Mise en œuvre d'UniVALplc sous Control Expert
Via Ethernet/IP (Partie 2)

Objectifs visés :

- ✓ ***Les commandes de mouvement*** *de type* ***PLC-Open***
- ✓ ***Diagnostic/Traitement de la défaillance de la baie CS9 sous UniVALplc***
- ✓ ***Mise en Conditions Initiales du robot Scara***
- ✓ ***Mise en œuvre de Blocs Fonctionnels***

0. Préambule

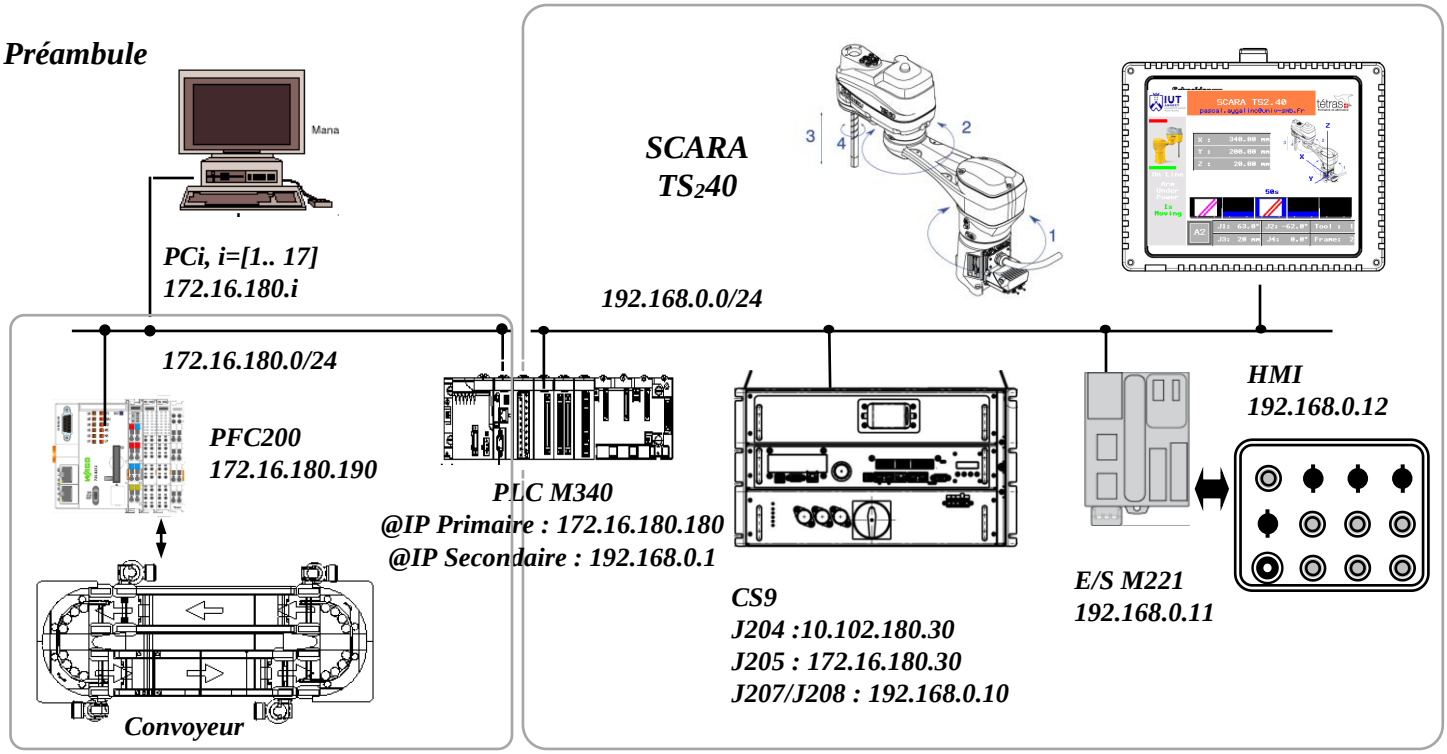


Figure 1 : *Cellule robotisée de la salle C180 :* ***(PLC M340, CS9, M221, HMI et PFC200)***

On rappelle qu'au niveau matériel, on dispose :

- ✓ *d'un robot* ***Stäubli Scara TS2 40 collaboratif*** *(4 axes)* et de son contrôleur ***CS9***, équipé de son ***pendant MCP*** et de son ***sélecteur de modes de marche WMS9***,
- ✓ *d'un convoyeur* ***TS2plus Bosch RexRoth*** sur lequel circulent des ***palettes*** portant des produits,
- ✓ *d'un* ***contrôleur PFC200*** du constructeur ***WAGO*** de type ***Little Endian*** qui gère une partie du convoyeur mais qui se comporte comme un ***ilot d'E/S*** accessible via ***Ethernet/IP*** pour la commande du ***Guichet***,
- ✓ *d'un* ***automate Schneider*** de type ***M340*** dont l'orientation des données est de type ***Little Endian***. Il est équipé de ***deux coupleurs réseaux BMX NOC 0401.2*** qui permettent d'obtenir :
  - pour le premier : une communication de type ***Ethernet*** avec les stations de développement ***(PCs)*** sur le ***réseau privatif de la salle C180 (classe B : 172.16.180.0/24)***. Elle est nécessaire pour le ***téléchargement du code*** dans l'automate et la ***surveillance de son exécution*** en mode connecté. Il permet aussi de scruter *le* ***contrôleur PFC200*** afin d'établir via ***Ethernet/IP*** la commande du ***Guichet*** du convoyeur sur lequel le robot ***Scara*** viendra saisir les pièces,
  - pour le second : un réseau de type ***Ethernet/IP*** pour ***les échanges périodiques*** à réaliser entre l'***API M340, la baie CS9 et le contrôleur M221***.
- ✓ *d'un* ***contrôleur Schneider*** de type ***M221*** dont la programmation réalisée permet l'accès au ***pupitre opérateur*** via ***Ethernet/IP*** et de fournir les informations de supervision à l'écran ***HMI*** à l'aide du protocole ***Modbus-Tcp***,

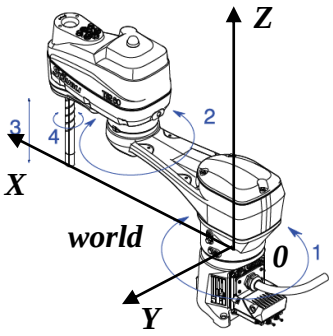
Lors des ***tps*** précédents, ont été étudiées et réalisées :

- ✓ la configuration du premier réseau ***Ethernet/IP*** comprenant un équipement (***PFC200***) et la communication à réaliser entre ***l'API M340*** et ce contrôleur. Elle a permis d'établir une première ébauche du mode de production normale (***mode F1***) du poste ***Guichet*** en ***isolation*** situé sur le convoyeur,
- ✓ la configuration du second réseau ***Ethernet/IP*** comprenant deux équipements *(la baie **CS9** et le contrôleur **M221**)* et la communication entre ***l'API M340*** avec la baie ***CS9***, ainsi qu'avec le contrôleur ***M221***,
- ✓ la structure de base logicielle du traitement utilisateur dans l'***API M340*** qui nécessite la ***réservation*** d'un objet du type ***T_StaeubliRobot*** pour représenter ***le robot*** *et* ***son contrôleur***. Cet objet doit être lu en début du cycle automate à l'aide du bloc fonctionnel ***VAL_ReadAxesGroup*** et doit être renvoyé à la baie en fin de cycle via ***VAL_WriteAxesGroup*** afin de transmettre les commandes de mouvement établies durant le cycle.

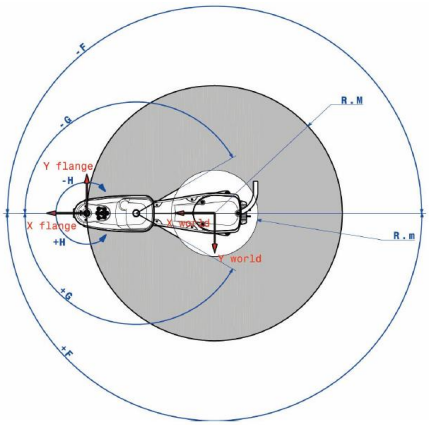
Ce ***tp*** portera sur la configuration initiale à donner à l'application ainsi que sur l'utilisation des différentes commandes de mouvement disponibles de type ***PLC-Open*** sur le serveur ***uniVALplc***. Pour ces commandes, les coordonnées des points qu'ils soient ***cartésiens*** *(type **T_CartesianPos**)* ou ***articulaires*** *(type **T_JointPos**)* sont directement issues de la mémoire de l'***API***.

Nota *: D'autres commandes de mouvement sont aussi disponibles mais ne seront pas vues ici car elles sont spécifiques à la baie **CS9**. Elles font toutes référence à des objets contenus dans des bases de données qu'il est possible de créer dans la baie à l'aide du pendant (cf. uniVALplc_client_4_5.pdf pour plus de détails)*

Le domaine de travail du robot ***Scara TS$_2$40*** et quelques-unes de ses caractéristiques vous sont donnés par le tableau suivant :



Rayon Max :R.M	***460 mm***
Rayon min : R.m	***150 mm***
Axe 1 : J1	***[-180°, 180°]*** *(550°/s)*
Axe 2 : J2	***[-141°, 141°]*** *(720°/s)*
Axe 3 : J3 (Z)	***[0, 200 mm]*** *(2500mm/s)*
Axe 4 : J4	***[-400°, 400°]*** *(2500°/s)*
Frein :	***Axe 3 et 4***



1. *Principe de la programmation robot*

Afin de permettre un éventuel lissage entre deux trajectoires se succédant *(cf. TYPE **eMC_BUFFER_MODE** : {Aborting := 0, Buffered (without blending) := 1, BlendingJoint := 6, BlendingCartesian := 7})*, l'ensemble de commandes de mouvement sont enregistrés dans une pile ***FIFO*** ***(First In, First Out** pour en conserver l'ordre)*. Elles seront ***dépilées*** au fur et mesure de leur exécution par le robot.

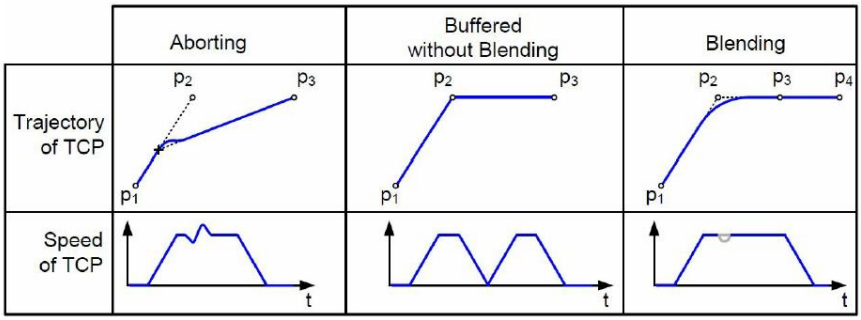


Figure 2 : ***TCP*** *:* ***T**ool **C**enter **P**oint*

Lors de la programmation de ces commandes de mouvement, il convient de connaitre ***parfaitement*** le ***fonctionnement*** *de* ***cette pile*** et de son ***influence*** sur le comportement du robot comme on a pu le voir en cours. Pour rappel :

- ✓ L'***exécution séquentielle*** des commandes de mouvement dans la pile débute ***dès que le bras est mis sous puissance*** obtenu à l'aide du bloc ***MC_GroupPower***
- ✓ Il est possible d'***ajouter*** des commandes de mouvement ***à tout moment*** *(un mouvement est en cours ou pas, le bras est ou n'est pas sous puissance, …).*
- ✓ Une commande de mouvement ***sort de la pile*** dès que ***sa trajectoire*** associée est ***totalement achevée***.
- ✓ Pour ***vider la pile*** de toutes les commandes de mouvement empilées, on dispose du bloc fonctionnel ***MC_GroupStop***.
- ✓ ***L'arrêt immédiat*** du robot durant l'exécution d'une trajectoire peut se faire via le ***bouton d'arrêt d'urgence*** (***E-Stop***), par un ***changement de mode de marche*** sur la baie (***WMS9)***, ou à l'aide des blocs fonctionnels ***MC_GroupInterrupt*** et ***MC_GroupStop***.
- ✓ Suite à un ***arrêt obtenu*** à l'aide du bloc fonctionnel ***MC_GroupInterrupt***, il est possible de ***reprendre les mouvements là où ils ont été stoppés*** grâce au bloc fonctionnel ***MC_GroupContinue***, ou bien ***de les supprimer*** en vidant la pile (***MC_GroupStop*** *vu précédemment).*
- ✓ Le ***contenu de la pile*** des commandes de mouvement ***est conservé même si le bras est hors puissance***. Une mise sous puissance du bras conduira ***éventuellement*** à une phase dite de « ***connexion*** » faite à petite vitesse pour rejoindre la trajectoire interrompue puis à la ***reprise des mouvements toujours présents dans la pile***.

On dispose donc pour ***les commandes de contrôle de mouvement*** de :

Name of Function block	*Description*
MC_GroupInterrupt	***Stop the movement*** *of the robot. The robot can continue along the commanded path after* ***MC_GroupContinue*** *is executed*
MC_GroupStop	***Stop the movement*** *of the robot and* ***flushes all movements*** *that have been commanded*
MC_GroupContinue	*Restart the movement of the robot after it has been stopped by* ***MC_GroupInterrupt***
MC_GroupReset	***Reset all pending errors*** *reported by the robot.*
MC_GroupPower	***Switch ON/OFF*** *the power stage of the robot*

Remarque : Lorsque l'on télécharge un nouveau programme dans l'automate, la communication entre l'***API M340*** et la baie ***CS9*** ainsi que celle avec le ***M221*** sont ***rompues***. Il en est de même avec le ***PFC200***. Les ***bits de vie*** entretenus par ces communications ne s'effectuant plus, la baie ***CS9*** passe ***en défaut***, interdisant toute commande sauf naturellement celle qui permet d'acquitter ces erreurs ***(MC_GroupReset)***. Pour le ***M221***, sa programmation conduit à faire ***clignoter*** à la fréquence de ***1 Hz*** la ***composante rouge*** de la ***colonne qu'il gère***. Pour le ***PFC200***, le ***poste Guichet*** devient ***non passant*** et ***sa colonne*** prend ***la couleur magenta***. Pour les deux contrôleurs ***M221*** et ***PFC200***, ils retrouvent leur fonctionnement normal dès que la communication devient opérationnelle. Pour la baie ***CS9***, il faudra mettre en place une procédure de redémarrage (*cf. 2.2*°mode ***D2 : Diagnostic/Traitement de la défaillance***)

Quant aux ***commandes de mouvement*** de type ***PLC-Open*** disponibles sur le serveur ***uniVALplc***, on distingue :

Name of Function Block	*Interpolated type*	*End Position*
MC_MoveAxisAbsolute	*Joint interpolated movement*	*joint position (**T_JointPos**)*
MC_MoveDirectAbsolute	*Joint interpolated movement*	*cartesian position (**T_CartesianPos**)*
MC_MoveLinearAbsolute	*Linear movement*	*cartesian position (**T_CartesianPos**)*
MC_MoveCircularAbsolute	*Circular movement*	*cartesian position (**T_CartesianPos**)*

Seule la première commande ne permet pas de prendre en compte la modélisation de ***l'outil*** situé à l'une des extrémités de l'axe linéaire ***J3*** (paramètre ***ToolNumber***) pour un robot ***Scara***. La position articulaire en absolue s'énonce dans le repère « ***world*** » alors que pour les autres commandes, il est possible de faire référence à un ***repère personnalisé*** (*paramètre « **CoordSystem** » pour désigner un repère de type **UserFrame**).* Pour plus de détails sur les paramètres de ces blocs fonctions, se reporter aux ***annexes*** ou à la documentation *« uniVALplc_client_4_5.pdf »* disponible sous *U:/Documents/LicencePro/RIF/UE2/Automatisme pour la robotique/Scara/uniVALplc/Doc»*

Nota : *Les conventions du constructeur **Stäubli** sont :*
*(**CoordSystem = 0**) ⇔ repère **WORLD** : (fPlcFrame[0] {X, Y, Z, Rx, Ry, Rz}={0, 0, 0, 0, 0, 0})*
*(**ToolNumber = 0**) ⇔ **FLANGE**(bride) : (tBankTool[0] {X, Y, Z, Rx, Ry, Rz}={0, 0, 0, 0, 0, 0})*

2. Gmma : Graphe des modes de marche et d'arrêt

Pour pouvoir utiliser l'outil ***GEMMA*** afin de concevoir une ***commande hiérarchisée***, ***les erreurs détectées*** à l'aide des ***bits de santé*** disponibles sur les réseaux ***NOC_ETHIP_CONVOYEUR*** et ***NOC_ETHIP_ROBOT*** devront être vues comme des ***pannes de la partie opérative***. Comme le montre le ***gmma*** de l'annexe, ***toute erreur de communication*** conduira le graphe dans ***l'état D1***, *et* ***le figera dans cet état*** tant que la communication n'est pas opérationnelle. Au retour de celle-ci, ***le mode D1*** s'exécutera.

*2.1 **Mode D1 : Arrêt d'Urgence***

Ce mode ***D1*** est le premier mode actif quand l'automate ***M340*** passe en ***RUN***. Sous réserves que la bibliothèque est initialisée et que la baie ***CS9*** est prête à recevoir des commandes (*cf. champ **Status.Initialized** et **Status.Online** de la structure **T_StaeubliRobot**)*, il aura pour but de ***mettre hors puissance*** le bras à l'aide d'une instance du bloc fonctionnel ***MC_GroupPower***. On maintiendra dans l'état le système de préhension *(distributeur bistable)* et les bloqueurs du ***Poste*** G***uichet*** devront interdire son accès (*la colonne **RVB** aura uniquement sa **composante rouge** allumée).*

*2.2 **Mode D2 : Diagnostic/Traitement de la défaillance***

Le mode ***D2*** aura pour objectif de donner une configuration du contrôleur propre de tout défaut *(**MC_GroupReset**)*, avec une pile de commandes de mouvement totalement vide *(**MC_GroupStop**)*, prête à en recevoir de nouvelles *(**MC_GroupContinue**)*, et avec un bras sous puissance *(**MC_GroupPower**).*

Ce mode devra donc réaliser de façon séquentielle ***les commandes de contrôle de mouvement*** suivantes :

a) *Acquitter les défauts présents (**MC_GroupReset**).*
b) *Vider la pile des commandes de mouvements (**MC_GroupStop**)*
c) *Autoriser de nouvelles commandes de mouvement (**MC_GroupContinue**)*
d) *Mettre le bras sous puissance (**MC_GroupPower**)*

Pour ce qui est du ***Poste Guichet***, son accès restera interdit mais on fera ***clignoter*** à la fréquence de ***1 Hz*** la ***composante rouge*** *de sa colonne.* Le système de préhension conservera quant à lui son état.

*Etablir le grafcet fonctionnel associé à la partie séquentielle de ce mode **D2** ainsi que les commandes combinatoires à réaliser sur le **Poste Guichet** et le système de préhension.*

*2.3 **Mode A5 : Remise En Route après Défaillance***

Pour simplifier ce mode, on considérera que l'on n'a pas besoin de déplacer le robot dans une position bien précise pour évacuer l'éventuelle pièce qui est restée saisie. Comme le préhenseur n'est pas instrumenté, c'est l'opérateur

qui autorisera la dépose par le biais du pupitre opérateur *(bouton bleu « autorisation de dépose »)*. Par ailleurs, il signalera que toutes les aires de traitement sont vides *(bouton « cuves vides »)*. Ce mode sera repris dans un prochain *tp* et fera appel à la conception d'un bloc fonctionnel utilisateur pour évacuer sur une aire de rebus l'éventuelle pièce restée saisie.

2.4 ***Mode A6 : Mise En Conditions Initiales***

On souhaite pouvoir positionner le robot au-dessus du guichet, prêt à saisir un gobelet mais sans pour autant entraver la circulation des palettes vides ou pleines. Les coordonnées cartésiennes dans le repère « ***world*** » respectant ces contraintes sont : (*X=90, Y=-410, Z=150, RX=-180, RY = 0, RZ =0*). Afin de sécuriser le déroulement de ce mode, on adopte la stratégie suivante ***si la position courante n'est pas celle voulue*** :

« On commencera par monter l'axe ***J3*** *en position* ***150*** *mm, sans modifier la position courante des autres axes à l'aide d'une commande de* ***mouvement linéaire****. Quand l'axe* ***J3*** *a fini sa course, une commande de mouvement de type* ***articulaire*** *permet de diriger le robot au-dessus du guichet pour terminer le positionnement voulu. Durant ce mode, la couleur de la colonne lumineuse sera jaune clignotant (période de* ***t#1s****) et les deux bloqueurs (****BE****,* ***BP****) du* ***Poste Guichet*** *seront toujours levés afin d'interdire son accès. Si au cours de celui-ci, un arrêt est demandé via le pupitre opérateur, on immobilisera le robot tout en maintenant le bras sous puissance. Il reprendra sa course dès que cet arrêt n'est plus actif. On signalera cette situation d'arrêt en allumant de façon clignotante le voyant rouge associé au bouton « arrêt demandé » ainsi que la colonne.*

2.4.1 Justifier la nature des commandes de mouvement que doit réaliser le robot dans ce mode. Indiquer les coordonnées des différentes positions finales à prendre en compte pour ces déplacements.

2.4.2 Proposer un grafcet fonctionnel réalisant cette mise en conditions initiales du robot

2.4.3 Quels problèmes rencontre-t-on au niveau de sa traduction en langage ***SFC*** *? Etablir les adaptations nécessaires pour effectuer son implantation.*

2.5 ***Mode A1 : Arrêt En Conditions Initiales***

Le bras du robot sera immobile mais sous puissance au-dessus du guichet passant. La colonne sera verte afin d'indiquer à l'opérateur que la cellule est prête pour traiter des pièces en mode automatique s'il le souhaite (cf Partie 3).

Partie TP

Le mise en œuvre associée à ce ***TD/TP*** reste générale et ne se substitue pas aux ***instructions de sécurité*** détaillées contenues dans les ***manuels*** des produits ***Schneider Electric SA*** et ***Stäubli***, et leurs caractéristiques techniques. Ces documents doivent ***être lus attentivement et appliqués*** avant de se lancer dans la partie ***TP***. On rappelle que ***le non-respect*** de ces ***instructions*** peut provoquer ***la mort***, ***des blessures graves*** ou ***des dommages matériels***.

3.1 Après s'être connecté(e) à une station PC, lancer la chaîne de développement ***Control Expert*** *à l'aide du raccourci disponible sur le bureau.*

3.2 Reprendre l'application du ***tp*** *précédent et la terminer si besoin.*

3.3 Programmation du ***Gmma****.*

*3.3.1 Réserver 6 objets booléens (****xD1****,* ***xD2****,* ***xA5****,* ***xA6****,* ***xA1*** *et* ***xF1A2****) nécessaire pour rendre la scrutation conditionnelle des sections relatives aux modes* ***D1, D2, A5, A6, A1*** *et* ***F1(A2)****.*

3.3.2 *Editer les caractéristiques des sections **F1_ProductionNormale** et **F1_ProductionNormalePost**. Les rendre **conditionnelles** à l'aide de l'objet booléen **xF1A2** (onglet **Condition**)*

3.3.3 *Créer trois sections de programme inconditionnelles associées au **gmma** : **GM_GmmaPre** et **GM_GmmaPost** en langage **ST**, et **GM_Gmma** en langage **SFC**. L'ordre de scrutation de ces 3 sections sera **GM_GmmaPre,** puis **GM_Gmma** et enfin **GM_GmmaPost**, et seront situées juste après la section **ReadEthIpRobot** établie au **TP n°2**.*

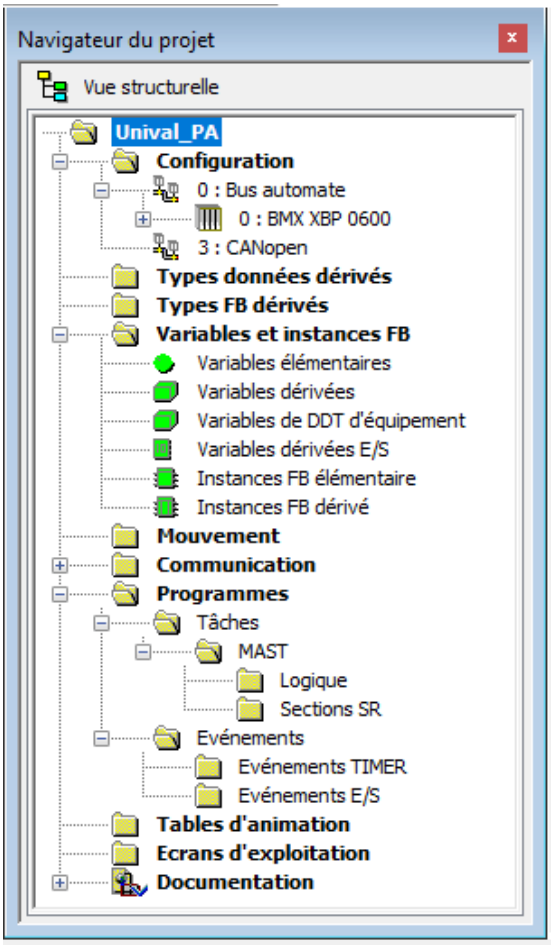


Figure 3

3.3.4 *Renseigner les corps des trois sections du **gmma** afin d'obtenir le fonctionnement suivant :*

- ✓ ***GM_GmmaPre** : Elle sera la deuxième à être scrutée dans le cycle **API**. Son objectif est :*
  - *d'initialiser le **Gmma** lors du **1er Cycle** en mode **RUN**, ou lorsqu'une **erreur de communication** est détectée via les bits de santé sur les réseaux **Ethernet/IP.** Réserver pour cela, un objet **xResult** de type **BOOL** nécessaire pour récupérer la valeur de retour de la fonction **InitChart**,*
  - *de positionner la période du bit de vie (champ **Command.LifebitPeriod** de la struct **T_Command** de l'objet **staeubliRobotScara** avec une valeur **t#150ms** (elle doit être comprise entre t#100ms et t#1s) lors du **1er Cycle** en mode **RUN***
- ✓ ***GM_Gmma** : Traduction du **gmma** en langage **SFC.** Le graphe **SFC** comportera 5 étapes avec pour étape initiale **GM_S_D1** correspondant au mode **D1** vu précédemment. Ne pas oublier d'associer les actions de type **P1** qui permettront d'initialiser les graphes des modes **D2**, **A5** et **A6** quand ceux-ci seront créés (leur conception a montré qu'ils n'étaient pas répétitifs).*
- ✓ ***GM_GmmaPost** : Il permet de positionner les bits **xD1**, **xD2**, **xA5**, **xA6**, **xA1** et **xF1A2** des sections conditionnelles associées aux modes **D1**, **D2**, **A5**, **A6**, **A1** et **F1(A2)** ainsi que de fournir à l'écran **hmi** l'état du **Gmma** sous forme hexadécimale (par exemple : hmiScara.m_Outputs.m_Screen.m_intStateGmma := **16#D1**).*

*3.4 Programmation du mode **D1 : Mode Arrêt d'Urgence***

3.4.1 *Créer une section conditionnelle (objet : **xD1**) **D1_ArretUrgence** en langage **ST**.*

3.4.2 *Réserver une instance du bloc fonctionnel **MC_GroupPower** avec l'éditeur de données, onglet **Bloc fonction***

3.4.3 *Pour effectuer et simplifier la programmation (éparpillement des données), on définira le type de données **TCtrlStatusGroupPower** avec l'éditeur de données, onglet **Types DDT.***

```
TYPE
    STRUCT TCtrlStatusGroupPower
        m_xEnable       : BOOL ;
        m_xStatus       : BOOL ;
        m_xError        : BOOL ;
        m_udiErrorID    : UDINT ;
    END_STRUCT
END_TYPE
```

*3.4.4 Réserver un objet **ctrlStatusGroupPower** de type **TCtrlStatusGroupPower** avec l'éditeur de données, onglet **Variables dérivées**.*

*3.4.5 Dans la section **WriteEthIpRobot**, faire scruter l'instance du bloc fonction **MC_GroupPower** en utilisant l'objet **ctrlStatusGroupPower** pour le passage des paramètres. Etablir aussi la commande du voyant qui permet de visualiser l'état sous puissance du bras (**hmiScara.m_Outputs.m_qToR.m_nqxArmUnderPower**).*

*3.4.6 Etablir le corps du mode **D1** qui doit mettre le bras hors tension si les conditions de reprise le permettent.*

*3.4.7 Renseigner la condition de passage **GM_T_D1toD2** de la section **GM_Gmma**. Etablir l'équation qui rend **xD1** actif dans la section **GM_GmmaPost** ainsi que de fournir à l'écran **hmi** l'état du **Gmma** sous forme hexadécimale (hmiScara.m_Outputs.m_Screen.m_intStateGmma)*

*3.5 Programmation du mode **D2 : Diagnostic/Traitement de la défaillance***

*3.5.1 Créer deux sections conditionnelles (objet : **xD2**) **D2_Diagnostic** en langage **SFC** et **D2_DiagnosticPost** en langage **ST**.*

*3.5.2 Réserver les instances des blocs fonctionnels associés aux commandes de contrôle de mouvement nécessaires pour effectuer ce mode (**MC_GroupReset**, **MC_GroupStop**, et **MC_GroupContinue**). Le nom des instances créées devra faire référence au mode qui les utilise (par exemple, l'instance du bloc fonctionnel **MC_GroupReset** réservée pour le mode **D2** aura pour nom **InstGroupResetD2**).*

*3.5.3 A l'aide de l'éditeur de données, onglet **Type DDT**, compléter le type **TCtrlStatusGroup** en rajoutant un nouveau champ **m_xExecute** de type **BOOL**.*

*3.5.4 Réserver les objets suivants pour les appels des blocs fonctionnels à l'aide de l'éditeur de données onglet **Variables** :*

Nom	**Type**
*ctrlStatusGroupReset***D2**	**TCtrlStatusGroup**
*ctrlStatusGroupStop***D2**	**TCtrlStatusGroup**
*ctrlStatusGroupContinue***D2**	**TCtrlStatusGroup**

*3.5.5 Etablir le graphe **SFC** associé au mode **D2_Diagnostic**.*

*3.5.6 Compléter la section **D2_DiagnosticPost** par la scrutation des blocs fonctionnels et les rendre actifs en fonction de l'état d'avancement du graphe **SFC**. Pour la mise sous puissance du bras, utiliser une **section action** de type **P1 (D2_AP1_ArmPowerOn)**.*

*3.5.7 Renseigner la condition de passage **GM_T_D2toA5** de la section **GM_Gmma** ainsi que l'action de type **P1** associée à l'étape **GM_S_D2** réalisant l'initialisation du graphe **SFC D2_Diagnostic (GM_AP1_InitChartD2)** puisque ce dernier n'est pas répétitif. Etablir l'équation qui rend **xD2** actif dans la section **GM_GmmaPost** et modifier l'affectation en hexadécimale de hmiScara.m_Outputs.m_Screen.m_intStateGmma par une structure de choix multiple afin de fournir à l'écran **hmi** l'état du **Gmma**.*

*3.5.8 Compiler puis transférer l'application. Mettre en **RUN** l'automate et vérifier son fonctionnement. Sur l'écran doit disparaitre l'avertissement **Error Pending**, et le **voyant blanc** du pendant doit être allumé, ainsi que celui du pupitre opérateur indiquant que le **bras est sous puissance**.*

*3.6 Programmation du mode **A5 : Remise En Route***

*3.6.1 Créer deux sections conditionnelles (objet : **xA5**) **A5_RemiseEnRoute** en langage **SFC** et **A5_RemiseEnRoutePost** en langage **ST.***

*3.6.2 Etablir le graphe **SFC** conformément à votre analyse faite en 2.3. C'est le bouton poussoir connecté sur le connecteur **J212** de la baie **CS9** : entrée rapide **TOR fin0 (J212-2/J212-7)** qui autorisera la dépose.*

*3.6.3 Renseigner la section **A5_RemiseEnRoutePost** afin de prendre en compte l'évolution du graphe pour la commande de dépose et d'obtenir **un flux non passant** des palettes dans le poste. La colonne lumineuse s'éclairera alternativement en rouge puis en jaune avec une période de **T#1s** durant ce mode.*

*3.6.4 Ecrire la condition de passage **GM_T_A5toA6** de la section **Gmma** ainsi que l'action de type **P1** associée à l'étape **GM_S_A5** réalisant l'initialisation du graphe SFC **A5_RemiseEnRoute (GM_AP1_InitChartA5)** puisque ce dernier n'est pas répétitif.*

*3.6.5 Etablir l'équation qui rend **xA5** actif dans la section **GM_GmmaPost**. Compléter la structure de choix multiple pour l'affectation en hexadécimale de la donnée hmiScara.m_Outputs.m_Screen.m_intStateGmma.*

*3.6.6 Compiler puis transférer l'application. Appeler l'enseignant avant de mettre en **RUN** l'automate.*

*3.7 Programmation du mode **A6 : Mise En conditions Initiales***

*3.7.1 Créer deux sections conditionnelles (objet : **xA6**) **A6_MiseEnCI** en langage **SFC** et **A6_MiseEnCIPost** en langage **ST***

*3.7.2 Créer les instances des blocs fonctionnels associées aux commandes de mouvement **MC_MoveLinearAbsolute** et **MC_MoveDirectAbsolute** ainsi que celles des blocs **MC_GroupInterrupt** et **MC_GroupContinue** permettant d'arrêter le robot dans ce mode à l'aide de l'élément de commande « Arrêt demandé » du pupitre opérateur.*

*3.7.3 Importer le type de données **TMoveAbsolute** qui vous facilitera les appels des blocs fonctionnels associés aux commandes de mouvement. Créer deux objets de ce type : **moveDirectAbsA6** et **moveLinearAbsA6***

*3.7.4 Créer un objet **cartesianPosInitialeA6** de type **T_Cartesian** pour définir les coordonnées initiales à donner au robot dans le repère « **world** ». **L'initialiser** avec les valeurs suivantes : (X=90, Y=-410, Z=150, RX=-180, RY = 0, RZ =0, CS=1, CE=1, CW=1).*

*3.7.5 Etablir le graphe **SFC** du mode **A6**. Comme le langage **SFC** n'autorise pas l'écriture de structures de choix comportant des transitions de synchronisation, ni des convergences de choix comportant des*

*transitions de parallélisme, vous serez amené à faire des adaptations pour traduire le grafcet fonctionnel. Par ailleurs, vous utiliserez des actions de type **P1** associées aux étapes pour établir le calcul des positions finales des mouvements ainsi que pour définir les propriétés à donner pour réaliser ces trajectoires (descripteur du mouvement) :*

- ✓ *pour les vitesses (**Velocity**) et accélérations angulaires (**Acceleration**, **Deceleration**) : **100.0***
- ✓ *pour les vitesses max du centre outil (TCP) **CartesianVel** : **500 mm/s**, **RotationalVel** :=**60 °/s***
- ✓ *comme on n'utilise pas la base de données de la baie **CS9**, le système de coordonnées sera « **world** » (**CoordSystem** :=0) et comme indice d'outil la bride « **flange** » (**ToolNumber** :=0).*
- ✓ *il est conseillé pour des soucis d'optimisation (durée et mécanique) d'effectuer un lissage entre les mouvements.*

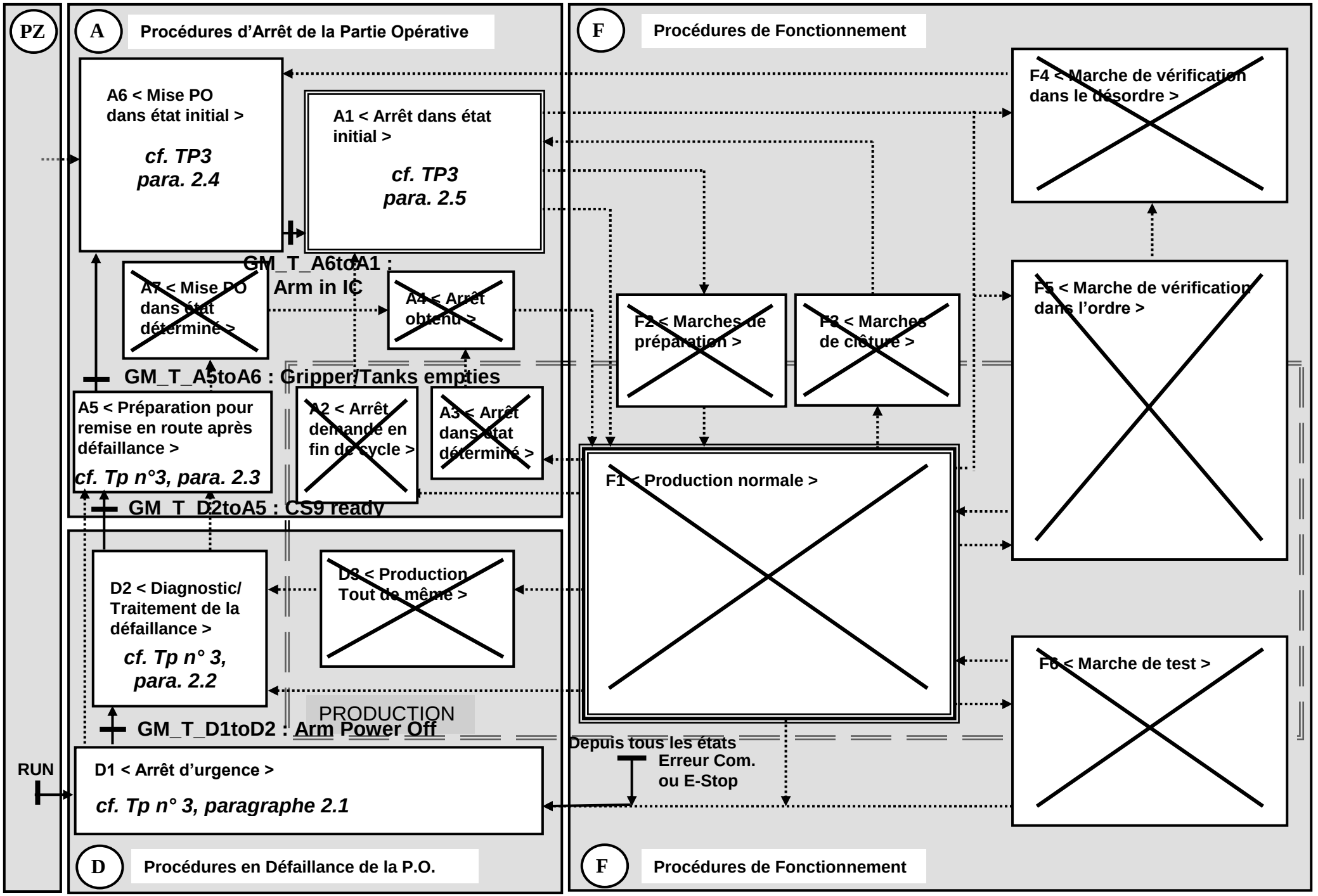
*3.7.6 Renseigner la section **A6_MiseEnCIPost** postérieur au graphe afin d'activer les blocs de contrôle et de commandes de mouvement, sans oublier l'animation de la colonne lumineuse et le voyant de bouton pousser-pousser « **arrêt demandé** ».*

*3.7.7 Pour terminer, écrire la condition de passage **GM_T_A6toA1** de la section **Gmma** ainsi que l'action de type **P1** associée à l'étape **GM_S_A6** réalisant l'initialisation du graphe SFC **A6_Mise EnCI** (**GM_AP1_InitChartA6**) puisque ce dernier n'est pas répétitif. Prendre en compte ce nouveau mode afin de fournir à l'écran **hmi** l'état du **Gmma** sous forme hexadécimale.*

*3.7.8 Compiler puis transférer l'application. Appeler l'enseignant avant de mettre en **RUN** l'automate. Sur l'écran doit apparaitre quand le robot est en mouvement une jauge verte proportionnelle au pourcentage déjà effectué du mouvement à réaliser. Le voyant blanc du pendant et celui du pupitre opérateur seront allumés car le **bras est sous puissance**.*

*3.8 Programmation du mode **A1 : Arrêt En conditions Initiales**.*

*3.8.1 Créer une section conditionnelle (objet : **xA1**) **A1_ArretEnCi** en langage ST qui permettra d'imposer une colonne verte afin d'indiquer à l'opérateur que la cellule est prête pour traiter des pièces en mode automatique s'il le souhaite (cf Partie 3) et un flux passant au niveau du poste **Guichet**.*


PZ
A
Procédures d'Arrêt de la Partie Opérative
A6 < Mise PO dans état initial >
cf. TP3 para. 2.4
A1 < Arrêt dans état initial >
cf. TP3 para. 2.5
GM_T_A6toA1 : Arm in IC
A7 < Mise PO dans état déterminé >
A4 < Arrêt obtenu >
GM_T_A5toA6 : Gripper/Tanks empties
A5 < Préparation pour remise en route après défaillance >
cf. Tp n°3, para. 2.3
A2 < Arrêt demandé en fin de cycle >
A3 < Arrêt dans état déterminé >
GM_T_D2toA5 : CS9 ready
D2 < Diagnostic/ Traitement de la défaillance >
cf. Tp n° 3, para. 2.2
D3 < Production Tout de même >
PRODUCTION
GM_T_D1toD2 : Arm Power Off
RUN
D1 < Arrêt d'urgence >
cf. Tp n° 3, paragraphe 2.1
D
Procédures en Défaillance de la P.O.
F
Procédures de Fonctionnement
F4 < Marche de vérification dans le désordre >
F2 < Marches de préparation >
F3 < Marches de clôture >
F5 < Marche de vérification dans l'ordre >
F1 < Production normale >
F6 < Marche de test >
Depuis tous les états
Erreur Com.
ou E-Stop
F
Procédures de Fonctionnement

Annexe 1 :

```
TYPE:                   (* This enumeration lists the supported buffer modes *)
eMC_BUFFER_MODE :  ( Aborting:=0,          (*A FB with buffer mode "Aborting" aborts any ongoing motion
                                             and starts the new motion immediately *)
                   Buffered:=1,            (* The next FB affects the axes group as soon as the previous
                                             motion is fully completed (Robot has reached commanded
                                             destination) *)
                   BlendingJoint:=6,       (* The current and the next motion FBs are blended, so the axes
                                             group will not stop between the motions. During blending, the
                                             robot is performing a joint interpolated motion *)
                   BlendingCartesian:=7);  (* The current and the next motion FBs are blended, so the axes
                                             group will not stop between the motions. The shape of the
                                             robot's motion during the blending is a Bezier curve *)
END TYPE

TYPE                    (* This enumeration defines the profile of the trajectory nearby the set points *)
eMC_TRANSITION_MODE  ( None:=0,            (* The motion blocks are not modified and no
                                             transition curve is inserted *)
                       Corner distance:=3, (* This mode is similar to 'Blending distances' except
                                             that leave and reach distances are equal *)
                       Blending distances:=10); (*leave and reach distances are differents *)
END TYPE

TYPE MC_TransitionParameter :
    STRUCT
        leave:  REAL; (* distance from the destination point at which the nominal trajectory is left *)
        reach:  REAL; (* distance from the destination point at which the nominal trajectory is joined *)
    END STRUCT
END TYPE

FUNCTION_BLOCK MC_MoveLinearAbsolute
    (* This function block commands a linear interpolated movement from the current robot position to the
    specified absolute position in the specified coordinate system. The values specified for the end-position are
    interpreted as cartesian values.
    The speed of the robot is calculated with both :
        • Parameters related to axes of the robot (Velocity / Acceleration / Deceleration)
        • Parameter related to the Tool Center Point (CartesianVel / RotationalVel)
    The final robot speed is set by the most limiting of these parameters. *)
    VAR_INPUT
        Execute          : BOOL ;     (* Rising edge triggers function execution *)
        Position         : T_CartesianPos ;(* Absolute end position for each dimension *)
        Velocity         : REAL ;     (* Joint velocity : % of nominal speed : [0.01,500]*)
        Acceleration     : REAL ;     (* Joint accelaration:% of nominal acceleration [0.01,500] *)
        Deceleration     : REAL ;     (* Joint decelaration:% of nominal deceleration [0.01,500] *)
        CartesianVel     : REAL ;     (* Max. cartesian velocity at Tool Center Point in mm/s *)
        RotationalVel    : REAL ;     (* Max. rotational velocity at Tool Center Point degree/s *)
        CoordSystem      : UINT ;     (* 0 : WORLD, others Coordinate System in a database *)
        ToolNumber       : UINT ;     (* 0 : FLANGE, others Index of the TOOL in a database *)
        BufferMode :     : UINT ;     (* cf. type eMC_BUFFER_MODE *)
        TransitionMode   : UINT ;     (* defined the profile of the trajectory nearby the set points*)
        TransitionParam  : MC_transitionParameter ;
    END VAR
```

VAR_IN_OUT

*AxesGroup : : **T_StaeubliRobot** ; (* Data block for a robot *)*

END VAR

VAR OUTPUT

Busy : BOOL ; (TRUE when function bloc is executing *)*

Done : BOOL ; (TRUE when function bloc has terminated with success *)*

Active : BOOL ; (TRUE when commanded movement is currently executed*)*

*CommandTransferred : BOOL ; (*TRUE when command motion is successfully buffered *)*

*CommandAborted : BOOL ; (*TRUE when aborted by another command *)*

Error : BOOL ; (TRUE when function bloc has terminated with error *)*

ErrorID : UDINT ; (error code *),*

MovementID : INT ; (identifier for this commanded movement *)*

END VAR

*FUNCTION_BLOCK **MC_MoveDirectAbsolute***

(This function block commands a **joint interpolated movement** from the current robot position to the specified absolute position in the specified coordinate system. The values specified for the end-position are interpreted as **cartesian values**.*

The speed of the robot is calculated with both :

- *Parameters related to axes of the robot (Velocity / Acceleration / Deceleration)*
- *Parameter related to the Tool Center Point (CartesianVel / RotationalVel)*

*The **final robot speed** is set by the **most limiting of these parameters**. *)*

VAR_INPUT

Execute : : BOOL ; (Rising edge triggers function execution *)*

*Position : : **T_CartesainPos** ; (* Absolute end positions for each dimension in the specified coordinate system *)*

Velocity : : REAL ; (Joint velocity : % of nominal speed : [0.01,500]*)*

Acceleration : REAL ; (Joint acceleration:% of nominal acceleration [0.01,500] *)*

Deceleration : REAL ; (Joint deceleration:% of nominal deceleration [0.01,500] *)*

CartesianVel : REAL ; (Max. cartesian velocity at Tool Center Point in mm/s *)*

RotationalVel : REAL ; (Max. rotational velocity at Tool Center Point degree/s *)*

CoordSystem : UINT ; (0 : WORLD, others Coordinate System in a database *)*

ToolNumber : UINT ; (0 : FLANGE, others Index of the TOOL in a database *)*

BufferMode : UINT ; (cf. type **eMC_BUFFER_MODE** *)*

TransitionMode : UINT ; (defined the profile of the robot trajectory *)*

*TransitionParam : **MC_transitionParameter** ;*

END VAR

VAR_IN_OUT

*AxesGroup : **T_StaeubliRobot** ; (* Data block for a robot *)*

END VAR

VAR OUTPUT

Busy : BOOL ; (TRUE when function block is executing *)*

Done : BOOL ; (TRUE when function block has terminated with success *)*

Active : BOOL ; (TRUE when commanded movement is currently executed*)*

*CommandTransferred : BOOL ; (*TRUE when command motion is successfully buffered *)*

*CommandAborted : BOOL ; (*TRUE when aborted by another command *)*

Error : BOOL ; (TRUE when function bloc has terminated with error *)*

ErrorID : UDINT ; (error code *),*

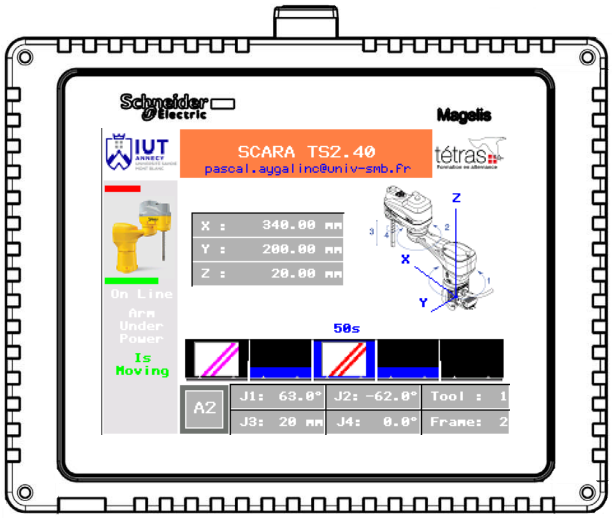
MovementID : INT ; (identifier for this commanded movement *)*

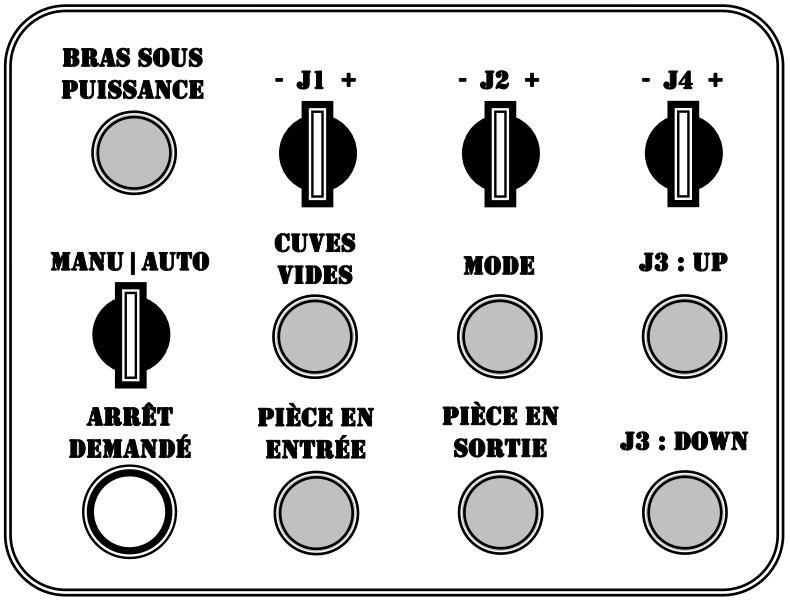
END VAR

```
FUNCTION_BLOCK MC_GroupPower
    (* This function block manages the arm power ON/OFF sequence *)
    VAR_INPUT
        Enable      : BOOL ;        (* TRUE= Command to switch power on,
                                       FALSE= Command to switch power off *)
    END_VAR
    VAR_IN_OUT
        AxesGroup   : T_StaeubliRobot ;   (* Data block for a robot *)
    END VAR
    VAR OUTPUT
        Status      : BOOL ;        (* Effective state of the robot power *)
        Error       : BOOL ;        (* Set when function block has terminated with error *)
        ErrorID     : UDINT ;       (* Error code *)
    END_VAR


FUNCTION_BLOCK MC_GroupReset
    (* This function blocks allows to reset all pending errors on robot side*)
    VAR_INPUT
        Execute     : BOOL ;         (* Rising edge triggers function execution *)
    END_VAR
    VAR_IN_OUT
        AxesGroup   : T_StaeubliRobot ;   (* Data block for a robot *)
    END VAR
    VAR OUTPUT
        Busy        : BOOL ;        (* TRUE when function block is executing *)
        Done        : BOOL ;        (* TRUE when function block has terminated with success *)
        Error       : BOOL ;        (* TRUE when function bloc has terminated with error *)
        ErrorID     : UDINT ;       (* error code *)
    END VAR


FUNCTION_BLOCK MC_GroupStop
    (* This function blocks commands a controlled motion stop. Moreover, this function cancels any movement
    that have been commanded and buffered on robot side *)
    VAR_INPUT
        Execute :   : BOOL ;        (* Rising edge starts function execution *)
    END_VAR
    VAR_IN_OUT
        AxesGroup   : T_StaeubliRobot ;   (* Data block for a robot *)
    END VAR
    VAR OUTPUT
        Busy        : BOOL ;        (* TRUE when function block is executing *)
        Done        : BOOL ;        (* TRUE when function block has terminated with success *)
        Error       : BOOL ;        (* TRUE when function bloc has terminated with error *)
        ErrorID     : UDINT ;       (* error code *)
    END VAR
```

```
FUNCTION_BLOCK MC_GroupContinue
    (* This function blocks allows to continue the movement that has been previously stopped by the
    MC_Interrupt function block *)
    VAR_INPUT
        Execute     : BOOL ;        (* Rising edge triggers function execution *)
    END_VAR
    VAR_IN_OUT
        AxesGroup   : T_StaeubliRobot ;   (* Data block for a robot *)
    END VAR
    VAR OUTPUT
        Busy        : BOOL ;        (* TRUE when function block is executing *)
        Done        : BOOL ;        (* TRUE when function block has terminated with success *)
        Error       : BOOL ;        (* TRUE when function bloc has terminated with error *)
        ErrorID     : UDINT ;       (* error code *)
    END VAR

TYPE TMoveAbsolute
    STRUCT
        m_xExecute          : BOOL ;    (* Rising edge triggers move execution *)
        m_cartesianPos      : T_CartesianPos ; (* Absolute end position for each dimension *)
        m_jointPos          : T_JointPos ;(* Absolute end position for each joint *)
        m_rVelocity         : REAL ;    (* Joint velocity : % of nominal speed *)
        m_rAcceleration     : REAL ;    (* Joint acceleration : % of nominal acceleration *)
        m_rDeceleration     : REAL ;    (* Joint deceleration : % of nominal deceleration *)
        m_rCartesianVel     : REAL ;    (* Max. cartesian velocity at Tool Center Point in mm/s *)
        m_rRotationalVel    : REAL ;    (* Max. rotational velocity at Tool Center Point degree/s *)
        m_uiCoordSystem     : UINT ;    (* Number of the coordinate system *)
        m_uiToolNumber      : UINT ;    (* Index of the tool in the bank *)
        m_uiBufferMode      : UINT ;    (* cf. type eMC_BUFFER_MODE *)
        m_uiTransitionMode  : UINT ;    (* cf. type eMC_TRANSITION_MODE *)
        m_transitionParam       : MC_transitionParameter ;
        m_xBusy                 : BOOL ;
        m_xDone                 : BOOL ;
        m_xActive               : BOOL ;
        m_xCommandTransferred   : BOOL ;
        m_xCommandAborted       : BOOL ;
        m_xError                : BOOL ;
        m_udiErrorID            : UDINT ;
        m_iMovementID           : INT ;
    END STRUCT
END TYPE

TYPE
    STRUCT TCtrlStatusGroupPower
        m_xEnable       : BOOL ;
        m_xStatus       : BOOL ;
        m_xError        : BOOL ;
        m_udiErrorID    : UDINT ;
    END_STRUCT
END_TYPE
```

Annexe 2 : ***Pupitre HMI, Eléments de commandes et Voyants***





TYPE ***THmi***

STRUCT (se reporter au* ***TP n°2*** *pour connaitre en détail les deux membres de cette structure *)*

m_Inputs *:* ***THmiInputs*** *;*

m_Outputs *:* ***THmiOutputs*** *;*

END_STRUCT

END_TYPE