

GOUILA Bastien

TORDJMAN Ruben

GOUDARD--BELLISSENS Aloïs

IUT ANNECY

Département GEII

Année 2016

# Simulation d'une partie opérative

# **SOMMAIRE**

## **I. Introduction**

A. Présentation Générale (Global Presentation)

B. Cahier des charges (Specifications)

1. Global Objective
2. Material and human resources
3. Tasks splitting

## **II. Utilisation rapide**

## **III. Communication avec OPC**

A. Definition

B. Configuration du serveur

C. Classe OPC en C#

## **IV. Programme en C#**

A. Classe CCommunication

B. Classe CPontRoulant

C. Classe CInterface

## **V. Configuration de l'automate avec Unity Pro XL**

A. Configuration matérielle

B. Programme de tests

## **VI. Conclusion**

# **I. Introduction**

## **A. Présentation Générale (Global Presentation)**

For the fourth semester we have to realize a project from the end of January to the end of Mars. The purpose of the project is to create a simulator of the surface treatment platform located in room C163 of the IUT.

With the simulator the person who wants to work on the surface treatment platform can, in a first time, test his program before transferring it on the physical platform. This person could either be a student or a teacher.

This project brings together several fields we have studied for 2 years : Industrial Network, Automation and Object Programming.

The simulator programmed in C# language is communicating with a PLC (Programmable Logic Controller) thanks to a OPC Server.



## **B. Cahier des charges (Specifications)**

### **1. Global Objective**

Creation of a surface treatment platform simulator in C# language with a communication with a PLC (Programmable Logic Controller).

*Proposed solution :*

Use the supervisor Client/Server software OPC to link the C# simulator with a Unity program in the PLC.

## 2. Material and human resources

### *Equipment :*

Multiple PLCs : MODICON TSX PREMIUM from Schneider Automation SA.

Physical Surface treatment.

PCs of the IUT : Virtual Machines with Visual Studio 2008, Unity Pro XL and OPC.

### *Team :*

This project is composed of three 2nd year students :

TORDJMAN Ruben, GOUDARD-BELLISENS Aloïs and GOUILA Bastien.

### *Other Specifications :*

- To create the simulator, we have to use the C# language.
- Project Management software : Mindview.
- Communication between the simulator and the PLC must be done with OPC.

## 3. Tasks splitting

We have separated our project in six parts :

- *Communication* : configure the OPC Server and the OPC Client (Simulator).
- *Interface* : prepare an environment where the user can interact.
- *PLC Configuration* : material configurations for the different PLCs (the one for the physical platform the others for the simulator).
- *Tests programs* : create small programs to test the proper functioning.
- *Simulator* : exchange information with the PLC to simulate the physical platform.
- *Pooling* : Gather all the functional parts and assemble them.

The next chart shows how the project was managed :

<u>Tasks\Person</u>	Ruben TORDJMAN	Aloïs GOUDARD-BELLISENS	Bastien GOUILA
<i>Communication</i>		X	X
<i>Interface</i>	X		X
<i>PLC Configuration</i>		X	X
<i>Tests programs</i>		X	X
<i>Simulator</i>	X		
<i>Pooling</i>	X		

## **II. Utilisation rapide**

### **Configuration de l'automate avec UNITY PRO XL :**

Pour commencer, nous avons mis à disposition des projets Unity préconçus :

**configuration\_finale\_simulateur.zef** dans le dossier

**Projet\_Automate\_Simulateur\Configuration\_Finale** pour le simulateur.

**configuration\_finale\_physique.zef** dans le dossier

**Projet\_Automate\_Physique\Configuration\_Finale** pour la partie physique.

Ces projets Unity permettent à l'utilisateur de commencer votre programme Unity avec un environnement déjà défini (Configuration matérielle, réseau...). Il ne reste plus qu'à programmer les parties en ST, SFC et Ladder dans la section MAST.

Si vous voulez passer du simulateur à la partie physique ou l'inverse, il vous suffit de changer d'environnement et d'importer et d'exporter vos sections MAST.

**!! Attention :** Sélectionnez bien l'adresse de votre automate dans "Automate" puis "Définir l'adresse". Vérifiez avec le plan de la salle C163 dans les annexes.

Si vous souhaitez éditer votre propre environnement, vous pouvez vous inspirer de la partie *III.A Configuration Matérielle* à une condition, **ne modifiez (ou supprimez) pas les certaines variables de base situées dans la section Variables et Instances**. Vous pouvez en ajouter ,cependant, **les variables des capteurs et des actionneurs ne doivent pas changer**. Elles sont essentielles pour le simulateur. Voir variables en annexes.

Sauvegardez votre travail Unity comme un fichier de type .stu. (Séparez les noms par des " \_ ").

Générez votre projet ,puis, transférez le sur l'automate et mettez le en mode RUN.

#### **Utilisation du simulateur :**

- Lancez le simulateur.
- Dans la fenêtre du simulateur où vous pouvez choisir votre automate, appuyez sur le bouton "Renseigner fichier Unity" puis sélectionnez votre fichier Unity que vous voulez simuler.
- Le simulateur va alors prendre votre fichier et le renseigner au serveur OPC. Connectez vous à l'automate. Votre simulateur est prêt.

#### **Précautions à prendre lors de l'utilisation du simulateur :**

- 1) **OPC ne fonctionne, et donc le simulateur, que si la version de votre projet** (qui s'incrémente après chaque génération) **est la même sur l'automate et dans le fichier que vous renseignez au simulateur.**

Si jamais vous obtenez un message d'erreur "*Unable to add this item*", fermez le simulateur, retransférez votre projet sur l'automate et relancez le simulateur.

- 2) **Si d'autres problèmes surviennent avec le serveur OPC**, il se peut que celui-ci n'ait pas été éteint correctement. Fermez le simulateur, sélectionnez ce logo dans la barre des tâches, faites clic-droit puis "*Exit*". Relancez alors le simulateur.

- 3) Le **simulateur commence avec l'arrêt d'urgence enclenché**, pensez à prendre cela en compte dans votre programme automate.

### III. Communication avec OPC

#### A. Définition

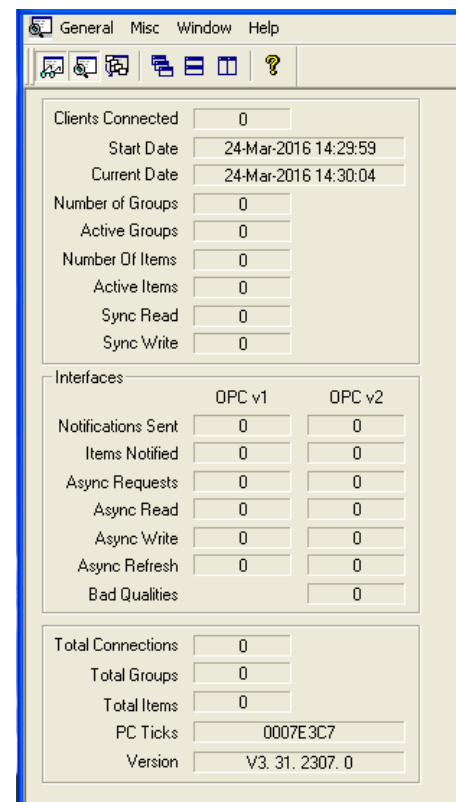
OPC (Object linking and Embedding for Process Control) est un ensemble d'application ayant pour but la supervision d'un ou plusieurs automates. Dans le cadre de notre PC, il nous permet de lier un programme sous UnityProXL (logiciel développé par Schneider) à un programme développé sous Visual Studio 8.

Pour ce faire nous avons utilisé la librairie SAOFSGRPXLib. Cette librairie contient des méthodes permettant de démarrer un serveur OPC, de l'initialiser et de lui ajouter des items.

#### B. Configuration du serveur

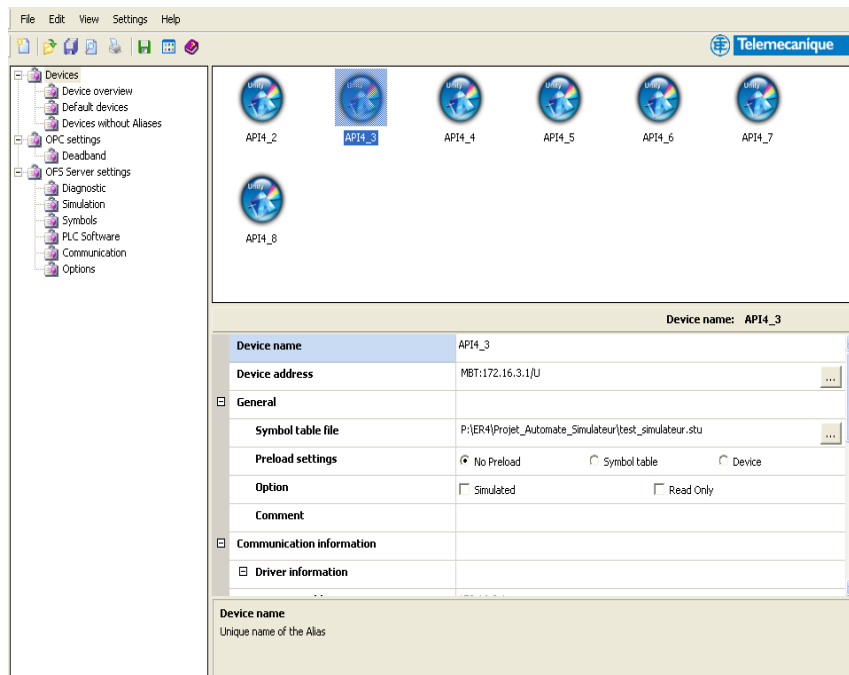
Un serveur OPC (voir figure) permet d'héberger des clients que l'on peut relier à des automates grâce un système de groupe. Pour cela, il faut créer un Alias de l'API auquel on souhaite se connecter (API4\_3 par exemple). Pour notre projet, le pont roulant est relié à l'automate numéro 10 de la salle C163. On peut donc y accéder depuis n'importe quel autre automate de la salle. Il suffit pour cela d'utiliser le bon Alias, c'est-à-dire l'Alias correspondant à l'automate auquel notre PC est relié. Tous les Alias sont présents sur la configuration OPC.

Cette dernière nécessite également un fichier appelé fichier de symbole. Ce fichier est un fichier de type STU (Unity).



Il servira à permettre l'importation des variables API sur le serveur OPC afin que l'on puisse les lire ou écrire dessus via le client. En effet, ils seront qualifiés d' « Item » par le serveur et l'on pourra les ajouter à un groupe d'item créé sur le client.

Grâce à ces items, le lien entre OPC et Unity est établi. Il reste donc à créer le lien entre OPC et Visual Studio 8. Cela passe par l'utilisation de la librairie SAOFSGRPXLib citée précédemment.



## C. Classe OPC en C#

La librairie SAOFSGRPXLib dispose de plusieurs méthodes dont les prototypes sont sur la figure ci-contre.

La méthode « Init » permet l'initialisation du client OPC en renseignant seulement le nom du serveur. En procédant ainsi, on force l'ouverture du serveur.

```
void SAOFSGroupX.Init (string ProgID, string RemoteServerName);
void SAOFSGroupX.AddItem(string AccessPath, string ItemID);
void SAOFSGroupX.Start(int RequestedUpdateRate);
void SAOFSGroupX.Read(void);
object GetValue(string Name);
void SAOFSGroupX.Write(string Name, bool Value);
void SAOFSGroupX.Stop(void);
```

Pour ce qui est de la méthode « AddItem », elle nous permet d'ajouter un des items d'un alias au client ouvert précédemment. Pour cela, cette méthode doit connaître le nom de l'alias ainsi que le nom de l'item en question grâce à des variables de type `String`. Il faut savoir que les variables de l'automate se voient ajouter un « ! » devant leur nom dans les services OPC.



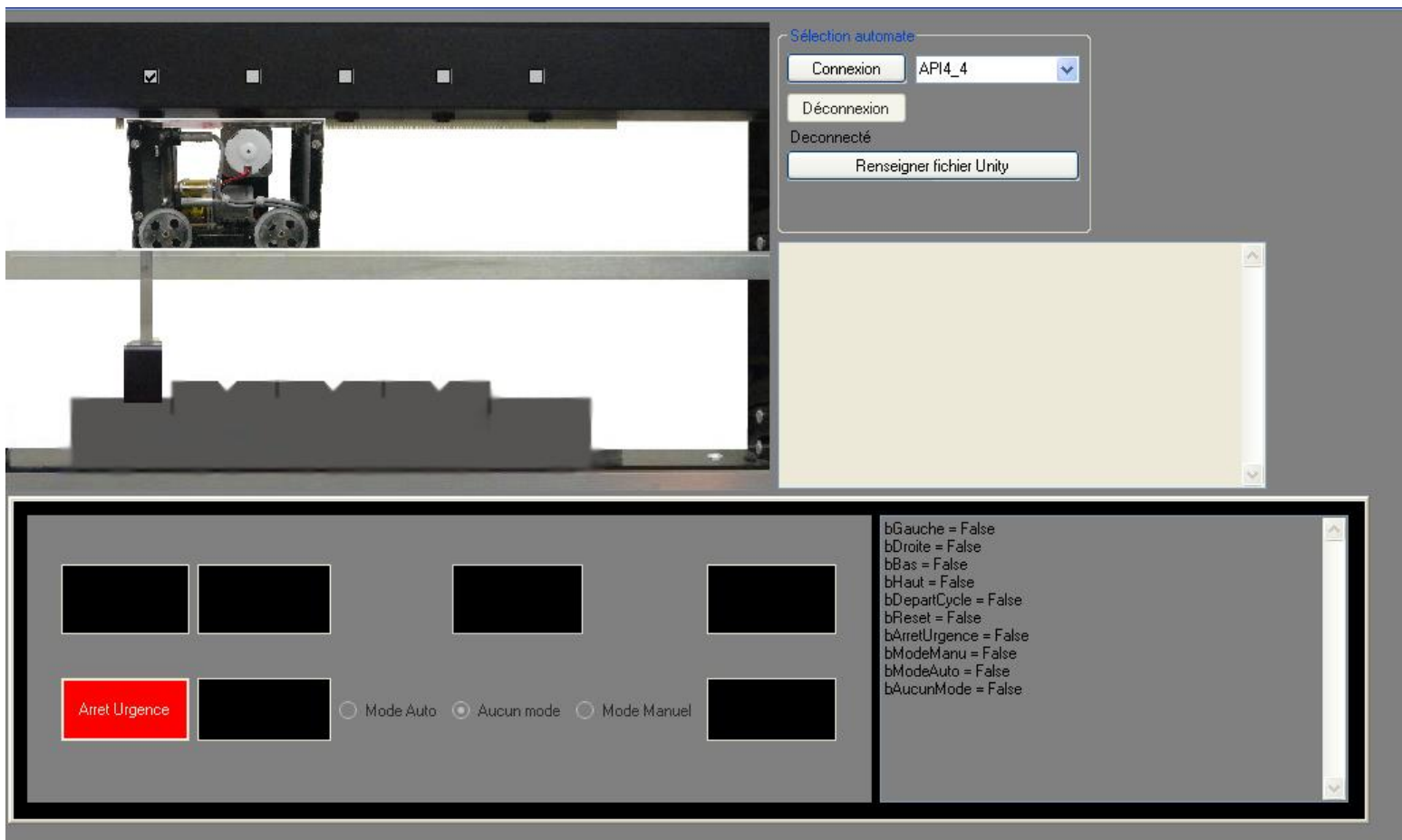
Par exemple, pour ajouter la variable « Exemple1 » de l'alias « API4\_1 », on renseignera la méthode de la façon suivante : AddItem (« API4\_1 » + « ! » + « Exemple1 »).

La méthode « Read » permet de lire les valeurs de tous les items présents sur le client. Si l'on souhaite récupérer leur valeur il faudra utiliser la fonction « GetValue » qui prend pour paramètres d'entrée des variables [String](#). Son exemple d'utilisation est le même que celui de la méthode AddItem.

Si l'on souhaite modifier l'état d'une variable, soit écrire, on devra se servir de la méthode « Write ». Cette méthode écrit la valeur du booléen reçu sur l'item portant le nom de la variable [String](#).

Les fonctions « Start » et « Stop » permettent respectivement de démarrer et d'arrêter le serveur OPC. La méthode « Start » prend une variable de type [Integer](#) en entrée. Il s'agit de la période de scrutations des variables en millisecondes. La méthode « Stop » quant à elle, prend la variable [String](#) contenant le nom du serveur à arrêter.

## IV. Programme en C#



### A. Classe CCommunication

Comme expliqué précédemment, la communication entre l'ordinateur et l'automate se fait via un dialogue client/serveur OPC. La librairie "SAOFSGRPXLib" offre de nombreuses fonctionnalités, et c'est la classe CCommunication qui y fait appel, au travers d'un objet de classe SAOFSGroupXClass.

Le constructeur de cette classe commence par détruire tout processus en rapport avec OFS, comme des restes de serveur inutilisés, dans le but de démarrer sur une base propre. Suit le démarrage du serveur OPC avec la méthode InitServeurOPC(sDirectoryStu) qui prend en paramètre le répertoire du fichier .stu de l'utilisateur pour éditer le registre.

Le constructeur se charge ensuite d'initialiser la connexion au serveur OPC, en ajoutant notamment toutes les variables utilisées dans l'automate. Pour ce faire, la méthode GetVarNameAdr(sXMLPath) récupère toutes les variables du projet, contenues dans le fichier XML généré par Unity, et retourne un tableau à deux dimensions, contenant les variables ainsi que leur adresse virtuelle. C'est la première colonne de ce tableau qui est utilisée au moment d'ajouter les variables au serveur (méthode AddItem()).

La souscription à l'évènement “\_ISAOPCGrpXCallback\_OnDataChangeEventHandler” permet d’être averti quand la valeur d’une des variables de l’automate change, et donc de mettre à jour l’état de toutes les variables côté programme C#.

Enfin, la classe CCommunication offre une méthode d’écriture de données sur l’API, ainsi qu’une méthode pour se déconnecter proprement.

## B. Classe CInterface

La partie opérative que nous avons simulé possède une console de contrôle, qui est représentée dans le simulateur par la classe CInterface. Cette classe possède les mêmes boutons que la console et permet, selon le programme chargé dans l’automate, de contrôler la partie ou opérative, ou de lancer des cycles automatiques.

Cette classe ayant été développée en dehors du reste du projet, dans une fenêtre indépendante du reste du simulateur, nous avons dû l’intégrer en utilisant les Multiple-Documents Interface, en tant qu’enfant de la fenêtre de CPontRoulant.

## C. Classe CPontRoulant

Les deux classes précédemment présentées sont liées par la classe CPontRoulant, car c’est elle qui crée leurs instances. En plus de faire le lien entre l’interface et le gestionnaire des communications PC/Automate, CPontRoulant gère la connexion et la déconnexion des différents automates, ainsi que la partie graphique du simulateur. Le choix de l’automate auquel se connecter se fait via un menu déroulant, et un OpenFileDialog permet à l’utilisateur de renseigner le répertoire de son fichier .stu.

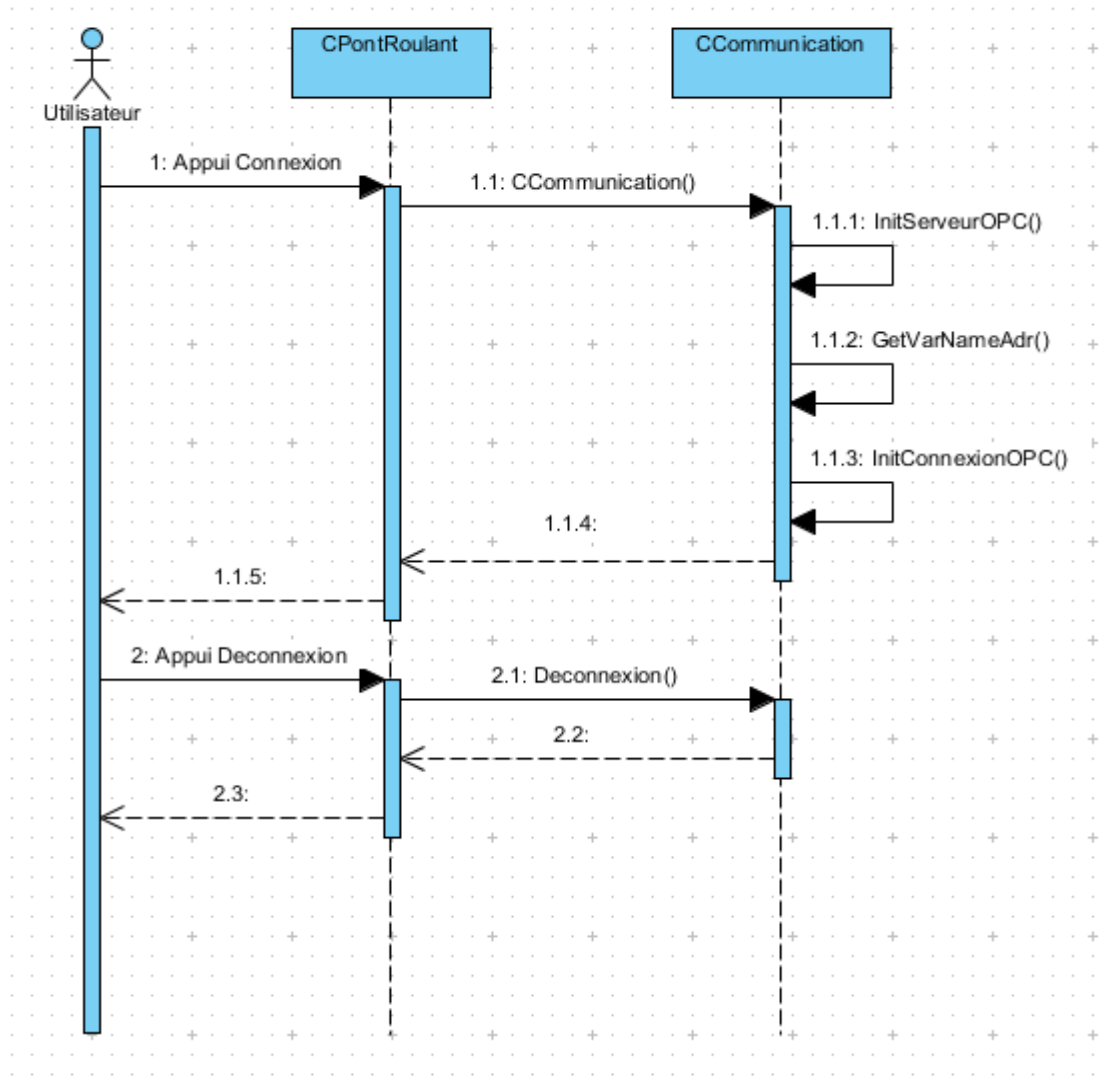
Après quelques essais utilisant des formes simples comme des rectangles et des cercles pour représenter le pont roulant, nous avons finalement opté pour l’utilisation de PictureBox dans lesquelles sont placées des photos retravaillées du pont roulant. Des CheckBox non cliquables sont utilisées pour symboliser les capteurs des cuves, en se validant et dévalidant selon la position du chariot.

Périodiquement, et grâce à un Timer, les valeurs des actionneurs de la classe CCommunication sont récupérées et mettent à jour les données du simulateur( méthode MAJEntrees()). Ces données sont ensuite traitées pour faire bouger les PicturesBox et donc animer le simulateur. Le chariot du pont roulant ayant pu activer un capteur après avoir bougé, ou le bras pouvant être arrivé en butée, il est important d’actualiser les sorties (méthode MAJSorties()).

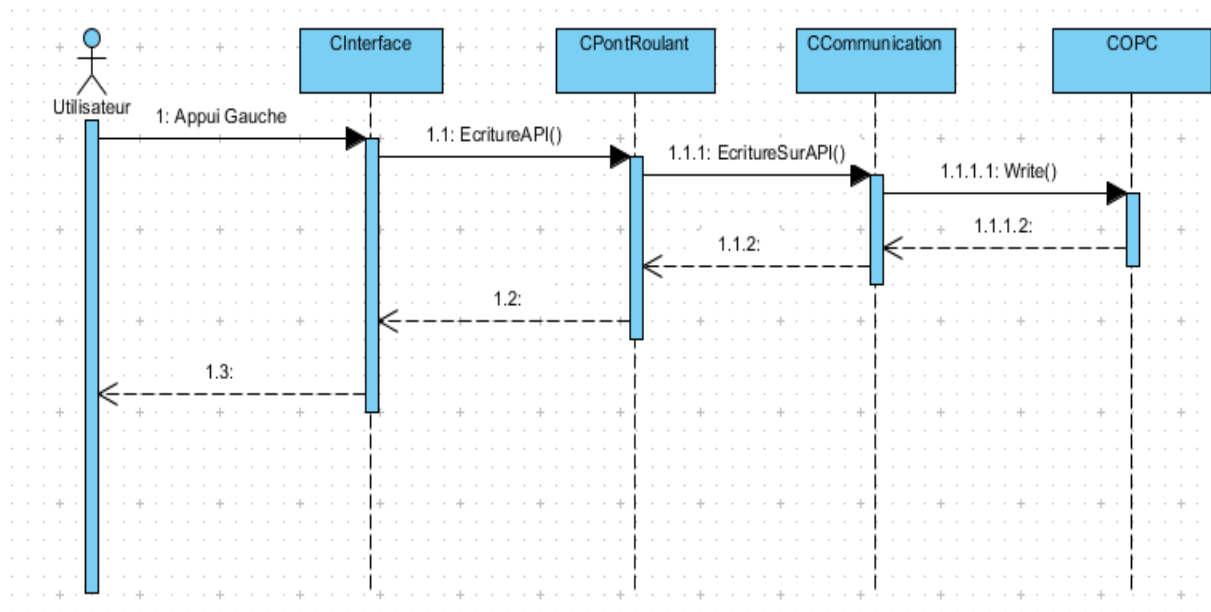
CPontRoulant intègre un dictionnaire qui associe à chaque bouton de l’interface une variable de l’automate, ce qui simplifie le transfert des données, car il n’y a plus besoin de devoir écrire des tests entiers pour savoir quel variable il faut modifier en fonction du bouton appuyé. Par exemple, en appuyant sur le bouton “Gauche”, le dictionnaire retourne

“iebDeplacementGauchePont”, qui est directement utilisé par la méthode Write() de la classe OPC, qui, on le rappelle, prend en paramètre le nom de la variable à modifier, ainsi que sa valeur booléenne.

- Séquence lors de la connexion et la déconnexion à l'automate.



- Exemple de séquence d'un appui sur le bouton Gauche de l'interface, en mode manuel.

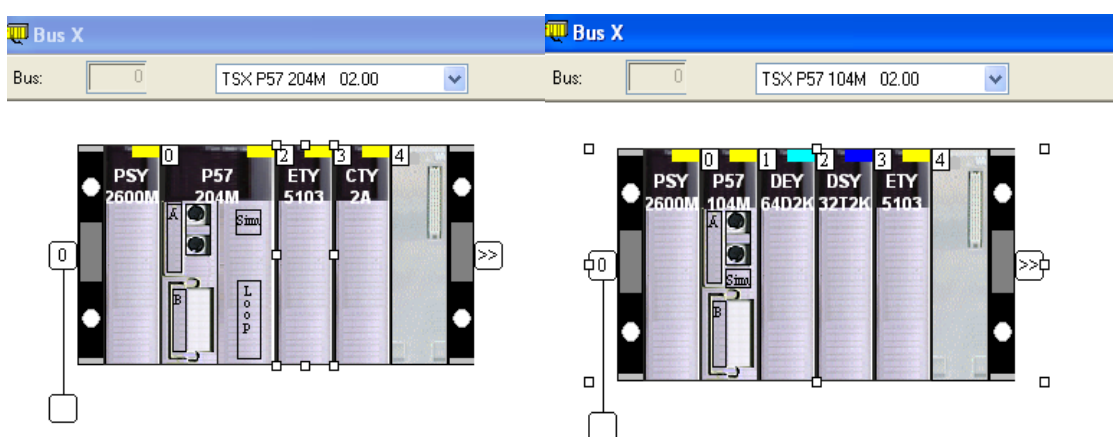


## V. Configuration de l'automate avec Unity Pro XL

### A. Configuration matérielle

#### Configuration des automates :

Les automates étant différents pour la partie physique ou le simulateur, il faut deux configurations matérielles :



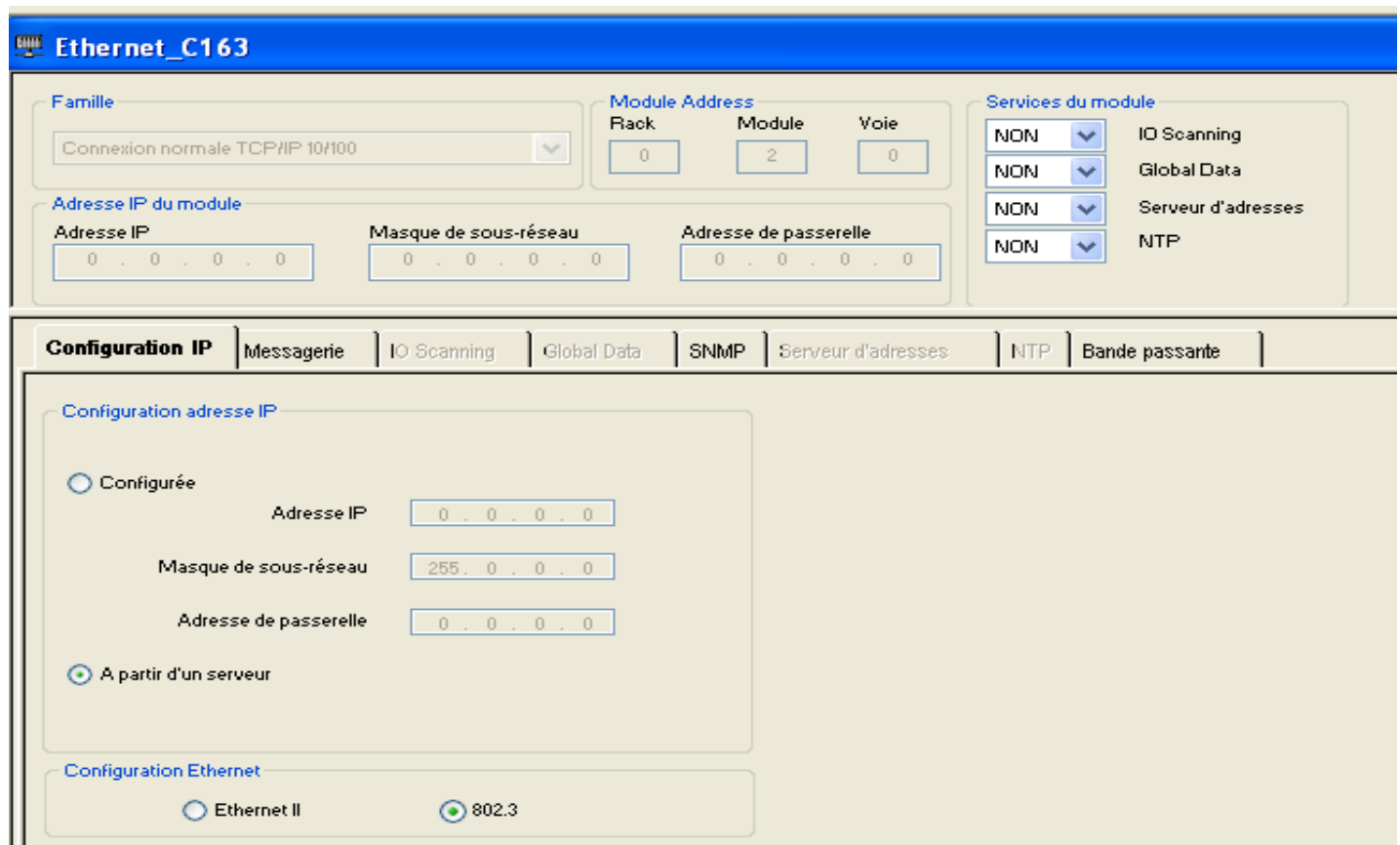
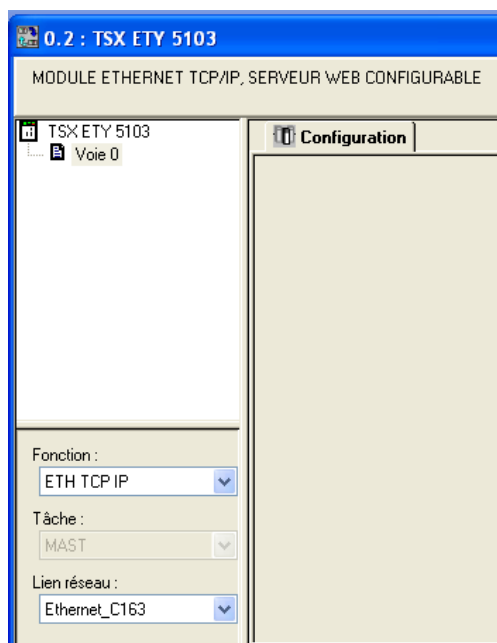
Mode\Module	TSX Premium	Processeur	Ethernet	Entrées	Sorties	Compteur
Physique	PSY 2600M	P57 104M	ETY 5103	DEY 64D2K	DSY 32T2K	X
Simulateur	PSY 2600M	P57 204M	ETY 5103	X	X	CTY2A

Pour les processeurs, il faut sélectionner les versions les plus antérieures pour éviter des problèmes de versions.

#### Configuration réseau :

Il faut maintenant relier l'automate au réseau de la salle. Pour cela, il faut créer un nouveau dans la section “Réseau”.

Il faut ensuite sélectionner la Configuration Ethernet avec “802.3” et la Configuration IP “A partir d’un serveur”, ainsi, l'automate se connectera au réseau à l'aide d'un serveur DNS qui lui donnera un alias et une adresse IP conforme au réseau de la salle.

Il faut ensuite relier ce nouveau réseau au module Ethernet de l'automate.

Il faut alors sélectionner le module Ethernet et placer dans la Voie0 le réseau que l'on vient de créer.

### Création des variables et de la table d'animation :

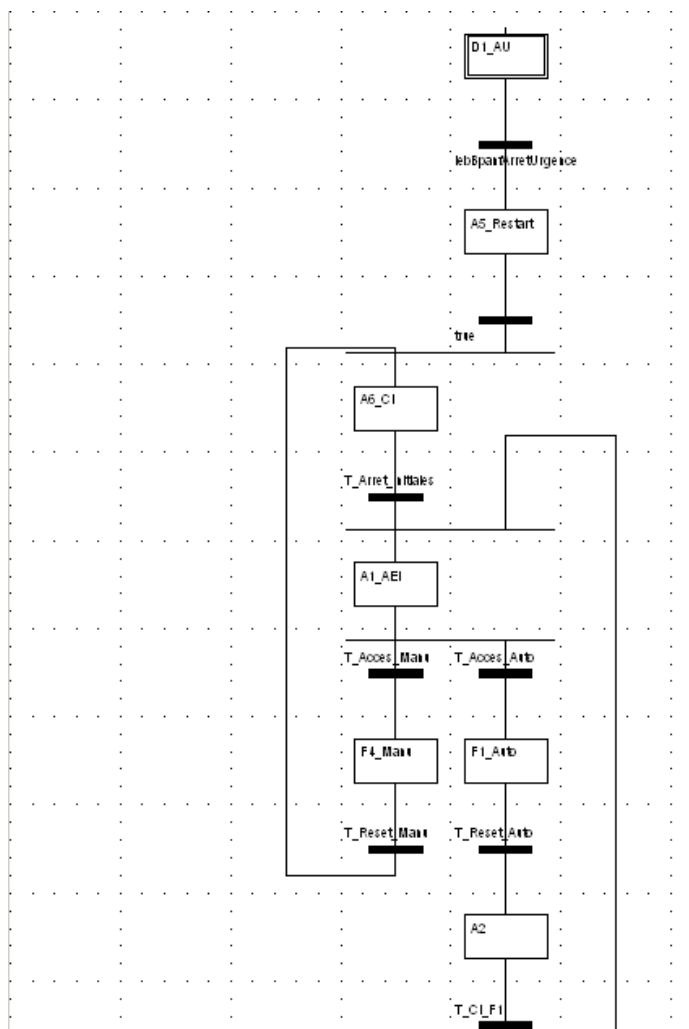
Dans la section “Variables et instances” les données utilisées dans le projet peuvent être rentrées. Certaines variables sont essentielles pour le bon fonctionnement du simulateur. Des variables peuvent être ajoutés librement.

Pour voir la liste des variables se référer aux annexes ou alors les importer dans cette section avec le fichier des variables mise à disposition.

Il faut ensuite créer une table d'animation pour que le simulateur puisse lire et écrire sur ces données. Il faut alors aller dans la section table d'animation et en créer une où l'on renseignera toutes les variables auxquelles accède l'automate.

## B. Programme de tests

Pour tester notre simulateur nous avons créé deux projets Unity simples : un pour le simulateur et un pour la partie physique.



Ces projets comporte un GEMMA composé de 7 modes :

*D1 :Arret d'Urgence*

*A5 : Reprise apres conditions initiales (non géré ici)*

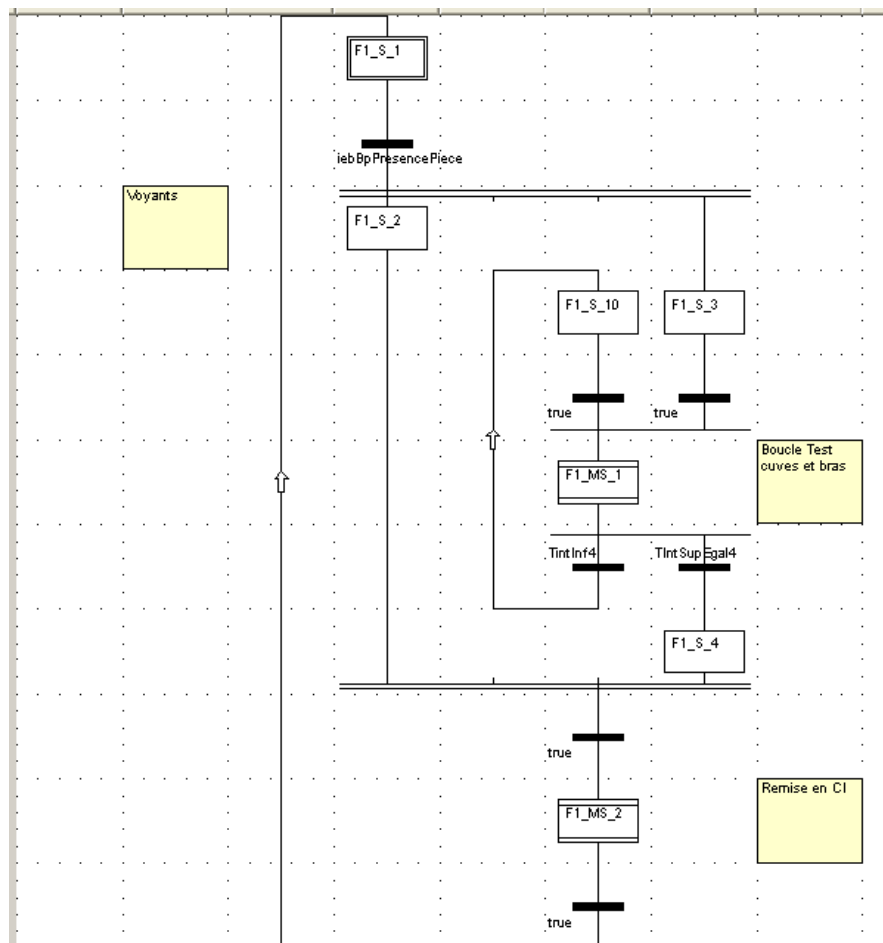
*A6 : Remise en conditions initiales. Cuve en bas et en position à la cuve d'entrée.*



*A1 : Attente de choix du mode*

*F4 : Mode Manuel*

*F1/A2 : Mode Auto.* Pour le test, ce fonctionnement est simple. Le convoyeur, prend la cuve en entrée après un appui sur *DepartCycle* et la dépose dans chaque cuve pendant 1 seconde avant de revenir en conditions initiales la cuve étant laissée dans la cuve de sortie. Un nouveau peut-être répété après un nouvel appui sur départ cycle. (Voir schéma suivant)



Pour voir la totalité du projet, se référer aux fichiers sources.

## **VI. Conclusion**

### **Schéma fonctionnel**



### **Bilan financier :**

The project « Surface treatment platform simulator » had been made by 4 students of Electrical Engineering and Industrial Computing who worked 80 hours with a professor. The writing of the report and the development of the interface had needed 5 extra hours.

In order to calculate the cost of our project we will take a base of 50€ per hour. Then we have a total cost, including the salaries and the working environment, about 12 750€ ( $50 \times 80h \times 3 \text{ students} + 5h \times 50 \times 3$ ).

The working environment is composed by the 2 PLC (Programmable Logic Controller), the software licences (OPC, UnityProXL, Visual Studio 8), and the loads.

### **Améliorations possibles :**

Le projet est fonctionnel et permet à n'importe qui d'utiliser le simulateur correctement et simplement. Cependant, si des personnes veulent l'améliorer, nous vous proposons quelques pistes :

- Nous n'avons pas eu le temps représenter la pièce sur le simulateur, elle peut être ajoutée.
- Lorsque l'on se trompe d'API, il se peut que le simulateur plante. Il faut alors arrêter le simulateur et le relancer. Nous n'avons pas eu le temps de réparer ce problème.
- Il y a dans la salle C262 une cellule Festo utilisée en AUTO2 pour les projets. Le problème est qu'il y a un seul automate pour toute la salle et il faut souvent attendre longtemps avant de pouvoir utiliser l'automate. Sur la base de ce simulateur, on

pourrait en créer un autre pour cette cellule et ainsi permettre aux étudiants de 1e année de pouvoir tester leurs programmes rapidement et sans risquer d'endommager matériellement la partie opérative.

- Nous n'avons pas géré un mode qui permettrait d'informer l'utilisateur qu'il aurait endommagé la machine physique s'il force sur les actionneurs. Il pourrait être ajouté.

### **Gestion du projet :**

Pour cette partie nous avons mis à votre disposition un fichier Mindview.

GOUILA Bastien

IUT Annecy

TORDJMAN Ruben

Département GEII

GOUDARD--BELLISSENS Aloïs

Année 2016

## **Résumé :**

Il n'est pas rare qu'un oubli de sécurité ou qu'une erreur de code mène la partie opérative d'un automate à s'abîmer, ou même casser une de ses pièces. Il serait intéressant de pouvoir tester son programme sans craindre d'abîmer quoi que ce soit; c'est exactement ce que permet un simulateur.

Notre projet a pour but de simuler le plus fidèlement possible la partie opérative d'un automate pont roulant, en utilisant le langage C# sous Visual Studio, ainsi qu'une couche OPC, gérant les communications entre PC et Automate. N'importe quel utilisateur ayant les connaissances suffisantes serait donc en mesure de tester son programme UnityProXL sur simulateur avant de l'envoyer sur la partie physique. On voit bien l'avantage qu'offre un simulateur en termes de sécurité pour le matériel, mais un simulateur s'avère aussi très utile dans le cadre des cours, quand une seule machine est disponible pour un grand nombre d'étudiants; les élèves pourront effectivement tester leurs programmes simultanément.

## **Mots clés :**

Programmation Orientée Objet, C#, Visual Studio, Automatisation, UnityProXL, OPC, Réseau, Pont Roulant, Simulateur.