombre de parenthèses doit être respecté au début et à la fin d'une opération.
- Pour différencier les variables, le caractère ';' doit être utilisé.
- Il faut bien respecter le nombre de paramètres dont a besoin la calculette sinon un message d'erreur sera enregistré dans le document logexepcion.txt

3.2. Restructuration dans la déclaration des Ports

Le projet initial a évolué au fil du temps, ce qui a fait que la déclaration des Ports dans le document de configuration ne correspondait pas au numéro de Port utilisé sur la carte. Ceci a provoqué une limitation dans les fonctionnalités des Ports.

L'un des principaux objectifs de la restructuration des Ports est la possibilité d'occuper la plupart des Ports avec la communication NMEA. Dans le document de configuration PlaySetDataSet, les Ports sont définis puis l'USU lit la configuration et les enregistre dans un ArrayList. La configuration du point de vue du programme est un objet qui contient comme attributs le numéro de Port à utiliser et la vitesse de transmission.

Le second objectif est d'avoir plus de clarté dans la déclaration des Ports dans le document de configuration mais aussi de pouvoir utiliser dans les différents Ports les protocoles de communication

développés pour gérer tous les périphériques (Bluetooth, IMU, SBG, Octans, H3000, NMEA183).

Afin de réaliser la modification, nous avons dû modifier le programme et également effectuer une fonction qui nous permettait d'avoir la rétrocompatibilité avec les anciens fichiers de configuration. Une classe a été créée pour gérer la disposition des Ports de communication configurés en NMEA (NMEA183, Octans et H3000).

La gestion des Ports se déroule en trois étapes :

- Dans un premier temps, les Ports déclarés par l'utilisateur sont initialisés en tenant compte de la rétrocompatibilité du système. Seuls les Ports déclarés comme NMEA183, OCTANS ou H3000 seront enregistrés dans une ArrayList.
- La deuxième étape consiste à définir les données qui seront extraites du Port de communication. Cette méthode fonctionne pour récupérer les données de NMEA183, Octans et H3000. La liste finale nous permettra dans la phase d'exécution d'extraire les informations précises de la chaîne de valeurs que ces périphériques envoient.
- Enfin, dans la phase d'exécution, le protocole de communication est identifié, ainsi que la liste des valeurs à récupérer. Ce processus est exécuté pendant toute la durée de fonctionnement de l'installation.

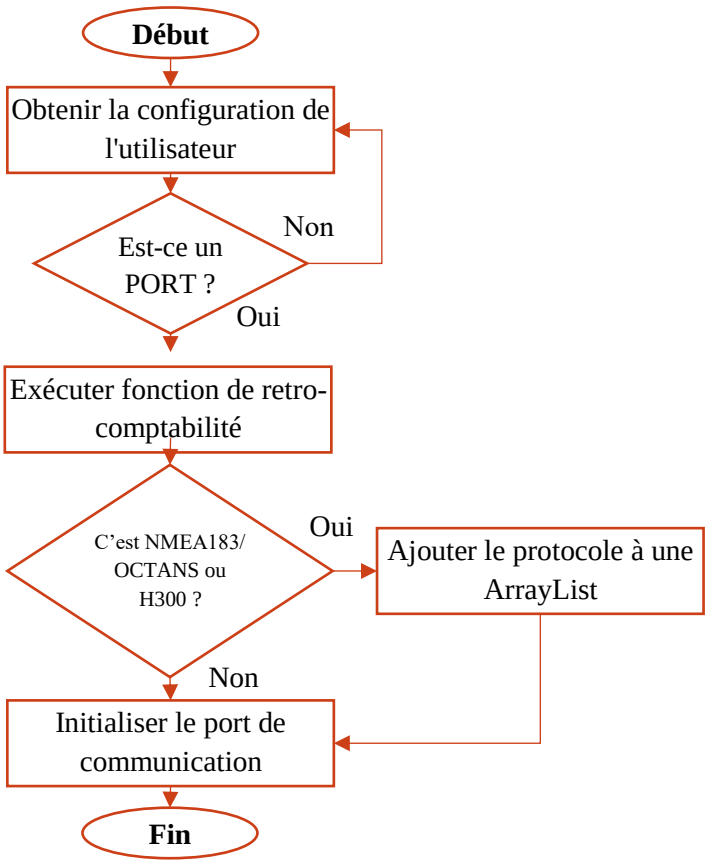
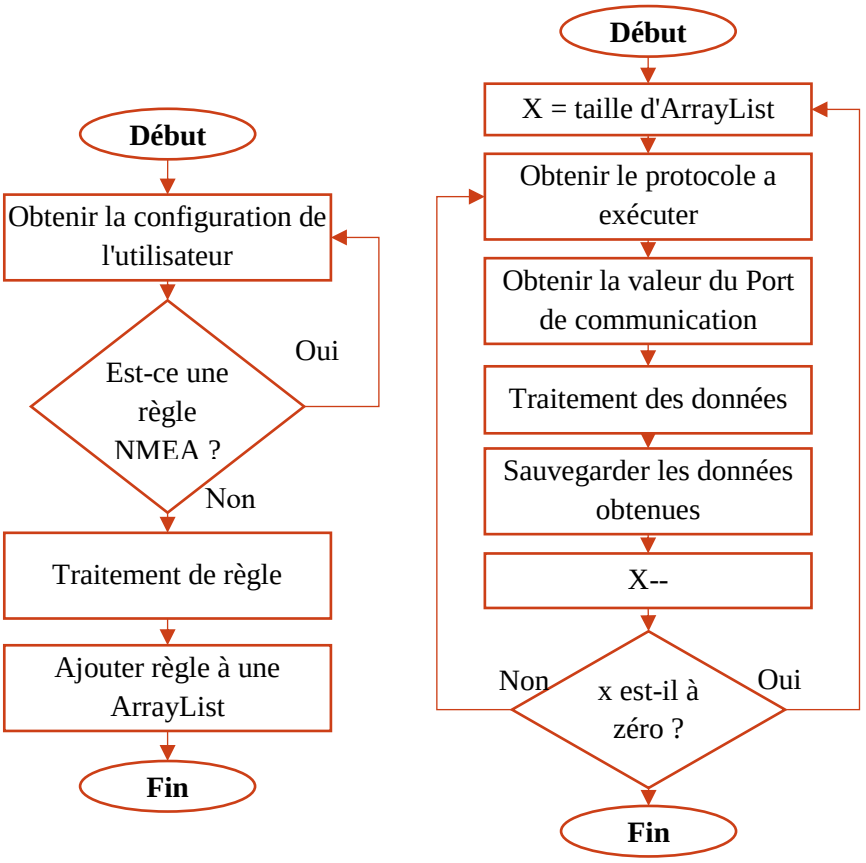


Diagramme 2. Initialisation du Port de communication



Obtention des règles Lecture du Port et traitement de données

Diagramme 3. Gestion des Ports de communication NMEA/OCTANS/H3000

Ci-dessous est présentée la différence entre l'ancien système de déclaration des Ports et l'actuel.

- La structure précédente :

Tableau 5. Ancien système de déclaration des Ports

SHORT NAME	Utilisation	Port physique
COM0	Remote control	COM2
	Récepteur BLE	COM2
	Centrale Octans sur NMEA183	COM2
	Centrale H3000 (HLink)	COM1
COM1	Centrale inertielle interne	COM4
	Remote control	COM4
	SBG Ellipse	COM1
COM2	NMEA183	COM1
	Centrale H3000 (HLink)	COM1
	NMEA183	COM2

SHORT NAME	Utilisation	Port physique
	NMEA183	COM5
	NMEA183	COM6
COM3	Centrale Octans sur NMEA183 Centrale H3000 (HLink)	COM6 COM4
	Auto-Pilot ACP H3000	COM6
COM6	SBG Ellipse	COM6

- La nouvelle restructuration :

Tableau 6. Actuelle système de déclaration des Ports

Colonne	Configuration
Short Name	• COM1 * • COM2 • COM4 • COM5 • COM6
Utilisation	• SBG • IMU • H3000 • NMEA183 • OCTANS • APC • BLE • BLE_ANDROID • SBG

a. Remarques

- *Lorsque le Port COM1 est déclaré pour utiliser l'IMU ou le BLE_ANDROID, le COM4 sera le vrai Port de communication utilisé. Cette action a été faite afin de garder la rétrocompatibilité avec les versions antérieures de DataSet.
- La multi-connexion est valable que pour le NMEA183, le H3000 et l'OCTANS. Sinon, un seul type de communication peut être déclaré dans le système général.
- Si plusieurs capteurs NMEA183 sont connectés dans le système, il faut vérifier que les informations envoyées par ceux-ci sont différentes les unes des autres. Sinon la valeur affichée dans l'interface pourrait être incorrecte.

3.3. Communication Multicast

La communication point à point est actuellement effectuée, c'est-à-dire que lorsqu'une plate-forme se connecte au IHM, la communication ne peut être effectuée qu'une par une. Si un deuxième IHM essaie d'établir une communication avec la plate-forme, l'une des interfaces cessera de fonctionner. Cette

configuration est très utile pour afficher les données dans une seule interface. Cependant, si nous avons plusieurs écrans, cela peut poser problème.

La communication UpSideUp- IHM se fait via unicast UDP. Cette communication peut avoir une vitesse de transfert d'information rapide. Le flux de fonctionnement simplifié est le suivant :

- Envoi en broadcast sur le Port Ethernet du IHM la commande #ETH,CONNECTION
- L'USU écoute le message et établit une communication directe avec l'interface qui a envoyé le message. Puis, il renvoie en retour son identifiant : !USU
- L'IHM reçoit l'identifiant et effectue une connexion au Port UDP pour établir une communication directe avec la plate-forme. L'interface est donc prête pour envoyer la commande #STFLX,1 afin de demander l'envoi d'informations en continu.
- L'USU reçoit la demande et envoie en stream les trames UDP contenant les différentes variables et commandes.
- Pour arrêter le stream UDP, la commande #STFLX,0 est envoyée à l'USU par l'IHM.
- Pour se déconnecter, la commande #ETH,DISCONNECTION est envoyée à la plateforme.

Pour l'affichage d'informations sur plusieurs écrans, deux options sont proposées :

a. Communication en utilisant un canal Multicast

Le mutlicast est une forme de diffusion d'un émetteur (source unique) vers un groupe de récepteurs. Les récepteurs intéressés par les messages adressés à ce groupe doivent s'y inscrire. Ainsi toutes les IHM qui souhaitent lire les informations envoyées par l'USU pourront le faire. Le flux de communication est le suivant :

- Envoi en broadcast sur le Port Ethernet du IHM la commande #ETH,CONNECTION
- L'USU enregistre l'adresse IP et le Port de communication de l`émetteur dans une liste d'utilisateurs actifs. Puis, il renvoie son identifiant plus l'adresse du canal Multicast : !USU, 224.255.2.33
- IHM reçoit le message et se joint au groupe Multicast. Parallèlement, il fait une connexion au Port UDP correspondant à l'USU précédemment connecté et envoie la commande #STFLX,1 pour demander à l'USU de démarrer le stream UDP.
- L'USU reçoit la demande et envoie en Multicast les trames qui contiennent les différentes variables et commandes.
- Pour arrêter l'envoi des données, IHM envoie la commande #STFLX,0. L'USU interrompt la communication lorsqu'aucun utilisateur n'écoute le message.
- Pour se déconnecter, la commande #ETH,DISCONNECTION est envoyée à l'USU et celui-ci supprime l'IHM déconnecté de sa liste d'utilisateurs actifs.

Cette solution fonctionne parfaitement dans un réseau câblé car la fidélité de la transmission est élevée. Toutefois, lorsque la communication multicast est effectuée via Wifi, le système ne peut pas transmettre le message et l'interface ne peut donc pas afficher d'informations. Cela se produit parce que certains routeurs bloquent généralement certains flux d'informations.

b. Rediffusion du message.

Ce type de communication ressemble à la configuration initiale. Cependant, son comportement est le suivant :

- Envoi en broadcast sur le Port Ethernet du IHM la commande #ETH,CONNECTION
- L'USU enregistre l'adresse IP et le Port de communication de l`émetteur dans une liste d'utilisateurs actifs. Puis, il renvoie son identifiant : !USU
- L'USU reçoit l'identifiant et effectue une connexion au Port UDP pour établir une communication directe avec la plate-forme. L'interface est donc prête pour envoyer la commande #STFLX,1 afin de demander l'envoi d'informations en continu.
- L'USU reçoit la demande et stream les trames à tous les utilisateurs actifs.
- Pour arrêter l'envoi des données, IHM envoie la commande #STFLX,0. L'USU interrompt la communication lorsqu'aucun utilisateur n'écoute le message.
- Pour se déconnecter, la commande #ETH,DISCONNECTION est envoyée à l'USU et celui-ci supprime l'IHM déconnecté de sa liste d'utilisateurs actifs.

Toute amélioration ou mise en œuvre d'une nouvelle fonctionnalité du système doit tenir compte de la rétrocompatibilité avec les fichiers de configuration précédents. C'est pourquoi il existe 3 modes de communication dans le système : point à point, multidiffusion et répétition des messages.

Ce mode de communication fonctionne bien dans un réseau filaire ou Wifi. Si le nombre d'utilisateurs est élevé, il peut y avoir une saturation dans l'utilisation des Ports. Par conséquent, il n'est pas recommandé si un grand nombre d'utilisateurs sont connectés.

3.4. Communication avec une API

Il est prévu de développer un projet en collaboration avec la société MADINTEC. Le but du projet sera de récupérer les données d'une API et de les afficher sur l'IHM. L'intermédiaire entre l'API et l'interface sera la plate-forme UpsideUp. Pour l'instant, nous n'avons pas l'API finale. Cependant, afin de faire avancer le développement du projet, un simulateur de l'API a été programmé.

Pour effectuer la communication, le protocole UDP sera utilisé. Le numéro de Port pour la communication USU- IHM sera différent de celui pour la communication USU -API, afin d'avoir un meilleur contrôle dans le flux des données. Pour s'assurer que la réception des données est adéquate, une communication synchronisée est proposée. Le flux qui sera suivi pour la communication UDP sera effectué comme le diagramme suivant :

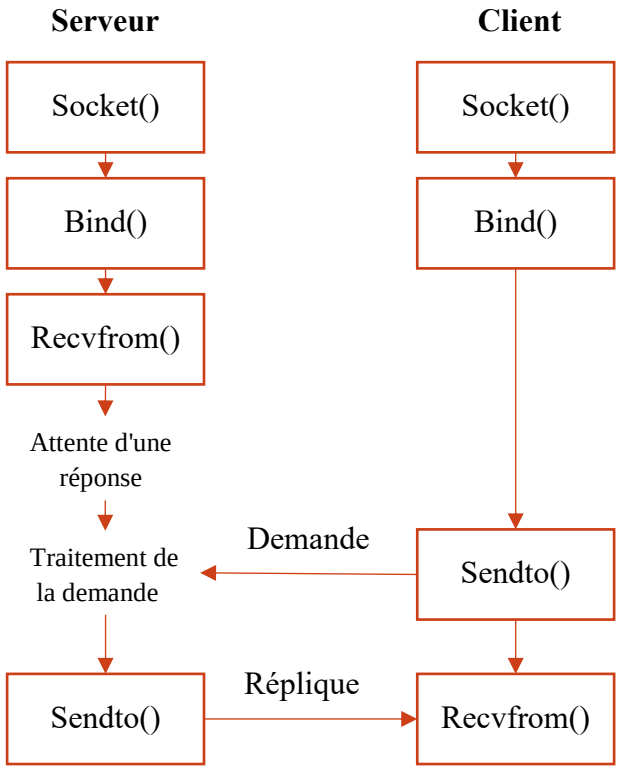


Diagramme 4. Communication UDP synchrone

La communication sera développée de la manière suivante :

- L'USU envoie en broadcast sur le Port la commande : ! APICONNECTION
- L'API reçoit la demande et envoie le message : ! APICONNECTED à l'adresse qui a envoyé la demande de connexion.
- L'USU reçoit le message et enregistre l'adresse IP de l'API afin d'établir une communication point à point. A ce moment, l'USU demande à l'API d'envoyer les informations souhaitées.
- L'API reçoit les informations et envoie les données demandées.
- Lorsque la plate-forme reçoit les données, celles-ci doivent être traitées afin d'être affichées dans l'IHM.

Une fois la connexion établie, la tâche est exécutée périodiquement. Le temps pour demander des informations à l'API sera défini dans le fichier de configuration. S'il n'est pas possible de recevoir ou d'envoyer des messages après un certain nombre de tentatives, le système ferme le Socket de communication et rétablit la connexion avec l'API. Les tentatives de reconnexion seront faites périodiquement, ce qui permet de rétablir la communication à tout moment. Lorsque la communication est rétablie, l'échange d'informations se normalise à nouveau.

L'idée générale de la classe est de pouvoir établir une communication avec n'importe quelle API. Cela permet d'avoir une classe générale, ce qui n'est pas seulement propre au projet de collaboration avec MADINTEC. La principale différence entre les distinctes API sera le traitement de l'information. Pourtant, cette méthode devrait évoluer en fonction de l'API à utiliser. Actuellement, le système exécute une analyse simple, juste pour obtenir un résultat et vérifier le fonctionnement de la classe; cette partie sera modifiée lorsque le système de MADIENTEC sera proportionné.

3.5. Récupération partielle des Logs

Dans le MicroSD de l'US, il y a un dossier appelé Logs dans lequel nous pouvons trouver les documents qui représentent des événements historiques. Pour créer un document, il doit être défini dans le fichier de configuration PlaySetDataset.

Ces fichiers nous permettent de sauvegarder les valeurs que nous voulons et de les gérer de manière spécifique. Pour leur création, leur définition sur le ficher PlaysetDataset doit être la suivante.

a. Création du log

La façon de déclarer l'utilisation d'un fichier log est la suivante :

Tableau 7. Définition d'un fichier LOG sur le ficher DataSet

Colonne	Configuration
Channel	Peu importe
Type	13 (String)
Source	12 (Ficher LOG)
Param	• 1 "Nom de dossier"\"nom de ficher"."extension de ficher" Exemple : LOG\LogDebug.csv • 2 Taille du fichier en bits • 3 Type de stockage : Archive, FIFO, ERASE " • 4 En-têtes de colonnes : "Titre 1;Titre2;Titre 3…" • 5 données qui seront insérées dans les colonnes : "Type+Channel; Type+Channel; Type+Channel …."

Description des paramètres :

- **Param 1 :** Le nom du fichier peut contenir des majuscules et des minuscules, mais ne doit pas contenir le caractère "-". Sinon, l'algorithme de sauvegarde partielle des fichiers ne fonctionne pas correctement. L'extension est également limitée à ".csv", en raison de la façon dont les données sont sauvegardées.
- **Param 2 :** Correspond à la taille maximale du fichier. Ça nous permet de créer différents fichiers, lorsque la taille aura été atteinte.
- **Param 3 :** Correspond à la façon dont l'utilisateur souhaite stocker les données :
  - Archive : Garde tous les fichiers. Si carte est pleine, l'USU essaye de supprimer les fichiers en mode FIF. S'il n'en reste plus qu'un de disponible est lancée l'erreur mémoire pleine.
  - FIFO : Garde tous les fichiers. Si la carte est pleine, l'USU supprime le plus ancien sauf un.
  - ERASE Suivi d'un espace puis d'un nombre spécifiant le nombre d'archive à garder. Si le nombre n'est pas précisé, le nombre par défaut est 1. Définit le nombre maximum de fichiers qui seront sauvegardés. Sinon ils seront supprimés.
- **Param 4 :** Correspond à la façon dont chaque colonne est appelée, ce qui permet d'identifier plus clairement les valeurs qui seront enregistrées dans le document.

- **Param 5 :** Correspond à la façon dont les données seront affichées : dernière valeur, Moyenne, Maximum ou Minimum.

b. Récupération des logs

Selon la configuration sélectionnée dans le fichier PlaysetDataSet sur le dossier LOGs, il y aura plusieurs ou au contraire peu de documents. Il y aura des documents qui représentent des événements historiques et ils auront le nom suivant : "nom de ficher – yyyyMMdd-HHmmss. extension de fichier ". Pour récupérer tous les fichiers LOGs, nous pouvons exécuter les activités suivantes : Via clé USB, Via FTP ou Via IHM.

Le but du projet est de trouver un moyen de diminuer le nombre de fichiers qui doivent être sauvegardés sur la clé USB. Le temps de transfert par fichier est de 17-20 secondes. Donc, s'il y a beaucoup de dossiers à transférer, le temps d'attente sera trop long. La solution à ce problème est la récupération partielle des fichiers. Cependant, la fonction de récupération de tous les fichiers est toujours disponible. Il est important à noter que les modifications doivent considérer la rétrocompatibilité avec les versions précédemment installées. Les modes de fonctionnement sont donc les suivants :

- Récupération partielle. Si c'est la première fois que cette action est effectuée, le système récupère tous les fichiers. Lors de la récupération de fichiers, l'USU enregistre la date à laquelle cette action a été effectuée.
  Lorsqu'une récupération partielle est à nouveau demandée, le flux est le suivant :
  - Le nom du fichier est comparé. S'il ne contient pas le caractère " - ", le fichier est récupéré.
  - Si le fichier a ce caractère, cela signifie qu'il s'agit d'un fichier historique. Alors la date est extraite et est comparée avec la date de la dernière récupération. Si le fichier est récent, il est récupéré. Sinon il n'est pas pris en compte.

  Cette action permet de récupérer les fichiers plus rapidement puisqu'il n'est pas nécessaire de récupérer tous les fichiers existants dans le dossier.

- Récupération totale. La récupération complète prend en compte tous les fichiers existants dans le dossier LOG, plus les fichiers Dataset et PlaySet. Lorsque cette option est exécutée, la date de la dernière récupération est également enregistrée.

Une classe spécifique a été créée pour ce projet. La classe est composée de trois méthodes. La première méthode correspond à l'obtention de la liste des fichiers à récupérer partiellement. La seconde méthode correspond à la liste de tous les fichiers. Enfin, la troisième méthode correspond au système de transfert de données MircoSd USB. Il était également nécessaire d'ajouter une combinaison spécifique pour que l'utilisateur puisse accéder au système de récupération partielle. Dans le diagramme suivant, il est possible de visualiser d'une manière simple le fonctionnement de la récupération des fichiers qui seront récupérés partiellement.

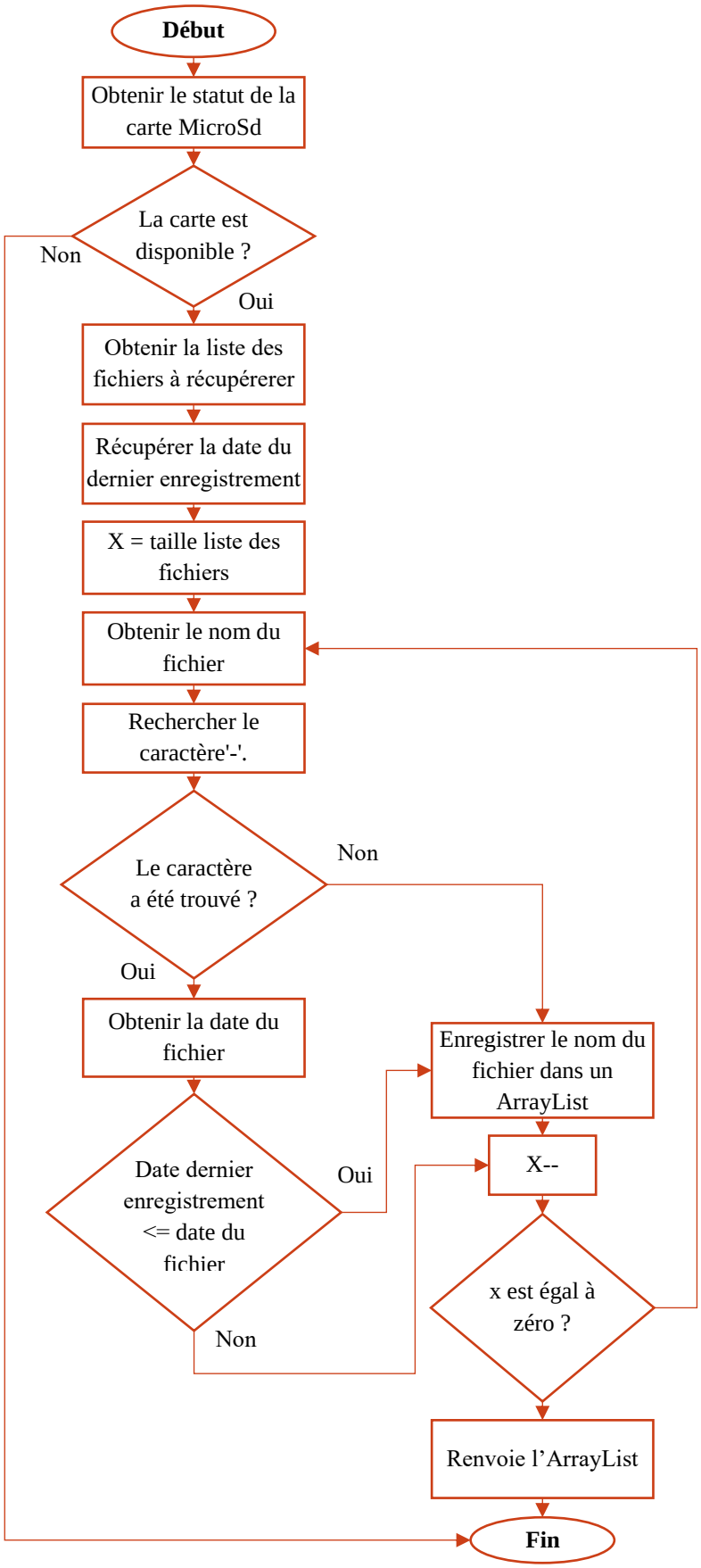


Diagramme 5. Obtention des fichers LOG partiels

3.6. Projets à développer

Il existe différents projets pour améliorer le système actuel. Certains projets qu'il est prévu de réaliser sont les suivants :

- Modifier le projet API : Une fois que MADITEC aura fournit les informations nécessaires, le projet devrait être modifié selon les caractéristiques de communication de l'API.
- Fonctions disponibles dans l'ancienne IHM : intégrer à la nouvelle IHM certaines fonctionnalités disponibles avec l'ancienne IHM :
  - Réplication du NMEA
  - Récupération de la valeur de n'importe quelle variable du Dataset
  - Remise à zéro des Maxs du NDR (par le bus CAN)
  - Actions sur l'USU (relais, alarmes, consigne pilote automatique...)
  - Calibration des entrées analogiques
- Threader CAN et COM : L'objectif est d'alléger le Thread Principal avec tout ce qui ne concerne pas la récupération de données qui peut être faite de manière asynchrone.

Il s'agira de réaliser la majorité des projets. Néanmoins cela m'assure une continuité de stage et surtout d'acquérir de nouvelles connaissances et compétences.

3.7. Activités secondaires

Certaines des activités secondaires qui sont développées dans l'organisation sont la vente des SPOT 3, 6 et 9 Leds, et le soudage des cartes électroniques pour la fabrication du USU. Les activités suivantes ont donc été réalisées :

a. Fabrication de SPOT

Le SPOT est un produit de vente récurrent : plus de 50 spots ont été fabriqués depuis le début de mon stage. Sa fabrication est rapide et demande peu de temps. Les activités lors de la fabrication de SPOT sont :

- Couper et souder les câbles d'alimentation LEDs.
- Coller et assembler les LEDs sur la boîte métallique.
- Coller les optiques sur les LEDs.
- Coller et assembler le plastique de protection des LEDs.
- Souder le système de contrôle de puissance des LEDs (driver, résistance thermique, diode).
- Couper et souder les câbles d'alimentation du SPOT.
- Tester son fonctionnement.

b. Soudage de cartes électroniques

Pour la vente au client ainsi que pour les fonctions de développement, il est parfois nécessaire de souder des composants électroniques sur une carte USU. Une fois tous les composants soudés, divers tests sont effectués pour vérifier le bon fonctionnement de la carte. Une fois le test terminé, un numéro de série est placé, puis l'USU est stocké.

c. Restructuration du code

Certaines modifications sont apportées dans le code pendant qu'une nouvelle fonctionnalité est en

cours de développement ou qu'une partie spécifique du programme est en cours de modification. Cela nous permet de faire évoluer le code et pourtant le firmware de l'USU. Les modifications apportées au code ont été les suivantes : ajouter des commentaires, ajouter des régions, créer de nouvelles classes, renommer des variables, etc.

4. Conclusion

Au début du stage, j'ai eu un défi vraiment important : la compréhension du fonctionnement du système. Il existe actuellement un document qui tente d'expliquer en termes généraux le fonctionnement de l'USU, mais il y a plusieurs situations qui ne sont pas complètement décrites et il faut comprendre le code pour savoir comment il fonctionne. Par conséquent, travailler avec un code non commenté et avec peu d'informations peut être un véritable défi. Cependant, cela m'a permis d'améliorer la façon dont je nomme mes variables et fonctions, puisque, pour faire un code compréhensible, il est nécessaire de programmer d'une manière claire.

Mon stage jusqu'à présent a été très fructueux car il m'a permis de compléter la formation du Master en Systèmes Embarqués. Un des éléments que j'ai pu améliorer est la programmation en langage C#. Cela m'a aussi aidé à mieux comprendre l'utilisation de la programmation orientée à objets pour les systèmes embarqués, sujet qui pourrait être mieux abordé dans les cours.

Au moment de travailler sur un projet de programmation très complexe, l'un des éléments importants est la possibilité d'utiliser le mode Debug. Ce mode nous permet de vérifier que le code fonctionne comme nous l'imaginions ainsi que d'identifier les erreurs qui provoquent l'arrêt du système. Programmer avec l'IDE Visual Studio pendant mon stage me permettra de travailler plus sereinement sur de futurs projets avec ce type d'outils, car j'ai développé les capacités nécessaires pour exploiter leur potentiel.

Un autre élément que j'ai pu renforcer est ma connaissance des réseaux. En d'autres termes, j'ai enfin compris comment fonctionnent les communications unicast, multicast et broadcast en utilisant le protocole UDP, le principal protocole que j'ai utilisé pour faire une communication locale.
J'ai aussi eu l'opportunité de travailler sur un projet d'envergure en utilisant la gestion de version de GIT. Cet outil est très efficace car il permet de travailler sur différents projets en même temps pour ensuite en faire une intégration.

Il est évident que l'organisation a travaillé dur pour développer ses produits, ce qui se reflète dans leur qualité et dans la façon dont ils fonctionnent. Cependant, il y a certains points qui peuvent être améliorés. Par exemple :

- Le langage de programmation Net MicroFramework, est peu documenté et est également limité dans l'utilisation de certaines bibliothèques, ce qui amène le développeur à créer ses propres bibliothèques. Cela augmente la complexité du projet et le temps de développement. L'incorporation d'une nouvelle carte pouvant intégrer un système d'exploitation LINUX permettrait d'avoir un système réellement embarqué et surtout la quantité de documentation existante sur Internet est immense. Cependant, la solution actuelle fonctionne de façon efficace et il n'est pas nécessaire d'apporter de tels changements à court terme.

- La carte peut gérer des threads, ce qui permet d'exécuter des tâches diverses et de ne pas avoir de programmation séquentielle. Le projet en cours gère différents threads. Néanmoins, ceux liés aux capteurs, les calculs et la mise à jour des données se font sous forme séquentielle. Une façon intéressante d'améliorer les performances de la carte serait d'utiliser un ou plusieurs threads dédiés à la mise à jour les données. Cette amélioration permettrait de raccourcir les délais d'exécution, mais n'aurait pas d'impact majeur sur le fonctionnement du système actuel.

Mon stage au sein d'Ocean Data Systems m'a permis d'améliorer mes compétences en programmation d'une façon remarquable. Aussi en tant qu'étranger, j'ai eu l'occasion de connaître l'environnement de travail d'une entreprise française. C'est pourquoi je remercie l'organisation de m'avoir donné cette opportunité et toute l'équipe de m'avoir accueilli. Je tiens à remercier mon tuteur Ugo Brunet qui m'a accompagné tout au long de ce stage.

5. Références

ODS. (2019). *Ocean Data Systems*. Récupéré sur http://www.oceandatasystem.com/
Wikipedia. (2019, 05 11). *Multicast*. Récupéré sur https://fr.wikipedia.org/wiki/Multicast