Automatisme pour la robotique

TP 2

v1.0

IUT d'Annecy, 9 rue de l'Arc en Ciel, 74940 Annecy

Configuration du réseau de la baie du robot

Préambule

Pour rappel, l'objectif de ce module est de réaliser l'intégration d'un robot dans une cellule automatisée. Le premier TP a permis de configurer le projet et de réaliser la communication avec l'automate commandant le convoyeur.

Dans ce TP, nous allons réaliser la communication entre l'automate et le contrôleur du robot.

Objectifs:

- Configurer la communication avec la baie CS9.
- Configurer la communication avec le contrôleur M221.

Les systèmes de production modernes intègrent de plus en plus de robots, et différentes structures de commande sont disponibles pour répondre à cette évolution. Le choix de la structure dépend non seulement des compétences de l'intégrateur, mais aussi de l'environnement dans lequel le robot opère. Les contrôleurs de robots actuels, qu'ils soient collaboratifs ou non, proposent des interfaces numériques et analogiques via des cartes additionnelles. Ces contrôleurs supportent aussi diverses technologies de communication, allant des réseaux de terrain traditionnels comme CanOpen ou Profibus DP, aux réseaux Ethernet industriels déterministes tels que EtherCAT, Ethernet/IP, PowerLink ou encore Profinet.

Dans ces configurations, les contrôleurs de robots peuvent être configurés aussi bien comme maîtres que comme esclaves. Dans le cadre de ce TP, nous allons utiliser un automate programmable pour piloter un robot, ce qui nous permet de l'intégrer dans notre cellule robotisée comme un objet industriel connecté. Le contrôleur du robot conserve sa fonction principale de « motion », en gérant les déplacements du robot. Toutefois, le séquencement de ces mouvements sera dicté par l'automate programmable.

La solution UniVALplc développée par Stäubli s'inscrit parfaitement dans cette logique. Ce TP, ainsi que les suivants, se concentrent sur l'intégration d'UniVALplc dans un réseau Ethernet/IP pour permettre la communication entre l'automate et le contrôleur de robot. L'objectif est d'utiliser les blocs fonctionnels pour commander les mouvements du robot et gérer les échanges de données avec les équipements périphériques, tels que la baie CS9 et le contrôleur M221.

1 Présentation de la cellule automatisée

1.1 Matériel

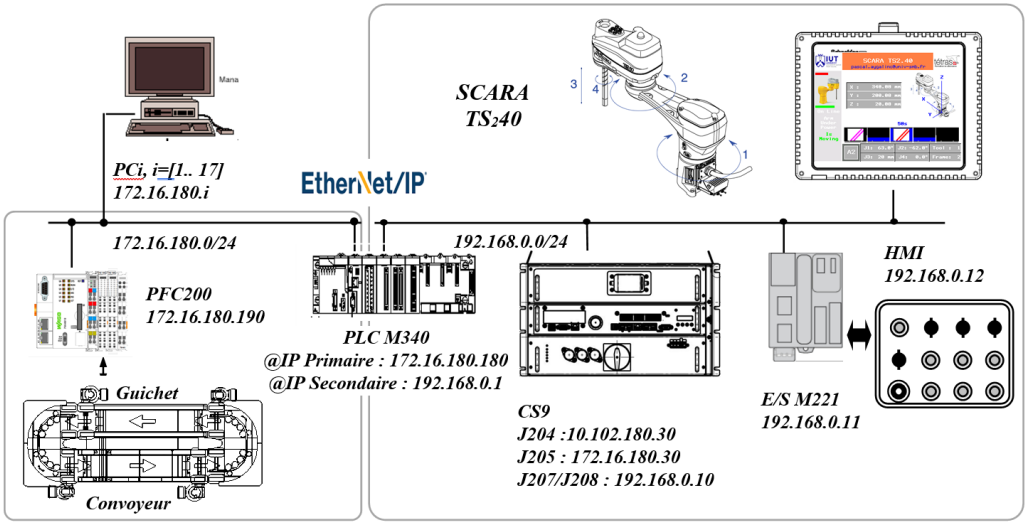

FIGURE 1 – Cellule robotisée

Pour réaliser ce TP, la cellule robotisée, représentée en Figure 1 est composée des éléments suivants :

- **Robot Stäubli Scara TS240** (4 axes, collaboratif) avec son contrôleur **CS9**, équipé d'un pendentif MCP et d'un sélecteur de modes de marche WMS9.
- **Convoyeur TS2plus** de Bosch RexRoth, sur lequel circulent des palettes transportant des produits.
- **Contrôleur PFC200** de WAGO, qui gère partiellement le convoyeur et se comporte comme un îlot d'E/S accessible via Ethernet/IP pour la commande du guichet.
- **Contrôleur M221** de Schneider, utilisé pour la gestion des E/S déportées et la communication avec l'interface HMI via le protocole Modbus-TCP.
- **Pupitre opérateur HMI** permettant de visualiser et superviser les actions du robot et du convoyeur.
- **Automate Schneider M340**, équipé de deux coupleurs réseaux *BMX NOC 0401.2*, chacun permettant, respectivement :
  - ◇ Une communication de type **Ethernet** avec les stations de développement (PC) et le contrôleur PFC200. *Adresse du réseau : 172.16.180.0/24*
  - ◇ Une communication de type **Ethernet/IP** avec la baie CS9 et le contrôleur M221. *Adresse du réseau : 192.168.0.0/24*

2 UniVALplc

2.1 Architecture UniVALplc pour le TP

A propos de UniVALplc

UniVALplc est une solution logicielle développée par Stäubli Robotics pour intégrer les robots de la gamme Stäubli dans des systèmes de production automatisés. Cette solution permet de commander les mouvements du robot et de gérer les échanges de données avec les équipements périphériques.

Architecture Client/Serveur UniVALplc

Serveur : Installé sur le contrôleur CS9 du robot. Il répondra aux commandes en provenance de l'automate M340 via le réseau Ethernet/IP.

- Les commandes sont interprétées pour générer des mouvements du robot.
- Les commandes peuvent faire référence à des objets internes du contrôleur CS9 (points, outils, frames, etc.)

Client : Bibliothèque de blocs fonctionnels fournie par le fabricant de l'automate **M340**.

2.2 Présentation du fonctionnement

Les blocs fonctionnels utilisés pour interagir avec le robot se répartissent en deux types :

- Les blocs spécifiques aux contrôleurs Stäubli, identifiables par le préfixe `VAL_`, permettent d'accéder aux fonctionnalités propres à UniVALplc.
- Les blocs conformes à la norme *plcOpen* (préfixe `MC_`), sont utilisés pour commander les mouvements standards, comme la gestion des axes et des positions.

Configuration du serveur

Pour que l'intégration soit réussie, il est nécessaire de respecter l'orientation des données de l'automate client. Dans notre cas, comme il s'agit d'un API M340 de Schneider, l'option *Little Endian* a été configurée lors de l'installation du serveur UniVALplc sur le contrôleur CS9. De plus, pour que les commandes envoyées par l'API soient acceptées, le sélecteur de modes de marche WMS9 du robot doit être positionné sur « Commande à distance ».

2.3 Programmation de l'API

Un objet clé utilisé dans la communication est l'instance du type `T_StaeubliRobot`, qui représente le robot et son contrôleur. Cet objet comprend trois champs principaux :

- `Status` : contient des informations sur l'état actuel du robot et de son contrôleur.
- `Command` : permet de transmettre des commandes au robot.
- `CommInterface` : gère l'interface de communication entre le contrôleur CS9 et l'automate.

Le processus de base à chaque cycle de l'API consiste en trois étapes :

1. Lire l'état du robot à partir d'une mémoire image (T->O) actualisée périodiquement par le réseau Ethernet/IP, en utilisant le bloc fonctionnel `VAL_ReadAxesGroup`.
2. Utiliser les blocs fonctionnels UniVALplc pour commander les mouvements du robot en se basant sur les informations reçues.
3. Écrire les commandes dans la mémoire image (O->T) à l'aide du bloc `VAL_WriteAxesGroup`, afin de les transmettre au contrôleur CS9 via Ethernet/IP.

Ce principe s'applique également pour l'accès aux E/S du contrôleur M221, où la lecture et l'écriture des données se font à travers des services d'assemblage sous Ethernet/IP.

3 Partie TD

3.1 Cartographie du Réseau Ethernet/IP

Lors du premier TP, la configuration du second coupleur BMX NOC 0401.2 (`NOC_ETHIP_ROBOT`) a été réalisée. Son image mémoire commence à l'adresse `%MW64` et comporte 128 mots en entrée ainsi que 128 mots en sortie. Les données produites et consommées par les deux équipements connectés (baie CS9 et contrôleur M221) sont récapitulées dans le tableau ci-dessous :

Équipement	Entrée (T->O) - Données produites	Sortie (O->T) - Données consommées
CS9 (Serveur UniVALplc)	148 octets	124 octets
M221 (Îlot E/S, HMI)	4 octets	40 octets

TABLE 1 – Données échangées sur le réseau Ethernet/IP

Préparation 1 : Cartographie de l'espace mémoire %MW

Question 1 À partir des informations du tableau et de l'organisation mémoire du coupleur `BMX NOC 0401.2` (cf. TP n°1), établir la cartographie complète de l'espace mémoire `%MW` de l'automate.

3.1.1 Déclaration des Objets

Le bloc fonctionnel `VAL_ReadAxesGroup` nécessite que les données produites par le serveur UniVALplc soient de type `T_FromRobot`, afin de les convertir en un objet `T_StaeubliRobot`. De même, pour le bloc `VAL_WriteAxesGroup`, les données consommées doivent être de type `T_ToRobot`.

Préparation 2 : Déclaration des objets `T_FromRobot` et `T_ToRobot`

Question 2 Utilisez les techniques de chevauchement et d'alias pour déclarer les objets suivants :
- `fromRobotScara` de type `T_FromRobot` (pour les données produites par la baie CS9).
- `toRobotScara` de type `T_ToRobot` (pour les données consommées par la baie CS9).

Pour le contrôleur M221, les blocs fonctionnels `FB_ReadHmi` et `FB_WriteHmi` seront utilisés pour gérer les échanges de données avec l'interface HMI.

Préparation 3 : Déclaration des objets `TFromHmi` et `TToHmi`

Question 3 Déclarez les objets suivants :
- `fromHmiScara` de type `TFromHmi` (pour les données produites par le contrôleur M221).
- `toHmiScara` de type `TToHmi` (pour les données consommées par le contrôleur M221).

4 Partie TP - Manipulations

Attention

Pour ne pas avoir à recommencer en cas de problème, sauvegarder régulièrement le projet.

Vérification 1 : Pré-requis

Étape 1 Vérifier le fonctionnement des manipulations du TP précédent

Configuration 1 : Options du projet

Étape 2 Dans `Outils->Options du projet`, configurer les options suivantes :

- Autoriser le chevauchement d'adresses :
  - `Général` ->Gestion des messages de génération : le chevauchement d'adresses ne génère aucun message.
- Autorisation des tableaux dynamiques :
  - `Variables` -> `Autoriser les tableaux dynamiques (ANY_ARRAY_XXX)` : cocher la case.

4.1 Configuration du réseau du Robot

Dans le TP précédent, nous avons configuré le premier coupleur `BMX NOC 0401.2` pour le réseau du guichet (`172.16.180.0/24`). Nous allons maintenant configurer le second coupleur pour le réseau `NOC_ETHIP_ROBOT` (`192.168.0.0/24`).

Configuration 2 : Ajout des équipements sur le réseau `NOC_ETHIP_ROBOT`

Étape 3 Vérifier la configuration de l'adresse IP

- Ouvrir le `Navigateur de DTM` (`Outils->Navigateur de DTM` puis double-clic sur le réseau).
- Rubrique `TCP/IP` : vérifier que l'adresse IP est bien `192.168.0.1/24`.

Étape 4 Ajout des équipements (pour chaque équipement : **Clique droit -> Ajouter**)

- Ajouter la baie CS9
  Type : `uniVALplc v4.6 - CS9 Adapter`
  Gestion des noms de `DTMs` : cs9Scara
- Ajouter le contrôleur M221
  Type : `TM221_for_Scara`
  Gestion des noms de `DTMs` : m221Scara

Après avoir configuré les interfaces IP des coupleurs `BMX NOC 0401.2`, il est essentiel de respecter l'ordre d'ajout des équipements sur le réseau `NOC_ETHIP_ROBOT`, afin de respecter le plan d'adressage mémoire défini précédemment.

Configuration 3 : Configuration des adresses des équipements

Étape 5 Configurer les adresses des équipements

1. Double-cliquer sur le coupleur réseau `NOC_ETHIP_ROBOT`.
2. Pour chacun des appareils ajoutés, configurer les adresses IP suivantes (*Onglet Paramétrage de l'adresse*) :
   - `[003] cs9Scara` : `192.168.0.10`
   - `[004] m221Scara` : `192.168.0.11`

Configuration 4 : Configuration des équipements

Étape 6 Configurer le cs9Scara

- Sur le `Navigateur de DTM`, double-cliquer sur le `cs9Scara`.
- Rubrique `Exclusive Owner` : Paramétrer la périodicité de la tâche Mast.

Étape 7 Configurer le m221Scara

- Sur le `Navigateur de DTM`, double-cliquer sur le `m221Scara`.
- Supprimer le service `Write To 150` et le remplacer par un service `Read From 100 / Write To 150`.
- Vérifier que ce nouveau service offre 4 octets en entrées et 40 octets en sortie, avec une lecture/écriture périodique de 30 ms

Vérification 2 : Vérification de l'espace mémoire

Étape 8 Sauvegarder, Vérifier et Compiler votre application

Étape 9 Dans l'éditeur de données : Vérifier les zones mémoires allouées

- ☐ `cs9Scara_IN : T_cs9Scara_IN AT %MW80`
- ☐ `cs9Scara_OUT : T_cs9Scara_OUT AT %MW208`
- ☐ `m221Scara_IN : T_m221Scara_IN AT %MW154`
- ☐ `m221Scara_OUT : T_m221Scara_OUT AT %MW270`

4.2 Mise en oeuvre de la communication

Dans cette section, nous allons créer les sous-programmes en charge de la communication avec la baie CS9. On propose, pour cela, d'ajouter deux sous-programmes à notre projet avec les noms et rôles suivants :

1. Une section `ReadEthIpRobot`, **première section à être exécutée**, elle aura pour rôle :
   (a) Lecture des données du robot via le bloc fonctionnel `VAL_ReadAxesGroup`
   (b) Lecture des informations depuis l'IHM via le bloc fonctionnel `BF_ReadHmi`
2. Une section `WriteEthIpRobot`, **dernière section à être exécutée**, elle aura pour rôle :
   (c) Écriture des informations vers l'IHM via le bloc fonctionnel `BF_WriteHmi`
   (d) Écriture des données du robot via le bloc fonctionnel `VAL_WriteAxesGroup`

Pour cela, ce TP suivra les étapes suivantes :

1. Importation des blocs fonctionnels nécessaires (HMI et CS9) dans le projet
2. Déclaration des variables nécessaires à la communication
3. Création des sections `ReadEthIpRobot` et `WriteEthIpRobot`
   (a) Réservation des variables
   (b) Instanciation des blocs fonctionnels
   (c) Appel des blocs de lecture et d'écriture dans les sections respectives.
4. Vérification du bon fonctionnement de la communication

4.2.1 Communication avec la baie CS9

Les blocs fonctionnels `VAL_ReadAxesGroup` et `VAL_WriteAxesGroup` sont fournis par la bibliothèque `UnivalPlc`. Puisque l'ajout d'une bibliothèque de types nécessite des droits d'administrateur, cela a été fait en amont du TP.

Importation des Fichiers EDS et de la Bibliothèque de Types

Ces manipulations ont été faites en amont du TP. Cet encart est à titre informatif.

1. Exécuter le programme de mise à jour de la bibliothèque de types sous *Control Expert* en mode administrateur.
2. Importer le fichier `FAMILY.DCS` pour ajouter la bibliothèque UniVALplc à l'environnement de développement.

Les fichiers et bibliothèques nécessaires sont disponible dans le dossier `Scara/uniVALplc/`

Manipulation 1 : Déclaration des variables nécessaires à la communication

Afin de rassembler les données provenant du bloc de communication avec le robot, on propose de déclarer une structure de données `T_CtrlStatusGroup`.

```
TYPE TCtrlStatusGroup
    STRUCT
        m_xEnable : BOOL ;
        m_xBusy : BOOL ;
        m_xDone : BOOL ;
        m_xError : BOOL ;
        m_udiErrorID : UDINT ;
        m_xTimeOut : BOOL ;
    END_STRUCT
END_TYPE
```

Étape 10 Définir ce type de données dans le dossier `Types données dérivés` du projet.

Manipulation 2 : Mise en oeuvre de la communication

Étape 11 Créer les deux sous-programmes décrits en début de cette sous-section et les ordonner afin qu'ils soient exécutés dans l'ordre adéquate.

Étape 12 Réserver les variables du tableau suivant. Pour certaines variables, l'adresse de lecture est à fixer selon la préparation (cf. 3.1.1).

Nom	Type	Adresse	Commentaire
ctrlStatusGroupRead	TCtrlStatusGroup		*État de la lecture*
ctrlStatusGroupWrite	TCtrlStatusGroup		*État de l'écriture*
fromRobotScara	T_FromRobot	*à fixer*	*Données du robot*
toRobotScara	T_ToRobot	*à fixer*	*Commandes vers le robot*
staeubliRobotScara	T_StaeubliRobot		*Structure de données du robot*

Étape 13 Instancier les blocs fonctionnels `VAL_ReadAxesGroup`, `VAL_WriteAxesGroup`, `BF_ReadHmi` et `BF_WriteHmi` dans le dossier `Instances FB dérivé`.

Étape 14 Appeler les blocs fonctionnels dans les sections `ReadEthIpRobot` et `WriteEthIpRobot` en respectant l'ordre de lecture et d'écriture.

- On conditionnera l'exécution des blocs au bon fonctionnement de la communication. Pour cela, prendre en compte le bit de santé `NOC_ETHIP_ROBOT_IN.HEALTH_BITS_IN[0].0`.

Vérification 3 : Fonctionnement de la communication

Étape 15 Créer une table d'animation nommée `scara` afin de visualiser les données suivantes :

- `NOC_ETHIP_ROBOT_IN.HEALTH_BITS_IN[0].0`
- `ctrlStatusGroupRead`
- `ctrlStatusGroupWrite`
- `staeubliRobotScara`

Étape 16 Vérifier, compiler puis transférer le programme.

Étape 17 Vérifier que les données échangées entre l'IHM et le robot sont correctes.

☐ Bit de santé du coupleur
☐ Absence d'erreur dans les variables `ctrlStatusGroupRead` et `ctrlStatusGroupWrite`
☐ Conformité des données du robot `staeubliRobotScara`

4.2.2 Communication avec le contrôleur M221

Les blocs fonctionnels `BF_ReadHmi` et `BF_WriteHmi` ont été créés par l'équipe enseignante et doivent être importés dans le projet.

Manipulation 3 : Importation des blocs fonctionnels HMI

Étape 18 **Clique-droit->Importer** sur le dossier `Type FB dérivés` dans l'arborescence du projet.

Étape 19 Sélectionner les fichiers `FB_ReadHmi.xdb` et `FB_WriteHmi.xdb` dans le dossier `U:\Documents\BUT\GEII\ModulesS5\Automatisme_Pour_Robotique\Scara`

Étape 20 Si l'application vous le demande, **Garder tout** puis valider.

Manipulation 4 : Déclaration des variables nécessaires à la communication

Étape 21 En suivant les mêmes étapes que pour la communication avec la baie CS9, réaliser la communication avec le contrôleur M221.

- L'appel du bloc `BF_ReadHmi` se fera **après** le bloc `VAL_ReadAxesGroup`.
- L'appel du bloc `BF_WriteHmi` se fera **avant** le bloc `VAL_WriteAxesGroup`.
- On utilisera le bit de santé `NOC_ETHIP_ROBOT_IN.HEALTH_BITS_IN[0].1` ainsi que le précédent pour conditionner l'exécution des blocs.

Vérification 4 : Communication avec l'IHM

Étape 22 Compiler, transférer puis vérifier le bon fonctionnement.

Étape 23 Faire valider par l'enseignant.

4.3 Mise en oeuvre du préhenseur

Cahier des charges : Test du préhenseur

- Le préhenseur sera activé pour saisir une pièce pendant 1 seconde lorsqu'une palette pleine est en poste.

Le système de préhension est réalisé à l'aide d'un « venturi » et d'une ventouse piloté par un distributeur **bistable**. Deux signaux électriques sont donc nécessaires pour le commander un pour saisir et un pour déposer.

Le robot de notre cellule ne disposant pas de cette fonctionnalité par défaut, elle a été réalisée en externe sur le connecteur J212 de la baie CS9. Par ailleurs, en l'absence de capteurs, on utilisera des durées *enveloppes* pour déterminer quand la saisie ou la dépose est effective.

Action	Nom	Connecteur	Durée enveloppe
Saisir	fastout0	J212-4/J212-9	T#300ms
Déposer	fastout1	J212-1/J212-5	T#1s

On propose de commander le préhenseur à l'aide d'une variable structurée de typé `T_Prehenseur` qui contiendra les ordres de saisie ou de dépose ainsi que les durées d'enveloppe associées. **Cette variable sera écrite par notre programme puis envoyée à la baie CS9 pour commander le préhenseur.**

Manipulation 5 : Déclaration d'une variable pour le préhenseur

Étape 24 Proposer et déclarer la structure de données `T_Prehenseur` dans le dossier `Types données dérivés` du projet.

Étape 25 Déclarer une variable `prehenseurScara` de type `T_Prehenseur`. Initialiser cette variable avec les caractéristiques ci-dessus

Manipulation 6 : Commande du préhenseur depuis le programme principal

Étape 26 Compléter les sections `F1_ProductionNormale` et `F1_ProductionNormale_Post` pour réaliser le cahier des charges du début de cette sous-section.

L'envoi des commandes à la baie CS9 se fait par l'intermédiaire du bloc `VAL_ReadWriteIO` fourni par la bibliothèque `uniVALplc`. Ce bloc permet de lire et d'écrire des données sur les entrées/sorties de la baie CS9. Son interface est donnée en annexe A.3.

Manipulation 7 : Envoi des commandes à la baie

Étape 27 Etablir une structure de donnée `T_J212IO` et en instancier une variable pour éviter l'éparpillement des données utilisées par le bloc `VAL_ReadWriteIO`.

Étape 28 Dans la section `WriteEthIpRobot`, faire le lien entre les variables structurées `T_Prehenseur` et `T_J212IO`.

Étape 29 Utiliser le bloc `VAL_ReadWriteIO` pour envoyer les commandes de saisie et de dépose à la baie CS9.

Vérification 5 : Fonctionnement

Étape 30 Vérifier le bon fonctionnement du préhenseur en exécutant le programme.

Étape 31 Faire valider par l'enseignant

Annexes

A UnivalPlc

A.1 Type pour robot Staubli

```
TYPE T_StaeubliRobot :
    STRUCT
        Status : T_Status ;
        Command : T_Command ;
        CommandInterface : T_CommandInterface ;
    END_STRUCT
END_TYPE


TYPE T_Status:
    STRUCT
        Initialized: BOOL ; (* True = The library is initialized *)
        Online: BOOL ; (* True = robot can receive commands *)
        ErrorPending: BOOL ; (* True = an error is pending *)
        IsMoving: BOOL ; (* True = robot is moving *)
        EStopActive: BOOL ; (* True = E-stop is pending *)
        DummyPlug: BOOL ; (* True = staubli teach pendant replaced by dummy
        plug *)
        ExternalMcp_Wms: BOOL ; (* True = Robot is properly configured to use both
        user- supplied WMS and Teach Pendant. *)
        ActualSpeed: REAL ; (* Cartesian speed of the current TCP *)
        ActualOverride: REAL ; (* Current override value [0.01 .. 100] *)
        ActualErrorNumber: UINT ; (* Total number of pending errors in robot *)
        ActualOperationMode: UINT ; (* Actual working mode 0= invalid , 1=manu,
        3=Auto, 4=remote(extaut) *)
        ActualErrorID: T_PendingErrors ; (* List of pending error on server side *)
        CartesianPos: T_CartesianPos ; (* Current cartesian position *)
        JointPos: T_JointPos ; (* Current joint position *)
        ActualCoordSystem: UINT ; (* Number of the user frame in which the position of
        the Tool Center Point is reported *)
        ActualTool: UINT ; (* Number of the Tool Center Point for which the
        position is reported *)
        RobotStateMachine: UINT ; (* State Machine of the robot *)
        MovementID: INT ; (* Identifier of motion currently executed *)
        MovementProgress: INT ; (* Percentage of actual movement that has been
        completed *)
        RobotModel: INT (* Robot model connected to controller *)
        ControllerModel: UINT ; (* Staubli controller 8=CS8C / 9=CS9 *)
        CS9Safety: T_CS9SftyFbk ; (* CS9 Only - Safety features *)
        Heartbeat: UINT ;
        ServerMajorVersion: UINT ; (* Major version of unival PLC server *)
        ServerMinorVersion: UINT ; (* Minor version of unival PLC server *)
        ServerEdit: UINT ; (* Edit of the uniVAL plc server *)
        ClientMajorVersion: UINT ; (* Major version of unival PLC client library *)
        ClientMinorVersion: UINT ; (* Minor version of unival PLC client library *)
        ClientEdit: UINT ; (* Edit of the unival PLC client library *)
    END_STRUCT
END_TYPE

TYPE T_Command :
    STRUCT
        EnableVerbose : BOOL ; (* TRUE=Enable tracing activity of the server. It is
        strongly recommended to enable trace for debugging
        purpose only. Starting with FALSE *)
        OverrideCmd : UINT ; (* Commanded Override [1..100] (monitor speed) Starting
        with 0 *)
        OperationModeCmd : UINT ; (* Commanded operation mode 0= invalid , 1= Manu,
        3=Auto, 4 remote(extAut) *)
        ToolCmd : UINT ; (* Select the Tool used for movement. Starting with 0*)
        CoordSystemCmd : UINT ; (* Select the coordinate system used for movement.
        Starting with 0 *)
        CS9Safety : T_CS9SftyCmd ; (* CS9 Only - Safety features *)
        LifebitPeriod : TIME ; (* Select the period of the internal lifebit
        (100ms<Period<1000ms) . starting with t#200ms *)
    END_STRUCT
END_TYPE
```

A.2 Blocs fonctions pour robots Staubli

```
FUNCTION_BLOCK VAL_ReadAxesGroup
    VAR_INPUT
        Enable : BOOL ; (* TRUE : contents of the FB is executed *)
    END_VAR
    VAR_IN_OUT
        AxesGroup : T_StaeubliRobot ; (* Data block for a robot *)
        IN : T_FromRobot ;
    END_VAR
    VAR_OUTPUT
        Error : BOOL ; (* set when function block has terminated with error *)
        ErrorID : UDINT ; (* error code *)
        Timeout : BOOL ; (* set when a timeout is detected *)
    END_VAR
```

```
FUNCTION_BLOCK VAL_WriteAxesGroup
    VAR_INPUT
        Enable : BOOL ; (TRUE : contents of the FB is executed *)
    END_VAR
    VAR_IN_OUT
        AxesGroup : T_StaeubliRobot ; (* Data block for a robot *)
    END_VAR
    VAR_OUTPUT
        Error : BOOL ; (* set when function block has terminated with error *)
        ErrorID : UDINT ; (* error code *)
        OUT : T_ToRobot ;
        Timeout : BOOL ; (* set when a timeout is detected *)
    END_VAR
```

A.3 Bloc pour Entrées/Sorties CS9

```
FUNCTION_BLOCK VAL_ReadWriteIO
    (* Update the digital outputs of the controller with values passed as parameter.
    Return the state of the digital inputs available on the controller *)
    VAR_INPUT
        Enable : BOOL ; (*function block content is executed as long as this input is TRUE *)
        Valve1 : BOOL ; (* 6 axis robot only : the internal valve *)
        Valve2 : BOOL ; (* 6 axis robot only : the internal valve *)
        Fout0 : BOOL ; (* Command Fast output (J212-4 / J212-9) CS9 controller *)
        Fout1 : BOOL ; (* Command Fast output (J212-1 / J212-5) CS9 controller *)
        EnableValve1 : BOOL ; (* 6 axis robot only : the internal valve 5-3 closed middle *)
        EnableValve2 : BOOL ; (* 6 axis robot only : the internal valve 5-3 closed middle *)
    END_VAR
    VAR_IN_OUT
        AxesGroup : T_StaeubliRobot; (* Data block for a robot *)
    END_VAR
    VAR_OUTPUT
        Valid : BOOL ; (*Set when function block is executing. Reset when Done or Error*)
        UsrInput0 : BOOL ; (* Digital input (J109-11 / J109-30) CS8 controller *)
        UsrInput1 : BOOL ; (* Digital input (J109-16 / J109-35) CS8 controller *)
        Fin0 : BOOL ; (* Fast input (J111-2/7) CS8 controller, J212-2/7 CS9 controller *)
        Fin1 : BOOL ; (* Fast input (J111-3/8) CS8 controller, J212-3/8 CS9 controller *)
        ValvesSafeState : BOOL ; (* CS9 controlling 5-3 ways valves ONLY *)
    END_VAR
```

B HMI

B.1 FB pour IHM

```
TYPE THmi
    STRUCT
        m_Inputs : THmiInputs ;
        m_Outputs : THmiOutputs ;
    END_STRUCT
END_TYPE
```