Reprise des messages de type XML dans une Queue de type MSMQueue

26/05/2025 - 04/06/2025

GUEREKOYAZANDE MALHON

Sommaire

1 Expression des besoins

1.1 Contexte, domaine, processus métier

Présentation d'Aperam Gueugnon

Aperam Gueugnon est un site industriel appartenant au groupe Aperam, situé en Saône-et-Loire. Il est spécialisé dans la production d'acier inoxydable, notamment dans la fabrication de produits plats destinés à des secteurs variés tels que l'automobile, le bâtiment ou encore l'électroménager. Le site se distingue par son expertise dans le laminage à froid et le recuit brillant.

Contexte du projet de stage

Dans le cadre de la modernisation de ses outils numériques, Aperam utilise des systèmes de messagerie tels que **MSMQueue** pour assurer les échanges entre ses applications. Mon projet de stage s'inscrit dans ce contexte : il consiste à développer un outil permettant de **reprendre des messages XML et de les rejouer dans une fille d'attente MSMQ** à partir d'un dossier d'archivage, afin de renforcer la gestion des erreurs et faciliter la reprise d'activité.

1.2 Demandeur, acteurs, utilisateurs

Le demandeur est le service informatique en charge des infrastructures et des échanges inter-systèmes. Les utilisateurs finaux sont les équipes techniques et les systèmes automatisés qui consomment les messages dans la queue.

1.3 Etude de l'existant, diagnostic

À ce jour, **aucun outil ne permet de relire ou de rejouer les messages XML à partir d'un dossier d'archivage** vers la MSMQueue. Lorsqu'un incident survient ou qu'un message doit être renvoyé, **aucune procédure n'est prévue** pour traiter ce besoin. Cela représente un **risque pour la fiabilité des échanges de données**, pouvant entraîner **des pertes d'informations** ou **des retards dans les traitements**.

Ce constat met en évidence une **lacune importante dans le processus**, ce qui justifie pleinement le développement d'un **outil dédié au rejouage de messages XML archivés**.

1.4 Description de la demande, objectifs, bénéfices attendus

Le projet vise à développer un outil en **Visual Basic .NET** doté d'une interface utilisateur intuitive. Cet outil permettra de naviguer et de sélectionner des messages XML archivés dans des répertoires de logs spécifiques (par exemple, `\\fil-srv-log\Acceptance\BZT\MES_TS03\EXEC_Log-Entry\` et `\\fil-srv-log\Acceptance\BZT\MES_TS04\EXEC_Log-Entry\`). **Ces répertoires de logs stockés sur un serveur nommé "loggingMES".** La fonctionnalité principale sera de "rejouer" ces messages XML en les envoyant vers une file d'attente **MSMQ (Microsoft Message Queuing)** spécifique.

1.5 Spécifications fonctionnelles

- L'outil développé devra pouvoir extraire les messages XML dans un dossier d'archivage, les analyser puis les rejouer selon un ordre défini.
- Il inclura une interface simple pour choisir la queue et lancer le processus.

1.6 Contraintes ou exigences (matérielles, techniques, délais, budget, ...)

- Le langage de développement utilisé est Visual Basic.NET.
- L'environnement de développement intégré (IDE) est Microsoft Visual Studio.
- La solution s'appuie sur une base de données de type SQL Server.

2 Conception, Spécifications Techniques

2.1 Description de la solution

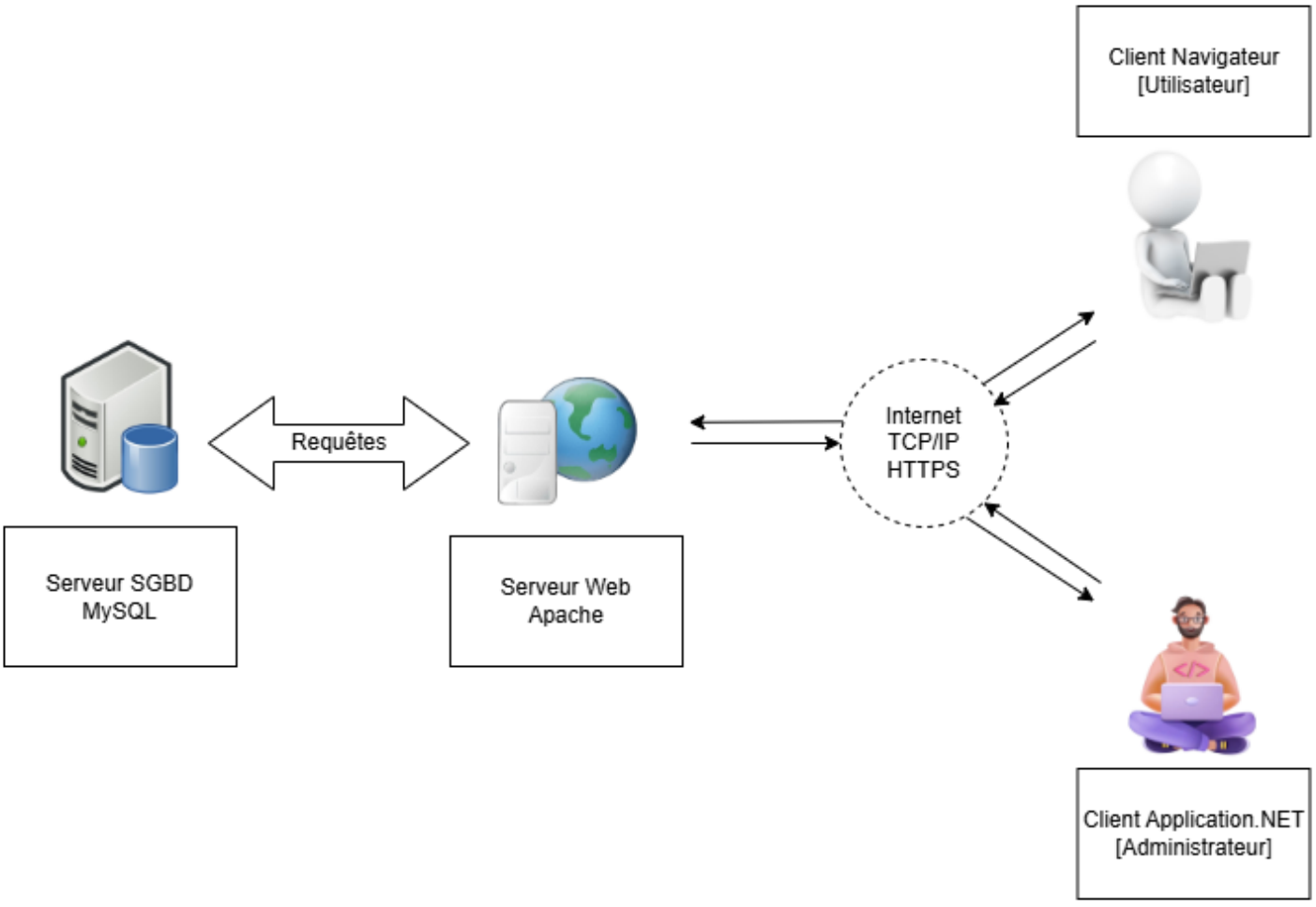
L'outil développé devra pouvoir extraire les messages XML dans un dossier d'archivage, les analyser puis les rejouer selon un ordre défini. Il inclura une interface simple pour choisir la queue et lancer le processus.

2.2 Outils logiciels de la solution

- Le **XML** (eXtensible Markup Language) est un format utilisé pour **organiser et structurer des données**. Il est lisible par les machines et les utilisateurs, ce qui le rend idéal pour **échanger, stocker ou archiver des messages métiers**. Dans cette application, les fichiers XML permettent de **retrouver et exploiter les messages** conservés dans un **dossier d'archivage**.

- **MSMQueue (Microsoft Message Queuing)** : technologie Microsoft permettant la communication asynchrone entre applications à travers des files de messages, utilisée ici pour gérer l'envoi, la réception et le rejouage de messages XML.

2.3 Architecture matérielle et logicielle de la solution (schémas)



2.4 Besoins techniques, ressources (humaines, matérielles, logicielles et budgétaires, coûts), ...

- **Schéma d'architecture de la solution :** Le document contient un schéma d'architecture qui représente les interactions entre un serveur de gestion de base de données de type SQL Server, un serveur Web Apache et des clients (application .NET) via Internet (TCP/IP HTTPS).

- **Composants logiciels clés :**

    - **XML (eXtensible Markup Language)** : Utilisé pour organiser et structurer les données des messages échangés, stockés et archivés. Il est lisible par les machines et les utilisateurs.

    - **MSMQueue (Microsoft Message Queuing)** : Technologie de communication asynchrone entre applications via des files de messages, utilisée pour l'envoi, la réception et le rejouage des messages XML.

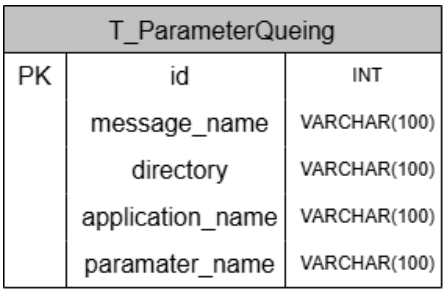
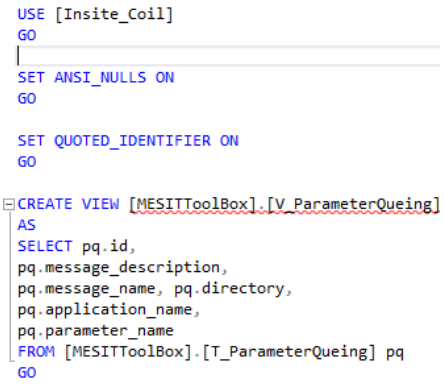
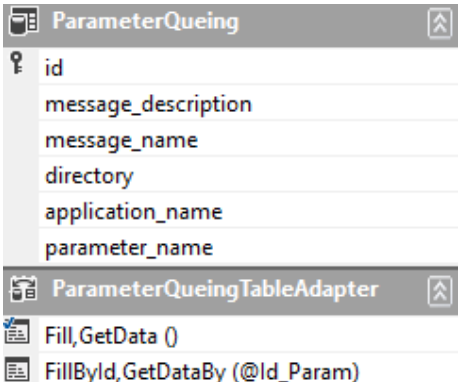
    - **Visual Basic.NET** : Langage de développement utilisé pour créer l'outil.

    - **Microsoft Visual Studio** : L'environnement de développement intégré (IDE) pour le développement de l'application.

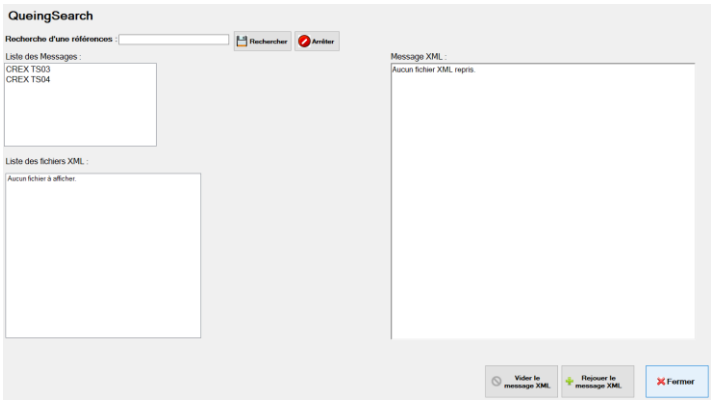
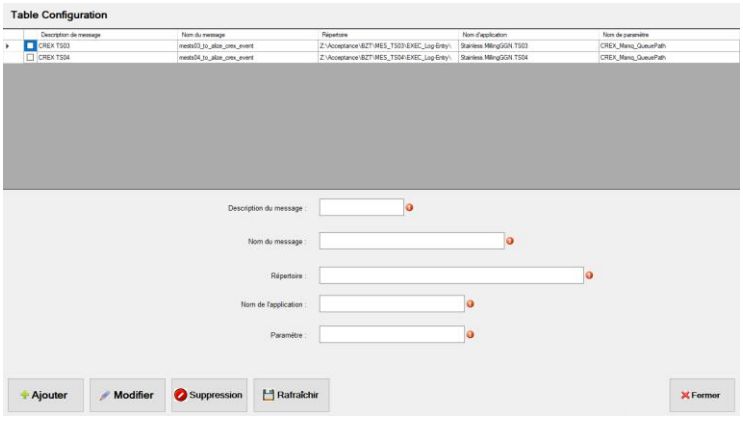
    - **SQL Server** : Le système de gestion de base de données utilisé est de type SQL Server.

    - **Interaction des composants :** L'outil en Visual Basic extraira, analysera et rejouera les messages XML depuis un dossier d'archivage vers une file MSMQueue. L'application interagira avec la base de données pour récupérer les paramètres des files d'attentes.

2.5 Analyse des données (modélisation, diagramme de classes, schéma relationnel)

Images	Actions / Correponsdance
	• Table ParameterQueing La **table ParameterQueing** est la table où sont consignée les informations des paramètres de l'application. • **message_name** : le nom du message ou de la file d'attente lié au paramètre. • **directory** : le chemin du répertoire associé au paramètre. • **application_name** : le nom de l'application utilisant ce paramètre. • **parameter_name** : le nom spécifique du paramètre.
	• Vue ParameterQueing La **vue ParameterQueing** simplifie l'accès aux données des paramètres pour les requêtes de lecture. Elle présente un sous-ensemble spécifique de colonnes de la table, offrant une interrogation optimisée sans exposer la structure complète de la table.
	• DataSet ParameterQueing L'application utilise le **DataSet ParameterQueing** comme un **cache de données en mémoire** pour gérer les configurations MSMQ. Ce DataSet permet de manipuler les données sans connexion constante à la base de données. Il est rempli et mis à jour par le **ParameterQueingTableAdapter**, qui assure la communication avec la base de données SQL Server pour récupérer et sauvegarder ces configurations.

2.6 IHM (Interfaces homme-machine)

Interfaces Homme-Machine	Fonctions des Interfaces Utilisateur
	**Formulaire QueingParameter :** • Zone de Recherche : o Un champ texte pour "**Recherche d'une référence**" où l'utilisateur peut taper son critère. o Un bouton "**Rechercher**" pour lancer la recherche. o Un bouton "**Arrêter**" pour interrompre une recherche en cours. • Panneaux de Résultats : o Une section "**Liste des Messages**" qui affiche les différents messages trouvés (par exemple, "CREX TS03", "CREX TS04"). o Une section "**Liste des fichiers XML**" qui listera les fichiers XML associés au message sélectionné dans la première liste. • Visualisation XML : o Une grande zone "**Message XML**" qui affichera le contenu détaillé du fichier XML sélectionné. Actuellement, elle indique "Aucun fichier XML repris." • Boutons d'Action : o "**Vider le message XML**" : pour effacer le contenu de la zone "Message XML". o "**Rejouer le message XML**" : pour relancer le traitement du message XML affiché. o "**Fermer**" : pour quitter l'application.
	**Formulaire "Table Configuration"** • Tableau de Configurations (Haut): Affiche une liste de configurations existantes avec des colonnes pour la description, le nom du message, le répertoire, le nom de l'application et le nom du paramètre. Permet la sélection multiple. • Détails de Configuration (Centre): Champs de saisie pour afficher, modifier ou créer une nouvelle configuration. Les champs correspondent aux colonnes du tableau ci-dessus. • Boutons d'Action (Bas): o **Ajouter**: Crée une nouvelle configuration. o **Modifier**: Met à jour la configuration sélectionnée. o **Suppression**: Supprime la ou les configurations sélectionnées. o **Rafraîchir**: Met à jour l'affichage de la table des configurations. o **Fermer**: Quitte le formulaire.

2.7 Conduite de projet : décomposition en tâches, structure équipes, planning (Gantt), durée

La période de stage pour la reprise des messages de type XML dans une Queue de type MSMQueue s'étend pendant toute la période du stage.

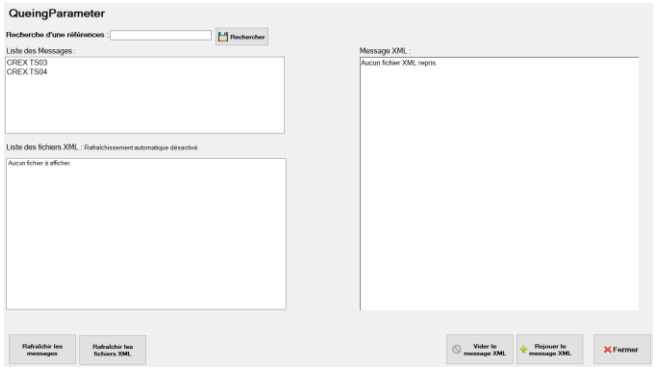
3 Développement

3.1 Réalisation des interfaces et programmes conformes aux spécifications fonctionnelles attendues

L'application charge les données de files d'attente depuis une procédure stockée SQL Server. Elle permet de rechercher des messages XML et d'afficher dynamiquement les fichiers XML associés. Un bouton "Rejouer le message XML" envoie le contenu XML sélectionné vers une file d'attente MSMQ.

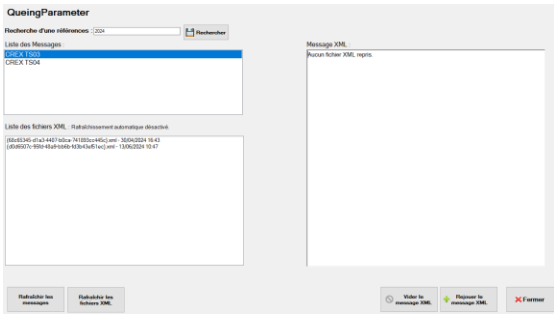
Interface utilisateur	Action / Extrait de code source
Chargement des données de la file d'attente des paramètres	• Procédure Stockée **"SELECT"**

```
USE [Insite_Coil]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO



CREATE PROCEDURE [MESITToolBox].[spc_SelectParameterQueing]
AS
        DECLARE @ERROR_NUMBER INT;

        SET @ERROR_NUMBER = 0


        BEGIN TRY
                SELECT *
                FROM MESITToolBox.V_ParameterQueing

        END TRY

        BEGIN CATCH

                DECLARE @ERROR_MESSAGE NVARCHAR(4000)
                DECLARE @ERROR_SEVERITY INT
                DECLARE @ERROR_STATE INT
                SELECT  @ERROR_NUMBER  =  ERROR_NUMBER(),
@ERROR_MESSAGE   =   ERROR_MESSAGE(),   @ERROR_SEVERITY   =
ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE()
                RAISERROR                (@ERROR_MESSAGE,
@ERROR_SEVERITY, @ERROR_STATE)

        END CATCH

GO
```



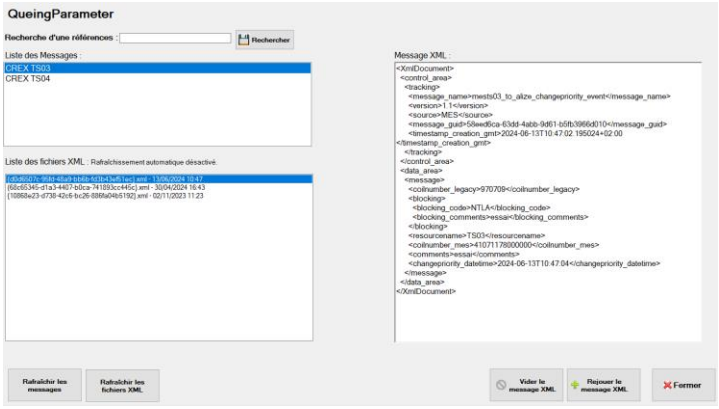
- **Chargement du formulaire**

```
    Private Sub QueingParameterForm_Load(sender As Object, e As EventArgs) Handles Me.Load
        ' 1. Charge les données principales du formulaire.
        LoadMessageParameters()

        ' 2. Configure le BackgroundWorker pour la recherche.
        searchWorker.WorkerReportsProgress = True
        searchWorker.WorkerSupportsCancellation = True

        ' 3. Active le bouton de recherche.
        btnSearch.Enabled = True

        ' 4. Configure et désactive le timer de rafraîchissement automatique.
        refreshTimer.Interval = 1000
        refreshTimer.Stop()
        lblTimer.Text = "Rafraîchissement automatique désactivé."
    End Sub
```



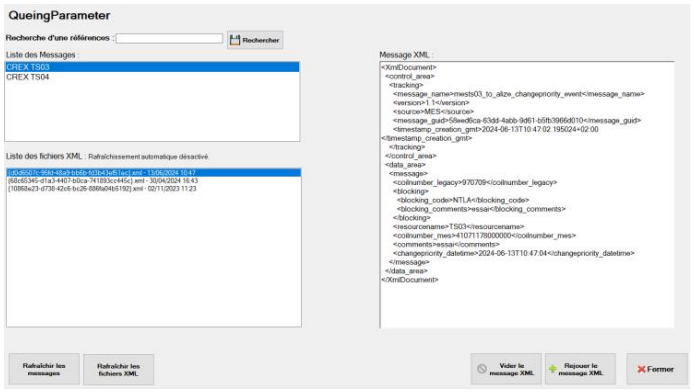
- **Bouton "Rechercher"** : lance la recherche

```
    Private Sub btnSearch_Click(sender As Object, e As EventArgs) Handles btnSearch.Click
        ' 1. Récupère le terme de recherche et initialise l'affichage.
        Dim searchTerm As String = txtSearch.Text.Trim().ToLower()
        lstFileXML.Items.Clear()
        rtbXmlPreview.Text = "Aucun fichier XML repris."

        ' 2. Valide le terme de recherche et détermine le chemin de recherche (défaut ou spécifique).
        Dim searchRootPath As String = xmlFolderPath ' Initialisation par défaut
        If String.IsNullOrEmpty(searchTerm) Then Return ' Sort si le terme est vide.
        If lstMessage.SelectedItem IsNot Nothing Then ' ... (logique de détermination du chemin réel si un message est sélectionné)

            ' 3. Vérifie l'accessibilité du dossier de recherche.
            If Not Directory.Exists(searchRootPath) Then
                lstFileXML.Items.Add("Dossier de recherche inaccessible.")
                Return
            End If

            ' 4. Prépare l'interface et lance la recherche en arrière-plan.
            btnSearch.Enabled = False
            searchWorker.RunWorkerAsync(New Object() {searchTerm, searchRootPath})
    End Sub
```



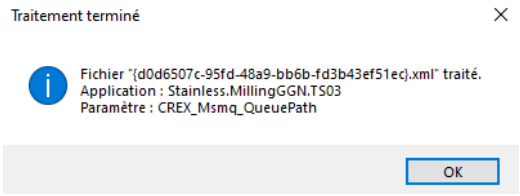
- Affichage de le liste des fichiers **“lstFileXML”**

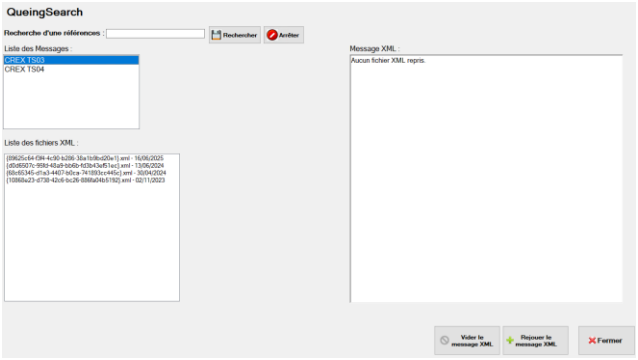
```
    Private Sub lstMessage_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstMessage.SelectedIndexChanged
        ' 1. Gère la dé-sélection ou la re-sélection du même élément.
        If lstMessage.SelectedIndex = lastSelectedIndexInLstMessage AndAlso lstMessage.SelectedIndex <> -1 Then
            ' Efface la sélection, vide l'affichage XML et réinitialise l'état des listes.
            lstMessage.ClearSelected()
            rtbXmlPreview.Text = "Aucun fichier XML repris."
            lastSelectedIndexInLstMessage = -1
            lstFileXML.Items.Clear()
            lstFileXML.Items.Add("Aucun fichier à afficher.")
            Return
        Else
            lastSelectedIndexInLstMessage = lstMessage.SelectedIndex ' Met à jour l'index de la dernière sélection.
        End If

        ' 2. Recharge les fichiers XML basés sur la nouvelle sélection (ou l'absence de sélection).
        LoadAndDisplayXmlFiles()
    End Sub
```

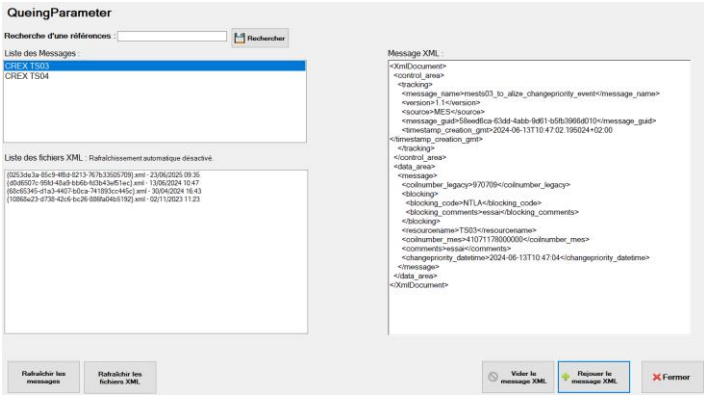


- Affichage du message XML dans **“rtbXmlPreview”**

```
Private Sub lstFileXML_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstFileXML.SelectedIndexChanged
        ' Gère la désélection ou la re-sélection du même élément.
        If lstFileXML.SelectedIndex = lastSelectedIndexInLstFileXML AndAlso lstFileXML.SelectedIndex <> -1 Then
            lstFileXML.ClearSelected()
            rtbXmlPreview.Text = "Aucun fichier XML repris."
            lastSelectedIndexInLstFileXML = -1
            Return
        Else
            lastSelectedIndexInLstFileXML = lstFileXML.SelectedIndex
        End If

        ' Valide l'élément sélectionné et détermine le chemin du fichier XML.
        If lstFileXML.SelectedItem Is Nothing OrElse Not TypeOf lstFileXML.SelectedItem Is XmlFileInfo Then Return
        Dim selectedXmlFileInfo As XmlFileInfo = CType(lstFileXML.SelectedItem, XmlFileInfo)
        Dim currentSelectedPathFromMessage As String = xmlFolderPath ' Défaut
        If lstMessage.SelectedItem IsNot Nothing Then ' ... (logique de chemin spécifique)
        End If
        Dim filePath As String = Path.Combine(currentSelectedPathFromMessage, selectedXmlFileInfo.FileName.Split({"-"}, StringSplitOptions.None)(0))

        ' Lit et affiche le contenu du fichier XML ou gère l'absence/l'erreur.
        If File.Exists(filePath) Then
            Try
                rtbXmlPreview.Text = File.ReadAllText(filePath)
            Catch ex As Exception
                rtbXmlPreview.Text = "Erreur lors de la lecture du fichier : " & ex.Message
            End Try
        Else
            rtbXmlPreview.Text = "Fichier introuvable à : " & filePath
        End If
    End Sub
```







- **Bouton "Rejouer le message XML" :** Création d'un fichier MSMQ rejouer vers la file d'attente

```
    Private Sub btnReplayToQueue_Click(sender As Object, e
As EventArgs) Handles btnReplayToQueue.Click

        '1. Affiche une confirmation de traitement du fichier
à l'utilisateur.
        MessageBox.Show("Fichier            """           &
selectedXmlFileInfo.FileName.Split({"                  -"},
StringSplitOptions.None)(0)     &     """     traité."     &
Environment.NewLine & _
                                    "Application   :   "   &
applicationName & Environment.NewLine & _
                                    "Paramètre    :    "    &
parameterName & Environment.NewLine, "Traitement terminé",
MessageBoxButtons.OK, MessageBoxIcon.Information)




        '2. Initialise et démarre le compte à rebours de
rafraîchissement visible à l'écran.
        timerCountdown = 12
        refreshTimer.Start()
        ' L'affichage du décompte se fait dans la méthode
RefreshTimer_Tick.
        ' ...
        timerCountdown -= 1
        lblTimer.Text   =   "Rafraîchissement   dans   "   &
timerCountdown.ToString() & "s..."
        ' ...




        '3. À la fin du compte à rebours, arrête le timer et
rafraîchit la liste des fichiers XML.
        If timerCountdown <= 0 Then
            refreshTimer.Stop()
            LoadAndDisplayXmlFiles()
            lblTimer.Text  =  "Rafraîchissement  automatique
désactivé."
        End If
    End Sub
```

Fermetture de l'application

- Bouton **"Fermer"**

```
Private Sub btnClose_Click(sender As Object, e As EventArgs)
Handles btnClose.Click
        Me.Close()
    End Sub
```

L'application **ConfigurationForm** est conçue pour gérer les paramètres de configuration des files d'attente. Elle offre des fonctionnalités de validation des champs de saisie, de chargement, d'ajout, de modification et de suppression de configurations via une interface utilisateur. Les données sont persistées dans une base de données, et l'application interagit avec une couche métier pour les opérations CRUD (Create, Read, Update, Delete).

Interface utilisateur	Action / Extrait de code source
Chargement des données de la file d'attente des paramètres	• Procédure Stockée **"SELECT"**

```
USE [Insite_Coil]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO



CREATE PROCEDURE [MESITToolBox].[spc_SelectParameterQueing]
AS
	DECLARE @ERROR_NUMBER INT;

	SET @ERROR_NUMBER = 0


	BEGIN TRY
		SELECT *
		FROM MESITToolBox.V_ParameterQueing

	END TRY

	BEGIN CATCH

		DECLARE @ERROR_MESSAGE NVARCHAR(4000)
		DECLARE @ERROR_SEVERITY INT
		DECLARE @ERROR_STATE INT
		SELECT	@ERROR_NUMBER	=	ERROR_NUMBER(),
@ERROR_MESSAGE	=	ERROR_MESSAGE(),	@ERROR_SEVERITY	=
ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE()
		RAISERROR	(@ERROR_MESSAGE,	@ERROR_SEVERITY,
@ERROR_STATE)

	END CATCH

GO
```

Ajout d'une nouvelle configuration de paramètre

- Procédure Stockée **“INSERT”**

```
USE [Insite_Coil]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [MESITToolBox].[spc_InsertParameterQueing]

    @description VARCHAR(100),
    @name VARCHAR(100),
    @directory VARCHAR(100),
         @application VARCHAR(100),
    @parameter VARCHAR(100)
AS
    DECLARE @QUERY NVARCHAR(1000);
    DECLARE @PARAMETERS NVARCHAR(500);

    BEGIN TRY
        SET @PARAMETERS = N'@parameterDescription VARCHAR(100),'
        SET @PARAMETERS = @PARAMETERS + N'@parameterName
VARCHAR(100),'
        SET @PARAMETERS = @PARAMETERS + N'@parameterDirectory
VARCHAR(100),'
                SET @PARAMETERS = @PARAMETERS +
N'@parameterApplication VARCHAR(100),'
        SET @PARAMETERS = @PARAMETERS + N'@parameterParameter
VARCHAR(100)';

        SET @QUERY = 'INSERT INTO
[MESITToolBox].T_ParameterQueing (message_description,
message_name, directory, application_name, parameter_name)
VALUES (@parameterDescription, @parameterName,
@parameterDirectory, @parameterApplication,
@parameterParameter)';

        PRINT @QUERY;

        EXEC sp_executesql @QUERY, @PARAMETERS,
@parameterDescription = @description, @parameterName = @name,
@parameterDirectory = @directory, @parameterApplication =
@application, @parameterParameter = @parameter;
    END TRY
    BEGIN CATCH
        DECLARE @ERROR_NUMBER INT;
        DECLARE @ERROR_MESSAGE NVARCHAR(4000);
        DECLARE @ERROR_SEVERITY INT;
        DECLARE @ERROR_STATE INT;

        SELECT @ERROR_NUMBER = ERROR_NUMBER(), @ERROR_MESSAGE =
ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(),
@ERROR_STATE = ERROR_STATE();
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY,
@ERROR_STATE);
    END CATCH;
GO
```

Mise à jour d'une configuration de paramètre existante

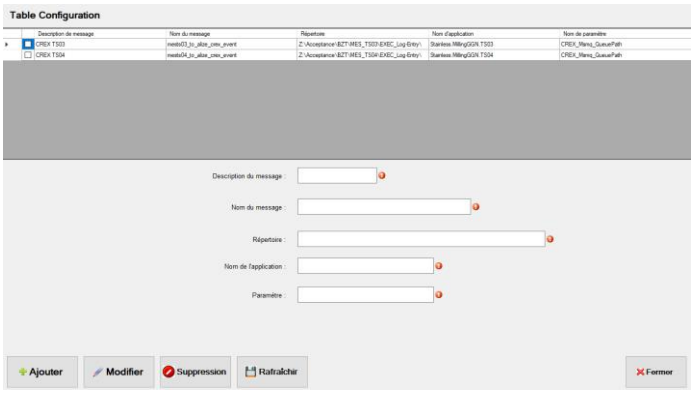
- Procédure Stockée **"UPDATE"**

```
USE [Insite_Coil]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [MESITToolBox].[spc_UpdateParameterQueing]
    @description VARCHAR(100),
    @name VARCHAR(100),
    @directory VARCHAR(100),
        @application VARCHAR(100),
    @parameter VARCHAR(100),
        @Id_Param INT
AS
    DECLARE @QUERY NVARCHAR(1000);
    DECLARE @PARAMETERS NVARCHAR(500);

    BEGIN TRY
        SET @PARAMETERS = N'@parameterDescription VARCHAR(100),'
        SET @PARAMETERS = @PARAMETERS + N'@parameterName VARCHAR(100),'
        SET @PARAMETERS = @PARAMETERS + N'@parameterDirectory VARCHAR(100),'
                SET @PARAMETERS = @PARAMETERS + N'@parameterApplication VARCHAR(100),'
        SET @PARAMETERS = @PARAMETERS + N'@parameterParameter VARCHAR(100),'
                SET @PARAMETERS = @PARAMETERS + N'@parameterId_Param INT';

        SET @QUERY = 'UPDATE [MESITToolBox].T_ParameterQueing SET message_description = @parameterDescription, message_name = @parameterName, directory = @parameterDirectory, application_name = @parameterApplication, parameter_name = @parameterParameter WHERE id = @parameterId_Param';

        PRINT @QUERY;

        EXEC sp_executesql @QUERY, @PARAMETERS, @parameterDescription = @description, @parameterName = @name, @parameterDirectory = @directory, @parameterApplication = @application, @parameterParameter = @parameter, @parameterId_Param = @Id_Param;
    END TRY
    BEGIN CATCH
        DECLARE @ERROR_NUMBER INT;
        DECLARE @ERROR_MESSAGE NVARCHAR(4000);
        DECLARE @ERROR_SEVERITY INT;
        DECLARE @ERROR_STATE INT;

        SELECT @ERROR_NUMBER = ERROR_NUMBER(), @ERROR_MESSAGE = ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE();
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE);
    END CATCH;
GO
```

Suppression de configurations de paramètres sélectionnées

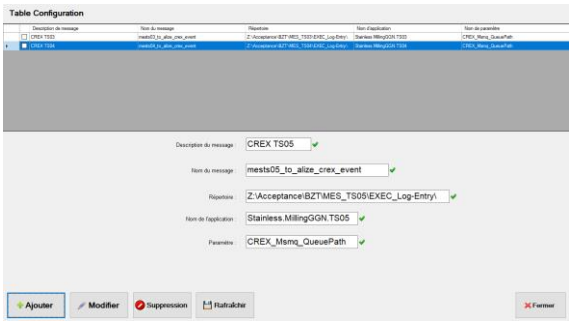
- Procédure Stockée **“DELETE”**

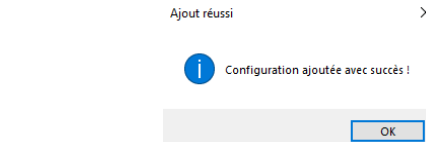
```
USE [Insite_Coil]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [MESITToolBox].[spc_DeleteParameterQueing]

	@Id_Param INT

AS
	DECLARE @QUERY NVARCHAR(1000);
	DECLARE @PARAMETERS NVARCHAR(500);

	BEGIN TRY

	SET @PARAMETERS = N'@parameterId_Param INT'

		SET	@QUERY	=	'DELETE	FROM [MESITToolBox].T_ParameterQueing WHERE id = @parameterId_Param';

		PRINT @QUERY

		EXEC	sp_executesql	@QUERY,	@PARAMETERS, @parameterId_Param = @Id_Param

	END TRY

	BEGIN CATCH

		DECLARE @ERROR_NUMBER INT
		DECLARE @ERROR_MESSAGE NVARCHAR(4000)
		DECLARE @ERROR_SEVERITY INT
		DECLARE @ERROR_STATE INT

		SELECT	@ERROR_NUMBER	=	ERROR_NUMBER(), @ERROR_MESSAGE	=	ERROR_MESSAGE(),	@ERROR_SEVERITY	=	ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE()
		RAISERROR	(@ERROR_MESSAGE,	@ERROR_SEVERITY, @ERROR_STATE)

	END CATCH
GO
```

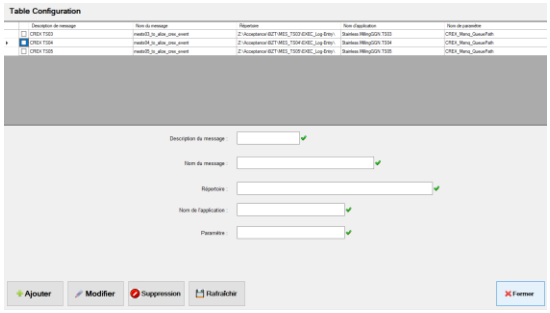


- **Chargement du formulaire** et Bouton **“Rafraîchir”**

```
	' Charge les paramètres de configuration existants dans la grille d'affichage du formulaire.
	LoadParametersIntoGrid()
	' Valide le champ de description du message au chargement du formulaire.
	DescValidation()
	' Valide le champ du nom du message au chargement du formulaire.
	MsgValidation()
	' Valide le champ du répertoire au chargement du formulaire.
	DirectoryValidation()
	' Valide le champ du nom de l'application au chargement du formulaire.
	AppliValidation()
	' Valide le champ du nom du paramètre au chargement du formulaire.
	ParamValidation()
```



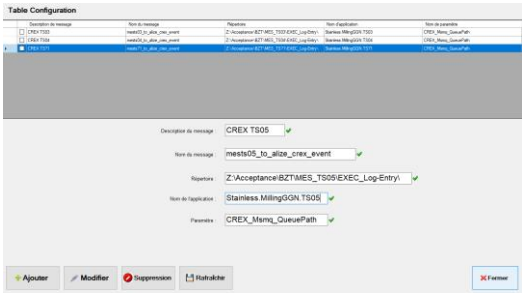


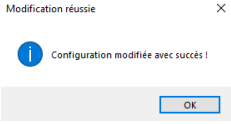


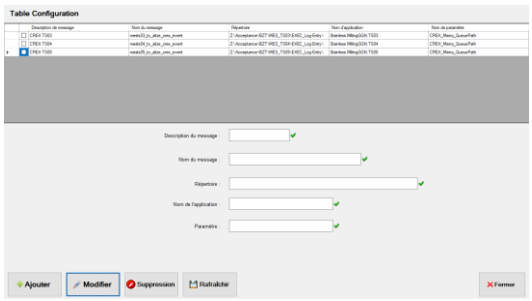
- Bouton **“Ajouter”**

```
        Dim msgdesc As String = txtDescMsg.Text
        Dim msgname As String = txtNameMsg.Text
        Dim directory As String = txtDirectory.Text
        Dim appli As String = txtAppli.Text
        Dim parameter As String = txtParam.Text
        If Not DescValidation() OrElse Not MsgValidation() OrElse
Not DirectoryValidation() OrElse Not AppliValidation() OrElse Not
ParamValidation() Then
            Me.DialogResult = DialogResult.None
            Return
        End If
            ' Ajout à la base de données
            Using            data            As            New
Stainless.MESITToolboxGGN.Bl.ParameterQueing(MetaData)
                data.AddParameterQueing(msgdesc,        msgname,
directory, appli, parameter)
            End Using


MessageBox.Show("Configuration ajoutée avec succès !", "Ajout
réussi", MessageBoxButtons.OK, MessageBoxIcon.Information)
```

```
        Finally
            LoadParametersIntoGrid()
            ClearInputFields()
        End Try
```



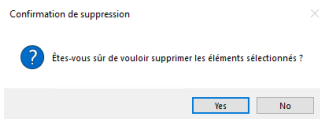


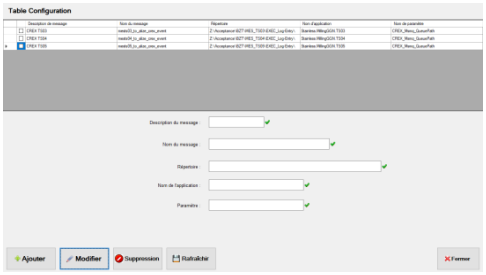


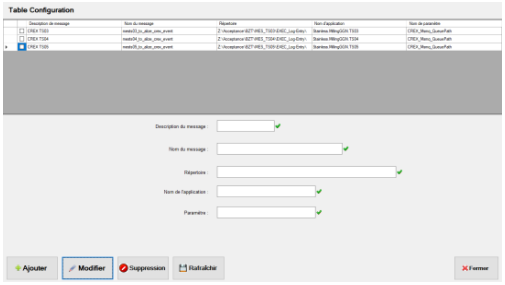
- Bouton **“Modifier”**

```
        Dim                    selectedRow                    As
Ent.DataSetParameterQueing.ParameterQueingRow = Nothing
        Dim idParam As Integer
        Dim msgdesc As String = txtDescMsg.Text
        Dim msgname As String = txtNameMsg.Text
        Dim directory As String = txtDirectory.Text
        Dim appli As String = txtAppli.Text
        Dim parameter As String = txtParam.Text

idParam = selectedRow.id

            Using            data            As            New
Stainless.MESITToolboxGGN.Bl.ParameterQueing(MetaData)
                data.UpdateParameterQueing(msgdesc,     msgname,
directory, appli, parameter, idParam)
            End Using


MessageBox.Show("Configuration    modifiée    avec    succès    !",
"Modification          réussie",          MessageBoxButtons.OK,
MessageBoxIcon.Information)
```

```
        Finally
            LoadParametersIntoGrid()
            ClearInputFields()
        End Try
```

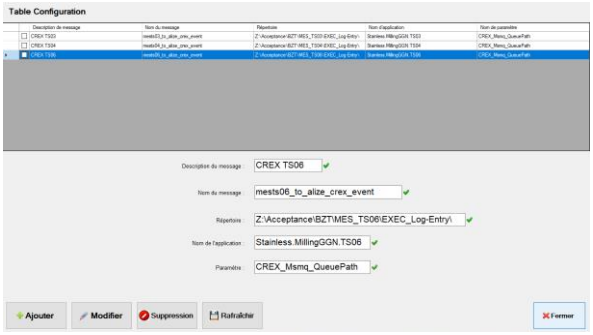






- Bouton **“Supprimer”**

```
result = MessageBox.Show("Êtes-vous sûr de vouloir supprimer les éléments sélectionnés ?", "Confirmation de suppression", MessageBoxButtons.YesNo, MessageBoxIcon.Question)



        Dim idParamSupp As New List(Of Integer)()
        Dim result As DialogResult
        Dim currentRow As Ent.DataSetParameterQueing.ParameterQueingRow

                For Each suppRow As DataGridViewRow In gvConfig.Rows
                    ' Pour chaque ligne de la grille, si la case à cocher est activée, son identifiant est ajouté à la liste de suppression.
                Next

                For Each selectedRow As DataGridViewRow In gvConfig.SelectedRows
                    ' Pour chaque ligne sélectionnée :
                    ' Si ce n'est pas une nouvelle ligne vide, son identifiant est extrait.
                    ' Cet identifiant est ensuite ajouté à la liste de suppression, uniquement s'il n'y est pas déjà.
                Next



        Finally
            LoadParametersIntoGrid()
            ClearInputFields()
        End Try
```



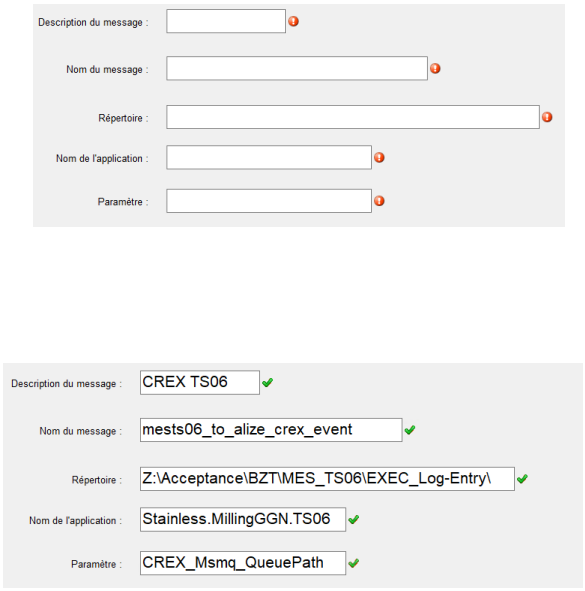
- Changement de sélection dans la grille **“gvParam”**

```
        Dim selectedRow As Ent.DataSetParameterQueing.ParameterQueingRow

        If gvConfig.SelectedRows.Count > 0 Then
            selectedRow = Me.SelectedRow
            PopulateInputFields(selectedRow)   '  Remplit les champs de saisie du formulaire avec les données de la ligne sélectionnée.
        Else
            ClearInputFields()
        End If
```

Fermeture de l’application

- Bouton **"Fermer"**

```
    Private Sub btnClose_Click(sender As Object, e As EventArgs) Handles btnClose.Click
        Me.Close()
    End Sub
```



- Validation des champs de saisie

```
' Les identifiants des lignes sélectionnées (non vides) sont ajoutés à la liste de suppression.

Private Sub Desc_Validated(sender As Object, e As EventArgs) Handles txtDescMsg.Validated
        DescValidation()
    End Sub

    Private Sub NameMsg_Validated(sender As Object, e As EventArgs) Handles txtNameMsg.Validated
        MsgValidation()
    End Sub

    Private Sub Directory_Validated(sender As Object, e As EventArgs) Handles txtDirectory.Validated
        DirectoryValidation()
    End Sub

    Private Sub Appli_Validated(sender As Object, e As EventArgs) Handles txtAppli.Validated
        AppliValidation()
    End Sub

    Private Sub Param_Validated(sender As Object, e As EventArgs) Handles txtParam.Validated
        ParamValidation()
    End Sub
```

3.3 Difficultés rencontrées (Bugs, Reste à faire)

- Difficulté initiale à comprendre le fonctionnement précis de MSMQueue dans le contexte Windows
- Reste à implémenter la logique complète de rejouage et de gestion des erreurs

Lors de la modification de la fonctionnalité de recherche par message, un problème a été rencontré concernant la gestion des répertoires. Contrairement à l'approche initiale qui cherchait à changer le répertoire parent de `C:` vers un emplacement réseau ou un autre lecteur (comme `Z:`), **il n'était en fait pas nécessaire de modifier le répertoire parent. Il suffisait simplement d'accéder à `\\fil-srv-log\`**. L'ancienne fonctionnalité, le bouton de recherche (`btnSearch`), détectait de manière incorrecte `Z:` comme son propre répertoire parent, ce qui créait un conflit alors qu'il n'aurait pas dû être affecté par ce changement.

En parallèle, suite à des manipulations réseau, un **problème d'accès à la base de données** a été identifié. Pour y remédier, il est nécessaire de **mettre à jour les fichiers XML lors de la relecture du MSMQ**.

5 Bilan

Ce stage, centré sur la **reprise de messages XML dans une queue de type MSMQ,** a représenté une expérience enrichissante et formatrice au sein du groupe Aperam à Gueugnon. L'objectif principal était de développer un **outil en Visual Basic .NET doté d'une interface utilisateur intuitive,** capable d'extraire, d'analyser et de rejouer des messages XML archivés vers une file d'attente MSMQ spécifique. Ce projet s'inscrivait dans le cadre de la modernisation

des outils numériques d'Aperam et visait à renforcer la **gestion des erreurs** et faciliter la reprise d'activité, en comblant une lacune existante où aucun outil ne permettait de relire ou rejouer ces messages.

Le développement de la solution a mobilisé des technologies clés telles que Visual Basic .NET et **Microsoft Visual Studio comme environnement de développement**. L'application interagit avec une base de données SQL Server pour gérer les configurations des files d'attente. Les messages sont au format XML, choisi pour sa capacité à structurer les données et à faciliter les échanges , et la **technologie MSMQ est au cœur de la communication asynchrone entre applications.**

Au cours de la phase de développement, plusieurs interfaces homme-machine ont été réalisées, notamment le formulaire `QueingParameter` pour la recherche et le rejouage des messages, et le formulaire `ConfigurationForm` pour la gestion des paramètres de configuration des files d'attente. L'application permet le **chargement des données depuis des procédures stockées SQL Server,** la recherche dynamique de fichiers XML, et le rejouage du contenu XML sélectionné vers une file d'attente MSMQ.

La solution est désormais fonctionnelle, permettant de répondre aux besoins identifiés, bien qu'elle ne soit pas encore déployée en production.

Malgré des avancées significatives, le projet a rencontré quelques difficultés. Une difficulté initiale a été la **compréhension précise du fonctionnement de MSMQ dans l'environnement Windows.** Un bug a également été identifié lors de la modification de la fonctionnalité de recherche par message, où le bouton de recherche (`btnSearch`) détecte incorrectement un répertoire parent, créant un conflit. Des problèmes d'accès à la base de données ont également été rencontrés suite à des manipulations réseau. Enfin, il reste à implémenter la logique complète de rejouage et de gestion des erreurs, et à mettre à jour les fichiers XML lors de la relecture du MSMQ.

Ce stage m'a permis **d'approfondir mes compétences en développement .NET,** de comprendre les enjeux des systèmes de messagerie asynchrone dans un environnement industriel, et de me familiariser avec la gestion des données via SQL Server. La confrontation aux difficultés techniques a renforcé ma capacité à résoudre des problèmes complexes et à adopter une approche méthodique. Ces apprentissages seront précieux pour ma future carrière professionnelle.