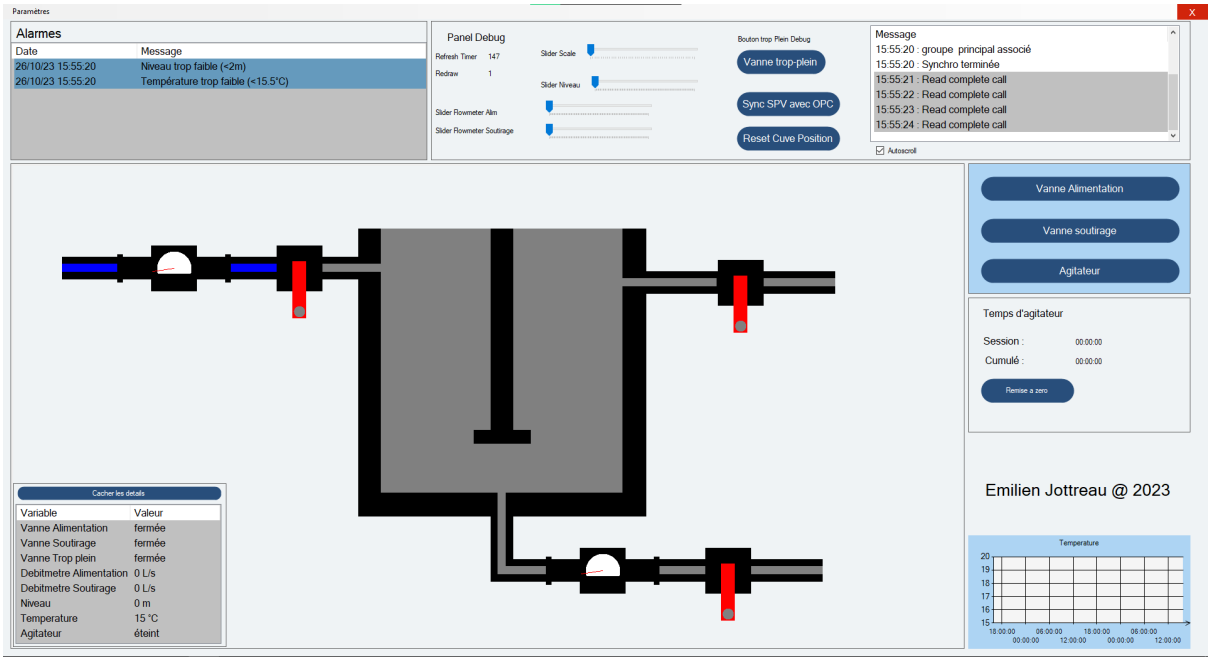
Compte-rendu Supervision

Emilien Jottreau 2023

1- Présentation de l’interface

Voici à quoi ressemble l’interface, elle fonctionne sur des écrans 1920x1080, il n’est pas recommandé de la tester sur des écrans plus petits, sinon les panels se chevauchent.



Interface au démarrage

Elle est segmentée, je vais vous détailler l’utilisation de ces parties.

1-a. Alarmes

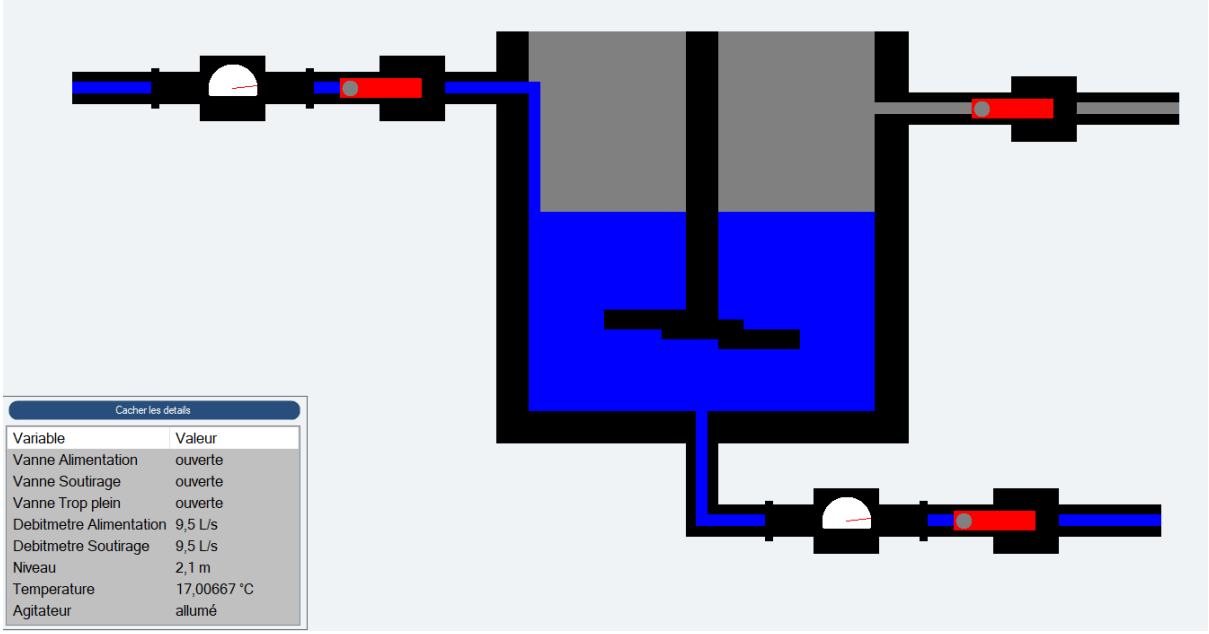
En haut à gauche, on retrouve la zone pour afficher les alarmes. Elles sont datées et s’affichent de la plus récente a la plus ancienne. En bleu, on retrouve les alarmes actives, elle passent au gris lorsque l’alarme est résolue.

1-b. Debug

En haut à droite, on a un panel utile uniquement pour le debug de l’application. On retrouve des sliders pour tester le déplacement des aiguilles des débitmètres, changer le niveau d’eau …
Il y a aussi un bouton pour synchroniser toutes les variables avec l’OPC, par exemple lorsque l’on met des breakpoints pour debugger et que l’on manque un changement de valeur. Une listbox a droite répertorie tous les évènements, sur l’image ci dessus on voit des messages relatifs à la lecture de tag OPC. Il existe toute sorte de message, comme l’écriture d’un tag OPC, le changement de chemin de log, le changement de taux de rafraîchissement …
En production, il faudrait cacher ce panel, il ne sert qu’aux développeurs pour tester les fonctionnalités. On pourrait laisser la listbox pour la maintenance pour détecter des défauts.

1-c. Synoptique

Au centre gauche, on a une représentation de la cuve, avec les vannes, les débitmètres et l'agitateur. Tous ces éléments sont animés pour refléter l'état des variables.



Cuve avec un état diffèrent

On voit ici que les vannes n'ont plus la même position, le niveau d'eau est monté, les aiguilles de débitmètres ont bougé, l'agitateur tourne (il faut le voir en lançant la supervision, c'est une succession d'images qui donne l'impression qu'il tourne).

Pour changer l'état des vannes, l'utilisateur à la possibilité de cliquer sur la représentation graphique au niveau des vannes.
Le graphique est déplaçable et redimensionnable, grâce au défilement de la souris ou d'un glisser-déplacer.

1-d. Détails

En bas à gauche, on voit la valeur des variable OPC affiché sous forme de tableau. Il possède un bouton pour le cacher, lorsqu'il est cliqué le tableau descends en dehors de la fenêtre et laisse apparaître un bouton pour l'afficher.

1-e. Commandes

Sur la droite, on retrouve 3 boutons pour interagir avec les vannes et l'agitateur. Le temps d'allumage de l'agitateur ainsi que le temps cumulé sont affichés. le temps

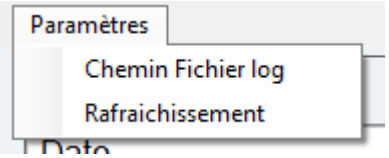
total peut être remis a zéro.

1-f. Graphique

En bas à droite, on retrouve un graphique de l'évolution de la température en fonction du temps. Malheureusement, avec le simulateur, la température n'évolue pas beaucoup avec la "deadband" qu'on a mis (détaillé par la suite) on ne voit rien apparaitre. Pour voir cette fonctionnalité, il faut mettre le simulateur en pause, et changer la valeur de la température a la main.

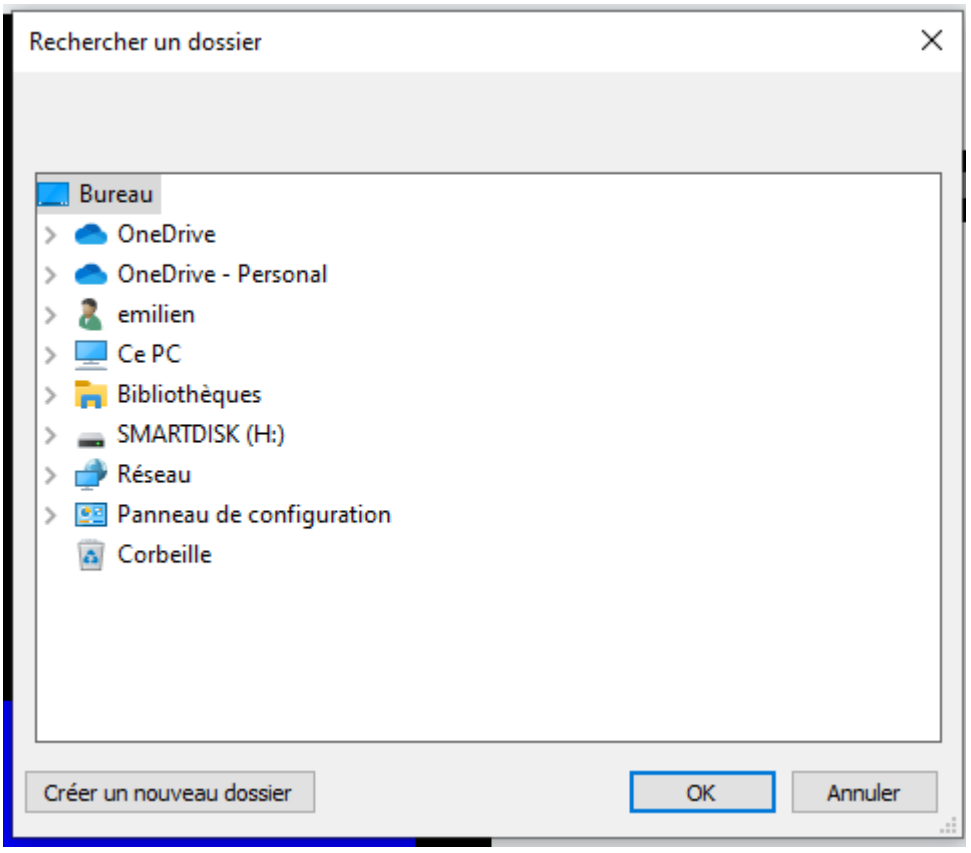
1-g. Menu paramètre

Dans le menu paramètre tout en haut, on peut sélectionner le dossier pour récupérer les fichiers log. On peut aussi changer le taux de rafraîchissement de l'affichage



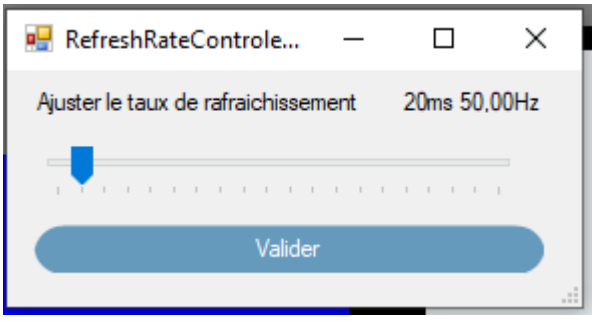
Menu paramètre

Pour changer la destination des fichiers log, j'utilise une fonction intégrée de .NET qui est "FolderBrowserDialog", il permet de naviguer dans les dossiers de l'ordinateur et de renvoyer le dossier choisi



Fenêtre de navigation dans les fichiers

On retrouve ensuite la fenêtre de changement de taux de rafraîchissement, cela affecte la partie synoptique, le tableau détail et le rafraîchissement de temps affichés pour l'agitateur et l'ajout de valeurs sur le graphique température.

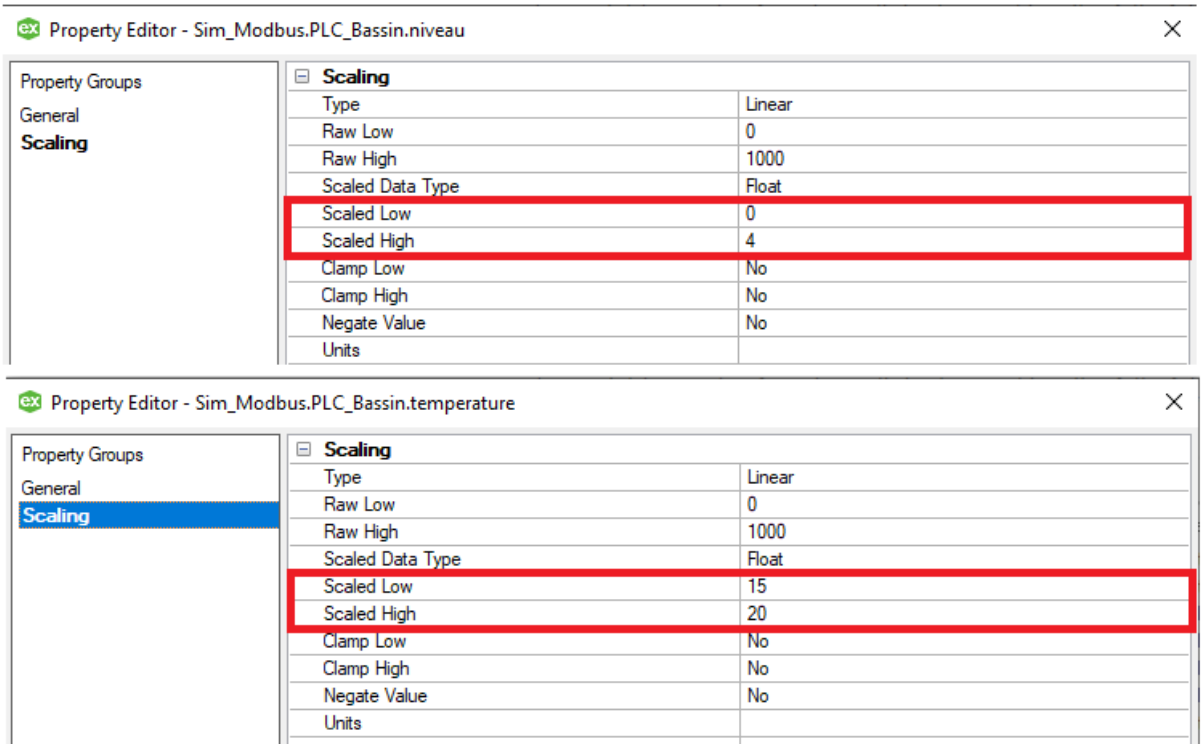


Fenêtre changement taux de rafraîchissement

2- Configuration serveur OPC

Dans le cadre de notre projet, nous utilisons KEPServeur. Il faut configurer l'espace d'adressage des variables OPC. Il faut charger le fichier "EspaceAdressageBassinEmilien.opf" dans KEPServer.

Au niveau de la configuration de KEPServeur, je vais détaillé ce qui a été fait. J'ai précisé sur la variable "niveau" et "temperature" les bornes maximales et minimales. Dans l'énoncé, il est précisé que, les valeurs de niveau vont de 0 à 4 mètres, les valeurs de température vont de 15 a 20 °C. Voici comment les paramétrer :

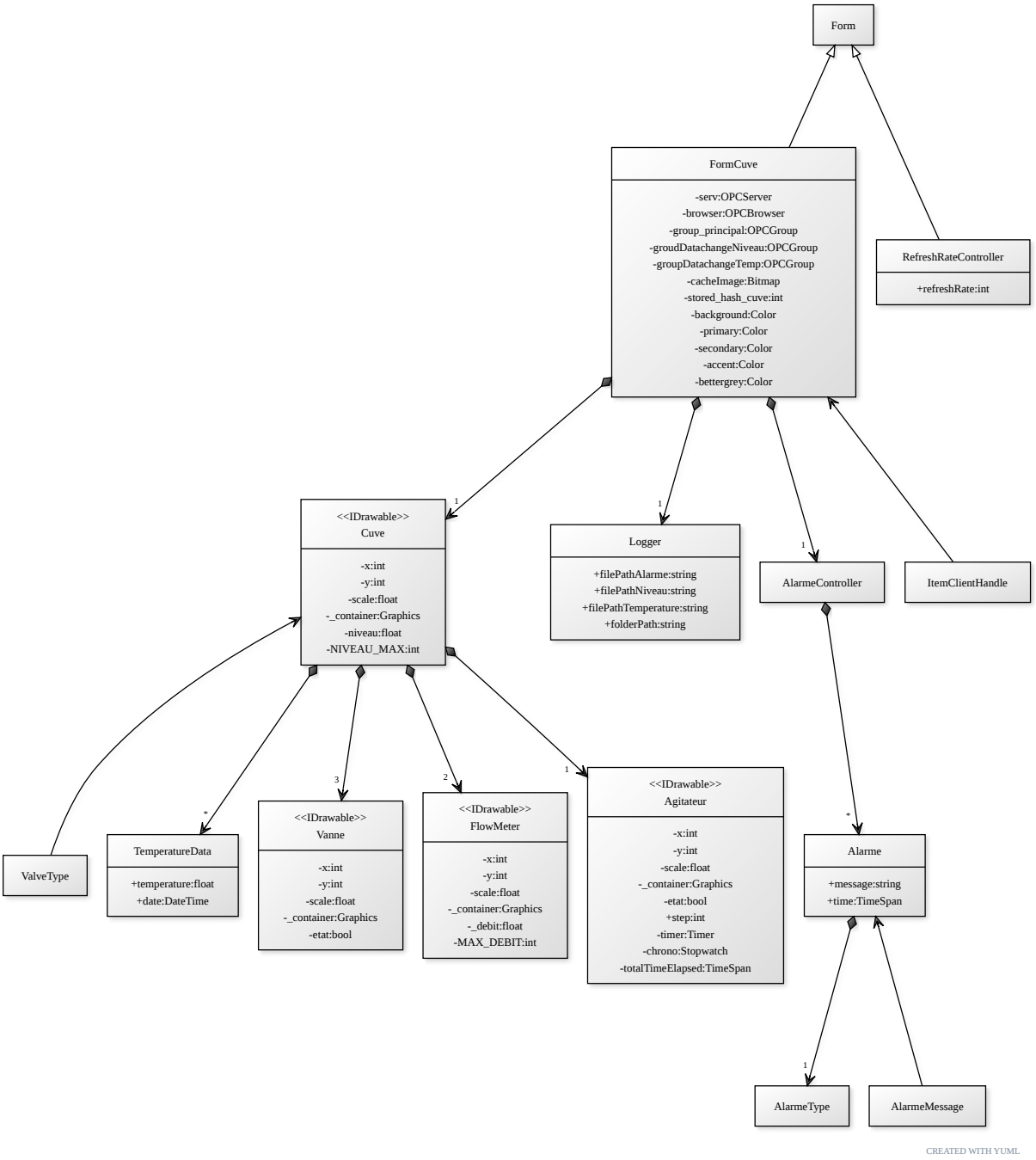


Configuration Scalling des deux variables

J'ai fait des tests et cela fonctionne comme prévu

3- Explication Technique

J'ai décidé d'organiser mon code avec des classes, voici le diagramme UML pour comprendre la structure de mon code. On ne voit pas les fonctions car je trouve cela plus lisible



3-a. Forms

Comme indiqué précédemment, j'ai deux form, le premier est principal, c'est celui qui est affiché au lancement de l'application. Le deuxième sert a gérer le taux de rafraichissement, il n'est pas très important, mais il se trouve dans le fichier RefreshRateControllerForm.cs.

3-b. Timers

Dans cette application, j'ai 3 timers.

1. timerUIRefresh, l'intervalle de temps entre deux activations est configurables grâce aux deuxième form. À chaque activation, il met a jour l'interface *(ligne 548 de FormCuve.cs)*
    - Redessine le dessine si un élément a changé de valeur (position, échelle, état …).
    - Met à jour les chronomètre de l'agitateur.
    - Déplace le panneau détail si besoin.
    - Met à jour le graphique de la température.
    - Fait en sorte que le dernier élément de la listbox debug soit visible (si la checkbox autoscroll est cochée)
2. timerRefreshDebit, il s'active toutes les secondes, il fait une requête asynchrone de lecture de la valeur des débits (détaillé au 3-e. Fonction OPC) *(ligne 515 de FormCuve.cs)*
3. timerSaveLogFile, il s'active toutes les minutes, il déclenche les fonctions du Logger pour faire les écritures dans les fichiers. *(ligne 826 de FormCuve.cs)*

3-c. Dessin

J'ai décidé de représenter les éléments physiques de la cuve par des classes qui implément une interface que j'ai créée: "IDrawable". Ce sont ces éléments que l'on voit dessiné sur le synoptique.

J'utilise une pictureBox pour afficher le dessin. Je stocke le dessin dans une Bitmap.

Pour réaliser ces dessins j'utilise la librairie System.Drawing. Le principe est simple on crée des formes géométriques, on donne une couleur et on le dessine sur un Graphics. La partie dessin se retrouve dans la fonction drawGraphic() des classes implémentant l'interface IDrawable.

```
2 références
public void drawGraphic()
{
    SolidBrush blackBrush = new SolidBrush(Color.Black);

    int H_TIGE = (int)(140 * this.scale);
    int W_TIGE = (int)(16 * this.scale);
    int W_PALE = (int)(10 * this.scale);
    int L_PALE = (int)(40 * this.scale);
    int OFFSET_PALE = (int)(5 * this.scale);


    Rectangle tige = new Rectangle(x, y, W_TIGE, H_TIGE);
    Rectangle bottom = new Rectangle(x - L_PALE / 2 + W_TIGE/2, y + tige.Height, L_PALE, W_PALE);
    Rectangle pale_gauche = new Rectangle(x - L_PALE, y + tige.Height, L_PALE, W_PALE);
    Rectangle pale_droite = new Rectangle(x + tige.Width, y + tige.Height, L_PALE, W_PALE);

    this._container.FillRectangle(blackBrush, tige);
    this._container.FillRectangle(blackBrush, bottom);

    if (this.step % 2 == 0) return;

    if (this.step == 1)
    {
        pale_gauche.Y -= OFFSET_PALE;
        pale_droite.Y += OFFSET_PALE;
    }
    else
    {
        pale_gauche.Y += OFFSET_PALE;
        pale_droite.Y -= OFFSET_PALE;
    }

    this._container.FillRectangle(blackBrush, pale_gauche);
    this._container.FillRectangle(blackBrush, pale_droite);
}
```

Fonction drawGraphic() *(ligne 51 de Agitateur.cs)*

Je vais prendre l'exemple de l'agitateur pour expliquer, le SolidBrush est en quelque sorte l'outil de dessin (avec la couleur). Ensuite, on définit des rectangles qui représentent les formes à dessiner, il faut leur donner l'emplacement, et les dimensions. Grâce à la fonction FillRectangle on peut le dessiner sur un Graphics. Je ne vais pas détailler toutes les fonctions drawGraphic(), elles se ressemblent sur le principe, celle de la cuve est la plus complexe. Une dernière chose à savoir est pour la superposition, le dernier objet dessiné se retrouve au premier plan donc l'ordre importe.

Ces fonctions de dessin sont un peu longue a executer. Pour ne pas avoir a redessiner alors que rien n'a changé, j'ai opté pour une solution de cache avec la Bitmap. Je calcule un hash qui me permet de représenter mon système avec un nombre. Je vais ensuite comparer ce hash avec sa dernière valeur connue et redessiner en fonction

```
1 référence
private void updateUI()
{
    int currentHash = getHashCuve();
    if (currentHash != stored_hash_cuve)
    {
        cuve.drawGraphic();
        nbDraw++;
        label2.Text = nbDraw.ToString();
        stored_hash_cuve = currentHash;
        pictureBoxCuve.Image = cacheImage;

        updatePanelDetailsText();
    }
```

Fonction updateUI() *(ligne 562 de FormCuve.cs)*

3-d. Fonctionnalité Arrière plan

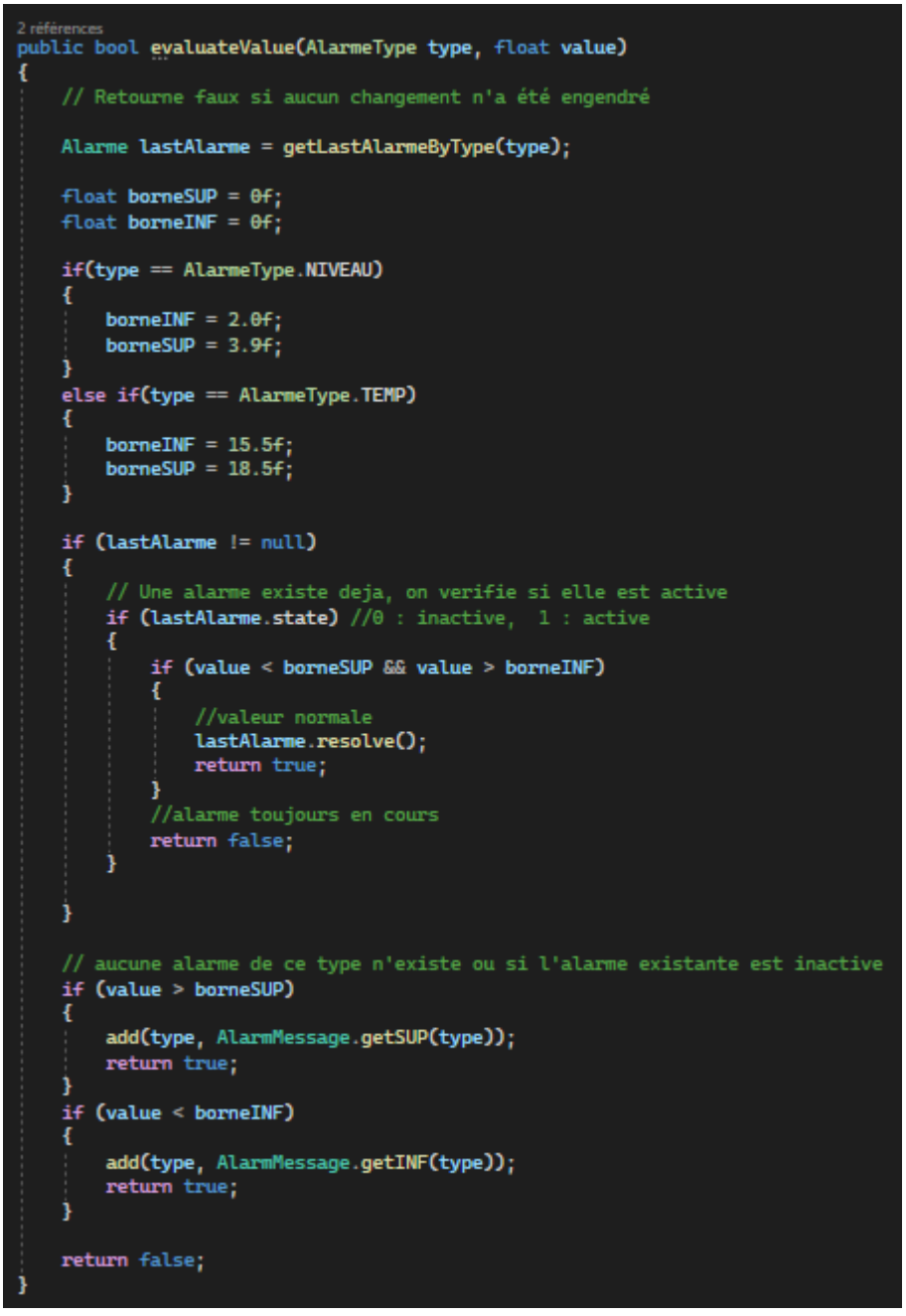
J'ai délégué les responsabilités d'écriture de log et de gestion des alarmes aux classes "Logger" et "AlarmeController"

Les fonctions du Logger sont appelées avec le timerSaveLogFile.

```
2 références
private void dataChange(int TransactionID, int NumItems, ref Array ClientHandles, ref Array ItemValues, ref Array Qualities, ref Array TimeStamps)
{
    bool updateAlarme = false;
    // ici on recoit temperature et niveau
    for (int i = 1; i <= NumItems; i++)
    {
        if ((int)ClientHandles.GetValue(i) == ItemClientHandle.TEMPERATURE)
        {
            addDebugInfo("Datachange Temperature called", Severity.DEBUG);
            bool is_added = cuve.addTemperature(Convert.ToSingle(ItemValues.GetValue(i)));
            if (!is_added) continue;

            if (alarmeController.evaluateValue(AlarmeType.TEMP, cuve.getLastTemperature())) updateAlarme = true;


            continue;
        }

        if ((int)ClientHandles.GetValue(i) == ItemClientHandle.NIVEAU)
        {
            addDebugInfo("Datachange Niveau called", Severity.DEBUG);
            bool is_added = cuve.setValue(Convert.ToSingle(ItemValues.GetValue(i)));
            if (!is_added) continue;

            if (alarmeController.evaluateValue(AlarmeType.NIVEAU, Convert.ToSingle(ItemValues.GetValue(i)))) updateAlarme = true;

            if (Convert.ToSingle(ItemValues.GetValue(i)) > 3.9f)
            {
                if (!(bool)cuve.vannes[ValveType.TROP_PLEIN].getValue()) setTropPleinValue(true);
            }
            else
            {
                if ((bool)cuve.vannes[ValveType.TROP_PLEIN].getValue()) setTropPleinValue(false);
            }

            continue;
        }
    }
    if (updateAlarme) updateAlarmePanel();
}
```

Fonction appelée lors d'un changement de valeur *(ligne 448 de FormCuve.cs)*

En ce qui concerne l'AlarmeController, son rôle commence à la réception de nouvelle valeur de température ou de niveau. Si la valeur est acceptée par mon objet cuve, c'est a dire que la valeur a changé et que l'on peut utiliser la fonction evaluateValue(). cette fonction indique vrai si un changement sur le panel Alarme doit être opéré.



```
2 références
public bool evaluateValue(AlarmeType type, float value)
{
    // Retourne faux si aucun changement n'a été engendré

    Alarme lastAlarme = getLastAlarmeByType(type);

    float borneSUP = 0f;
    float borneINF = 0f;

    if(type == AlarmeType.NIVEAU)
    {
        borneINF = 2.0f;
        borneSUP = 3.9f;
    }
    else if(type == AlarmeType.TEMP)
    {
        borneINF = 15.5f;
        borneSUP = 18.5f;
    }

    if (lastAlarme != null)
    {
        // Une alarme existe deja, on verifie si elle est active
        if (lastAlarme.state) //0 : inactive,  1 : active
        {
            if (value < borneSUP && value > borneINF)
            {
                //valeur normale
                lastAlarme.resolve();
                return true;
            }
            //alarme toujours en cours
            return false;
        }

    }

    // aucune alarme de ce type n'existe ou si l'alarme existante est inactive
    if (value > borneSUP)
    {
        add(type, AlarmMessage.getSUP(type));
        return true;
    }
    if (value < borneINF)
    {
        add(type, AlarmMessage.getINF(type));
        return true;
    }

    return false;
}
```

Fonction AlarmeController.evaluateValue() *(ligne 124 de Alarme.cs)*

cette fonction a un comportement différent en fonction du type de valeur, mais globalement, elle retire l'alarme déjà présente si la valeur retrouve une valeur

“normale”, sinon elle ajoute des alarmes, mais pas de doublons. Si elle renvoie vrai la fonction updateAlarmePanel() est appelée

```
1 référence
private void addAlarmAtTop(Alarme al)
{

    ListViewItem item = new ListViewItem(al.getTime(), 0);
    item.SubItems.Add(al.message);

    if (al.state) item.BackColor = secondary;
    else item.BackColor = betterGray;

    listViewAlarme.Items.Insert(0, item);
}

2 références
private void updateAlarmePanel()
{
    listViewAlarme.Items.Clear();
    foreach (Alarme al in alarmeController) { addAlarmAtTop(al); }
}
```

Fonctions pour mettre à jour le panel *(ligne 600 de FormCuve.cs)*

Cette fonction rafraichie le panel alarme et ajoute les alarmes dans un ordre décroissant du temps

3-e. Fonction OPC

Au vu des variables et des fonctionnalités demandées, il a fallu faire des choix pour regrouper les items OPC.

J’ai fait le choix d’avoir 3 groupes :

- Un groupe pour l’item Niveau (1)
- Un groupe pour l’item Température (2)
- Et un dernier groupe pour les autres variables (3)

Pour le premier et le deuxième groupe, j’utilise la fonctionnalité datachange (notification d’un changement de valeur si les critères sont réunis).

```
291             float borneINF = 0f;
292             float borneSUP = 4f;
293             float interval = 0.1f;
294
295             groupDatachangeNiveau = groups.Add("DataChangeNiveau"); // ne pas oublier de mettre le scaling sur niveau et temperature
296             groupDatachangeNiveau.IsSubscribed = true;
297             groupDatachangeNiveau.DataChange += dataChange;
298             groupDatachangeNiveau.UpdateRate = 5000; //5sec
299             groupDatachangeNiveau.DeadBand = interval * 100 / (borneSUP - borneINF); // 2.5% 0.025
300
301             borneINF = 15;
302             borneSUP = 20;
303             interval = 0.3f;
304
305             groupDatachangeTemp = groups.Add("DataChangeTemperature");
306             groupDatachangeTemp.IsSubscribed = true;
307             groupDatachangeTemp.DataChange += dataChange;
308             groupDatachangeTemp.UpdateRate = 5000; //5sec
309             groupDatachangeTemp.DeadBand = interval * 100 / (borneSUP - borneINF); // 6%
310
311             addDebugInfo("groupe niveau et temperature cree", Severity.DEBUG);
312             //ajout des items:
313             foreach (string leaf in list)
314             {
315                 if (leaf.Contains("niveau")) { groupDatachangeNiveau.OPCItems.AddItem(leaf, ItemClientHandle.NIVEAU); continue; }
316                 if (leaf.Contains("temperature")) { groupDatachangeTemp.OPCItems.AddItem(leaf, ItemClientHandle.TEMPERATURE); continue; }
317             }
```

Création du groupe (1) et (2) *(ligne 291 de FormCuve.cs)*

J’ai mis la propriété updateRate a 5000ms, subscribe a true. pour la deadBand qui s’exprime en pourcentage il y a deux valeurs différentes.

Pour le calcul de la deadband je me suis basé sur ce site :

> UA Part 8: DataAccess - 6.2 PercentDeadband
>
> The DataChangeFilter in OPC 10000-4 defines the conditions under which a data change notification shall be reported. This filter contains a deadbandValue which can be of type AbsoluteDeadband or PercentDeadband. OPC 10000-4 already specifies the behaviour of the AbsoluteDeadband. This sub-
>
> https://reference.opcfoundation.org/Core/Part8/v104/docs/6.2

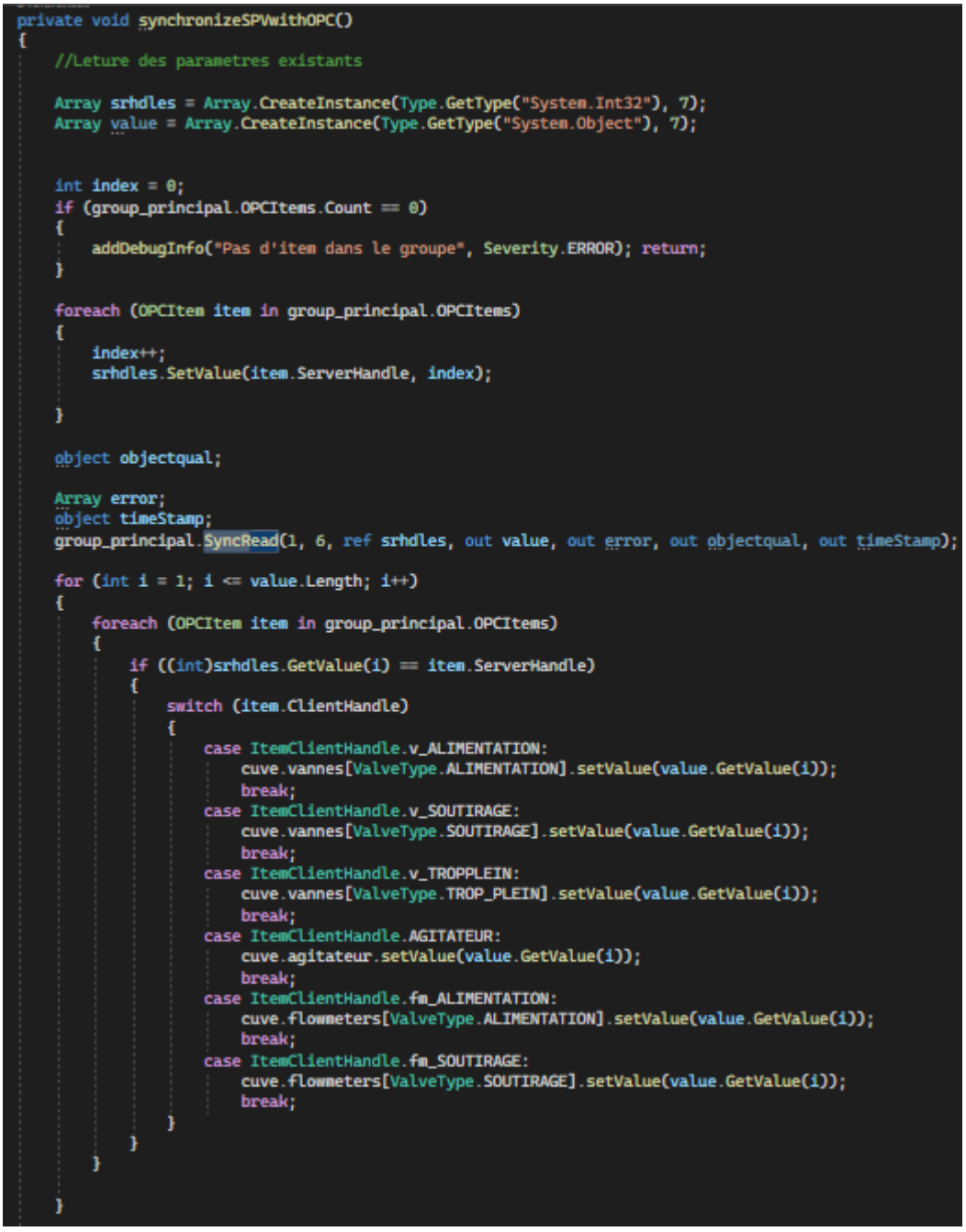
Plus particulièrement,

> “DataChange if (absolute value of (last cached value - current value) > (*deadbandValue*/100.0) * ((high–low) of *EURange*)))”

C’est pour cela qu’on retrouve un calcul dans mon code.

Quand les conditions sont réunies, la fonction dataChange() est appelée.

Pour le dernier groupe, j’utilise SyncRead pour l’initialisation au lancement de la supervision,



```
private void synchronizeSPVwithOPC()
{
    //Leture des parametres existants

    Array srhdles = Array.CreateInstance(Type.GetType("System.Int32"), 7);
    Array value = Array.CreateInstance(Type.GetType("System.Object"), 7);


    int index = 0;
    if (group_principal.OPCItems.Count == 0)
    {
        addDebugInfo("Pas d'item dans le groupe", Severity.ERROR); return;
    }

    foreach (OPCItem item in group_principal.OPCItems)
    {
        index++;
        srhdles.SetValue(item.ServerHandle, index);

    }

    object objectqual;

    Array error;
    object timeStamp;
    group_principal.SyncRead(1, 6, ref srhdles, out value, out error, out objectqual, out timeStamp);

    for (int i = 1; i <= value.Length; i++)
    {
        foreach (OPCItem item in group_principal.OPCItems)
        {
            if ((int)srhdles.GetValue(i) == item.ServerHandle)
            {
                switch (item.ClientHandle)
                {
                    case ItemClientHandle.v_ALIMENTATION:
                        cuve.vannes[ValveType.ALIMENTATION].setValue(value.GetValue(i));
                        break;
                    case ItemClientHandle.v_SOUTIRAGE:
                        cuve.vannes[ValveType.SOUTIRAGE].setValue(value.GetValue(i));
                        break;
                    case ItemClientHandle.v_TROPPLEIN:
                        cuve.vannes[ValveType.TROP_PLEIN].setValue(value.GetValue(i));
                        break;
                    case ItemClientHandle.AGITATEUR:
                        cuve.agitateur.setValue(value.GetValue(i));
                        break;
                    case ItemClientHandle.fm_ALIMENTATION:
                        cuve.flowmeters[ValveType.ALIMENTATION].setValue(value.GetValue(i));
                        break;
                    case ItemClientHandle.fm_SOUTIRAGE:
                        cuve.flowmeters[ValveType.SOUTIRAGE].setValue(value.GetValue(i));
                        break;
                }
            }
        }

    }
```

Fonction synchronizeSPVwithOPC() *(ligne 344 de FormCuve.cs)*

C’est une lecture synchrone de toutes les variables du dernier groupe.

Sinon grâce au timerRefreshDebit qui s’exécute toutes les secondes je lance une lecture asynchrone des débits et grâce a la fonction de callback readComplete(), je récupère les valeurs de débit que j’assigne.

```
1 référence
private void readComplete(int TransactionID, int NumItems, ref Array ClientHandles, ref Array ItemValues, ref Array Qualities, ref Array TimeStamps, ref Array Errors)
{
    addDebugInfo("Read complete call", Severity.DEBUG);
    // ici on recoit le debit
    for (int i = 1; i <= NumItems; i++)
    {
        if ((int)ClientHandles.GetValue(i) == ItemClientHandle.fm_ALIMENTATION)
        {
            cuve.flowmeters[ValveType.ALIMENTATION].setValue(ItemValues.GetValue(i));
            continue;
        }

        if ((int)ClientHandles.GetValue(i) == ItemClientHandle.fm_SOUTIRAGE)
        {
            cuve.flowmeters[ValveType.SOUTIRAGE].setValue(ItemValues.GetValue(i));
            continue;
        }
    }
}
```

Fonction readComplete() *(ligne 490 de FormCuve.cs)*

Pour écrire les valeurs des vannes et de l’agitateur, j’utilise AsyncWrite

Toutes les valeurs des tags que je récupère sont stockées dans les objets IDrawable (Cuve, Vanne, Flowmeter, Agitateur)