

TO54 – Projet de développement Sujet :

Relais de protection de réseaux électriques : mise en œuvre du protocole de l'IEC 61850

Encadré par M. Robin Roche

Département Energie – P2022

HIRTH Sébastien

Filière électronique et système embarqué (ESE)

TEANI Hugo

Filière réseau et conversion de l'énergie électrique (RCEE)

Table des matières

Introduction.....	4
I – Présentation du Projet : Définition des problématiques.....	5
1) Données initiales du projet	5
2) Objectif de notre projet.....	5
II – Le protocole de communication IEC61850, utilité par rapport à notre IHM	5
1) La norme IEC 61850.....	5
2) Présentation de l'IHM gérée par la norme IEC 61850.....	6
3) Gestion de projet.....	7
III – Le Projet.....	8
IV - Interface Graphique de l'application	8
1) Présentation de la fenêtre d'accueil	11
2) Présentation de la fenêtre permettant d'ajouter des IED	14
3) Présentation de la fenêtre permettant de gérer/monitored les IED relais	22
4) Présentation de la fenêtre permettant de gérer les messages Goose	24
V – L'application de la norme IEC-61850 à notre IHM à l'aide de la librairie en C#.....	25
1) Construction de la Librairie	25
2) Connexion aux IEDs	29
3) Réception de Message Goose	36
Conclusion	51
Annexe : Guide d'utilisation de l'IHM	53

*« La théorie, c'est quand on sait tout et que rien ne fonctionne.
La pratique, c'est quand tout fonctionne et que personne ne sait pourquoi. »*

Albert Einstein.

Introduction

Le projet détaillé au travers de ce rapport rend compte d'un travail d'un semestre d'étude au sein de l'établissement.

L'université de Technologie de Belfort Montbéliard propose à ces étudiants en filière énergie à la suite de leur stage ST40 de choisir une filière de spécialité. Cette première année leur permet d'acquérir des connaissances scientifiques. Par la suite, en filière les enseignants nous poussent à travailler en mode projet c'est-à-dire en autonomie sur un sujet proposé aux étudiants qui rend compte du travail et de ce qui a été vu lors de la première année.

Le projet détaillé par la suite rentre dans le cadre de l'UV TO54 (projet de développement) proposé par Monsieur Robin Roche. Il est autorisé pour les filières RCEE (Réseau et conversion de l'énergie électrique) et ESE (Électronique et système embarqué). Le sujet rentre pleinement dans la filière énergie puisqu'il traite d'un sujet primordial de la sécurité des biens et des personnes des réseaux électriques de distribution.

En effet, les réseaux électriques de distributions impliquent la mise en œuvre d'équipements de protection assurant le bon fonctionnement des installations et garantissant la sécurité des utilisateurs, à savoir dans notre cas l'étude des relais de protection des réseaux haute tensions.

Dans un souci d'uniformisation des procédés il a été décidé de créer un nouveau protocole de communication dont le but est d'homogénéiser les interactions entre les équipements présents dans une installation quel que soit la marque du produit utilisé. Ce nouveau protocole de communication s'appelle IEC 61850 et sera détaillé par la suite.

L'objectif de ce projet est de créer une interface de supervision des relais (F650 de GE) de la maquette utilisée dans le cadre de l'UV ER57 dirigée par monsieur Robin Roche permettant de communiquer à l'aide de la norme IEC 61850. Notre rôle au sein de ce projet est de reprendre le travail qui a effectué lors des précédents semestres et de l'améliorer. En effet lors des précédents semestres, des étudiants ont créé deux servantes avec une IHM gérée par le protocole TCP/IP à l'aide d'un automate Siemens. Le problème de ce protocole est qu'il est différent du protocole IEC 61850, puisque le simple ajout d'un relais d'une autre marque que Général Electric, ne permettrait pas de communiquer simplement entre eux.

L'objectif est donc de configurer les relais présents sur les servantes afin qu'ils puissent communiquer entre eux ainsi qu'avec des équipements externes en suivant le protocole IEC 61850. La supervision devra donc respecter la norme IEC 61850 et être open-source. La programmation de la supervision s'effectuera dans le langage C# qui possède une librairie sur la norme IEC 61850.

Le cahier des charges concernant notre IHM sera défini par la suite. Nous allons commencer dans ce rapport à rappeler les aspects techniques du protocole de communication IEC 61850, puis nous verrons dans une première partie comment l'interface graphique a été créé, puis dans une deuxième partie nous verrons comment elle a été adaptée pour communiquer à l'aide de la norme IEC 61850. Enfin dans une dernière partie nous effectuerons quelques tests de notre IHM, et relevons les potentiels axes d'amélioration pour les prochains étudiants qui travailleront dessus.

I – Présentation du Projet : Définition des problématiques

1) Données initiales du projet

Nous sommes donc le troisième groupe à travailler sur ce projet. Le premier étudiant à avoir travailler sur le projet fût HIRAT Bastien, son travail avait pour objectif d'améliorer la compréhension de la norme IEC 61850 appliquée à des relais de protection de général Electrique F650 GE. Il a facilité la compréhension des messages Goose (issue de la norme IEC 61850) et à réussit à faire communiquer entre eux plusieurs IED (Intelligent Electronic Device) pour réaliser une sélectivité sur le réseau électrique.

Le deuxième groupe à avoir travaillé sur le sujet fût BACONNIER Juliette, MELHE Kevin, et HELL Ludovic leurs travails avaient pour objectif d'améliorer la compréhension de la norme IEC 61850, et de créer deux servantes mobiles comportant les relais de protection, et enfin la création d'une IHM en effectuant une liaison client-serveur en utilisant le protocole TCP/IP avec un automate Siemens.

2) Objectif de notre projet

Nous sommes donc le troisième binôme à travailler sur le projet, Hugo TEANI et Sébastien HIRTH, notre objectif est de reprendre le travail qui a été fait par les anciennes équipes, pour réaliser une IHM open source communiquant à l'aide de la norme IEC 61850. A terme cela permettrait de faire communiquer dans un même réseau plusieurs IED de marque différente. Actuellement, l'IHM mis en place ne respecte pas cette norme, elle lit simplement chaque registre des différents relais, ce qui permet d'obtenir les données des courants et des tensions de chaque phase, et d'afficher l'état des sorties. En revanche elle ne permet pas de lire les messages Goose, elle se contente seulement de lire dans le registre du Relais si le « TRIP » est déclencher ou non. Nous n'avons donc sur cette IHM aucune idée sur quelle phase ni en combien de temps le défaut a été éliminé ni de quel type de défaut il s'agit.

II – Le protocole de communication IEC61850, utilité par rapport à notre IHM

1) La norme IEC 61850

Le protocole IEC 61850 est une norme de communication utilisée par les systèmes de protection des sous-stations électrique dans le secteur de la production d'énergie électrique. Elle est surtout utilisée au niveau des postes de distribution du réseau électrique par exemple dans les plateformes HTA/BT. Ces stations sont équipées d'IED la plupart du temps comme dans notre cas d'étude de relais de protection basés sur des microprocesseurs, de dispositifs de mesure (Capteur de courant et de tension), d'enregistreurs de défauts et d'événements. Ils permettent la surveillance en temps réel du réseau et la gestion autonome de celui-ci

La norme IEC 61850 est un ensemble de protocoles et de spécifications qui permet à différents appareils d'un même réseau de communiquer entre eux. Il peut fonctionner sur des réseaux types clients/serveur TCP/IP (Ethernet) et prend en charge les message GOOSE/GSSE, qui sont des messages envoyés sur les réseaux rapidement inférieurs à la milliseconde pour pouvoir être traiter le plus rapidement et permettre au réseau électrique de ne pas être endommagé. Elle prend également en charge les messages de type MMS (Manufacturing Message Specification) et XML (eXtensible Markup Language) qui ne seront pas traité dans ce rapport.

Le principal avantage de cette norme est comme dit précédemment, elle permet d'assurer l'interopérabilité entre plusieurs IED provenant de différents fabricants. Chaque dispositif électronique intelligent est repéré sur le réseau par son adresse IP, son port, à travers lequel il est possible d'effectuer le paramétrage et la supervision à distance.

La communication et la supervision des IED présents sur notre réseau s'effectue donc via une liaison Ethernet local. Cette liaison nous permet donc à travers des switches de configurer chaque IED indépendamment et de visionner à l'aide d'une supervision l'état de chaque IED.

2) Présentation de l'IHM gérée par la norme IEC 61850

Afin de ne pas augmenter le coût déjà élevé du projet, nous avons décidé de créer notre application de supervision à l'aide d'un outil de programmation open Source en C#. Afin de développer notre application nous nous sommes aidés de l'IDE Visual Studio. Nous avons décidé de programmer en C# car il y existait une bibliothèque déjà bien fournie.

Le cahier des charges de notre supervision peut être décrit par le diagramme Bête à Corne et le diagramme Pieuvre suivant :

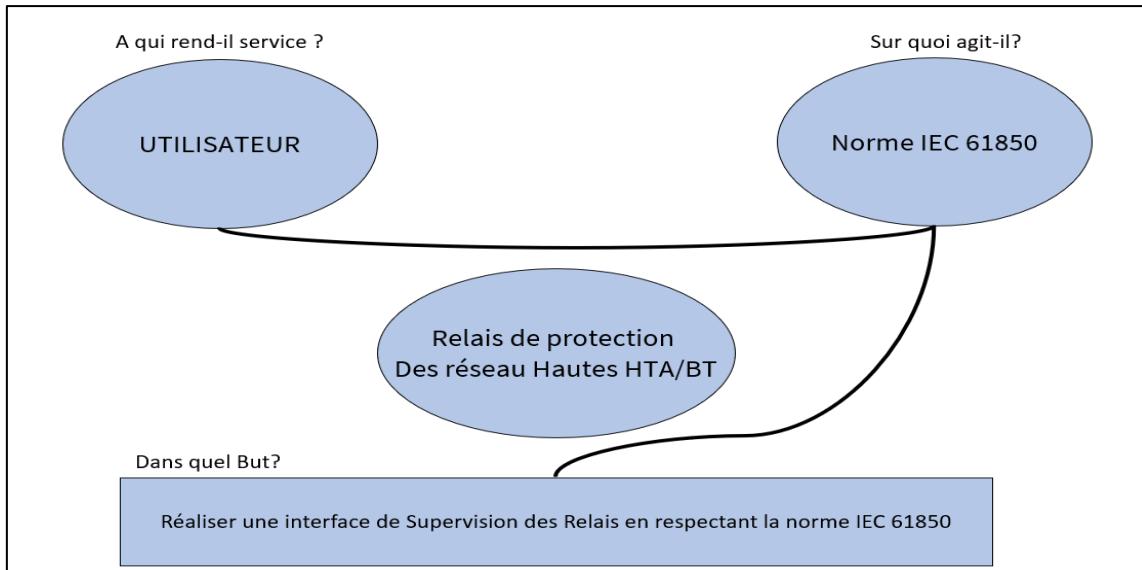


Figure 1 : Bête à Corne du projet

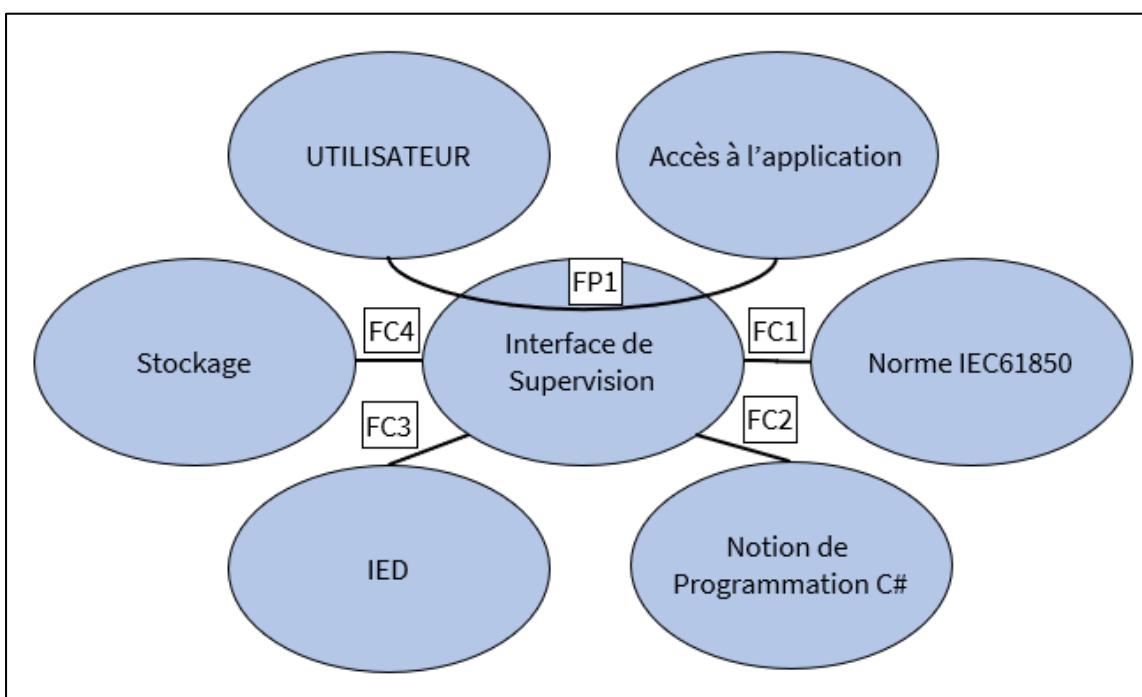


Figure 2 : Diagramme Pieuvre du projet

Fonction principale :

FP1 : L'Utilisateur doit pouvoir accéder à la supervision des IED en respectant la norme IEC61850, il doit pouvoir gérer l'ensemble des IED sur le réseau.

Fonctions secondaires :

FC1 : L'application de supervision des IED doit respecter en tout point la norme et le protocole de communication de l'IEC 61850 (Réception des message SV et message Goose)

FC2 : L'application devant être open-source il faut avoir des prérequis en programmation C# pour pouvoir réaliser le projet

FC3 : Le but est à terme de pouvoir ajouter à la supervision n'importe quelle IED (de marque différente) sans avoir à retoucher le programme

FC4 : Récupérer et stocker les différents messages Goose (Permet de détecter de quel défaut il s'agit, sur quelle phase est survenue le problème, sa durée, le temps de déclenchement, et l'horodatage de l'erreur)

3) Gestion de projet

Une fois notre projet bien appréhendé et défini notre sujet, il nous a fallu réaliser un diagramme de Gantt nous permettant de répartir les tâches et de nous fixer des objectifs précis pour chaque semaine. En fonction de notre temps libre et des problèmes survenus, nous essayons de nous voir le vendredi dans la matinée et parfois si nécessaire les jeudis dans l'après-midi. Bien entendu il y a beaucoup de travail personnel réalisé chacun sur notre temps libre. Voici ci-dessous le diagramme de Gantt prévisionnel réalisé en tout début de projet.

Diagramme de Gantt

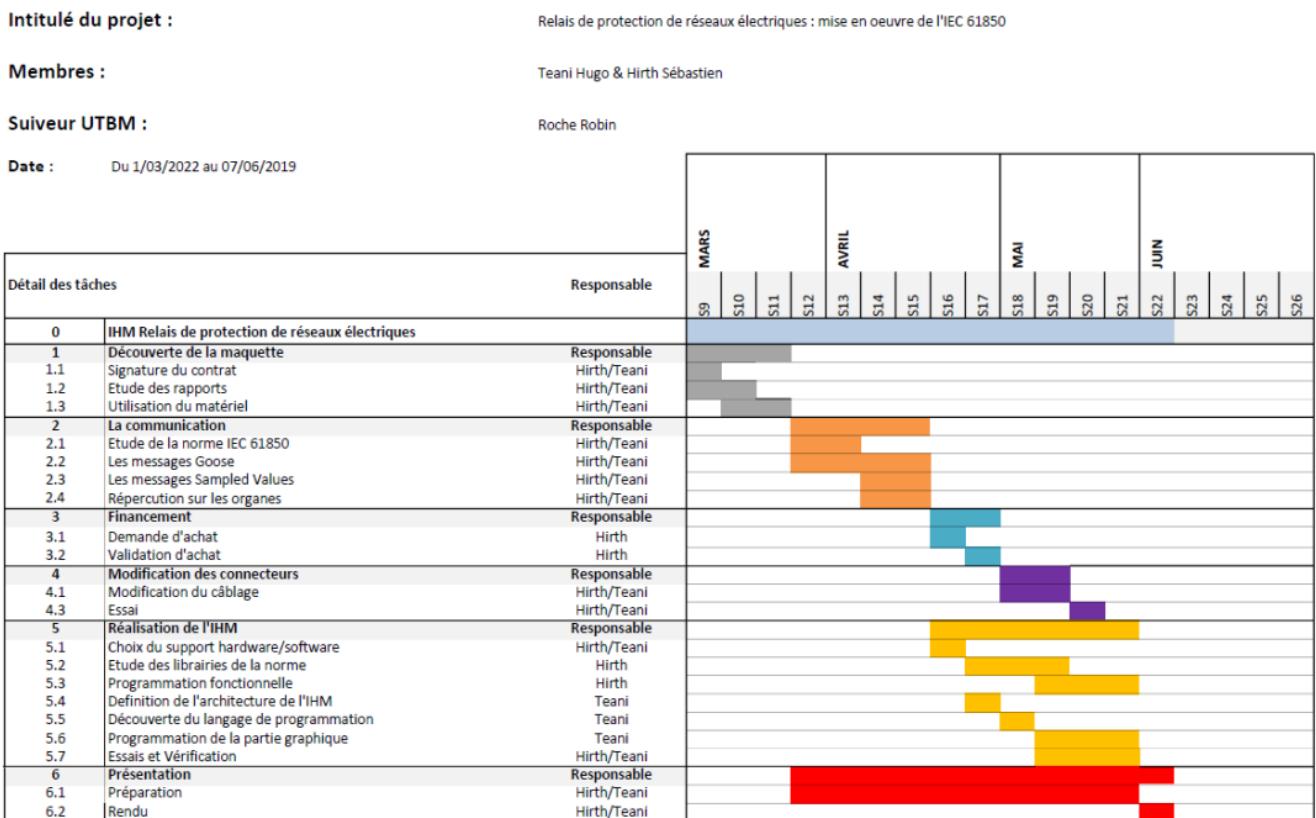


Figure 3 : Diagramme de Gantt du projet

III – Le Projet

Après avoir bien appréhendé le sujet et réalisé des essais sur les maquettes, nous avons décidé de répartir notre travail en deux grandes parties, indépendantes qui nous a permis d'avancer en simultané et s'aider mutuellement lorsqu'il y avait des problèmes. La première partie avait pour but de créer l'interface graphique de notre application de supervision, avec l'ensemble des caractéristiques énoncé précédemment, cette partie a été traité par Hugo TEANI. La deuxième partie quant à elle avait pour objectif de faire communiquer les IED de la maquette avec la norme IEC 61850 afin de réceptionner les message Goose et les Samples Values qui transit sur le réseau Ethernet local, cette partie a été traité par Sébastien HIRTH.

IV - Interface Graphique de l'application

Le sujet initialement proposé par Monsieur Robin Roche était de réaliser cette IHM à l'aide de la programmation en python. Après quelque semaine de recherche nous avons décidé de réaliser notre application en C# car la librairie permettant de réaliser le protocole IEC 61850 était plus fournit en C# qu'en langage python.

Une fois le langage de programmation choisi, il a fallu choisir un IDE (environnement de développement intégré), nous nous sommes donc tournés sur l'IDE Visual Studio, qui est un IDE totalement gratuit et disponible directement sur les ordinateurs de l'UTBM.

Sachant que notre application à terme est censée fonctionner sur un ordinateur, avec pour OS, Windows, nous nous sommes donc tournés vers l'utilisation de **Windows form** pour créer notre application de façon conviviale et simple, sachant que nous sommes en FISE énergie et non Informatique.

La programmation en C# de Windows form à l'aide de l'IED Visual Studio est très intuitive, elle permet d'ajouter simplement des contrôles (présent dans la boîte à outil) sur un formulaire. Chacun de ces outils est défini selon **ces propriétés**, selon **ces événements** (action particulière à exécuter en fonction de l'évènement choisi), et peuvent même avoir **des méthodes** (qui permet de caractériser notre outil).

Notre application a tout de suite été définie comme suit :

- La page d'accueil de notre application a pour objectif d'ajouter les IED présent sur le réseau local. Cette première page est la plus importante car elle permet de définir chaque IED, on doit pouvoir les modifier, les supprimer, les classer, etc... Il doit donc y avoir une interface conviviale pour ajouter les IED à la main, après avoir discuté avec Monsieur Robin Roche nous avons également décidé que l'utilisateur puisse rentrer directement les IED via le biais dans un premier temps d'un fichier Excel puis finalement d'un fichier txt, ce qui a pu être réalisé.
- La deuxième page accessible depuis la première a pour objectif de fournir les informations concernant les IED relais uniquement, on doit pouvoir retrouver l'ensemble des caractéristiques des IED enregistrées sur la page d'accueil. Chaque relais étant défini dans une classe, on doit pouvoir visualiser pour chaque relais les informations de bases à savoir la tension ainsi que le courant circulant dans les trois phases de chacun d'entre eux. Les relais F650 n'étant pas compatible avec cette partie de la norme, la réception des SV et leur stockage dans l'IHM ne sont donc pas implémentés.
- Enfin la troisième page accessible également depuis la première a pour objectif de « sniffer » les messages Goose provenant du réseau local, notre interface a pour but d'obtenir le même résultat que le logiciel IEDScout, à savoir le temps de déclenchement, l'horodatage du message Goose, le nom de l'IED qui provoque le Défaut sur le réseau, le type de défaut, et enfin la phase sur laquelle est intervenu le défaut.

Avant de commencer la programmation de ce que l'on souhaite obtenir il faut réaliser un schéma sur papier des trois fenêtres pour bien cerner ce que nous devons réaliser.

Page d'accueil

Fichier | Ajouter des IED à partir d'email |

Image de IED	Marque _____	Modèle _____					
	Nom de l'IED _____						
	Adresse IP _____						
	Adresse de sous réseau _____						
	Port IED _____						
<input type="button" value="Ajouter"/> <input type="button" value="Rechercher"/> <input type="button" value="Effacer"/>							
Marque	Modèle	Image	Nom IED	IP	Adresse sous réseau	Port	VID
<input type="button" value="Modifier"/> <input type="button" value="Supprimer"/> <input type="button" value="Vider"/>							

Tableau de Stockage

Figure 6 : Page d'accueil de l'application

Interface des Relais

liste déroulante des relais

Image IED	Marque _____	Modèle _____
	IP _____	Adresse sous réseau _____
	Port _____	VID _____
<input type="checkbox"/> VA : _____ V		
<input type="checkbox"/> VB : _____ V		
<input type="checkbox"/> VC : _____ V		
<input type="checkbox"/> IA : _____ V		
<input type="checkbox"/> IB : _____ V		
<input type="checkbox"/> IC : _____ V		

Figure 5 : Page d'interface des relais de protection

Interface Goose

Time	Nom de l'IED	Type de Défaut	Phase 1	Phase 2	Phase 3

Figure 4 : Page d'interface des messages Gooses

Une fois les trois fenêtres définies il a fallu configurer une fenêtre d'accueil permettant d'accéder avec un bouton à l'une des fenêtres présentées ci-dessus. On a défini sur papier le visuel de cette page que vous pouvez retrouver ci-dessous :



Figure 7 : Visuel de la page d'accueil de l'application

Avant de commencer à détailler pour chaque fenêtre comment elles ont été programmées, vous trouverez ci-dessous un visuel de chaque fenêtre une fois l'interface terminée. L'application a été retranscrite en anglais afin d'être le plus professionnelle possible.

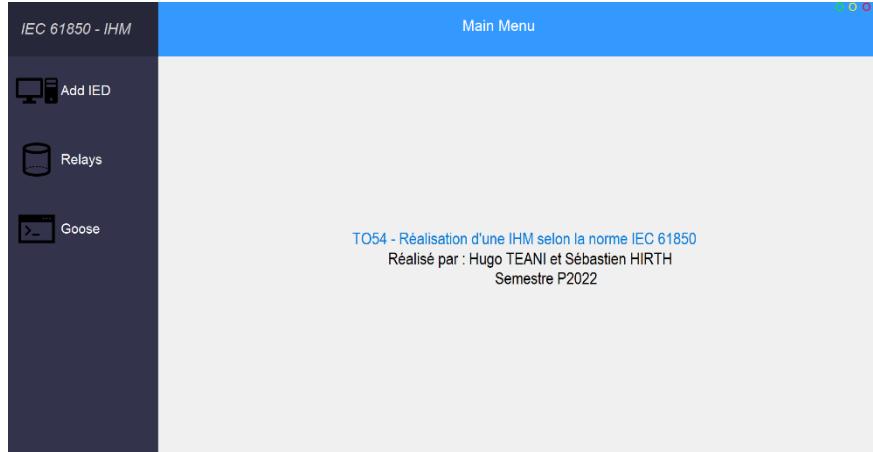


Figure 10 : Page d'accueil de notre IHM

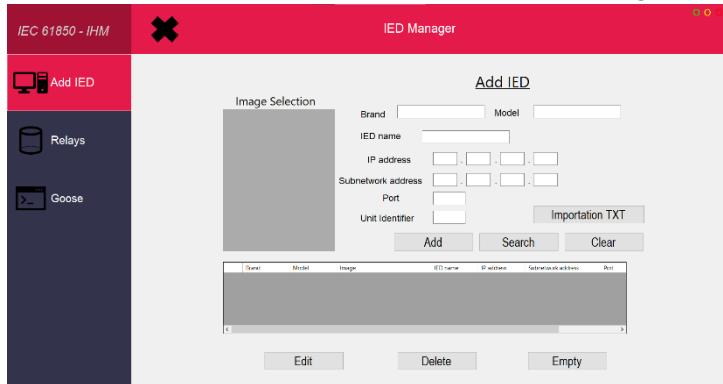


Figure 11 : Fenêtre Ajout des IED de l'IHM

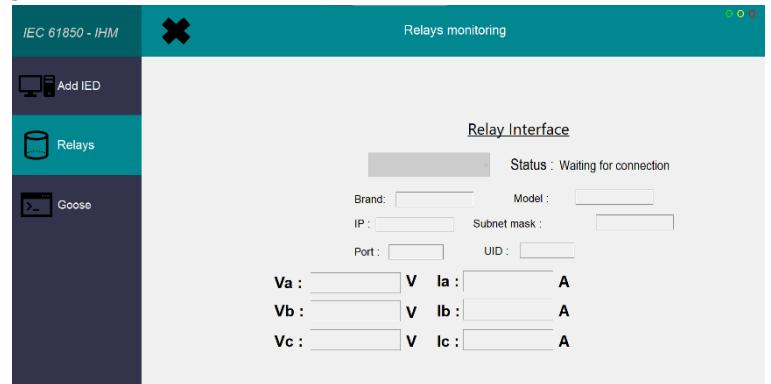


Figure 9 : Fenêtre de l'interface des Relais de l'IHM

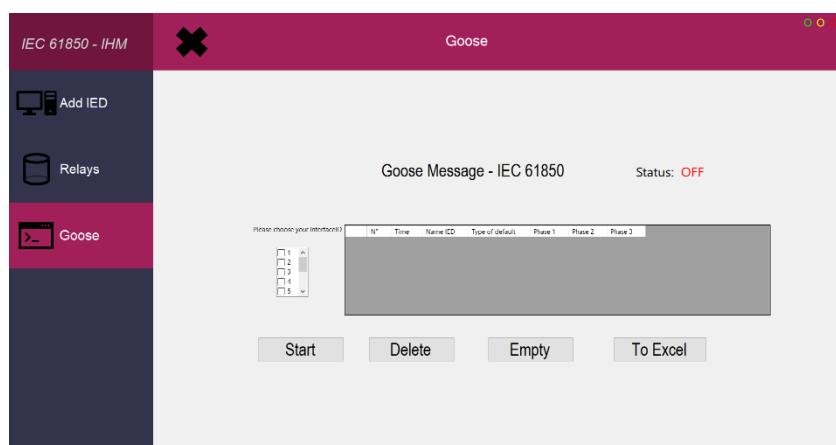


Figure 12 : Fenêtre d'exportation des Goose de l'IHM

Remarque importante :

Afin de bien visualiser l'application sur tout type de d'écran et de ne pas avoir de potentiel bug d'affichage il est nécessaire de privilégier un mode d'affichage avec un zoom à 100%. Si cela n'est pas le cas il faut suivre la démarche suivante : Bureau → Click droit → Paramètre d'affichage → Mise à l'échelle et disposition → 100 %. Comme vous pouvez le voir ci-contre.

Mise à l'échelle et disposition

Certaines applications n'appliquent pas les modifications d'échelle avant d'être fermées et réouvertes.

Modifier la taille du texte, des applications et d'autres éléments

Figure 8 : Mise à l'échelle de l'application

1) Présentation de la fenêtre d'accueil

Le visuel de la page d'accueil de l'application a été présenté ci-dessus nous allons voir dans cette partie comment a été configuré chaque item de cette fenêtre. Afin de rendre l'interface, plus visuelle à regarder nous avons utilisés un fichier au format C# avec des couleurs de pré-enregistré. Durant les prochaines explications nous n'expliquerons pas les éléments visuels de l'application mais la partie fonctionnelle de l'application. La page d'accueil possède donc 3 boutons permettant de sélectionner chacune des trois fenêtres, d'un bouton de retour au menu principal et des boutons habituels d'une application à savoir réduire, agrandir, fermer.

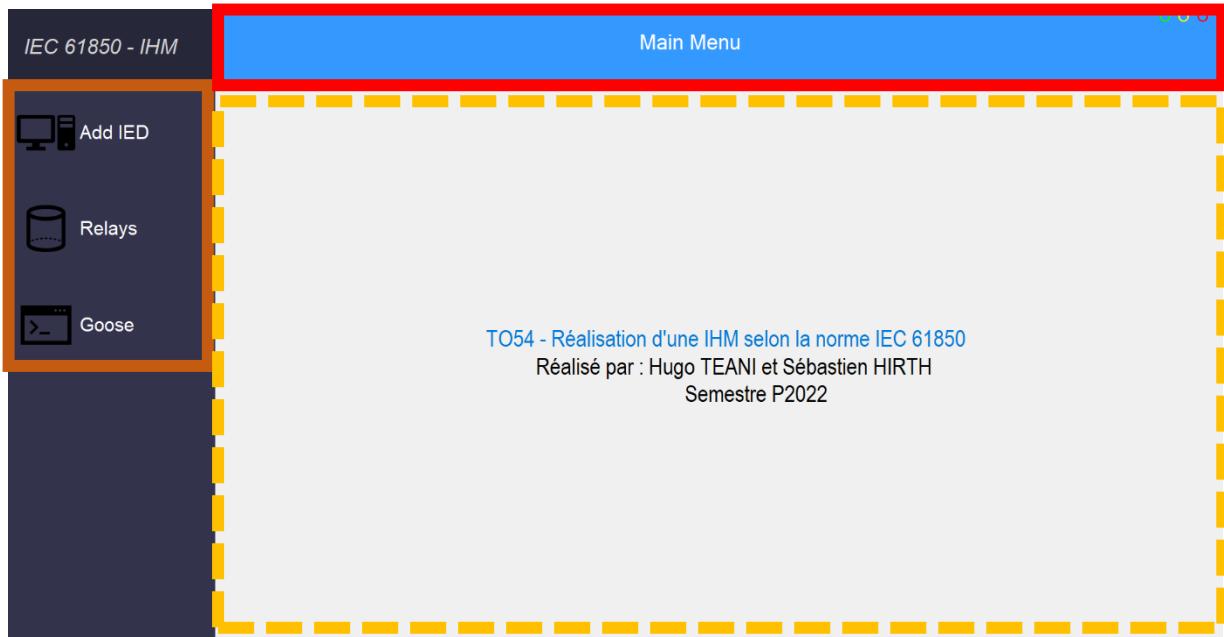


Figure 13 : Page Accueil

- En encadré rouge : c'est un panel d'affichage permettant de savoir sur quelle fenêtre on se situe, ce panel possède aussi les boutons réduire, agrandir, fermer
- En encadré Orange : c'est un panel qui contient les boutons de sélection des fenêtres
- En jaune en pointillé : c'est un panel qui contient la fenêtre en cours de sélection, l'affichage de chacune des fenêtres se fera au sein de ce panel

Explication et détail du code derrière cette fenêtre principale :

- **Les boutons de fermeture, agrandissement, et de réduction de la fenêtre**
 - Bouton de réduction (Bouton vert)



Quand on clique sur ce bouton cela déclenche l'événement suivant qui permet de réduire la fenêtre.

```
208     private void btnMinimize_Click(object sender, EventArgs e)
209     {
210         WindowState = FormWindowState.Minimized;
211     }
212 }
```

- Bouton d'agrandissement (Bouton Jaune)

Quand on clique sur ce bouton cela déclenche l'événement suivant qui permet d'agrandir la fenêtre ou le cas échéant de la réduire si elle est déjà au maximum.

```
200     private void btnMaximize_Click(object sender, EventArgs e)
201     {
202         if (WindowState == FormWindowState.Normal)
203             WindowState = FormWindowState.Maximized;
204         else
205             WindowState = FormWindowState.Normal;
206     }
207 }
```

- o Bouton fermeture (Bouton rouge)

Quand on clique sur ce bouton cela déclenche l'événement suivant qui permet de fermer l'application et de fermer tous les objets encore présents dans la mémoire de l'ordinateur, on commence par supprimer tous les objets présents dans Liste Relais (qui stock les IED, que l'on verra par la suite), puis on clôture l'ensemble des fenêtres (objet FORM) avant de quitter l'application.

```
1 référence
154  private void btnClose_Click(object sender, EventArgs e)
155  {
156      //Fermeture des objets et de l'application
157      for (int i = Gestion.ListeRelais.Count() - 1; i > 0; i--)
158      {
159          Gestion.ListeRelais.RemoveAt(i);
160      }
161      Gestion.FNGoose.Close();
162      Gestion.FNrelais.Close();
163      Gestion.IEDforms.Close();
164      Application.Exit();
165  }
166 }
```

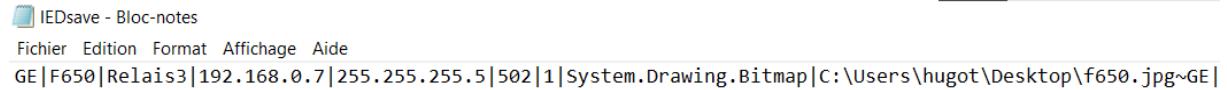
Nous avons en plus ajouté une fonctionnalité permettant de sauvegarder les IED (vue dans une prochaine partie), lorsque l'on ferme l'application dans un fichier texte .txt. On réalise cette manipulation en réalisant la procédure suivante :

- 1^{ère} étape : On teste si dans la liste de relais il y a présence de relais
 - o S'il n'y a pas de relais on exécute le code vu précédemment qui ferme l'application et les objets
- 2^{ème} étape : On créer un fichier texte à la racine du projet où se situe l'exécutable du projet. Puis on stock les données de chaque relais dans ce fichier. Pour pouvoir les réutiliser au lancement de l'application, on sépare donc chaque IED par les caractères suivants « ~ » et les données de chaque relais par les caractères suivants « | ».
- 3^{ème} étape : Après avoir stocké les données dans le fichier on le ferme puis on exécute le code vu précédemment

```
154  private void btnClose_Click(object sender, EventArgs e)
155  {
156      if (Gestion.ListeRelais.Count > 0)
157      {
158          string execPath = AppDomain.CurrentDomain.BaseDirectory;
159          string pathTxtWrite = execPath + "IEDsave.txt";
160          // On supprime les données du précédent fichier
161          System.IO.File.WriteAllText(pathTxtWrite, string.Empty);
162          // Ecriture des relais dans un fichier TXT
163          StreamWriter fichiertxt = new StreamWriter(pathTxtWrite); // path du fichier à mettre en variable global pour laisser le choix à l'utilisateur de l'emplacement
164          //MessageBox.Show(Gestion.ListeRelais[0].getpath());
165          // Ecriture des relais dans le fichier
166
167          for (int i = 0; i < Gestion.ListeRelais.Count; i++)
168          {
169              fichiertxt.WriteLine(Gestion.ListeRelais[i].getMarque().ToString() + "|");
170              fichiertxt.WriteLine(Gestion.ListeRelais[i].getModele().ToString() + "|");
171              fichiertxt.WriteLine(Gestion.ListeRelais[i].getNom().ToString() + "|");
172              fichiertxt.WriteLine(Gestion.ListeRelais[i].getIP().ToString() + "|");
173              fichiertxt.WriteLine(Gestion.ListeRelais[i].getMAC().ToString() + "|");
174              fichiertxt.WriteLine(Gestion.ListeRelais[i].getPort().ToString() + "|");
175              fichiertxt.WriteLine(Gestion.ListeRelais[i].getID().ToString() + "|");
176              fichiertxt.WriteLine(Gestion.ListeRelais[i].getIMG() + "|");
177              fichiertxt.WriteLine(Gestion.ListeRelais[i].getpath().ToString());
178              fichiertxt.WriteLine("~");
179          }
180          fichiertxt.Close();
181      }
182
183
184      //Fermeture des objets et de l'application
185      for (int i = Gestion.ListeRelais.Count - 1; i > 0; i--)
186      {
187          Gestion.ListeRelais.RemoveAt(i);
188      }
189      Gestion.FNGoose.Close();
190      Gestion.FNrelais.Close();
191      Gestion.IEDforms.Close();
192      Application.Exit();
193  }
194 }
```

Remarques importantes : Le stockage des IED dans le fichier .txt prend en compte le chemin d'accès des images que vous avez choisi. Si vous déplacez où supprimez l'image d'emplacement, l'application ne se lancera plus. Pour régler le problème il faut supprimer le fichier **IEDsave** dans le dossier de l'application puis relancer l'application.

Voici le fichier rempli une fois l'application quittée :



```
Fichier Edition Format Affichage Aide  
GE|F650|Relais3|192.168.0.7|255.255.255.5|502|1|System.Drawing.Bitmap|C:\Users\hugot\Desktop\f650.jpg~GE|
```

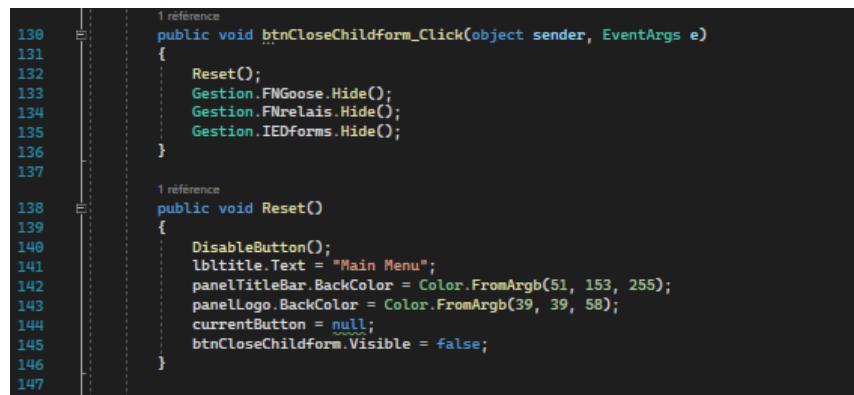
Figure 14 : Fichier de stockage des IED (IEDsave)

Nous verrons donc dans la partie sur la fenêtre d'ajout des IED comment ce fichier va être utilisé pour réimplémenter les IED dans l'IHM automatiquement.

- **Bouton pour revenir au Menu principal**



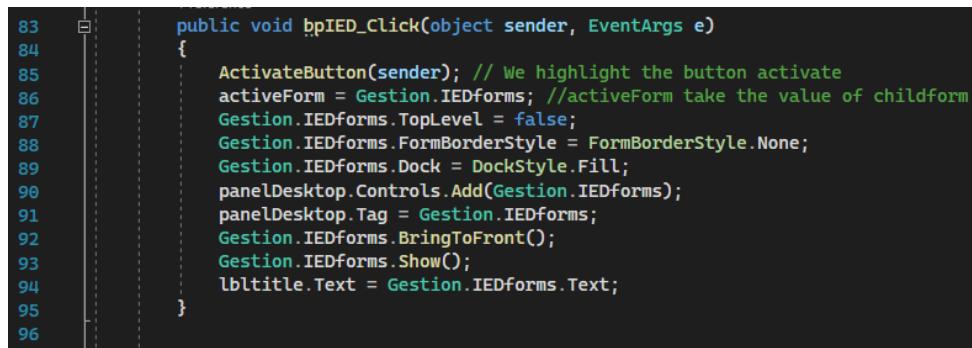
Le bouton permet de revenir au menu principal en cachant les fenêtres exécutées encore en arrière-plan, et en changeant, le label contenu dans le panel, ainsi que plusieurs manipulations graphiques pour rendre l'application plus attractive.



```
1 référence  
130     public void btnCloseChildform_Click(object sender, EventArgs e)  
131     {  
132         Reset();  
133         Gestion.FNGoose.Hide();  
134         Gestion.FNrelais.Hide();  
135         Gestion.IEDforms.Hide();  
136     }  
137  
138     1 référence  
139     public void Reset()  
140     {  
141         DisableButton();  
142         lbltitle.Text = "Main Menu";  
143         panelTitleBar.BackColor = Color.FromArgb(51, 153, 255);  
144         panelLogo.BackColor = Color.FromArgb(39, 39, 58);  
145         currentButton = null;  
146         btnCloseChildform.Visible = false;  
147     }
```

- **Bouton permettant d'accéder aux autres fenêtres**

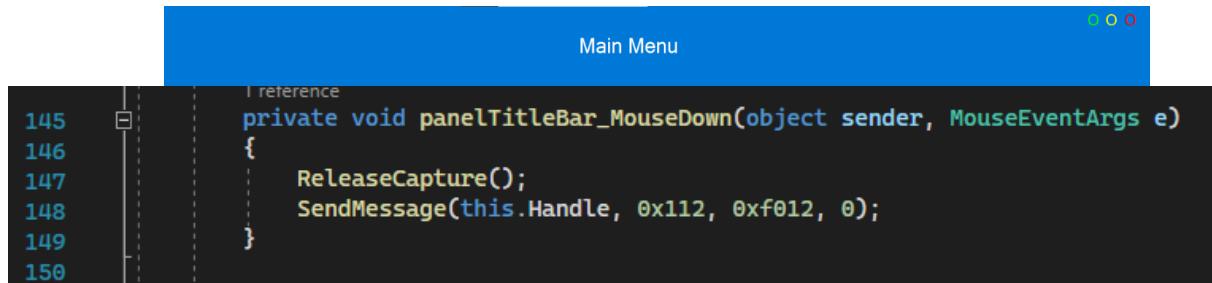
Les événements déclenchés par les trois boutons sont identiques dans la façon d'être programmé, ils font cependant référence à leur fenêtre respective. Nous allons donc détailler un seul bouton. Ci-dessous les lignes 85 à 89 permettent de faire un rendu visuel lors du clic sur le bouton. Les lignes 90 à 93 permettent d'ajouter dans le panel prévu à cet effet la fenêtre en question et par la commande Show(), de l'afficher à l'écran. La ligne 94 permet de changer le texte du label présent sur la fenêtre principale en prenant le texte du nom de la fenêtre.



```
83     public void bpIED_Click(object sender, EventArgs e)  
84     {  
85         ActivateButton(sender); // We highlight the button activate  
86         activeForm = Gestion.IEDforms; //activeForm take the value of childform  
87         Gestion.IEDforms.TopLevel = false;  
88         Gestion.IEDforms.FormBorderStyle = FormBorderStyle.None;  
89         Gestion.IEDforms.Dock = DockStyle.Fill;  
90         panelDesktop.Controls.Add(Gestion.IEDforms);  
91         panelDesktop.Tag = Gestion.IEDforms;  
92         Gestion.IEDforms.BringToFront();  
93         Gestion.IEDforms.Show();  
94         lbltitle.Text = Gestion.IEDforms.Text;  
95     }  
96 }
```

- **Faire bouger la fenêtre**

Une fonctionnalité a été rajouté à l'aide d'un événement si l'utilisateur reste appuyé avec le click gauche sur le panel ci-dessous alors il peut déplacer la fenêtre de l'application pour la mettre sur un autre écran par exemple.



```

Main Menu

private void panelTitleBar_MouseDown(object sender, MouseEventArgs e)
{
    ReleaseCapture();
    SendMessage(this.Handle, 0x112, 0xf012, 0);
}

```

2) Présentation de la fenêtre permettant d'ajouter des IED

Nous allons dans les prochaines parties détailler comment a été conçu chaque fenêtres constituant l'IHM. Nous allons commencer par détailler la fenêtre permettant de gérer les IED sur le réseau. Cette fenêtre possède une multitude de fonctionnalité. En parallèle de cette présentation de fenêtre, nous allons également détailler la classe Relais, et la ListeRelais qui stock les IED présent sur le réseau.

Remarque importante :

Par la suite nous utilisons Excel, pour l'export des messages Goose, il faut impérativement que vous possédiez Excel sur votre ordinateur pour pouvoir faire fonctionner pleinement l'IHM.

Cette fenêtre est constituée de la façon suivante :

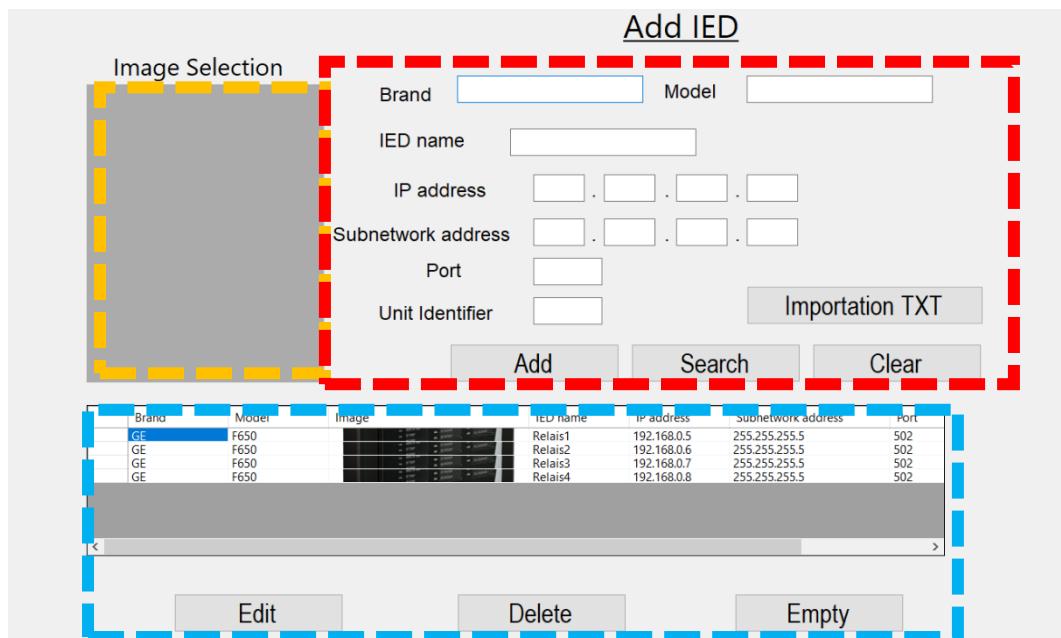


Figure 15 : Fenêtre Gestion des IED

- En encadré rouge : cette partie permet d'ajouter les IED en lui donnant ces caractéristiques sur le réseau
- En encadré Orange : En cliquant sur le fond gris une boite de dialogue s'ouvre permettant de choisir une photo de l'IED
- En encadré Bleu : C'est un tableau qui stock chaque relais ajouté sur le réseau

- **Programmation de la récupération des IED au lancement de la fenêtre**

Lors de la fermeture du programme, comme nous l'avons vu dans la précédente partie, les IED sont stockés dans un fichier à la racine du projet nommé « IED Save ». Le but de cette partie est de charger les IED à partir de ce fichier lorsque l'événement de lancement de la fenêtre est demandé par l'utilisateur en cliquant sur le bouton d'ouverture de la fenêtre. Voyons comment cet événement est géré. Voici la trame stockée pour un IED dans le fichier IEDsave :

GE | F650 | Relais1 | 192.168.0.5 | 255.255.255.5 | 502 | 1 | System.Drawing.Bitmap | C:\Users\hugot\Desktop\f650.jpg~

Analyse de la trame :

- ~ : Ces caractères nous permettent de faire la coupure entre deux IED
- | : Ces caractères nous permettent de faire la coupure des données au sein d'un même IED.

Il faut donc réaliser un programme permettant d'extraire les IED de ce fichier et de le restocker dans notre IHM à la fois dans le tableau des IED mais également de créer pour chaque IED un nouvel objet de type relais. Afin de l'expliquer le plus facilement possible nous allons voir la fonction à l'aide de l'organigramme suivant :

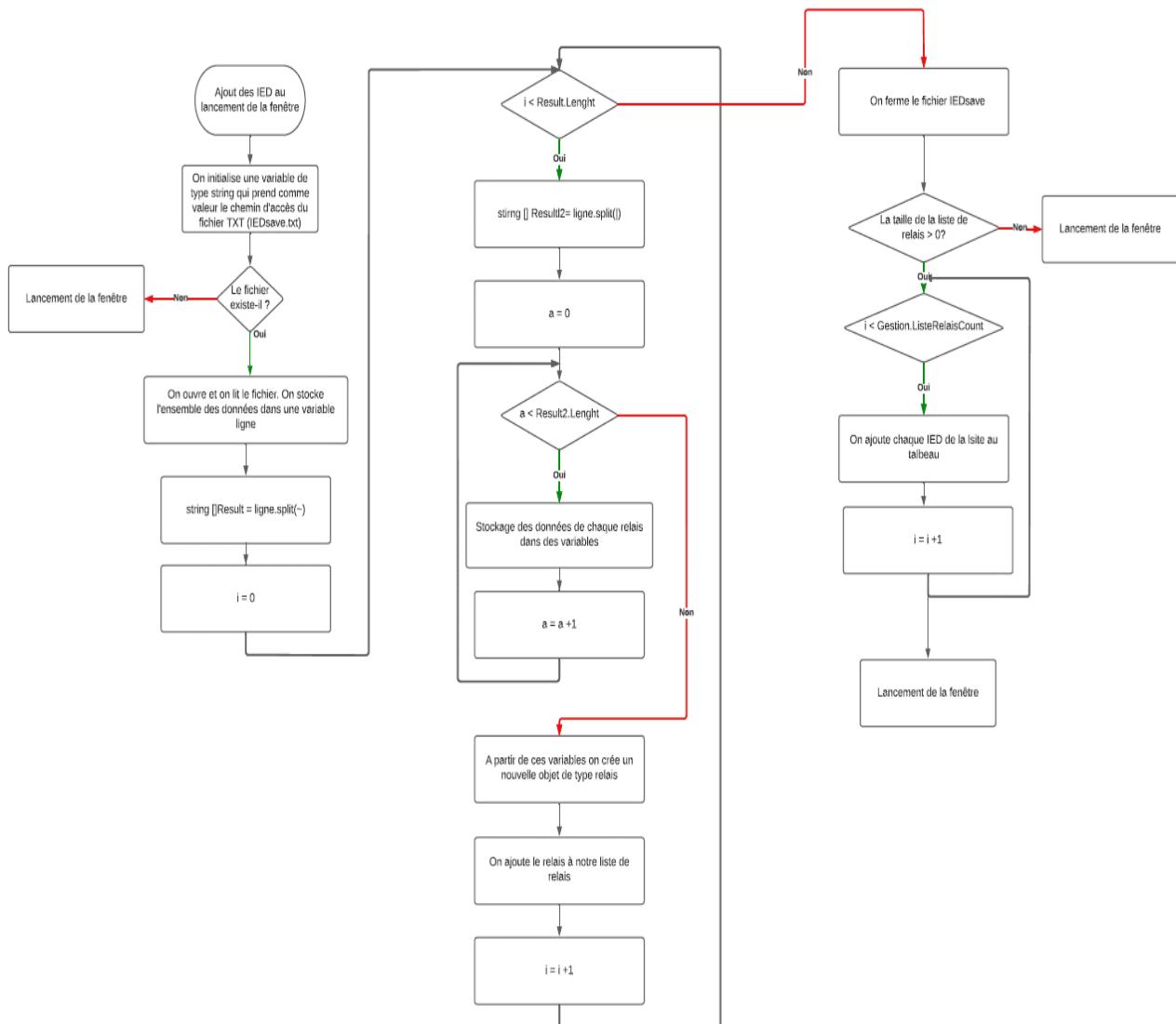


Figure 16 : Organigramme de la fonction permettant d'ajouter les IED au lancement de l'application

Le code réalisé à l'aide de cet organigramme est le suivant :

```
    Informer();
    private void IDEnews_Load(object sender, EventArgs e)
    {
        //MessageBox.Show("fonctionnement");
        string pathload = "C:\\Windows\\Temp\\attchtemp\\";
        string pathloadfile = pathload + "IDEnews.txt";
        if (File.Exists(pathloadfile))
        {
            StreamReader lecturefichier = new StreamReader(pathloadfile);
            //Déclaration d'une variable qui lit toutes les lignes
            string ligne = lecturefichier.ReadToEnd();
            string[] result = ligne.Split(new char[] { '\n' });
            string[] result2 = result[0].Split(new char[] { ';' });
            for (int i = 0; i < result.Length - 1; i++)
            {
                //MessageBox.Show(result[i].Length.ToString()); //7
                string[] result3 = result[i].Split(' ');
                string[] result4 = result3[0].Split(' ');
                for (int a = 0; a < result4.Length - 1; a++)
                {
                    //MessageBox.Show(result4[a].ToString()); //8
                    ResultIDrelais = result4[0];
                    ResultIDrelai = result4[1];
                    ResultNom = result4[2];
                    ResultIP = result4[3];
                    ResultMAC = result4[4];
                    ResultPort = result4[5];
                    ResultRelais = result4[6];
                    ResultpathImage = result4[7];
                    //MessageBox.Show(ResultpathImage);
                    ImageID_E Image = new Bitmap(ResultpathImage);
                }
                // Stockage des relais dans une liste (Relais liste préalablement créée)
                Relais relais = new Relais(ResultIDrelais, ResultIDrelai, ResultNom, Convert.ToInt32(ResultPort), Convert.ToInt32(ResultID), ImageID_E.Image, ResultpathImage); // Définit un objet relais
                Gestion.ListeRelais.Add(relais);
                Gestion.ListeRelais.Add(relais);
                Gestion.Relais.RefreshList();
            }
            lecturefichier.Close();
        }
        if (Gestion.ListeRelais.Count > 0)
        {
            for (int i = 0; i < Gestion.ListeRelais.Count; i++)
            {
                //MessageBox.Show(Gestion.ListeRelais[i].getIP().ToString());
                GridDonnees.Rows.Add(Gestion.ListeRelais[i].getID(), Gestion.ListeRelais[i].getNom(), Gestion.ListeRelais[i].getIP().ToString(), Gestion.ListeRelais[i].getPort(), Gestion.ListeRelais[i].getMAC().ToString(), Gestion.ListeRelais[i].getPort(), Gestion.ListeRelais[i].getIP().ToString(), Gestion.ListeRelais[i].getNom());
            }
        }
        ImageID_E.Image = null;
    }
}
```

- Programmation du bouton Ajouter

Lorsque l'on clique sur le bouton Ajouter de la fenêtre cela déclenche un évènement, dont celui-ci à plusieurs conséquences. Lorsque le click est réalisé, on commence tout d'abord à regarder si l'IED que l'on souhaite ajouter n'existe pas déjà dans la liste des IED, on appelle donc la fonction **SameIED()**. Si l'IED n'existe pas déjà alors on fait un test pour voir si l'ensemble des données (labels) ont bien été remplies par l'utilisateur à l'aide d'une deuxième fonction **TESTIED()**. Si cela est le cas on ajoute alors l'IED dans le tableau de la forme et on créer un nouvel objet de la classe **Relais** contenant les données de l'IED. Si ce n'est pas le cas alors on affiche un message d'erreur qui permet à l'utilisateur de changer le label qu'il a mal rempli.

La fonction **SameIED()** ci-dessous renvoie un booléen, si une IED porte déjà le même nom dans le tableau alors la variable *Same* se met à TRUE le cas échéant elle se met à FALSE.

```
32     public bool SameID()
33     {
34         bool Same = false; //On initialise la variable comme fausse
35         if (GridDonnee.Rows.Count > 0) // On recherche si la donnée est identique uniquement lorsque la liste comporte au moins une ligne
36         {
37             for (int i = 0; i < GridDonnee.Rows.Count; i++) // On parcours toutes les données déjà dans le tableau
38             {
39                 if (GridDonnee.Rows[i].Cells[3].Value.ToString() == Nomtxt.Text) //Si une valeur du tableau est identique à la valeur rentrée par l'utilisateur alors on passe le flag à vrai
40                 {
41                     Same = true;
42                     break;
43                 }
44             }
45         }
46         return Same;
47     }
48 }
```

La fonction **TESTIED()** ci-dessous renvoie également un booléen, si l'utilisateur ne rentre pas toutes les données correctement alors la variable *FlagTESTIED* passe à FALSE, au contraire si toutes les données sont correctes le flag passe à TRUE.

```
172     public bool TESTED()
173     {
174         bool FlagTESTED = false;
175         if (Nonetxt.Text.Length > 0 && Nonetxt.Text != "" && Modeletxt.Text.Length > 0 && Modeletxt.Text != "" && Marquetxt.Text.Length > 0 && Marquetxt.Text != "" && ImageIED.Image != null)
176         {
177             FlagTESTED = true;
178         }
179         return FlagTESTED;
180     }
181 }
```

Afin de voir comment sont utilisées ces deux fonctions pour l'événement ajouter nous allons réaliser un organigramme permettant de bien comprendre le fonctionnement du bouton AJOUTER qui est complexe :

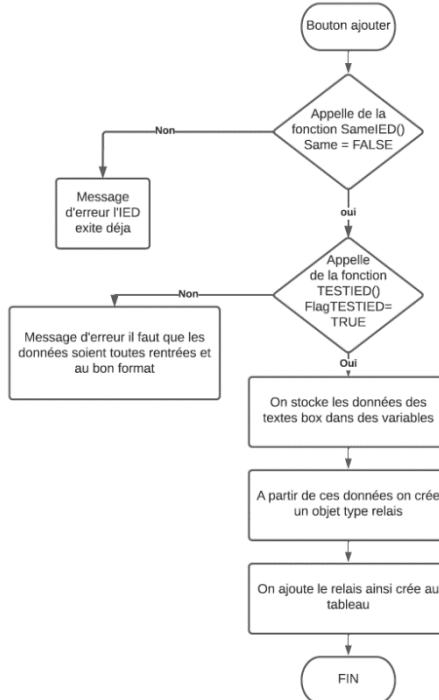


Figure 17 : Organigramme du bouton ajouter

Ci-dessous le code de cette fonction :

```

138     {
139         public void Ajouter_Click(object sender, EventArgs e)
140         {
141             if (!SameIED()) //Si la méthode renvoie Same = Faux alors c'est qu'il n'existe pas de relais sous ce nom on peut donc le créer
142             {
143                 if (TESTIED())
144                 {
145                     try
146                     {
147                         // Stockage des résultats
148                         ResultMarque = Marquetxt.Text.ToString();
149                         ResultModelo = Modeletxt.Text.ToString();
150                         ResultNom = Nometxt.Text.ToString();
151                         ResultIP = (float.Parse(IP1txt.Text) + "." + float.Parse(IP2txt.Text) + "." + float.Parse(IP3txt.Text) + "." + float.Parse(IP4txt.Text)).ToString();
152                         ResultMAC = (float.Parse(MAC1txt.Text) + ":" + float.Parse(MAC2txt.Text) + ":" + float.Parse(MAC3txt.Text) + ":" + float.Parse(MAC4txt.Text)).ToString();
153                         ResultPort = (float.Parse(Porttxt.Text)).ToString();
154                         ResultUID = (float.Parse(UUIDtxt.Text)).ToString();
155                         GridDonnee.Rows.Add(ResultMarque, ResultModelo, ImageIED.Image, ResultIP, ResultMAC, ResultPort, ResultUID);
156                         // Stockage des relais dans une liste (Relais list préalablement créée)
157                         Relais relais = new Relais(ResultMarque, ResultModelo, ResultNom, ResultIP, ResultMAC, Convert.ToInt32(ResultPort), Convert.ToInt32(ResultUID), ImageIED.Image, pathIMG); // Définit un objet relais
158                         Gestion.ListeRelais.Add(relais); // Ajout du relais à la liste de relais
159                         Gestion.FnRelais.RefreshListe();
160                     }
161                     catch
162                     {
163                         MessageBox.Show("Please note the format", "Error",
164                         MessageBoxButtons.OK, MessageBoxIcon.Error);
165                     }
166                 }
167                 else
168                 {
169                     MessageBox.Show("The data of the IED is not correct please correct it", "Error",
170                         MessageBoxButtons.OK, MessageBoxIcon.Error);
171                 }
172             }
173         }
174     }

```

- Choix de l'image correspondant à notre IED

Chaque IED sur le réseau est référencé par rapport à sa marque, son modèle, son nom, son adresse IP, son masque de sous réseau, son port, son UID et enfin une image afin de bien voir à quoi ressemble l'IED branché sur le réseau. Une fois cliqué à l'endroit prévu une boîte de dialogue s'ouvre nous demandant de choisir une image

```

49     {
50         public void ImageIED_Click(object sender, EventArgs e)
51         {
52             openFileDialog.Multiselect = false; //Permet à l'utilisateur de ne choisir que un fichier image
53             if (openFileDialog.ShowDialog() == DialogResult.OK)
54             {
55                 ImageIED.Image = new Bitmap(openFileDialog.FileName); // Permet de créer une instance de notre image
56             }
57         }

```

- **Classe Relais et ListeRelais**

Afin d'avoir accès à cette classe dans l'ensemble des fenêtres de l'application, nous avons déclaré notre objet dans le fichier Program.cs qui s'exécute au lancement de l'application. La ListeRelais permet de stocker comme son nom l'indique l'ensemble des relais de la classe Relais :

```
17  public static class Gestion
18  {
19      public static List<Relais> ListeRelais = new List<Relais>(); //Créer une liste permettant de stocker les relais
```

Quant à la classe relais, elle est située à la racine du projet et est nommée Relais.cs. Son objectif est qu'à chaque ajout d'IED (Vue précédent), elle créer un nouvel objet de type relais, qui stock dans cet objet l'ensemble des caractéristiques du relais vue précédemment. Le classe relais rassemble donc tous les attributs suivants :

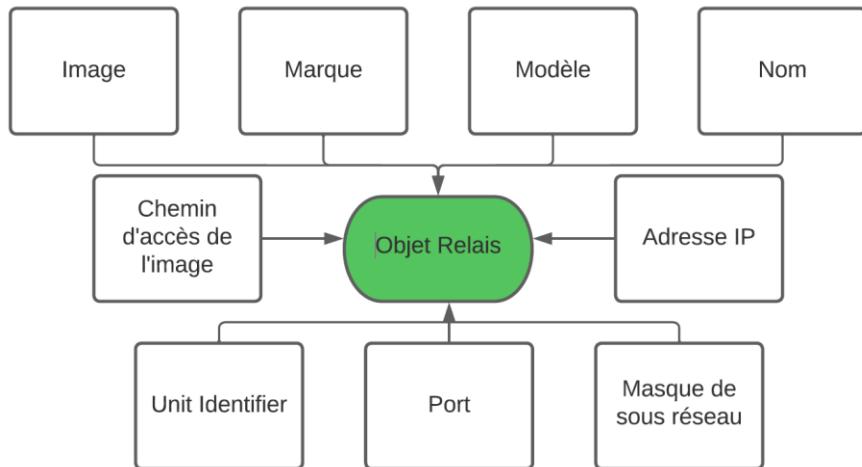


Figure 18 : Attribut de notre classe Relais.cs

- **Bouton effacer**

Ce bouton sert simplement à effacer l'ensemble des labels de la fenêtre et à mettre la PictureBox de l'image vide

```
58  public void Effacer_Click(object sender, EventArgs e)
59  {
60      Marquetxt.Text = "";
61      Modeletxt.Text = "";
62      Nomtxt.Text = "";
63      IP1txt.Text = "";
64      IP2txt.Text = "";
65      IP3txt.Text = "";
66      IP4txt.Text = "";
67      MAC1txt.Text = "";
68      MAC2txt.Text = "";
69      MAC3txt.Text = "";
70      MAC4txt.Text = "";
71      Porttxt.Text = "";
72      UIDtxt.Text = "";
73      ImageIED.Image = null;
74  }
```

- **Bouton Rechercher**

Lorsqu'il commence à avoir beaucoup d'IED sur le réseau il peut être difficile de trouver si un IED est déjà présent sur le réseau ou non. En entrant le nom de l'IED dans le label prévu à cet effet et en appuyant par la suite sur le bouton rechercher on obtient une réponse qui nous indique de la présence ou non de l'IED. Cet événement se sert seulement de la valeur de retour du booléens de la fonction **SameIED()** ;

```

107     public void Rechercher_Click(object sender, EventArgs e)
108     {
109         if (SameIED()) //Si la méthode renvoie Same = Vrai
110         {
111             MessageBox.Show("The IED exists", "Found!",
112             MessageBoxButtons.OK, MessageBoxIcon.Information);
113         }
114         else
115         {
116             MessageBox.Show("The IED doesn't exists", "Error",
117             MessageBoxButtons.OK, MessageBoxIcon.Error);
118         }
119     }

```

- **Bouton AjouterIED depuis un fichier texte**

Nous avons décidé avec Monsieur Robin Roche de rajouter une fonctionnalité qui permet à l'aide d'un fichier texte d'ajouter directement les IED, à la fois dans la liste de relais mais également dans le tableau de la fenêtre. Cette étape était difficile car il fallait pouvoir réussir à lire dans un fichier texte pour pouvoir extraire depuis ce fichier les données pour pouvoir les ajouter dans notre IHM.

Pour ce faire il faut disposer de notre fichier source texte nommée IED. Ce fichier texte est constitué de la façon suivante :

```

IED - Bloc-notes
Fichier Edition Format Affichage Aide
GE|F650|Relais1|192.168.0.5|255.255.255.5|502|1|System.Drawing.Bitmap|C:\Users\hugot\Desktop\f650.jpg~

```

Figure 19 : Capture d'écran du fichier IED

Remarque Importante :

Pour que cela fonctionne depuis chez vous il faut penser à modifier les chemins d'accès menant à vous images d'IED. Si cela n'est pas fait l'application ne va plus fonctionner.

La programmation de cette partie comprend reprend ce qui a été réalisé lors du lancement de la fenêtre et le chargement des IED. La seule chose qui change c'est lorsque que l'on appui sur le bouton d'importation une boite de dialogue s'ouvre nous demandant d'ouvrir un fichier .txt une fois le fichier IED.txt ouvert la manipulation pour séparer les données des IED et les IED entre eux reste la même que précédemment.

Après avoir ajouté et ouvert le fichier on obtient le résultat suivant sur l'IHM :

	Brand	Model	Image	IED name	IP address	Subnetwork address	Port	UnitIdentifier
	GE	F650	[Image Placeholder]	Relais1	192.168.0.5	255.255.255.5	502	1
	GE	F650	[Image Placeholder]	Relais2	192.168.0.6	255.255.255.5	502	1
	GE	F650	[Image Placeholder]	Relais4	192.168.0.8	255.255.255.5	502	1
	GE	F650	[Image Placeholder]	Relais3	0.168.0.7	255.255.255.5	502	1

Figure 20 : Résultat obtenu à la suite de l'ouverture du fichier IED.txt

- **Bouton Modifier**

Il arrive parfois en rentrant les caractéristiques d'un IED de se tromper une fois que l'IED a été ajouté. En cliquant sur la ligne de l'IED à modifier et en appuyant sur le bouton on parvient à changer le label qui n'était pas correct. On peut par la suite valider notre choix. L'ancien IED est alors supprimé et le nouveau le remplace, à la fois dans la liste du tableau mais également dans la liste des relais qui permet de communiquer entre les fenêtres. Le code pour cette modification fait 100 lignes et est détaillé pour chaque ligne du code. Afin d'être plus parlant on réalise un exemple, en reprend l'exemple ci-dessus et en modifiant le relais 3.

On clique sur la ligne du relais 3 puis on appuie sur le bouton modifier, on obtient le résultat suivant :

The screenshot shows the 'Add IED' interface. At the top, there is an 'IED Image Selection' section with a GE F650 device image. Below it, form fields include 'Brand' (GE), 'Model' (F650), 'IED name' (Relais3), 'IP address' (0.168.0.7), 'Subnetwork address' (255.255.255.5), 'Port' (502), and 'Unit Identifier' (1). There is also an 'Importation TXT' button. Below the form is a table listing IEDs:

Brand	Model	Image	IED name	IP address	Subnetwork address	Port	Unit identifier
GE	F650		Relais1	192.168.0.5	255.255.255.5	502	1
GE	F650		Relais2	192.168.0.6	255.255.255.5	502	1
GE	F650		Relais4	192.168.0.8	255.255.255.5	502	1
GE	F650		Relais3	0.168.0.7	255.255.255.5	502	1

At the bottom are buttons for 'Validate', 'Delete', and 'Empty'. The 'Modify' button is highlighted with a blue border.

Figure 21 : Présentation de la partie Bouton Modifier

On peut alors modifier ce que l'on a à changer. Pour le test on modifie seulement le nom de l'IED. Puis on appuie sur le bouton Valider en bas à gauche. On vérifie par la suite que dans la liste il est bien changé, comme sur la capture ci-dessous :

The screenshot shows the 'Add IED' interface. The 'IED name' field now contains 'testModifieation'. The 'Importation TXT' button is visible. Below the form is a table listing IEDs:

Brand	Model	Image	IED name	IP address	Subnetwork address	Port	Unit identifier
GE	F650		Relais1	192.168.0.5	255.255.255.5	502	1
GE	F650		Relais2	192.168.0.6	255.255.255.5	502	1
GE	F650		Relais4	192.168.0.8	255.255.255.5	502	1
GE	F650		testModifieation	0.168.0.7	255.255.255.5	502	1

At the bottom are buttons for 'Modify', 'Delete', and 'Empty'. The 'Modify' button is highlighted with a blue border.

Figure 22 : Résultat obtenu de la modification

- **Bouton Supprimer**

Ce bouton permet une fois une ligne du tableau sélectionnée de supprimer l'IED, à la fois du tableau mais aussi de la liste des relais. Cela est un peu complexe car l'index de la ligne n'est pas forcément identique à celui de la classe Liste de relais. Il faut donc faire attention de supprimer le bon IED dans le tableau mais également dans la liste de relais. Comme vous pouvez le voir sur le code ci-dessous :

```
82     public void Supprimer_Click(object sender, EventArgs e)
83     {
84         if (GridDonnee.Rows.Count > 0)
85         {
86             int Ligne = GridDonnee.CurrentRow.Index; //On initialise une variable qui prend comme paramètre le nom de la ligne du tableau sélectionnée
87             int indextab = 0;
88             for (int i = 0; i < Gestion.ListeRelais.Count; i++)
89             {
90                 if (Gestion.ListeRelais[i].getNom() == GridDonnee.CurrentRow.Cells[3].Value.ToString())
91                 {
92                     indextab = i;
93                     break;
94                 }
95             }
96             GridDonnee.Rows.RemoveAt(Ligne);
97             Gestion.ListeRelais.RemoveAt(indextab); //Supprime dans la liste de relais le relais sélectionné
98             Gestion.FNrelais.RefreshListe();
99         }
100     }
101     else
102     {
103     }
104 }
105 }
106 }
```

- **Bouton Vider**

Ce bouton permet de supprimer l'intégralité des IED du tableau et de la liste des relais.

```
76     public void Vider_Click(object sender, EventArgs e)
77     {
78         GridDonnee.Rows.Clear(); // On supprime toutes les données du tableau
79         Gestion.ListeRelais.Clear(); //Supprime toute la liste de relais sélectionnées
80         Gestion.FNrelais.RefreshListe();
81     }
```

3) Présentation de la fenêtre permettant de gérer/monitorer les IED relais

Cette deuxième fenêtre permet de suivre en temps réel les données fournies par les IED relais enregistrés dans la première fenêtre. Un certain nombre d'information sont présents sur cette fenêtre. On peut à l'aide d'une liste déroulante choisir l'un des IED présent sur le réseau qui a été enregistré lors de la première étape. Cette fenêtre nous permet de voir les caractéristiques de l'IED sélectionné. On peut également voir si l'IED est connecté ou non sur le réseau à l'aide d'un label qui sera présenté dans la deuxième partie de Sébastien sur la IEC-61850. Des labels ont également été prévus afin de récupérer les Samples Values (SV) de nos relais à savoir les tensions et courants de phases, mais nous avons appris bien trop tard dans le projet que les relais ne sont pas compatibles avec cette caractéristique de la norme. Nous les avons cependant laissés, si d'autre groupe continue notre projet sur cette norme avec de nouveau relais de protection d'une autre marque compatible.

Cette fenêtre se présente de la façon suivante :

The screenshot shows a window titled "Relay Interface". At the top right, it says "Status : Waiting for Connection". Below that, there are fields for "Brand" and "Model", both with empty input boxes. Underneath are fields for "IP", "Subnet mask", "Port", and "UID", each with an empty input box. At the bottom, there are three sets of phase values labeled "Va", "Vb", and "Vc", each followed by a voltage value (V) and a current value (Ia, Ib, Ic) in Amperes (A). The values are currently empty.

Figure 23 : Fenêtre Monitoring des IED

En fonctionnement cette fenêtre se présente de la façon suivante :

The screenshot shows the same "Relay Interface" window, but now with "Status : Disconnected" in red text. To the left of the window is a small image of a black industrial control unit. The phase value fields at the bottom are now populated with data: Va = 230V, Ia = 10A; Vb = 230V, Ib = 10A; Vc = 230V, Ic = 10A. To the right of the window, there is a large blue sidebar with the text "Relais1", "Relais2", "Relais4", and "TestModification" stacked vertically.

Figure 24 : Fenêtre Monitoring des IED en fonctionnement

- La fenêtre se compose donc d'une liste déroulante qui permet de naviguer entre les différents IED présent sur le réseau comme vous pouvez le voir ci-dessus.
 - Lorsque l'on clique sur un relais dans la liste déroulante les données rentrées dans la première fenêtre stockées dans la liste de relais, s'affichent à l'écran
 - Un label nous permet de connaître l'état du relais sur le réseau
 - Différents labels nous permettent d'afficher les données fournies par les SV (Samples Value) de la norme IEC-61850, qui ne sont pas compatibles avec nos relais

- Programmation de l'événement lors d'une clique sur la liste déroulante

Lorsque l'on clique sur un item de la liste déroulante alors l'index de la liste déroulante sélectionné change. Nos données étant stockés dans la liste de relais, il suffit seulement de piocher dans la liste des relais au bon index pour les afficher sur les différents labels à l'écran.

```
    public void ListeDerouante_SelectedIndexChanged(object sender, EventArgs e)
    {
        for (int i = 0; i < Gestion.ListeRelais.Count(); i++)
        {
            if (Gestion.ListeRelais[i].getNom() == ListeDerouante.Text)
            {
                FNRMarque.Text = Gestion.ListeRelais[i].getNom();
                FNRMmodele.Text = Gestion.ListeRelais[i].getModele();
                FNRIp.Text = Gestion.ListeRelais[i].getIP();
                FNRMAC.Text = Gestion.ListeRelais[i].getMAC();
                FNRPoRT.Text = Gestion.ListeRelais[i].getPort().ToString();
                FNRIUID.Text = Gestion.ListeRelais[i].getUID().ToString();
                FNRIImageIED.Image = Gestion.ListeRelais[i].getIMG();
                return;
            }
        }
    }
}
```

- Programmation de la fonction qui permet de rafraîchir la liste des relais lors d'un changement des IED

Afin de faire communiquer les deux fenêtres en elles (IEDforms et FNRELAIS) il faut une fonction qui permette lors d'un ajout , d'une modification ou lors de la suppression d'un IED dans la première fenêtre de venir le modifier également dans la deuxième. C'est fonction s'appelle **RefreshListe()** elle est appelée dans la première fenêtre (IEDforms)dans les évènements des boutons de suppression, d'ajout et de modification. Cette fonction une fois appelée supprime toutes les données dans la liste déroulante puis elles les rajoutent avec la nouvelle liste des relais modifiée.

```
7 références
20     public void RefreshListe()
21     {
22         ListeDeroulante.Items.Clear();
23         FNRMarque.Text = "";
24         FNRModele.Text = "";
25         FNRPID.Text = "";
26         FNRMAC.Text = "";
27         FNRPPort.Text = "";
28         FNRUID.Text = "";
29         ListeDeroulante.Text = "";
30         for (int i = 0; i < Gestion.ListeRelais.Count(); i++)
31         {
32             ListeDeroulante.Items.Add(Gestion.ListeRelais[i].getNom());
33         }
34     }
35     if (ListeDeroulante.Items.Count > 0)
36     {
37         ListeDeroulante.SelectedIndex = 0;
38         ListeDeroulante.Enabled = true;
39     }
40     else
41     {
42         ListeDeroulante.Enabled = false;
43     }
44 }
```

4) Présentation de la fenêtre permettant de gérer les messages Goose

Cette troisième et dernière fenêtre permet d'obtenir en temps réel les messages Goose interceptés sur le réseau Local. Cette fenêtre comprend une multitude de bouton comme vous pouvez le voir-ci-dessous :

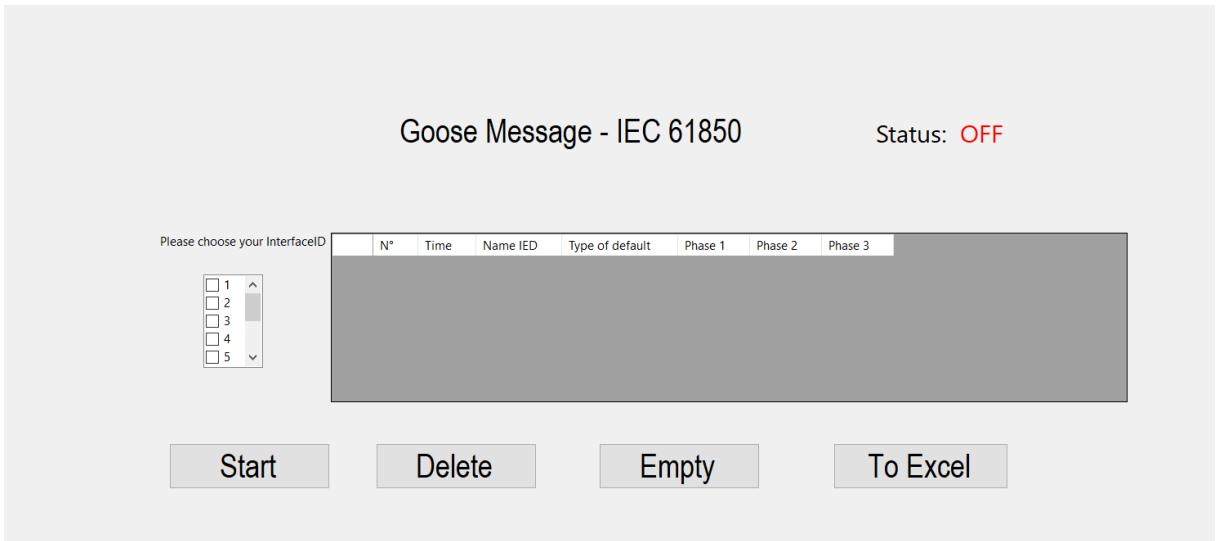


Figure 25 : Fenêtre réception des messages Goose

La Checked box list, le bouton Start, ainsi que le label Status seront expliqués dans la prochaine partie. Nous allons nous intéresser ici à la conception graphique de cette fenêtre. Nous retrouvons donc les mêmes fonctionnalités que précédemment à savoir la suppression d'une ligne et la suppression totale des données du tableau. La fonctionnalité mise en avant ici est la possibilité d'exporter le tableau de message Goose au format Excel pour pouvoir faire du post-traitement.

- Bouton d'exportation sur Excel.

Afin de pouvoir utiliser les fonctionnalités d'Excel dans notre application il faut ajouter les références suivantes à notre projet :

```
using Excel = Microsoft.Office.Interop.Excel;
using System.Runtime.InteropServices;
```

On peut donc par la suite se servir de l'ajout de ces références afin de créer une nouvelle instance de l'application Excel dans notre application nommé ExportExcel. Il suffit par la suite de créer un programme permettant de recopier les cellules de notre tableau dans les cellules de notre Excel. Voici-dessous le code de cette fonction :

```
68  public void ExportExcel_Click(object sender, EventArgs e)
69  {
70      if (GridGoose.Rows.Count > 0)
71      {
72          Excel.Application ExportExcel = new Excel.Application();
73          ExportExcel.Application.Workbooks.Add(Type.Missing); //Créer un nouveau classeur excel actif
74
75          // Recopie le texte au dessus des données pour chaque colonne
76          for (int i = 1; i < GridGoose.Columns.Count + 1; i++)
77          {
78              ExportExcel.Cells[1, i] = GridGoose.Columns[i - 1].HeaderText;
79          }
80
81          //Recopie les lignes et les colonnes dans le fichier excel
82          // i index de la ligne et j index de la colonne
83          for (int i = 0; i < GridGoose.Rows.Count - 1; i++)
84          {
85              for (int j = 0; j < GridGoose.Columns.Count; j++)
86              {
87                  ExportExcel.Cells[i + 2, j + 1] = GridGoose.Rows[i].Cells[j].Value.ToString(); // Commence dans la case Excel Ligne 2 et colonne 1
88              }
89          }
90          ExportExcel.Columns.AutoFit(); // Permet de donner la bonne taille de cellule en fonction de la dimension de la donnée
91          ExportExcel.Visible = true; // On affiche la fenêtre à l'écran
92      }
93  }
```

V – L'application de la norme IEC-61850 à notre IHM à l'aide de la librairie en C#

Un des enjeux majeurs de ce projet fut la mise en place de la norme IEC61850. En effet, dans le cahier des charges, il était prévu de télécharger et utiliser une librairie open source. Nous avons pris la décision de partir sur *libIEC61850* de la firme allemande Mz-Automation. Ce choix a été conforté par une grande partie de sites internet traitant de la norme, qui la recommandait. Celle-ci, propose un code open-source mais aussi leurs services pour des application « built-in ». L'utilisation de cette librairie fut bien plus compliquée que prévue. En effet, l'absence de commentaires dans le code ou même d'information générale quant à son utilisation, ont rendu le travail plus compliqué. C'est ce qui va être détaillé dans les différentes sous-parties ci-dessous. A noter qu'aucun des deux membres du groupes ne sont dans le département informatique, ce qui a rendu la tâche difficile.

1) Construction de la Librairie

Un des premiers enjeu et défi à relever quant à la mise en place de cette norme est la construction de la librairie. En effet, celle-ci n'est pas prête à être directement utilisée après son téléchargement, des prérequis sont nécessaires. Pour ce faire, il faut installer un premier logiciel tiers, **CMake**. CMake est un système de construction logicielle multiplateforme. Il permet de vérifier les prérequis nécessaires à la construction, de déterminer les dépendances entre les différents composants d'un projet, afin de planifier une construction ordonnée et adaptée à la plateforme. A l'aide d'un script créé par MZ-Automation, il va permettre de créer un projet Visual Studio en C/C++. Lors de son installation, il ne faut pas oublier de cocher l'option « Add to PATH ». Celle-ci permet d'utiliser CMake directement par l'invite de commande Windows 10.

Un des premiers problèmes ici, est que nous avions tenté d'utiliser CMake via son interface graphique. Pour une raison qui nous est encore inconnue à ce jour, cela ne fonctionnait pas.

Une fois ceci fait, il faut télécharger le logiciel tiers Winpcap avec son dossier « Developper Kit ». Il s'agit d'un driver qui ajoute au système d'exploitation un accès réseau très bas niveau. Il est un élément essentiel pour capter les messages Goose circulant sur le réseau. Une fois ceci fait, l'ordinateur doit être redémarrer afin de finaliser l'installation et l'exécution du pilote.

Le dossier « Developper Kit », est à placer dans le dossier « third_party », situé à la racine de la librairie, puis Winpcap. Il doit contenir les dossier « Lib » et « Include ».

TO54 > Lib-Code Source > LIB RENDU > libiec61850-1.5 > third_party > winpcap >

<input type="checkbox"/> Nom	Statut	Modifié le
Include	✓ 8	30/05/2022 11:58
Lib	✓ 8	30/05/2022 11:58
README	✓ 8	15/04/2022 23:05

Figure 26 : Dossier winpcap dans third_party

Un des problèmes qui a été rencontré ici est que Winpcap est de base en 32bits. Nous utilisons un système d'exploitation en 64bits (architecture x86). Lors de la compilation réalisée plus tard dans la

chronologie de construction de la librairie, une multitude d'erreurs apparaîtra. Dans ce cas, il fallait prendre le contenu du dossier « x64 », dans le dossier Include et le mettre à l'intérieur de celui-ci (ce qui a pour effet d'écraser les fichiers du même nom en 32 bits). Les prérequis pour la construction de la librairie sont donc établis.

Pour effectuer cette construction, on crée un dossier dans la librairie qui va accueillir les fichiers. Puis, on ouvre l'invite de commande en mode administrateur et on pointe vers la librairie en tapant « cd + lien du dossier » :

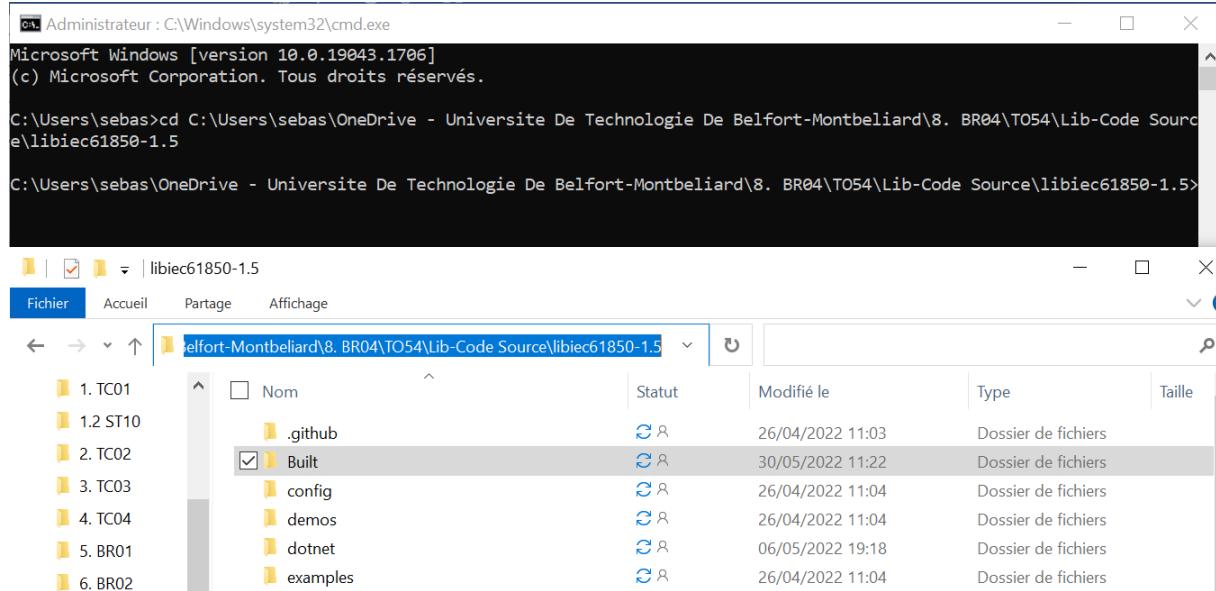


Figure 27 : Ouverture du dossier "Built" via l'invite de commande

Puis, on tape « cd Built » dans le but de pointer vers le dossier créé précédemment.

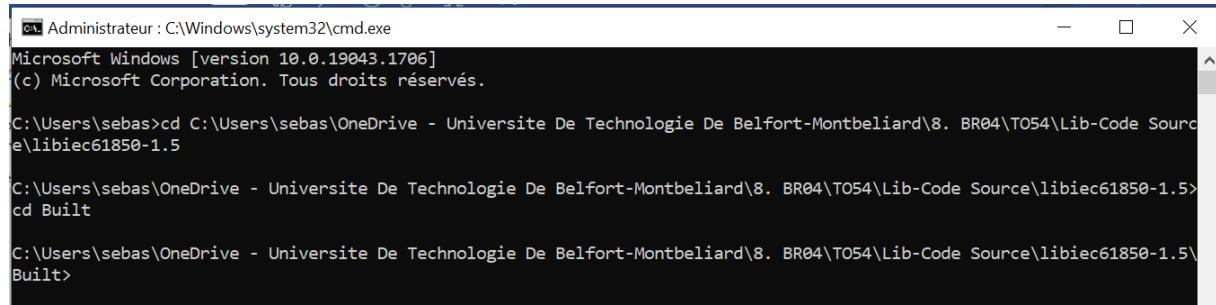


Figure 28 : Invite de commande pointant vers le dossier d'accueil

Nous allons à présent utiliser l'api CMake en tapant « **cmake -G "Visual Studio 17 2022" .. -A x64** » puis « Entrer ». Ceci a pour but de créer un projet pour Visual Studio 2022. Il est possible de changer la version en modifiant l'argument. Détailons la commande :

- CMake : invocation de l'API
- -G spécifie le système de génération cible qui est précisé à la suite : VS2022
- .. Indique que les fichiers et le script sont dans le dossier parent à celui qui a été pointé
- -A renseigne la plateforme cible, ici x64

La génération débute. On peut avoir une vision des Logs :

```

Administrator : C:\Windows\system32\cmd.exe
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\sebas>cd C:/Users/sebas/OneDrive - Université De Technologie De Belfort-Montbéliard/8. BR04\TO54\Lib-Code Source\libiec61850-1.5
C:\Users\sebas>cd Built

C:\Users\sebas\OneDrive - Université De Technologie De Belfort-Montbéliard/8. BR04\TO54\Lib-Code Source\libiec61850-1.5>cd Built

-- Selecting Windows SDK version 10.0.19041.0 to target Windows 10.0.19043.
-- The C compiler identification is MSVC 19.31.31107.0
-- The CXX compiler identification is MSVC 19.31.31107.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Program Files/Microsoft Visual Studio/2022/Community/VC/Tools/MSVC/14.31.31103/bin/Hostx64/x64/cl.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/Program Files/Microsoft Visual Studio/2022/Community/VC/Tools/MSVC/14.31.31103/bin/Hostx64/x64/cl.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Looking for clock_gettime in rt
-- Looking for clock_gettime in rt - not found
Found winpcap --> compile ethernet HAL layer (required for GOOSE/SV support)
Found winpcap --> can compile with GOOSE support
server-example-logging: sqlite not found
Found winpcap --> compile examples for GOOSE and SV
Found winpcap --> can compile with GOOSE support
-- Generating RC file : C:/Users/sebas/OneDrive - Université De Technologie De Belfort-Montbéliard/8. BR04\TO54\Lib-Code Source\libiec61850-1.5/Built/src/version.rc
-- Performing Test COMPILER_HAS_DEPRECATED_ATTR
-- Performing Test COMPILER_HAS_DEPRECATED_ATTR - Failed
-- Performing Test COMPILER_HAS_DEPRECATED
-- Performing Test COMPILER_HAS_DEPRECATED - Success
-- Could NOT find Doxygen (missing: DOXYGEN_EXECUTABLE)
-- Configuring done
-- Generating done
-- Build files have been written to: C:/Users/sebas/OneDrive - Université De Technologie De Belfort-Montbéliard/8. BR04\TO54\Lib-Code Source\libiec61850-1.5/Built

```

Figure 29 : Logs CMake de la génération de la librairie

Sur la figure ci-dessus, une ligne est surlignée. Elle spécifie si l'opération de placement du kit développeur winpcap dans le dossier « third_party » a été correctement réalisé (si le dossier avec la mauvaise architecture a été copié, le même message sera spécifié.)

Le dossier Built contient à présent un grand nombre de fichiers :

Nom	Statut	Modifié le	Type	Taille
CMakeFiles	🕒	30/05/2022 11:27	Dossier de fichiers	
config	🕒	30/05/2022 11:27	Dossier de fichiers	
examples	🕒	30/05/2022 11:27	Dossier de fichiers	
hal	🕒	30/05/2022 11:27	Dossier de fichiers	
src	🕒	30/05/2022 11:27	Dossier de fichiers	
ALL_BUILD.vcxproj	🕒	30/05/2022 11:27	VC++ Project	68 Ko
ALL_BUILD.vcxproj.filters	🕒	30/05/2022 11:27	VC++ Project Filters F...	1 Ko
cmake_install	🕒	30/05/2022 11:27	Fichier source CMake	9 Ko
CMakeCache	🕒	30/05/2022 11:27	Document texte	21 Ko
CPackConfig	🕒	30/05/2022 11:27	Fichier source CMake	5 Ko
CPackSourceConfig	🕒	30/05/2022 11:27	Fichier source CMake	5 Ko
CTestTestfile	🕒	30/05/2022 11:27	Fichier source CMake	1 Ko
INSTALL.vcxproj	🕒	30/05/2022 11:27	VC++ Project	11 Ko
INSTALL.vcxproj.filters	🕒	30/05/2022 11:27	VC++ Project Filters F...	1 Ko
<input checked="" type="checkbox"/> libiec61850.sln	🕒	30/05/2022 11:27	Visual Studio Solution	51 Ko
PACKAGE.vcxproj	🕒	30/05/2022 11:27	VC++ Project	12 Ko
PACKAGE.vcxproj.filters	🕒	30/05/2022 11:27	VC++ Project Filters F...	1 Ko
RUN_TESTS.vcxproj	🕒	30/05/2022 11:27	VC++ Project	10 Ko
RUN_TESTS.vcxproj.filters	🕒	30/05/2022 11:27	VC++ Project Filters F...	1 Ko
ZERO_CHECK.vcxproj	🕒	30/05/2022 11:27	VC++ Project	112 Ko
ZERO_CHECK.vcxproj.filters	🕒	30/05/2022 11:27	VC++ Project Filters F...	1 Ko

Figure 30 : Dossier Built après la génération

On exécute le fichier projet « libiec61850.sln » avec Visual Studio.

La liste déroulante à droite contient tous les projets. Pour Windows, celui qui nous intéresse est iec61850-shared. Pour l'utiliser, il faut le définir en tant que projet de démarrage disponible dans ses propriétés. On l'exécute. Cela peut prendre quelques minutes. A la fin de la génération, un message d'erreur peut apparaître. S'il s'agit d'un message équivalent à celui sur la capture ci-dessous, il ne faut pas en tenir compte.

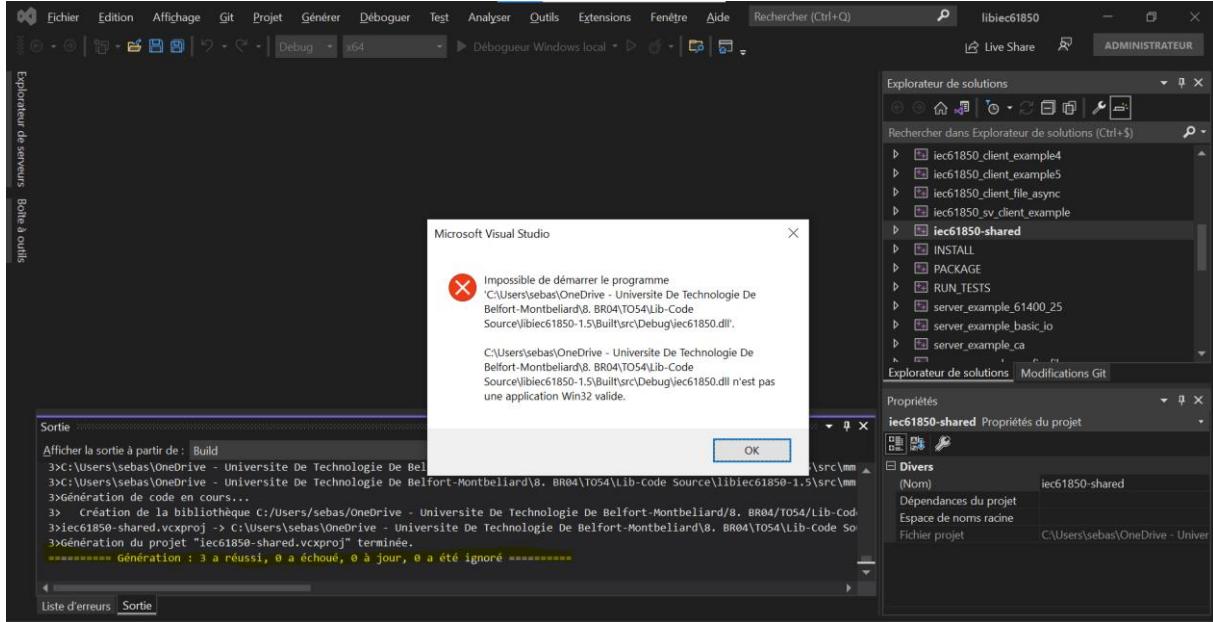


Figure 31 : Message d'erreur à la fin de la génération de iec61850-shared

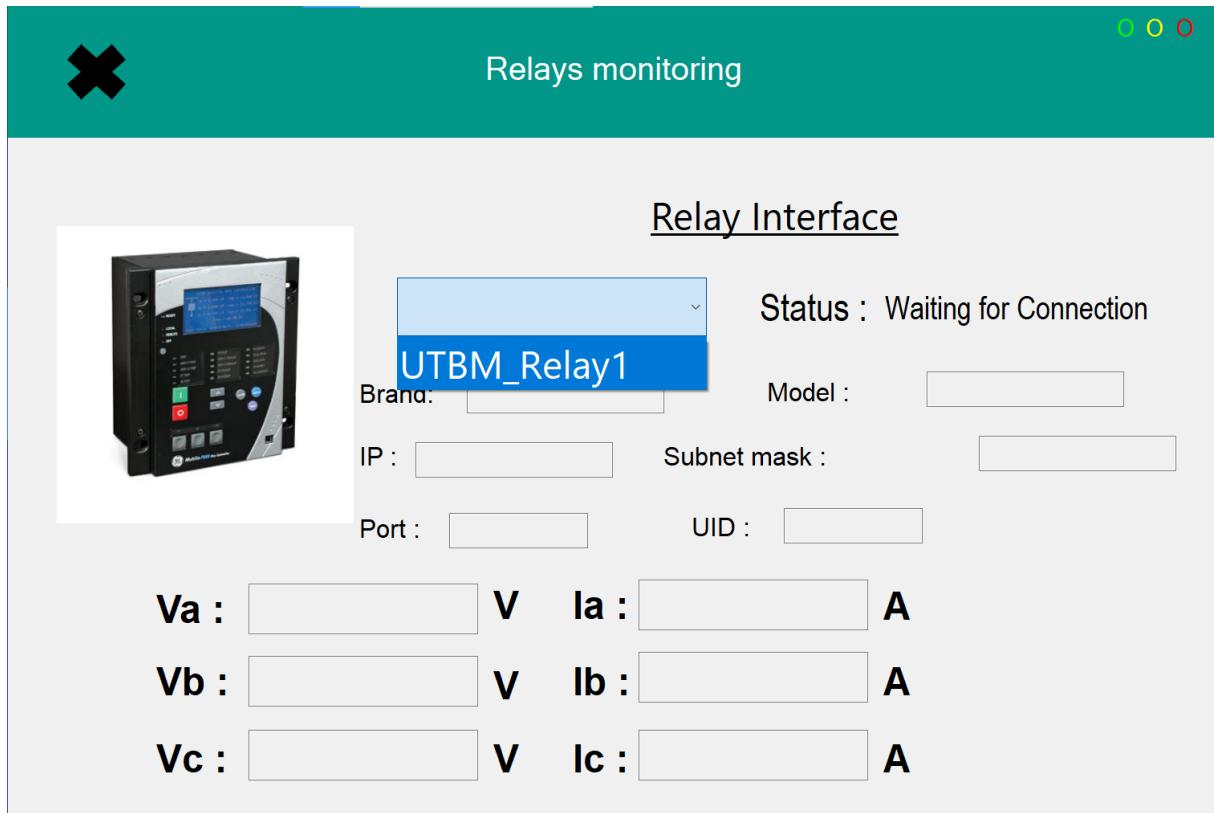
Cette opération a pour objectif de créer une librairie sous la forme d'un fichier avec extension « .dll ». Cette extension est uniquement reconnue par un système d'exploitation Windows. Pour linux, le projet à exécuter est iec61850. Pour trouver le fichier généré, il suffit de se rendre à libiec61850-1.5\Built\src\Debug. Il doit être inséré dans le dossier contenant les librairies pour le système d'exploitation. Pour Windows, ce dossier est dans C:\Windows\System32.

Il ne reste plus qu'à inclure la librairie « IEC61850forCsharp » au projet et à installer le logiciel npcap qui est un driver et une librairie permettant d'intercepter des paquets circulant sur le réseau. La librairie est, en théorie prête à être utilisée.

Cette partie fut compliquée à réaliser puisqu'elle comporte la découverte et l'utilisation de la librairie qui, comme vous avez pu le voir, comporte beaucoup de prérequis peu voire pas expliqués. Nous avons à plusieurs fois contacté un développeur de MZ-Automation, M. Michael Zilligith. Il a su nous aider pour la construction tout en restant assez vague. C'est pourquoi cela nous a pris beaucoup de temps, à comprendre le principe de construction avec toutes les erreurs que nous avons pu avoir et qui n'ont pas été détaillées ici car la liste est beaucoup trop grande.

2) Connexion aux IEDs

L'objectif de cette partie est d'expliquer comment est réalisé la connexion aux IEDs. Celle-ci a été implémentée afin de vérifier la présence d'un appareil sur le réseau ainsi que sa bonne configuration. Pour ce faire, le code d'un exemple de la librairie en application console a été repris et adapté. Cette action est déclenchée par le chargement des paramètres d'un IED dans l'onglet « Relays ».



La communication avec l'utilisateur est effectuée par des « MessageBox » et un texte Statut. On peut représenter l'algorithme de connexion par une modélisation de type organigramme :

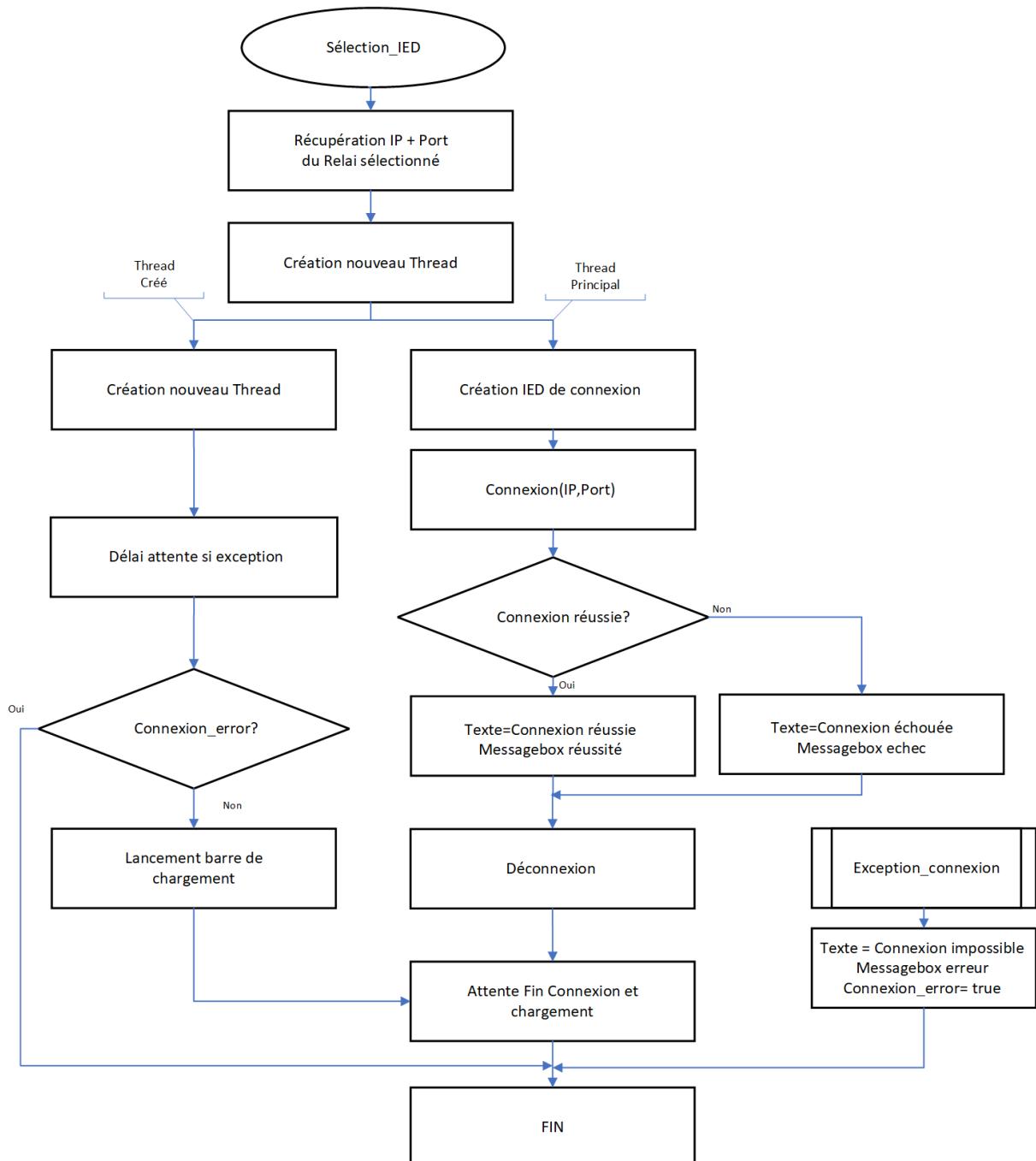


Figure 32 : Organigramme représentant l'algorithme de connexion

Cela représente le code suivant :

```

1. public void ListeDeroulante_SelectedIndexChanged(object sender, EventArgs e)
2. {
3.     for (int i = 0; i < Gestion.ListeRelais.Count(); i++)
4.     {
5.         if (Gestion.ListeRelais[i].getNom() == ListeDeroulante.Text)
6.         {
7.             myGui_Connection = new Thread(New_Thread_create); //nouveau thread
8.             myGui_Connection.IsBackground = true;
  
```

```

9.             myGui_Connection.Start();      //démarrage en tâche de fond
10.
11.            try
12.            {
13.                IedConnection con = new IedConnection();
14.                var args = Environment.GetCommandLineArgs();
15.                string hostname;
16.
17.                hostname = Gestion.ListeRelais[i].getIP();
18.                int port = Gestion.ListeRelais[i].getPort();
19.
20.                if (args.Length > 1)
21.                    port = Int32.Parse(args[1]);
22.
23.                try
24.                {
25.                    con.Connect(hostname, port);
26.
27.                    List<string> serverDirectory = con.GetServerDirectory(false);
28.
29.                    foreach (string entry in serverDirectory)
30.                    {
31.                        connexion_flag = 1;
32.                    }
33.                }
34.                catch (IedConnectionException f)
35.                {
36.                    connexion_flag = 2;
37.                }
38.                // déconnexion
39.                con.Dispose();
40.            }
41.            catch (Exception ex)
42.            {
43.                ETATIED.Text = "Unable to Connect";
44.                ETATIED.ForeColor = Color.Red;
45.                connection_error = true;
46.                MessageBox.Show("Failed to connect, please check that both DLLs
are in System32 Folder", "Internal Error",
47.                               MessageBoxButtons.OK, MessageBoxIcon.Error);
48.                connexion_flag = 0;
49.            }
50.
51.        }
52.    }
53.}
54.
55. private void New_Thread_create() //creation nouveau thread
56.{
57.    try
58.    {
59.        Launch_Progress_Bar();
60.    }
61.    catch (Exception ex)
62.    {
63.        //Progressbar form = pas d'erreur possible
64.    }
65.}
66.
67. void Launch_Progress_Bar()
68.{
69.    Thread.Sleep(200);
70.    if (connection_error == false)
71.    {

```

```

72.         flag_co = true;
73.         myForm = new Chargement();
74.         myForm.Show();
75.         myForm.Location = new Point(750, 250);
76.         for (int i = 0; i < 100; i++)
77.         {
78.             myForm.progressBar_statut.Value = i;
79.             Thread.Sleep(90);
80.             //myForm.label_statut.Text = i + " %";
81.         }
82.         flag_co = false;
83.         myForm.Hide();
84.     }
85.     else
86.     {
87.         connection_error = false;
88.     }
89.
90. }

```

Pour synthétiser cette fonction, lors de la sélection d'un équipement compatible IEC61850, on récupère son adresse IP et son port de connexion (qui est de 102 pour une connexion avec cette norme). Une fois ceci fait, on créer un nouveau thread afin de laisser la connexion s'exécuter et pouvoir constamment avoir un fil de calcul dédié à l'interface GUI (interface graphique, lancement des fonctions de l'application). Une connexion va être lancée à l'aide de la librairie. Si le fichier généré précédemment (iec61850.dll ou .lib) est manquant, une exception va ressortir. A ce moment, l'utilisateur sera averti qu'il manque probablement ce fichier (ou que winpcap n'est pas actif).

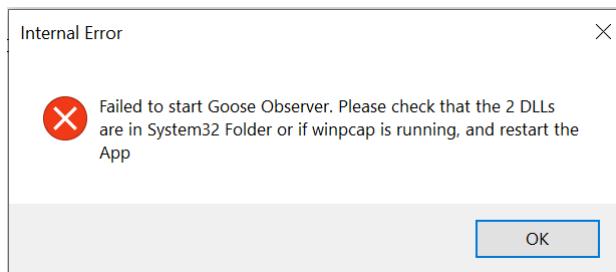


Figure 33 : Message d'erreur interne

Une exception est un bout de programme qui permet de sortir de la fonction si une erreur à lieu. En effet, au lieu d'arrêter le programme complet à la suite d'un bug, il sort uniquement de la fonction. La fonction parent doit comporter un traitement de cette exception. En fin de compte cela supprime l'instabilité et permet de communiquer à l'utilisateur les potentielles erreurs, dans le cas de fonctions critique. Cela se met sous la forme :

```

1. Try
2. {
3.     //appel de la function sensible
4. }
5. Catch (Exception e)
6. {
7.     //est le code erreur retourné
8. }

```

Si aucune exception n'est renvoyée, il va lancer la connexion avec un timeout de 10 secondes. La connexion dure à peu près le même temps car, la librairie prévoit la récupération de la configuration

de l'IED. Ceci n'a pas été prévu dans le cahier des charges et n'a donc pas été implémenté dans l'application. En même temps su l'autre fil de calcul, on affiche une autre fenêtre de chargement dépendantes des délais qui indique à l'utilisateur l'état d'avancement de la connexion. Visual Studio n'est pas stable lorsque 2 threads évoluent en même temps sans aucun lien. C'est pourquoi il persiste un bug quant à la barre de chargement où son design ne fonctionne pas. Voici sa conception :

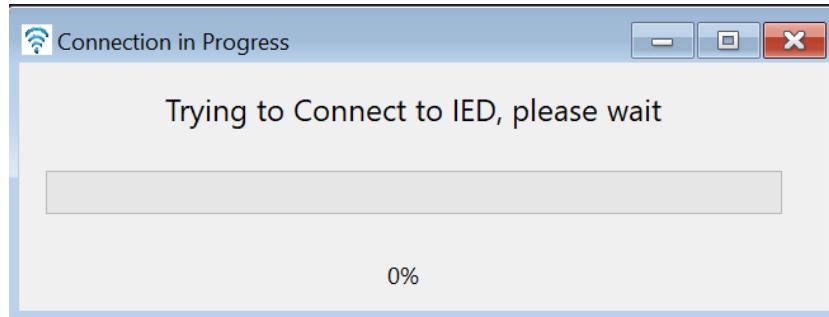


Figure 34 - Design initial de la fenêtre de chargement

Lorsque l'on appelle le nouveau thread et l'ouverture de cette fenêtre :



Figure 35 : Design de la barre de chargement sur le nouveau Thread

On remarque que les deux textes ne sont pas affichés. En revanche la barre de progression évolue. Etant un problème d'esthétique plus que fonctionnel, nous nous sommes concentrés sur la mise en place de la norme iec61850. Une fois terminé, on la cache (elle reste en mémoire).

Pour poursuivre avec le thread principal, si le timeout est dépassé, on affiche un message d'erreur à l'utilisateur et le texte indiquant le statut passe en rouge avec le même message :

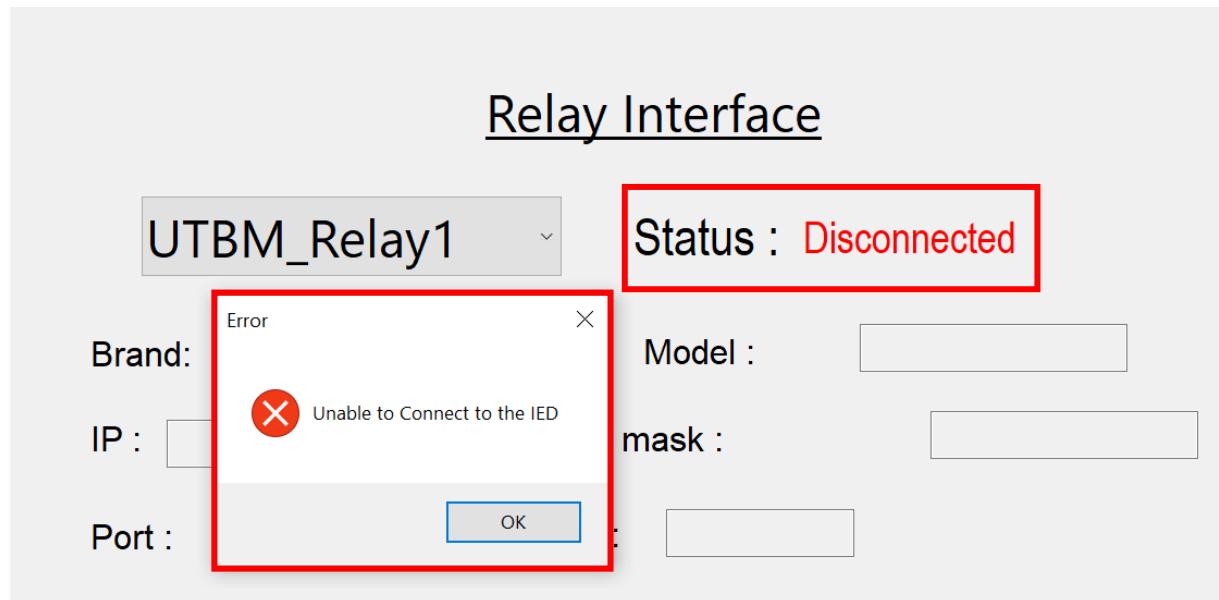


Figure 36 : Message de connexion échouée

A ce moment, l'erreur peut venir d'une erreur réseau (câble débranché), d'une mauvaise configuration interne de l'IED, ou bien d'un mauvais paramétrage de l'application. Pour retenter une connexion, il est possible de sélectionner un nouvel IED ou bien de se déplacer vers un nouvel onglet et d'y revenir.

Dans le cas où une connexion est établie, on informe de la même manière l'utilisateur avec cette fois ci des messages de réussite et un texte en vert :

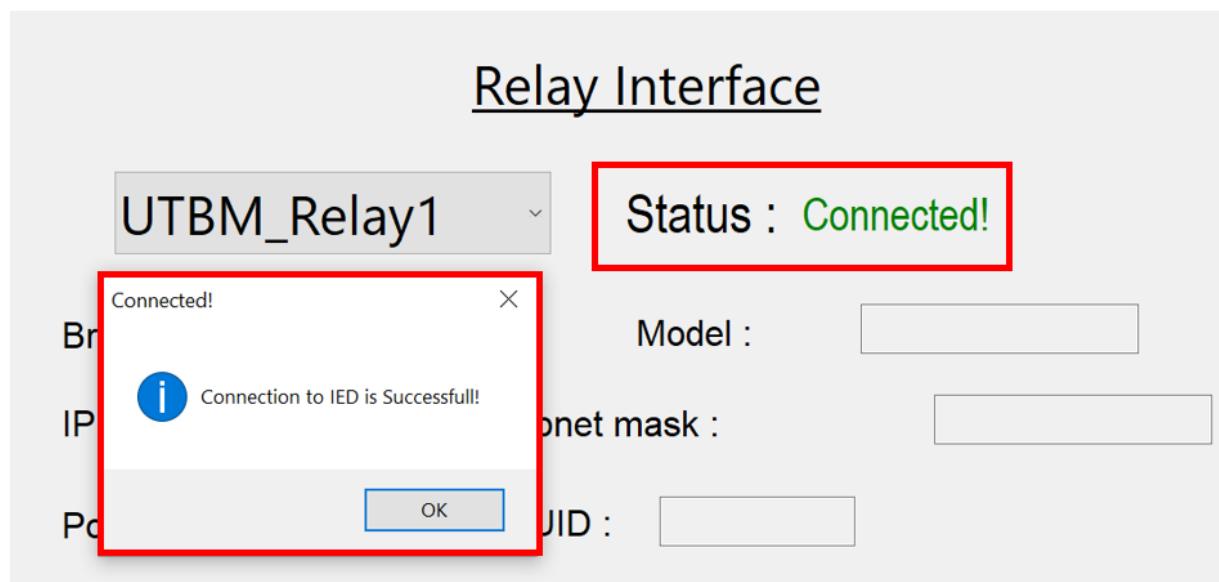


Figure 37 : Message de connexion réussie

A noter que si l'utilisateur reste sur cet onglet « Relay Interface » après la réussite d'une connexion et que l'IED venait à être déconnecté (matériellement par exemple), l'affichage ne changera pas. En effet, le programme se déconnecte directement après la réussite de l'opération afin de décharger les deux entités : IED et application.

Ces fonctions d'affichage sont réalisées par le code ci-dessous :

```
1. if(connexion_flag != 0)
2. {
3.     while (flag_co) ;
4.     if (connexion_flag == 1)
5.     {
6.         ETATIED.Text = "Connected!";
7.         ETATIED.ForeColor = Color.Green;
8.         MessageBox.Show("Connection to IED is Successfull",
9.                         "Connected!",
10.                        MessageBoxButtons.OK, MessageBoxIcon.Information);
11.    }
12.    else if (connexion_flag == 2)
13.    {
14.        ETATIED.Text = "Disconnected";
15.        ETATIED.ForeColor = Color.Red;
16.        MessageBox.Show("Unable to Connect to the IED", "Error",
17.                         MessageBoxButtons.OK, MessageBoxIcon.Error);
18.    }
}
```

Une fois toutes les opérations en arrière-plan réalisées, on effectue l'affichage à l'aide d'un système de drapeau :

- connexion_flag :
 - =0 détecte s'il y a eu une exception
 - =1 connexion réussie
 - =2 connexion échouée
- while(flag_co) : attente de la fin de la connexion + fenêtre de chargement

La connexion est donc fonctionnelle en tout point. Les seuls problèmes que l'on peut relever ici est l'affichage des textes sur la barre de chargement qui sont manquants ainsi que le temps de connexion. Celui-ci est recommandé par la norme donc difficilement améliorable. Cette partie fut bien plus simple par rapport à la partie précédente car il s'agissait uniquement de comprendre comment importer la librairie à notre projet et s'en servir. Le reste n'était que de la logique. Le niveau est bien remonté avec la tentative d'acquisition Goose.

3) Réception de Message Goose

Dans cette partie nous allons aborder la réception de messages Goose sur le réseau local. Il s'agit ici de la partie la plus dure du projet, un parcours semé d'obstacles. En effet, tout comme la connexion, la librairie possède une application console en C# exemple. Avant de l'intégrer, nous voulions la tester. Tout d'abord celle-ci ne compilait pas. Nous avions des erreurs de génération. Après maintes recherches, nous avons décocher « Préférer 32 bits » sous les propriétés de Build et le programme a compilé :

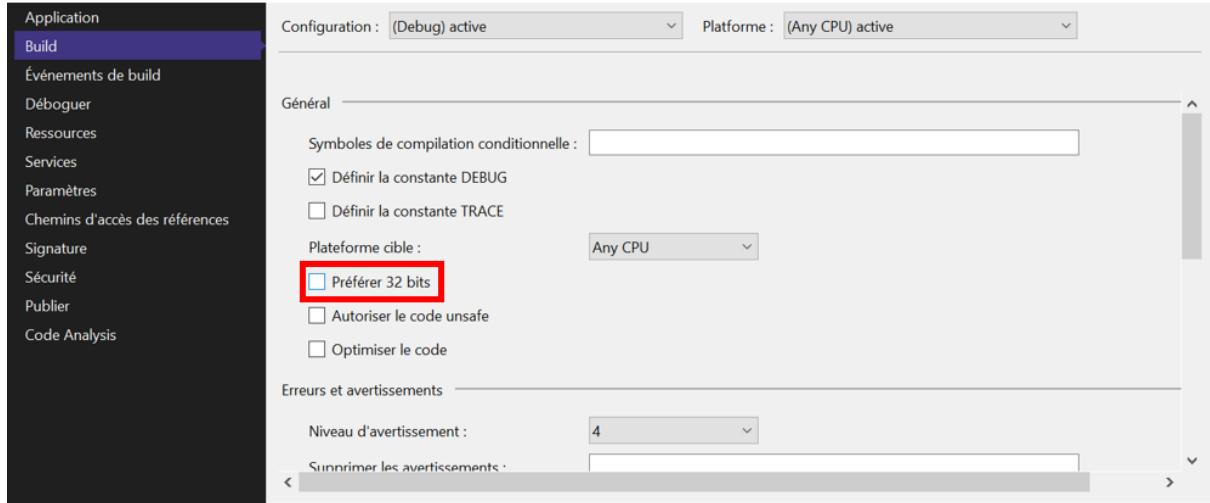


Figure 38 : Modification des préférences de built

Voici un exemple du code :

```
1. public static void Main (string[] args)
2. {
3.     Console.WriteLine ("Starting GOOSE subscriber...");
4.
5.     GooseReceiver receiver = new GooseReceiver ();
6.
7.     receiver.SetInterfaceId ("6");
8.     //receiver.SetInterfaceId("0"); // on windows use the interface index
9.     starting with 0
10.    GooseSubscriber subscriber = new GooseSubscriber
11.    ("simpleIOPort/LLN0$GO$gcbAnalogValues");
12.    // APP-ID has to match the APP-ID of the publisher
13.    subscriber.SetAppId(0);
14.    subscriber.SetListener (gooseListener, null);
15.
16.    receiver.AddSubscriber (subscriber);
17.
18.    GooseSubscriber subscriber2 = new
19.    GooseSubscriber("simpleIOPort/LLN0$GO$gcbEvents");
20.    subscriber2.SetAppId(0);
21.    subscriber2.SetListener(gooseListener, null);
22.
23.    receiver.AddSubscriber(subscriber2);
24.
25.    receiver.Start ();
```

Après l'étude de cette fonction principale, on remarque qu'il y a deux paramètres à modifier :

- L'Application ID (qui est à 0 d'après les CID de nos relais)
- L'interfaceID : il s'agit de l'interface réseau utilisé.

Aucune indication ni explication n'a pu être trouvée pour ce dernier paramètre, autant par la documentation que par le développeur. Nous avons tenté plusieurs valeurs tel que l'ordre des cartes réseaux voire leurs indexées, sans succès. Nous avons également tenté une à une plusieurs valeurs, avec le même résultat : aucun message n'apparaissait, ou un message d'erreur « ExceptionAccessViolation » (pour des valeurs entre 3 et 20). Le comportement était aléatoire suivant les jours ce qui rendait le fait de tirer une conclusion, assez compliqué.

Après de longues recherches, nous avons pu obtenir un code en C, application console permettant de réaliser cette fonction. Il fallait tout simplement renseigner l'interface ID. Au-delà de 6, un message d'erreur apparaissait spécifiant qu'aucune instance matérielle n'existe, et ce pour toutes les tentatives supérieures à ce nombre. En y allant un par un nous avons pu trouver le bon : nous recevions des messages Goose ! Cet ID ne fonctionne pas sur le code en C#.

Un problème s'est donc posé. Pour récapituler la situation, nous avons un projet en C#, une connexion qui s'établie en C# mais une réception de Goose en C. Nous avions pensé à changer complètement de langage de programmation mais ce fut tard. Il restait plus qu'une solution créer un fichier dll à partir du début, afin de faire le lien entre les deux langages de programmation.

Les fichiers ayant pour extension .dll (Dynamic Link Library) sont des fichiers système compilés (non compréhensible par un éditeur de texte) très bas niveau, essentiels au fonctionnement des logiciels et de Windows. Beaucoup de problèmes, fausses pistes et mauvaises orientations ont été rencontrés dans la construction d'un tel fichier. Seuls les plus importants, nécessaire aux dll finales seront énoncés.

Pour commencer, étudions le principe de communication entre les deux langages de programmation. Dans le dll, chaque fonction qui va être appelée depuis du code en C# doit être spécifiée comme ceci:

```
extern "C" __declspec(dllexport) BSTR send_data(const char* tableName)
```

Etudions les différents arguments :

- extern spécifie la volonté de rendre la fonction accessible par l'extérieur
- « C » indique le langage de programmation à l'entité voulant y accéder
- __declspec(dllexport) indique la méthode d'exportation
- BSTR est le type de variable que va retourner la fonction
- Send_data est le nom de la fonction
- Const char* est le type d'argument que va recevoir la fonction associée à son nom de variable

Du côté C#, il va falloir importer le dll et la fonction que l'on va vouloir utiliser :

```
[DllImport("HIRTH_TEANI.dll", CallingConvention = CallingConvention.Cdecl)]
[return: MarshalAs(UnmanagedType.BStr)]
```

Il s'agit de tout spécifier à nouveau. DLL Import va importer le dll à l'adresse spécifiée. Pour simplifier les opérations, les dll seront dans le dossier System32. Par défaut, s'il n'y a pas d'adresse, Visual Studio va les chercher dans ce dossier-ci. Nous sommes dans un cas à part ici, où les dll ne sont pas capables de retourner des tableaux ou chaînes de caractères. Le type BSTR (*wchar-t) est utilisé ce qui complexifie grandement l'importation.

Ceci vu, nous pouvons commencer par la partie algorithmique et mise en pratique. Le fichier comporte trois fonctions :

- osKernelStart()
- Goose_Listener()
- Send_data()

osKernelStart est la fonction responsable du lancement du sniffer. Il s'agit d'un noyau (Kernel) indépendant et autonome.

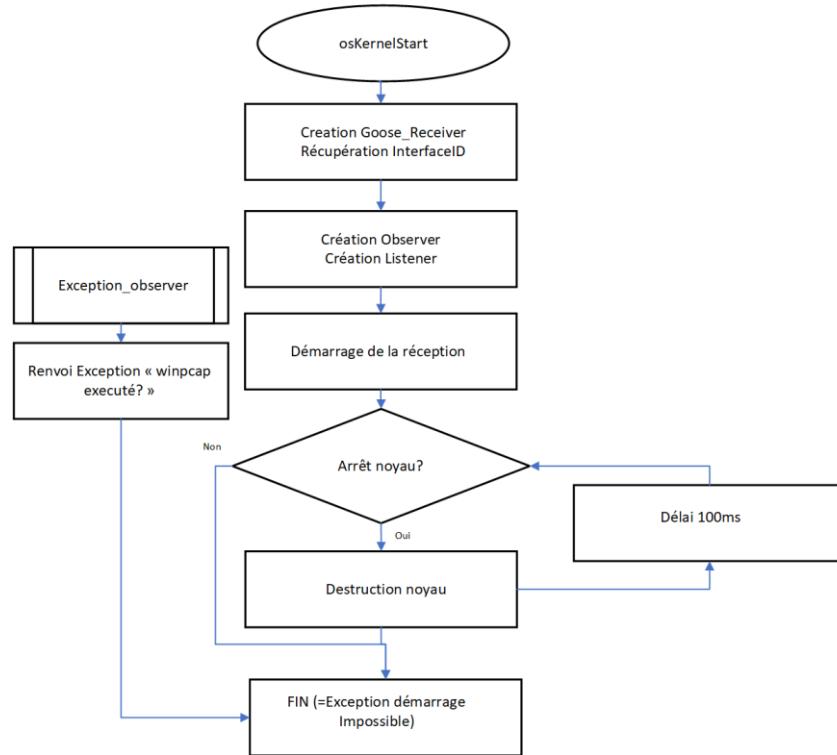


Figure 39 : Organigramme de la fonction de démarrage du noyau

Et voici son code correspondant :

```

1. extern "C" __declspec(dllexport) void
2. osKernelStart(char* connexion)
3. {
4.     GooseReceiver receiver = GooseReceiver_create();
5.     GooseReceiver_setInterfaceId(receiver, connexion);
6.     try
7.     {
8.
9.         GooseSubscriber subscriber = GooseSubscriber_create("", NULL);
10.        GooseSubscriber_setObserver(subscriber);
11.        GooseSubscriber_setListener(subscriber, gooseListener, NULL);
12.
13.
14.        GooseReceiver_start(receiver);
15.        running = 1;
16.    }
17.    catch (std::invalid_argument)
18.    {
  
```

```

19.         throw std::invalid_argument("Failed to start Goose Observer, Please check if winpcap
20.             is running ");
21.
22.
23.     if (GooseReceiver_isRunning(receiver)) {
24.         signal(SIGINT, sigint_handler);
25.
26.         while (running) {
27.             Thread_sleep(100);
28.         }
29.     }
30.     else {
31.         throw std::invalid_argument("Failed to start Goose Observer, Please check if dll is
32.             in System32 ");
33.     }
34.     GooseReceiver_stop(receiver);
35.
36.     GooseReceiver_destroy(receiver);
37. }
```

Dans ce cas, la fonction prend en paramètre un caractère de type `char*` spécifiant l’`InterfaceID`. Une fois ceci fait, on créer la connexion à l’aide de la librairie (qui a dû être modifiée). En créant les différentes instances nécessaires au démarrage du noyau, il peut y avoir une exception. Afin de la gérer, on l’a renvoi. Si elle arrive à ce moment-ci, il est fort probable que `winpcap` ne soit installé ou en cours d’exécution.

Si cette opération est effectuée avec succès, on lance le noyau. Il peut également y avoir un bug ici dans le cas où l’ID d’interface ne représente aucun matériel sur l’ordinateur. On renvoi également un code d’erreur. Sinon, on regarde toutes les 100ms si l’utilisateur veut stopper le noyau. Si c’est le cas, le seul moyen pour l’arrêter est de le kill (détruire). Ceci est réalisé en passant la variable `running` à 0 dans une autre fonction :

```

1. extern "C" __declspec(dllexport)
2. void sigint_handler(int signalId)
3. {
4.     running = 0;
5. }
```

Une fois lancé, à chaque réception d’une trame supposée `Goose`, une interruption se déclenche, passant la fonction `gooseListener` à l’état « élue ». L’organigramme de celle-ci est présent page suivante. :

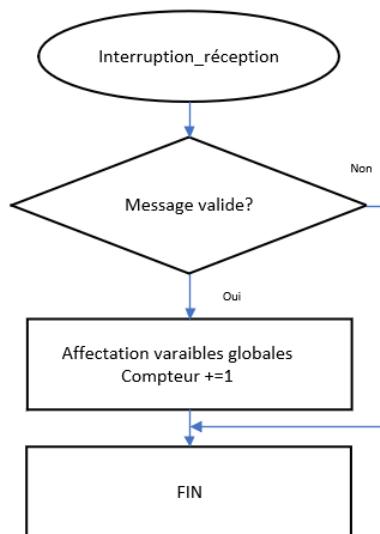


Figure 40 : Organigramme de la fonction interruption

Cette fonction est assez simple, on exécute tout d'abord l'algorithme de vérification de la trame qui nous renvoi un booléen. Si elle est conforme, on récupère à l'aide de variables globales le GoID, GocbRef et les données. On incrémente également un compteur. Son utilité va être détaillée par la suite.

```

1. void gooseListener(GooseSubscriber subscriber, void* parameter)
2. {
3.     GooseSubscriber_isValid(subscriber) ? valid_msg = true : valid_msg = 0; //check si la
   trame est valide
4.     if (valid_msg == true) //tri si valide
5.     {
6.         GoID = GooseSubscriber_getGoId(subscriber);
7.         GocbRef = GooseSubscriber_getGoCbRef(subscriber);
8.         MmsValue* values = GooseSubscriber_getDataSetValues(subscriber);
9.         char buffer[1024];
10.        MmsValue_printToBuffer(values, buffer, 1024);
11.        strcpy(Data, buffer);
12.        goose_compteur++;
13.        valid_msg = false;
14.    }
15. }

```

Encore une fois le code a dû être repensé car la pile prévue pour la donnée par la librairie était trop petite. Il s'agit d'une des nombreuses erreurs qu'il a fallu déboguer par élimination.

Enfin, nous avons la fonction send_data qui va être appelée de manière périodique par le projet. Elle permet, à l'aide des variables globales assignées précédemment, constituer une chaîne de caractère à renvoyer.

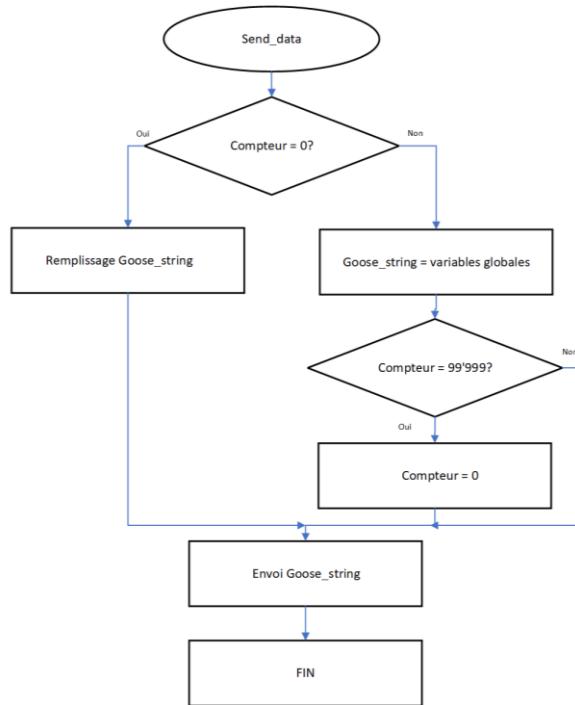


Figure 41 : Organigramme de la fonction envoi de données

En C/C++ cela donne le code suivant :

```

1. extern "C" __declspec(dllexport)
2. BSTR send_data(const char* tableName)
3.
4. {
5.     if (goose_compteur == 0)
6.     {
7.         strcpy(Goose_string, "0");
8.         strcat_s(Goose_string, "|");
9.         strcat_s(Goose_string, "UTCM_Relay1CON/LLN0$GO$PIOC1");
10.        strcat_s(Goose_string, "|");
11.        strcat_s(Goose_string, "Relais1_PIOC");
12.        strcat_s(Goose_string, "|");
13.        strcat_s(Goose_string,
14.        "{{false,0,false,0,false,0,000000000000,20000330173129,716Z");
15.    }
16.    else
17.    {
18.        char goose_compteur1[5];
19.        _itoa_s(goose_compteur, goose_compteur1, 10);
20.
21.        strcpy_s(Goose_string, goose_compteur1);
22.        strcat_s(Goose_string, "|");
23.        strcat_s(Goose_string, GocbRef);
24.        strcat_s(Goose_string, "|");
25.        strcat_s(Goose_string, GoID);
26.        strcat_s(Goose_string, "|");
27.        strcat_s(Goose_string, Data);
28.        if (goose_compteur == 99999)
29.        {
30.            goose_compteur = 0;
31.        }
32.    BSTR bstrText = _com_util::ConvertStringToBSTR(Goose_string);
33.    return SysAllocString(bstrText);
34. }
```

On commence par vérifier l'état du compteur. S'il est à 0, cela veut dire que le code en C# demande à recevoir les trames reçues alors que l'interruption n'a pas encore été déclenchée. Pour ne pas renvoyer des valeurs « null » et donc déclencher des exceptions instables, on remplit les chaînes de caractère avec une trame de base (elle sera rejetée par le code en C#).

Si le compteur a été incrémenté, une interruption a été déclenchée avec un message valide d'enregistré. Lors de la récupération des données, on met bout à bout les différentes chaînes de caractère en commençant par l'indice du compteur et en finissant par la donnée avec un délimiteur pour favoriser le découpage : « | »

Si le compteur est trop élevé par rapport à la chaîne de caractère qui lui ait octroyée, on le réinitialise. Puis on convertit la chaîne en BSTR (wchar-t*) afin qu'elle puisse être comprise par le C# au niveau de l'encodage du texte (UTF-8) mais aussi du buffer à lui allouer.

Pour que le code puisse compiler il y a des paramètres à ajouter dans les propriétés :

- Définir la sortie en dll

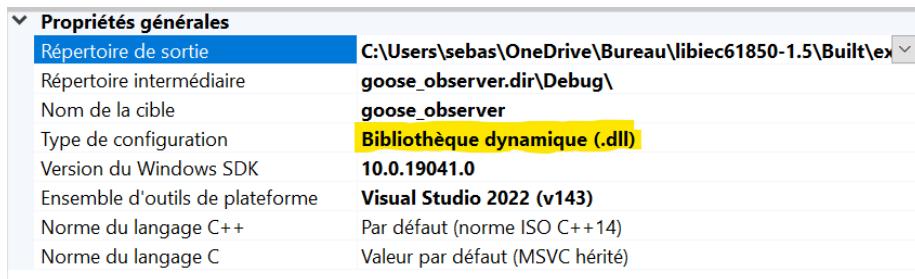


Figure 42 : Paramétrage de compilation en dll

- Modifier l'extension et le langage commun en /clr

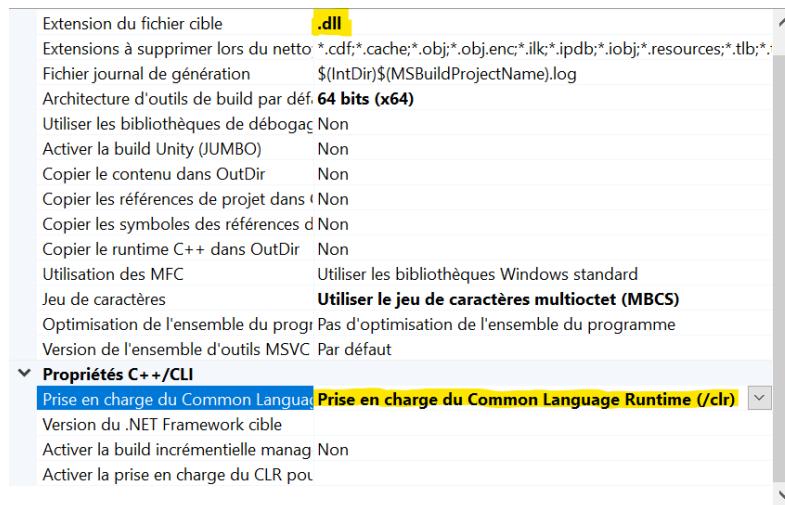


Figure 43 : Modification de l'extension et du Common Language

- Ajout d'une commande au build C/C++

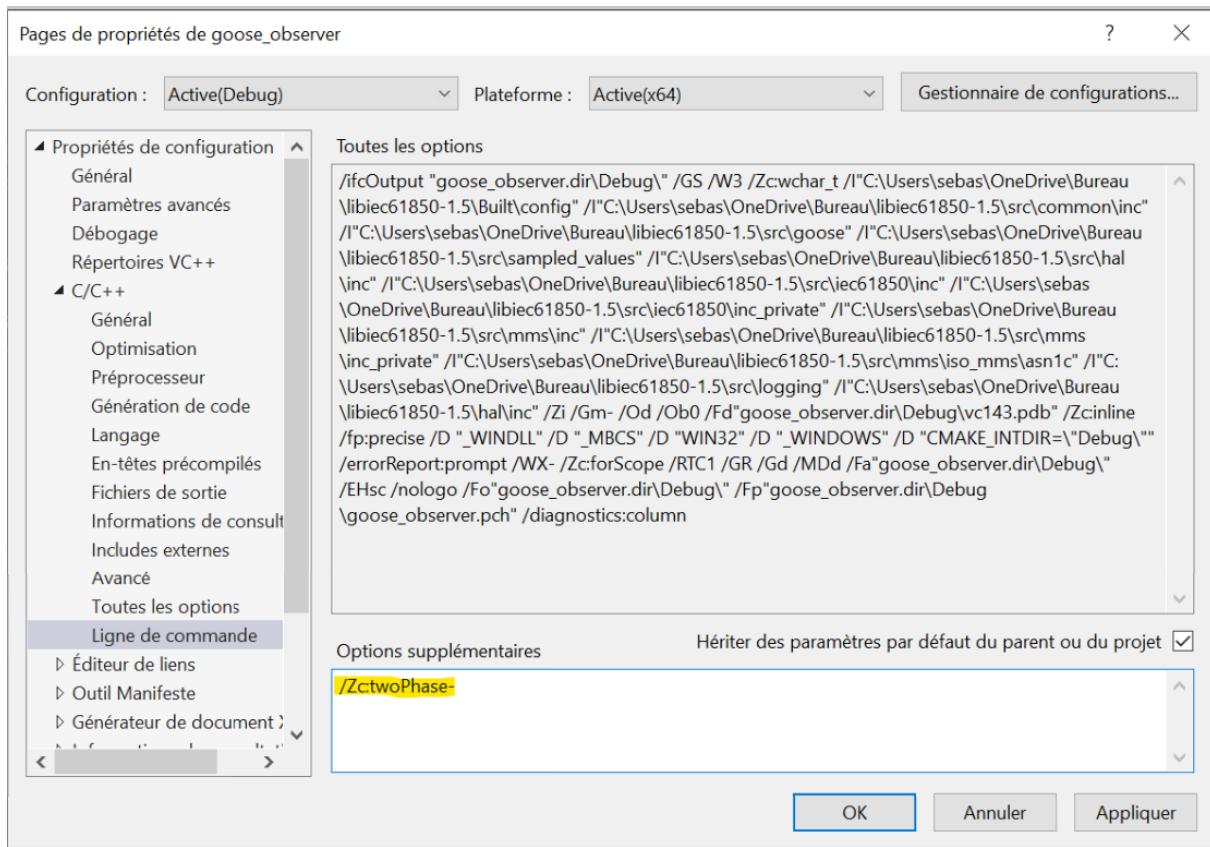


Figure 44 : Ajout d'une ligne de commande dans l'onglet C/C++

- Modification du langage pour utiliser wchar_t (fonction send_data)

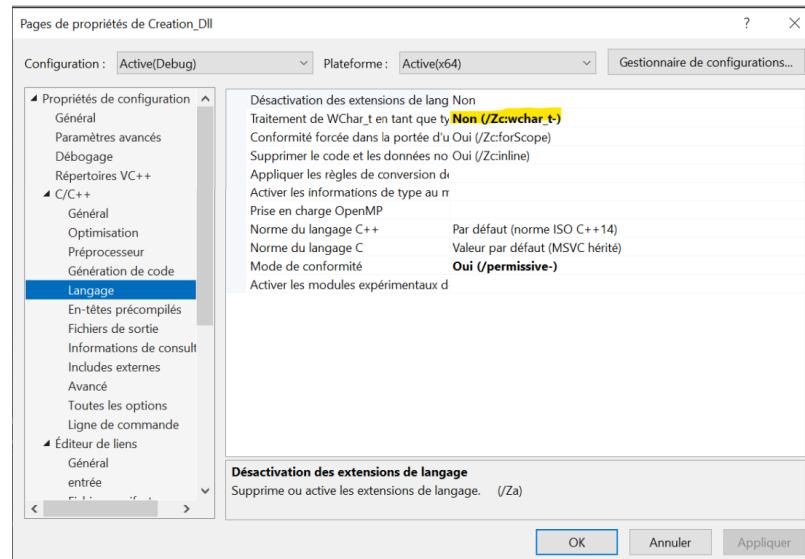


Figure 45 : Modification des paramètres wchar_t

- Ajout d'une dépendance externe

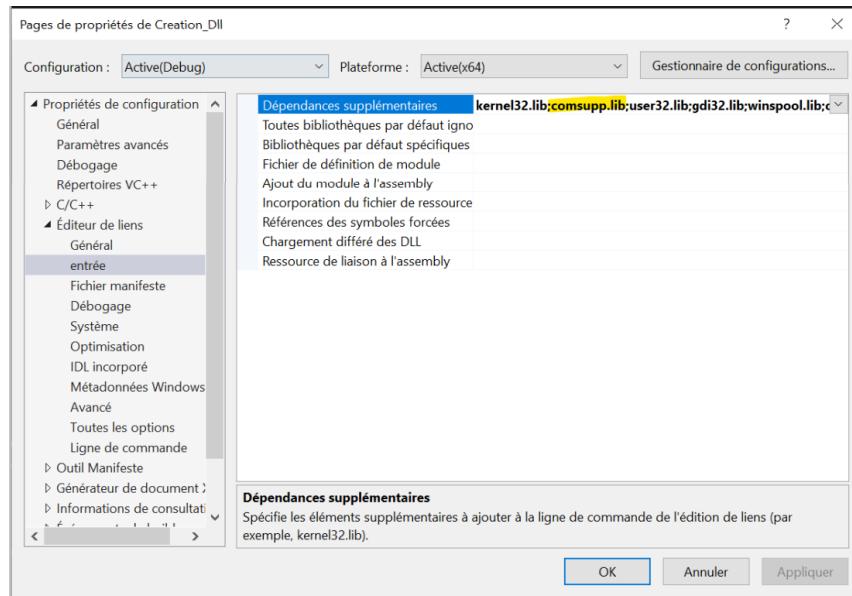


Figure 46 : Ajout d'une dépendance externe

La gestion des bugs fut compliquée sur le dll. Pourtant, les problèmes furent présents et en nombre. En effet, il n'est pas possible de debug ce type de fichier. Un exemple qui reflète ceci est le crash de l'application avec un code d'erreur « L buffer is too small & 0 ». La première étape est de modifier l'extension du dll en .exe pour avoir une vue d'ensemble sur la situation et de d'afficher sur la console les variables au lieu de les envoyer. Pas de problème de ce côté. En enlevant la sécurité sur strcat (=strcat_s qui permet de concaténer deux chaînes de caractères de manière sécuritaire au niveau de l'espace alloué), on enlevait l'erreur et on était en mesure de recevoir la trame en C#. C'est là qu'on s'est aperçu que la trame data était remplacée par 938 caractères 'l'. Strcat renvoyait une erreur étant donné que le tableau Goose_string ne peut contenir que 200 caractères.

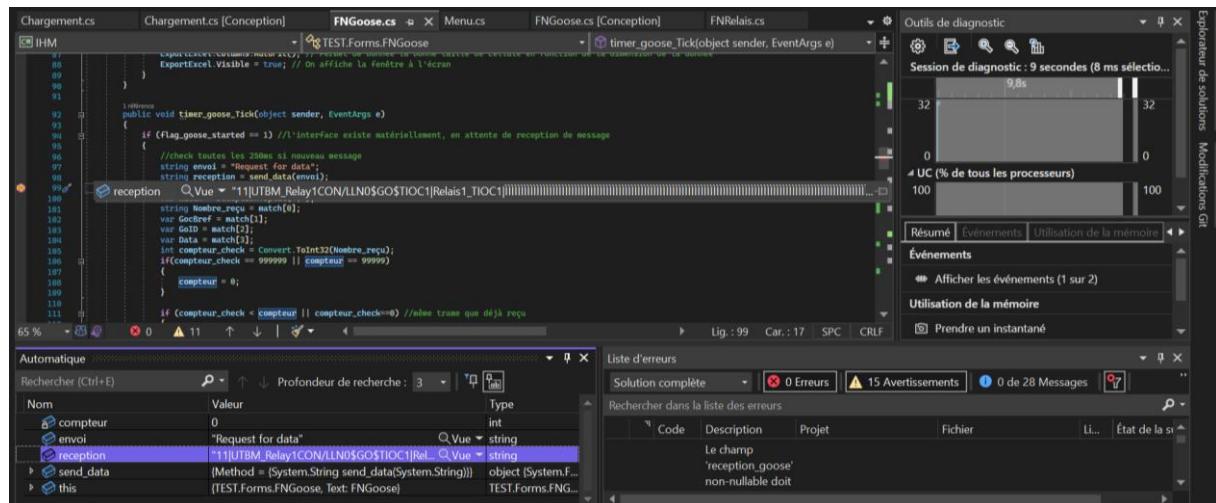


Figure 47 : Capture d'écran de la trame erronée

Le problème est celui de la pile qui était trop petite et donc prenait le buffer à l'adresse mémoire suivante. Vous pouvez à présent concevoir la difficulté afin de remonter petit à petit à l'erreur.

Le dll étant prêt, nous pouvons l'utiliser à travers notre projet, dans l'onglet Goose. Pour rappel, voici son aspect :

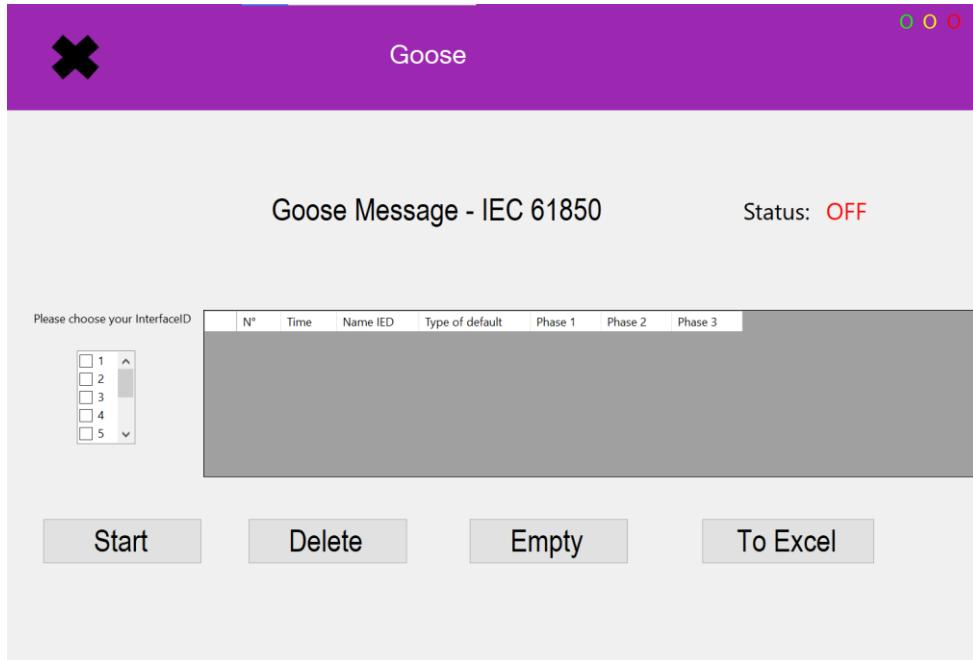


Figure 48 : Onglet Goose de l'application

Pour démarrer le sniffer, l'utilisateur doit cocher une des 10 cases correspondant à son InterfaceID et appuyer sur le bouton start. La case cochée correspond à l'argument de la fonction osKernelStart appelée par l'interaction avec le bouton. Si aucune des cases n'est cochée, un message d'erreur apparaitra spécifiant qu'une case doit être cochée. Si une case est cochée et que l'utilisateur en coche une autre, la première vient à être décochée. Si l'ID n'existe pas, comme vu précédemment dans le dll, un message d'erreur apparaitra correspondant au renvoi de l'exception :

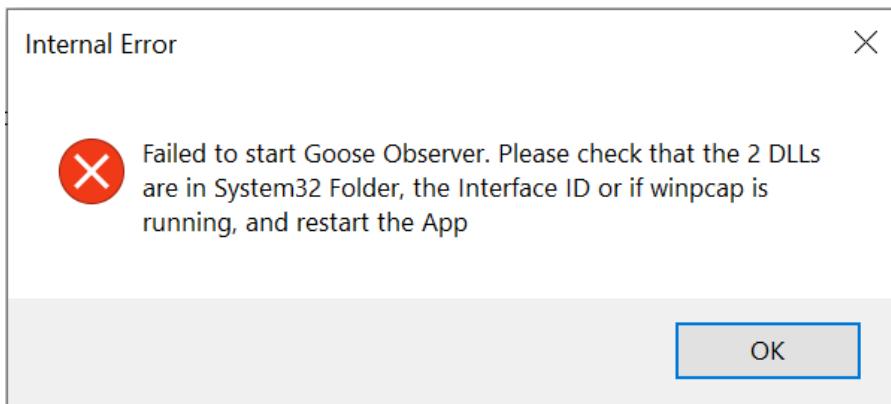


Figure 49 : Message d'erreur lors du démarrage

Cette exception peut survenir dans plusieurs cas, cité dans le message d'erreur :

- 1 voire 2 DLL manquent ou ne sont pas trouvés
- L'ID n'est pas associé à un matériel
- Winpcap n'est pas en cours d'exécution

Si toutes ces conditions sont remplies, l'appui sur le bouton « Start » va lancer un timer, et afficher un warning :

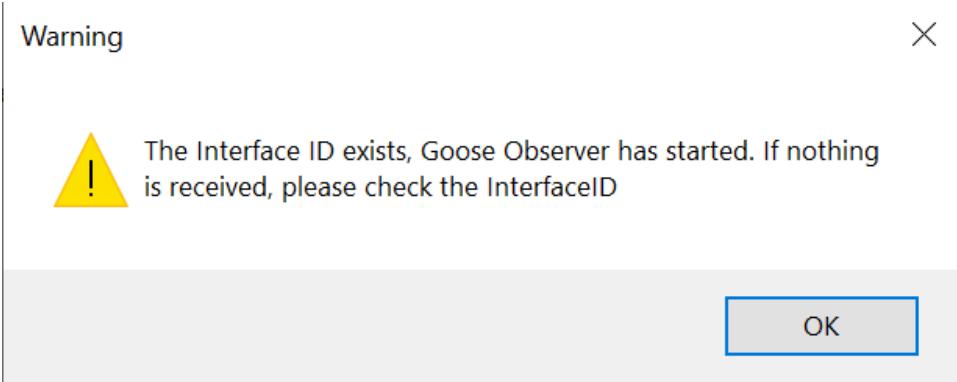


Figure 50 : Message de prévention lors du démarrage du sniffer

Si toutes les conditions sont remplies, cela ne veut pas dire qu'il y aura réception de trames Goose. Pour ce faire il faut que l'Interface ID fasse référence à la carte réseau sur laquelle sont connectés les IEDs. Pour le moment, il ne nous a pas été possible de faire un quelconque lien afin de déterminer ce paramètre. Il faut donc y aller un par un comme le spécifie le message.

L'affichage du statut est aussi modifié selon l'état du sniffer. Voici la fonction correspondant avec la pression sur le bouton start :

```
1. private void Button_Start_Click(object sender, EventArgs e)
2. {
3.     checkedItems =
4.     InterfaceID_list.CheckedItems.Cast<object>().Aggregate(string.Empty, (current, item) =>
5.     current + item.ToString()); //regarde quelle valeur est cochée
6.     if(checkedItems == "") //pas de case cochée
7.     {
8.         MessageBox.Show("Please select an Interface ID and press Start Button ", "Error",
9.         MessageBoxButtons.OK, MessageBoxIcon.Error);
10.    }
11.    else
12.    {
13.        //int ID_int = Convert.ToInt32(checkedItems);
14.        //byte[] intBytes = BitConverter.GetBytes(ID_int);
15.        //Array.Reverse(intBytes);
16.        //Interface_ID = intBytes; //conversion
17.        myGoose_thread = new Thread(New_Thread_create); //création nouveau thread
18.        myGoose_thread.IsBackground = true;
19.        myGoose_thread.Start(); //démarrage en tâche de fond
20.
21.        Button_Start.Visible = false; //goose_observer dans la boucle infinie,
22.        impossible de rémarrer, bouton start désactivé
23.        Boutton_Stop.Visible = true; //goose_observer dans la boucle infinie,
24.        impossible de rémarrer, bouton start désactivé
25.    }
26.    Thread.Sleep(200);
27.    if(flag_goose_started == 1) //l'interface existe matériellement, en attente de
28.    reception de message
29.    {
30.        ON_OFF_Label.Text = "ON";
31.        ON_OFF_Label.ForeColor = Color.Green;
32.        MessageBox.Show("The Interface ID exists, Goose Observer has started. If
33.        nothing is received, please check the InterfaceID", "Warning",
34.        MessageBoxButtons.OK, MessageBoxIcon.Warning);
```

```

29.             timer_goose.Enabled = true;
30.         }
31.         else if (flag_goose_started == 2) //l'interface n'existe pas matériellement,
impossible de démarrer
32.         {
33.             ON_OFF_Label.Text = "OFF";
34.             ON_OFF_Label.ForeColor = Color.Red;
35.         }
36.     }

```

Pour le sniffer (osKernelStart), on met le noyau sur un Thread auxiliaire afin de libérer la file de calcul principale.

```

37. private void New_Thread_create() //creation nouveau thread
38.     {
39.         try
40.         {
41.             flag_goose_started = 1; // a démarré
42.             Launch_Goose_Observer();
43.         }
44.         catch (Exception ex)
45.         {
46.             flag_goose_started = 2; //echec démarrage
47.             MessageBox.Show("Failed to start Goose Observer. Please check that the 2 DLLs
are in System32 Folder, the Interface ID or if winpcap is running, and restart the App ",
"Internal Error",
48.                         MessageBoxButtons.OK, MessageBoxIcon.Error);
49.         }
50.     }
51.
52.     //#[MethodImpl(MethodImplOptions.NoInlining)]
53.     void Launch_Goose_Observer()
54.     {
55.         osKernelStart(checkedItems); //fonction DLL
56.     }

```

On détecte les potentielles Exceptions prévues dans le code du DLL. On note également la présence d'un flag permettant la communication inter-thread de l'état du sniffer.

De plus, le bouton start est rendu invisible pour laisser place au bouton stop dans le cas où aucun message d'erreur n'est apparu.

Pour le bouton stop :

```

57.     private void Boutton_Stop_Click(object sender, EventArgs e)
58.     {
59.         Button_Start.Visible = true; //goose_observer dans la boucle infinie, impossible
de rémarrer, bouton start désactivé
60.         Boutton_Stop.Visible = false; //goose_observer dans la boucle infinie,
impossible de rémarrer, bouton start désactivé
61.         ON_OFF_Label.Text = "OFF";
62.         ON_OFF_Label.ForeColor = Color.Red;
63.         timer_goose.Enabled = false;
64.         sigint_handler(0);
65.     }

```

Comme le seul moyen pour stopper un noyau est de le tuer il a d'abord été imaginé un code détruisant le thread créé :

```

66. /*Code pour kill un thread et afficher son statut (ne fonctionne pas car impossible en C#)
67.
68.     myGoose_thread.Interrupt();
69.     myGoose_thread.Interrupt();
70.     if ((myGoose_thread.ThreadState & ThreadState.Running) == ThreadState.Running)
71.     {
72.         MessageBox.Show("Thread is still running", "Warning",
73.                         MessageBoxButtons.OK, MessageBoxIcon.Warning);
74.     }*/

```

Cependant ceci n'est plus géré avec les derniers Framework car cela engendrait beaucoup trop d'erreurs et d'instabilité. Dans notre cas on stoppe le timer ce qui revient à arrêter d'interroger le dll. Cela permet aussi de libérer de la puissance de calcul et de réduire la consommation électrique. De plus, on appelle la fonction du DLL qui va détruire le noyau. A l'inverse, on s'occupe de l'affichage en masquant le bouton stop et en affichant le bouton start.

Le cœur de la récupération des messages est exécuté à chaque tic de timer, c'est-à-dire toutes les 50ms. A noter que le timer reste actif lorsque l'on change d'onglet donc le sniffer également. Selon l'ordinateur utilisé, une perte de performance peut être ressentie en cas de plusieurs utilisation simultanées de l'application (exemple tentative de connexion + sniffer).

On commence par regarder si le sniffer est bien actif, alors toutes les 50ms, alors on récupère les informations retransmises par le dll dans une string. On incrémente le compteur local, initialisé au début à 0. On décompose ensuite la trame pour la retrouver dans la même forme que dans l'interruption du dll.

```

75. public void timer_goose_Tick(object sender, EventArgs e)
76.     {
77.         if (flag_goose_started == 1) //l'interface existe matériellement, en attente de
reception de message
78.         {
79.             //check toutes les 250ms si nouveau message
80.             string envoi = "Request for data";
81.             string reception = send_data(envoi);
82.             compteur++; //compteur synchro avec le DLL pour savoir si nouveau message
reçu entre chaque check
83.             var match = reception.Split('|');
84.             string Nombre_reçu = match[0];
85.             var GocBref = match[1];
86.             var GoID = match[2];
87.             var Data = match[3];
88.             int compteur_check = Convert.ToInt32(Nombre_reçu);

```

A ce moment on compare les deux compteurs. Si le compteur reçu est égal à 0, cela veut dire qu'aucune trame n'a été réceptionnée, on ne fait rien. De la même manière que dans le dll, si le compteur est trop grand, on le réinitialise.

Si le compteur local est plus grand que le compteur reçu, alors aucune trame, depuis la dernière enregistrée, n'a été réceptionnée.

```

89.             if(compteur_check == 999999 || compteur == 99999)
90.             {
91.                 compteur = 0;
92.             }
93.
94.             if (compteur_check < compteur || compteur_check==0) //même trame que déjà
95.             reçu
96.             {
97.                 compteur--;

```

Sinon, on réalise un deuxième découpage de chaque variable, pour obtenir les données que l'on veut :

- gocbRef, découpage jusqu'à « CON » : UTBM_Relay1CON/LLNO\$GO\$PIOC1 → UTBM_Relay1
- goID, découpage jusque « _ » : Relais1_PIOC → PIOC
- Data :
 - Shift de 2 vers la gauche pour supprimer les deux « {} » du début
 - Séparation par des « , » pour récupérer l'état des trois phases

On incrémente le numéro de la ligne et on synchronise les deux compteurs (local et dll).

```

98.         else
99.         {
100.             compteur_tableau++;
101.             compteur = compteur_check; //resynchro des compteurs
102.             string[] token_GocBref = GocBref.Split(new[] { "CON" }, StringSplitOptions.None);
103.             Nom_relais = token_GocBref[0];
104.
105.             string[] token_GoID = GoID.Split('_');
106.             Nom_Defaut = token_GoID[1];
107.
108.             string Data1 = Data.Substring(2, Data.Length - 2) + Data.Substring(0, 2);
// shift de 2 pour enlever les {
109.
110.             string[] token_Data = Data.Split(',');
111.             Nom_Phase1 = token_Data[2];
112.             Nom_Phase2 = token_Data[4];
113.             Nom_Phase3 = token_Data[6];

```

On récupère le temps grâce à l'horloge Windows et on affiche les informations dans le tableau :

```

114.             //obtention du temps
115.             DateTime dateTime = DateTime.Now;
116.             Time_actual = dateTime.ToString("ddd, dd MMMM yyyy HH:mm:ss");
117.             //Ajout de la ligne du défaut dans le tableau datagridview du goose
118.             GridGoose.Rows.Add(compteur_tableau, Time_actual, Nom_relais, Nom_Defaut,
119.             Nom_Phase1, Nom_Phase2, Nom_Phase3);
120.         }

```

Concernant l'affichage, voici le résultat obtenu :

The screenshot shows a software application window titled "Goose". At the top left is a large black "X" icon. At the top right are three colored circles (green, orange, red). The main area displays a table titled "Goose Message - IEC 61850" with the status "Status: ON". A dropdown menu on the left says "Please choose your InterfaceID" with options 1 through 5, where 5 is selected. The table has the following data:

N°	Time	Name IED	Type of default	Phase 1	Phase 2	Phase 3
1	vendredi, 17 juin 2022 00:04:26	UTBM_Relay1	PIOC	true	true	true
2	vendredi, 17 juin 2022 00:04:26	UTBM_Relay1	PIOC	true	true	true
3	vendredi, 17 juin 2022 00:04:26	UTBM_Relay1	PIOC	true	true	true
4	vendredi, 17 juin 2022 00:04:26	UTBM_Relay1	PIOC	true	true	true
5	vendredi, 17 juin 2022 00:04:26	UTBM_Relay1	PIOC	true	true	true
6	vendredi, 17 juin 2022 00:04:26	UTBM_Relay1	PIOC	true	true	true
7	vendredi, 17 juin 2022 00:04:26	UTBM_Relay1	PIOC	true	true	true
8	vendredi, 17 juin 2022 00:04:26	UTBM_Relay1	PIOC	true	true	true

At the bottom are four buttons: "Stop", "Delete", "Empty", and "To Excel".

Figure 51 : Aperçu de la simulation de réception de messages Goose

Nous avons donc, en termes d'application de la norme IEC61850, réalisé une connexion et un sniffer Goose. Cela peut paraître peu pour un travail d'un semestre mais il a fallu partir de bien plus bas qu'initialement prévu. La librairie comporte aussi un code de récupération de Sample Values. Ne pouvant pas le tester, nous avons préféré ne pas l'implémenter afin de garantir une application la plus stable pos

Conclusion

A travers ce projet TO54, nous avons pu développer énormément nos compétences dans le domaine de l'informatique avec la conception de cette IHM en C#, tout en restant dans le domaine de l'énergie puisque le but principal était de montrer des relais de protection présent dans les sous-stations électrique. Ce projet n'a pas toujours été facile et nous a pris énormément de temps pour la résolution de bug, les problèmes de libraires, etc... Nous sommes donc arrivés à terme à réceptionner des messages Goose selon la norme IEC-61850, on peut donc désormais avec notre IHM récupérer pour n'importe quel type d'IED les potentiels message Goose transitant sur le réseau Ethernet local. Il serait judicieux d'ailleurs d'essayer avec de nouveau IED. Les IED que nous possérons à l'UTBM ne permettent que de répondre à cette partie de la norme, bien que la partie pour la réception des Samples Values soit implémenté graphiquement elle ne fonctionne pas car les IED relais GE F650 ne gèrent pas ce type de données.

En ce qui concerne (Hugo), malgré le fait que je suis en énergie et non en informatique, j'ai quand même apprécié travailler sur ce projet. J'ai pu développer mes affinités avec ce domaine ce qui m'a permis d'obtenir de nouvelles compétences tout en sortant de ma zone de confort. Au tout début de ce projet je ne connaissais absolument pas la programmation en C#, il a fallu donc apprendre sur le tas avec l'aide de Sébastien qui avait déjà pratiqué ce langage. Je tenais à le remercier pour sa patience car sans lui le projet n'aurait pas été à son terme. Nous avons su ensemble surmonter les difficultés lorsqu'il y en avait. Le projet était difficile car quand nous pensions avancer, on était parfois bloqué par des bugs qui ont mis pour certain des semaines à être résolus. Au tout début du semestre je n'étais pas très confiant mais petit à petit j'ai réussi à mieux comprendre le langage C#(WindowsForm) ce qui me permettait au fil de l'avancement de me corriger moi-même. Finalement même si ce projet fut complexe j'ai pu apprendre de nouvelles techniques qui me serviront un jour dans mon métier d'ingénieur.

En ce qui concerne (Sébastien), la compréhension du sujet fut assez compliquée. Ceci était dû au caractère aléatoire des maquettes notamment. Il était important pour moi de comprendre entièrement le fonctionnement et l'utilité. Au début il était question de trouver une librairie et de simplement l'utiliser. La situation a commencé à se complexifier lorsqu'il y avait des manipulations à faire pour utiliser la librairie. Il y avait très peu d'explications, associé à mon manque de connaissances informatique quant à l'application de librairies, ont rendu le travail particulièrement dur. Mais ceci s'est encore complexifié lorsque la fonction sniffer de messages Goose de celle-ci ne fonctionnait pas. C'est ici que pour le bien du projet, je me suis lancé dans un défi « fou », modifier la librairie et créer une dll. A quelques semaines du rendu, le temps pressait. Les nuits commençaient à être très courtes. Le nombre de problèmes rencontrés est indénombrable sans compter de la difficulté et les fausses pistes de résolution. Cette modification a été intégralement effectué par mes soins sans aucune aide (ce n'est pas faute d'avoir demandé à plusieurs reprises). Je pense qu'Hugo et moi-même, nous nous sommes surpassés par rapport au travail initialement demandé, sur un domaine qui ne nous est par directement concerné dans le cadre de nos études. Je pense que cela a été une expérience certes très éprouvante mais particulièrement enrichissante dans des domaines où peu de gens osent s'y aventurer, qui saura me mettre en valeur.

Enfin si nous devions donner un avis global sur le travail réalisé durant ce semestre, nous dirions que dans l'ensemble nous sommes satisfaits et heureux de voir notre application fonctionner. Nous avons pu aller au bout du projet en réalisant une IHM facile à utiliser et répondant au cahier des charges qui nous avait été confié c'est-à-dire répondant à la norme IEC-61850. Nous avons énormément travaillé sur ce projet, sans compter réellement nos heures, c'est pourquoi nous pensons avoir remplis notre contrat et nos objectifs pour ce projet même s'il reste toujours des axes d'amélioration à développer.

Annexe : Guide d'utilisation de l'IHM

Comment lancer l'application :

- Copier les deux fichiers iec61850 et HIRTH_TEANI en .lib ou .dll dans votre dossier System32 (dossier fichier système)
- Ouvrir le port 102 de votre ordinateur et lever les exceptions du pare-feu pour notre IHM
- Installer WINPCAP puis NPCAP (dossier programme)
- Ouvrir le dossier contenant l'application (dossier application)

 IEDsave	14/06/2022 18:05	Document texte	1 Ko
 IHM.deps	14/06/2022 12:39	JSON File	41 Ko
 IHM.dll	14/06/2022 16:04	Extension de l'app...	615 Ko
 IHM	14/06/2022 16:04	Application	146 Ko
 IHM.pdb	14/06/2022 16:04	Program Debug D...	67 Ko
 IHM.runtimeconfig	14/06/2022 12:39	JSON File	1 Ko
 Microsoft.CSharp.dll	14/03/2022 18:29	Extension de l'app...	1 021 Ko
 Microsoft.DiaSymReader.Native.amd64.dll	17/12/2020 00:57	Extension de l'app...	1 772 Ko
• Exécuter l'application IHM			
 IHM	14/06/2022 16:04	Application	146 Ko

- L'application se lance et est fonctionnelle
- Si un message d'erreur apparaît lors de l'ouverture de l'application, c'est que l'application ou la sauvegarde voire les liens des images ont été modifiés. Pour résoudre ce problème il suffit de supprimer le fichier texte IEDsave s'il est présent dans votre dossier
- Pour tout autre problème merci de suivre les indications ci-dessous après avoir lu entièrement notre rapport.

Nous contacter en cas de problème sur l'application :

- Sebastien.hirth@utbm.fr
- Hugo.teani@utbm.fr