REPUPLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Schneider Electric

ECOLE NATIONALE SUPERIEURE POLYTECHNIQUE

Département de Génie Electrique

Spécialité : Automatique

Projet de fin d'études en vue de l'obtention du Diplôme d'Ingénieur d'Etat en Automatique

Développement de blocs fonction sous UNITY PRO. Application aux régulateurs P.I.D. auto-ajustables

Réalisé par :

M^lle^ FEKNOUS Baya

Mr HAMI Elias

Proposé et dirigé par :

Pr E.M.BERKOUK

Mr M.MAARADJI

Juin 2009

Ecole Nationale Supérieure Polytechnique, 10 Av. Hassan Badi, El Harrach, Alger, Algérie

REMERCIEMENTS

Ce travail a été effectué à l'Ecole Nationale Supérieure Polytechnique d'Alger.

Tout d'abord, nous tenons à remercier notre promoteur Pr E.M. BERKOUK pour ses conseils et son aide tout au long de notre projet.

Nous remercions également toute l'équipe du « Service Automatique » de Schneider Electric Algérie pour leur disponibilité et leur soutien pendant notre stage de fin d'études, plus particulièrement Mr M.MAARADJI, Mr A.MOSSAAB et MR A. YOUNSI

Nous exprimons notre profonde reconnaissance aux membres du jury qui nous ont fait l'honneur d'étudier notre travail.

Que tous nos professeurs qui ont contribué à notre formation trouvent ici notre plus profonde gratitude.

Nous adressons nos plus sincères remerciements à tous ceux qui ont contribué, de près ou de loin, à l'aboutissement de ce travail.

Enfin, nous souhaitons dédier ce mémoire à nos parents. Rien n'aurait été possible sans leur soutien, confiance et générosité.

Dédicaces

Je dédie ce modeste travail, tout d'abord à ma chère maman qui a toujours été là pour moi et a fait beaucoup plus que ce qu'est censé faire une mère.

A mon père, sans qui rien n'aurait était possible.

A mon adoré frère au cœur d'or, qui me soutient depuis toujours de toutes les façons possibles et imaginables.

A ma tante Soussou, qui est pour moi une deuxième mère et son fils Rafik qui m'a tant fait rire durant cette période avec ses bêtises.

Et puis une dédicace spéciale à mes amis très chers,

D'abord, mon binôme Elias, avec qui ce fut un plaisir de travailler durant ces 3 dernières années, la bande des Soufaha (Ghylou, poupoune, zak, chriklou, omar, khelloudi ,rahim...) que je porte dans mon cœur.

Sans oublier ma meilleure amie Fifi, qui est la sœur que je n'ai jamais eu.

Baya

DEDICACE

A mes très chers parents

A ma tante Zahoua

A la mémoire de ma grand-mère et de mes grands-pères

A mes frères et sœur : Mahdi, Idir, Zakaria et Sabrina

A ma binôme : Baya

A mes amis : Omar, Khaloudi, Slimane, Chriklou…

A tous ceux qui ont su croire en moi

A tous ceux qui me sont chers

Je dédie ce modeste travail

Elias

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GENERALE

INTRODUCTION GENERALE

Depuis le début des années 80, les automates programmables sont intégrés pour le contrôle des différents processus industriels. A l'origine, l'automate programmable était considéré comme une machine séquentielle, capable de suppléer des automatismes réalisés en logique traditionnelle, en apportant toutefois de profonds bouleversements dans la manière de concevoir et d'organiser le contrôle d'un processus.

L'intégration de l'automate programmable renforce le degré de fiabilité de l'équipement et offre une très grande adaptabilité face aux évolutions de l'environnement. Aujourd'hui, l'automate programmable n'est plus seulement une machine séquentielle mais il est beaucoup plus considéré comme un calculateur de processus grâce aux énormes progrès quant à la structure de base, la qualité et la diversité des outils proposés. Son intégration sur Fieldbus (Profibus, WorldFip), sur Ethernet (Standard TCP-IP), accroît ses possibilités et constitue un passage obligé pour augmenter la performance des processus. **[1]**

Les systèmes de régulation et de commande de processus possèdent aujourd'hui une importance primordiale grâce à l'emploi toujours croissant des automates programmables dont la qualité des performances ne cesse de croître. Les processus demandent toujours une plus grande précision, ainsi qu'une parfaite stabilité. La recherche pour l'amélioration de la qualité converge tout naturellement vers l'amélioration des boucles de régulation, de manière à ce que la structure soit correctement adaptée pour répondre parfaitement aux spécifications.

Pour bien étudier les automates programmables industriels et leur programmation, nous avons effectué un stage au sein de l'entreprise SCHNEIDER ELECTRIC ALGERIE.

Schneider Electric Algérie est une filière de la société Scheider Electric qui est une entreprise française possédant une large offre de produits dans la distribution électrique, le contrôle du bâtiment et les automatismes de machines. L'entreprise possède plusieurs marques : Merlin Gerin, Square D, Télémécanique…

Notre travail consiste à réaliser pour le logiciel de programmation d'automates « Unity Pro », un régulateur P.I.D. *autoajustable* qui n'existe toujours pas dans la bibliothèque du logiciel et dont les ingénieurs de l'entreprise avaient besoin. Ce bloc devra identifier les paramètres du système dynamique à réguler, et d'après ces paramètres et la dynamique désirée, il calculera les paramètres d'un P.I.D. intégré pour pouvoir réguler le système et ce malgré son changement, ce qui est très souvent le cas en industrie.

Pour y arriver, nous avons du passer par quelques étapes que nous avons illustré en quatre grands chapitres.

Dans le premier chapitre, nous avons parlé des systèmes automatisés, donc des automates programmables en général (architecture, langages de programmation, réseaux de communication…), puis de la gamme télémécanique en particulier, avec les détails des différents modules constituant un automate de cette gamme.

Le second chapitre explicite le logiciel de programmation d'automates « Unity Pro ». Ainsi que les différents langages de programmation qu'il admet. Plusieurs exemples seront réalisés et simulé grâce aux outils du logiciel.

Nous n'avons pas oublié la partie supervision. Le troisième chapitre aborde le logiciel « Vijeo Designer » qui sert à programmer les interfaces Homme-machine (HMI). Un exemple avec plusieurs écrans, alarme, graphe… sera illustré à la fin de ce chapitre.

Ces trois derniers chapitres constituent une préparation au projet qui nous a été soumis. Le dernier chapitre présente plusieurs parties. La première partie est consacrée à la simulation numérique. La seconde aborde la régulation. Puis l'identification sera traitée, et enfin, l'élaboration du P.I.D. autoajustable.

Nous finirons par une conclusion générale sur les résultats obtenus, les connaissances assimilées durant ce projet et les perspectives en vue.

CHAPITRE I

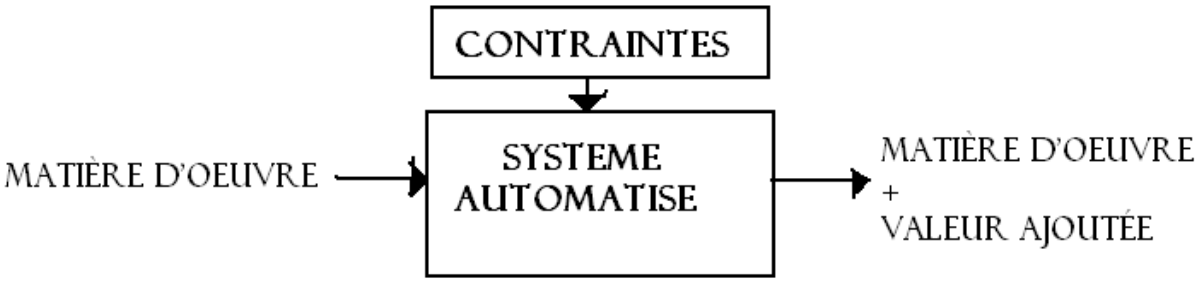
LES AUTOMATES PROGRAMMABLES INDUSTRIELS

1. Introduction [2]

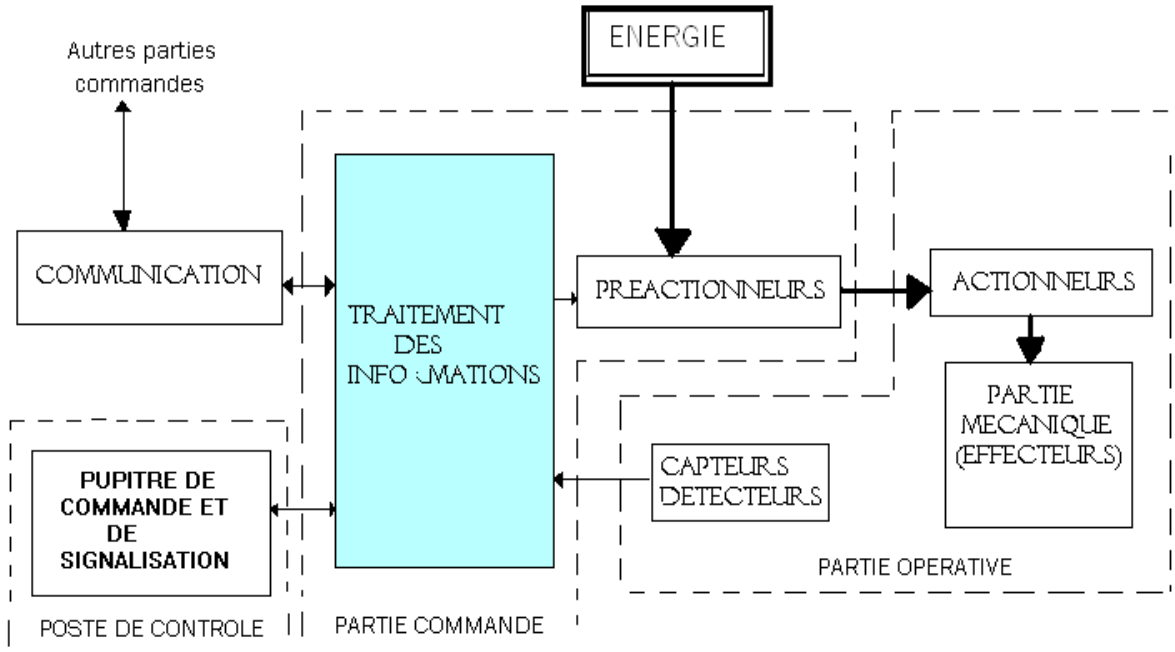
L'objectif de l'automatisation des systèmes est de produire, en ayant recours le moins possible à l'homme, des produits de qualité et ce pour un coût le plus faible possible.

Un système automatisé est un ensemble d'éléments en interaction, et organisés dans un but précis : agir sur une matière d'oeuvre afin de lui donner une valeur ajoutée.

Le système automatisé est soumis à des contraintes : énergétiques, de configuration, de réglage et d'exploitation qui interviennent dans tous les modes de marche et d'arrêt du système.



Tout système automatisé peut se décomposer selon le schéma ci-après :



- **Partie opérative** :

- Elle agit sur la matière d'oeuvre afin de lui donner sa valeur ajoutée.
- Les actionneurs (moteurs, vérins) agissent sur la partie mécanique du système qui agit à son tour sur la matière d'oeuvre.
- Les capteurs / détecteurs permettent d'acquérir les divers états du système.
-

- **Partie commande** :

- Elle donne les ordres de fonctionnement à la partie opérative.
- Les préactionneurs permettent de commander les actionneurs; ils assurent le transfert d'énergie entre la source de puissance (réseau électrique, pneumatique …) et les actionneurs. Exemple : contacteur, distributeur …
- Ces préactionneurs sont commandés à leur tour par le bloc traitement des informations.
- Celui-ci reçoit les consignes du pupitre de commande (opérateur) et les informations de la partie opérative transmises par les capteurs / détecteurs.
- En fonction de ces consignes et de son programme de gestion des tâches (implanté dans un automate programmable ou réalisé par des relais (on parle de logique câblée)), elle va commander les préactionneurs et renvoyer des informations au pupitre de signalisation ou à d'autres systèmes de commande et/ou de supervision en utilisant un réseau et un protocole de communication.

- **Poste de contrôle :**

- Composé des pupitres de commande et de signalisation, il permet à l'opérateur de commander le système (marche, arrêt, départ cycle …).
- Il permet également de visualiser les différents états du système à l'aide de voyants, de terminal de dialogue ou d'interface homme-machine (IHM).

2. Généralités sur les A.P.I.

L'automate programmable industriel (A.P.I) ou Programmable Logic Controller (PLC) est un appareil électronique programmable. Il est défini suivant la norme française EN-61131-1, adapte à l'environnement industriel, et réalise des fonctions d'automatisme pour assurer la commande de préactionneurs et d'actionneurs à partir d'informations logiques, analogiques ou numériques. C'est aujourd'hui le constituant essentiel des automatismes. On le trouve non seulement dans tous les secteurs de l'industrie, mais aussi dans les services et dans l'agriculture.

La force principale d'un automate programmable industriel API réside dans sa grande capacité de communication avec l'environnement industriel. Outre son unité centrale et son alimentation, il est constitué essentiellement de modules d'entrées/sorties, qui lui servent d'interface de communication avec le processus industriel de conduite.

Et il a comme rôles principaux dans un processus :

- D'assurer l'acquisition de l'information fournie par les capteurs;
- En faire le traitement ;
- Elaborer la commande des actionneurs ;
- Assurer également la communication pour l'échange d'informations avec l'environnement. **[3]**

3. Architecture des automates [4]

3.1. Aspect extérieur :

Les automates peuvent être de type **compact** ou **modulaire.**

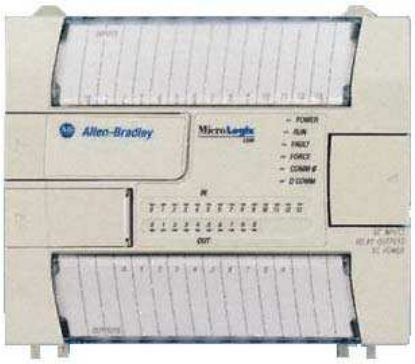
De type *compact*, on distinguera les *modules de programmation* (LOGO de Siemens, ZELIO de Schneider, MILLENIUM de Crouzet ...) des *microautomates*.

Il intègre le processeur, l'alimentation, les entrées et les sorties. Selon les modèles et les fabricants, il pourra réaliser certaines fonctions supplémentaires (comptage rapide, E/S analogiques ...) et recevoir des extensions en nombre limité.

Ces automates, de fonctionnement simple, sont généralement destinés à la commande de petits automatismes.

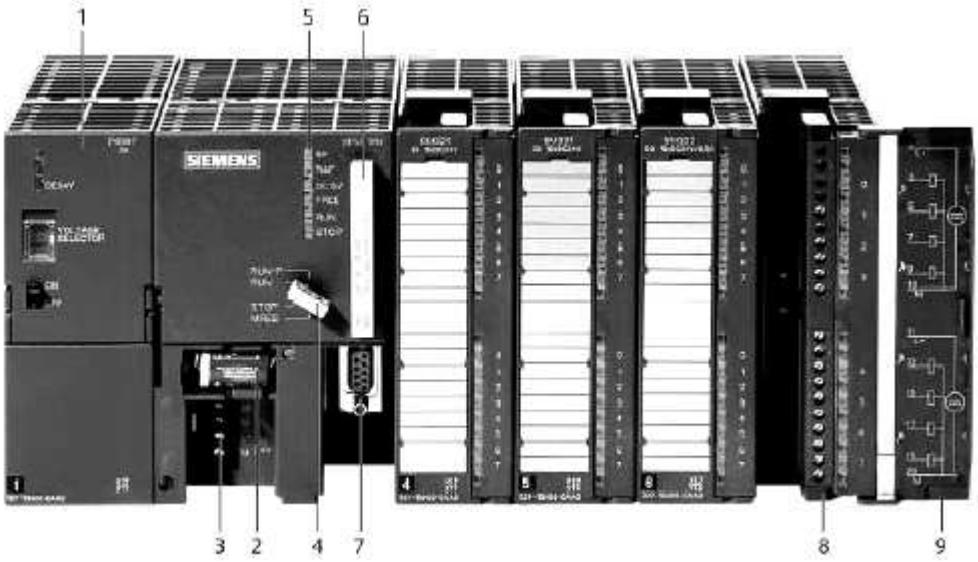
De type *modulaire*, le processeur, l'alimentation et les interfaces d'entrées / sorties résident dans des unités séparées (**modules**) et sont fixées sur un ou plusieurs **racks** contenant le "fond de panier" (bus plus connecteurs).

Ces automates sont intégrés dans les automatismes complexes où puissance, capacité de traitement et flexibilité sont nécessaires.



Automate compact (Allen Bradley)

Automate modulaire (Modicon)



Automate modulaire (Siemens)

1 Module d'alimentation
2 Pile de sauvegarde
3 Connexion au 24VCC
4 Commutateur de mode (à clé)
5 LED de signalisation d'état et de défauts
6 Carte mémoire
7 Interface multipoint (MPI)
8 Connecteur frontal
9 Volet en face avant

Figure I.1 : Aspect extérieur d'un automate

3.2. Structure interne :

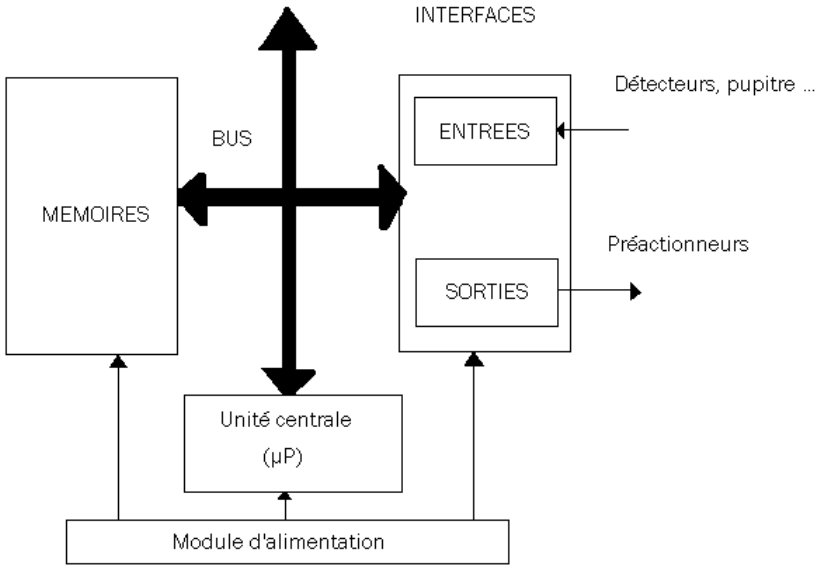


Figure I.2 Structure interne d'un automate

3.2.1. Le module d'alimentation

Il est composé de blocs qui permettent de fournir à l'automate l'énergie nécessaire à son fonctionnement, il convertit la tension du réseau (AC 220 V) en tension de service (DC 24V, 12V ou 5V) et assure l'alimentation de l'automate ainsi que circuits de charge.

Un voyant est positionné en générale sur la façade pour indiquer la mise sous tension de l'automate.

3.2.2. L'unité centrale "CPU"

La CPU est une carte électronique bâtie autour d'un ou plusieurs processeurs, elle comprend aussi des moyens de stockage, qui sert à sauvegarder les programmes et les données.

- **Le processeur**

Le processeur est chargé d'exécuter le programme utilisateur, il doit assurer des opérations logiques et arithmétiques ainsi que des fonctions de temporisation et de comptage. Il peut être issu de la technologie câblée ou de la technologie à microprocesseur.

En général un microprocesseur est composé d'une Unité Arithmétique et Logique (UAL), de registres, un Décodeur d'Instructions, un Compteur Programme et une horloge **[5].**

- **La mémoire**

La mémoire est l'élément fonctionnel qui peut recevoir, conserver et restituer l'information.

Elle est découpée en zones :

- une zone mémoire programme,

- une zone mémoire donnée.

- une autre pour les variables internes.

Pour un automate, il faut connaître la capacité mémoire minimale utile et la capacité maximale que l'on peut obtenir par diverses extensions.

3.2.3. Le module d'entrées/sorties

Le module E/S assure le rôle d'interface pour la partie commande, qui distingue une partie opérative (les sorties), où les actionneurs agissent physiquement sur le processus, et une partie d'acquisitions (les entrées) récupérant les informations sur l'état de ce processus et

coordonnant en conséquence les actions pour atteindre les objectifs prescrits (matérialisés par des consignes).

En plus d'assurer la communication entre la CPU et les organes externes, le module d'E/S doit garantir une protection contre les parasites électriques, c'est pour quoi la plus part des modules E/S font appel au découplage optoélectronique.

Il existe deux types d'interface E/S:

- Le module E/S Tout Ou Rien (TOR) **:** Permet de raccorder l'automate à des capteurs TOR (boutons poussoirs, fins de course, capteurs de proximité, capteurs photoélectriques …) ou à des préactionneurs (vannes, contacteurs, voyant pneumatique, électrovannes, relais de puissance, LED….). L'état de chaque entrée ou sortie est visualisé par une diode électroluminescente. Le nombre d'entrées sur une carte est de : 4, 8, 16, 32.
- Le module E/S analogique **:** Permet de traiter les signaux analogiques. Il est muni d'un convertisseur analogique/numérique pour les entrées et un autre numérique/analogique pour les sorties. Il existe des modules à 2, 4, 8 voies.

3.2.4. Les interfaces

- Le module de fonction "FM" (Les cartes spécialisés).

Le module de fonction ou «Function Module » est un module additionnel ou des cartes spécialisées peuvent être connectés. Ces cartes comportent un processeur spécifique ou une carte électronique spécialisée, elles assurent non seulement la liaison avec le monde extérieur mais aussi une partie du traitement pour soulager le processeur. On peut citer : les cartes d'axe, les concentrateurs de communication, les cartes E/S déportées, les cartes de comptage rapide, les cartes de pesage, les cartes de régulations PID…

- Le module de communication

Le module de communication comprend les consoles et les boîtiers de tests.

→Les consoles

Les consoles permettent la programmation, le paramétrage et les relevés d'informations, ils peuvent également afficher le résultat de l'autotest comprenant l'état des modules d'entrées et

de sorties, l'état de la mémoire, de la batterie, etc. Ils sont équipés (pour la plupart) d'un écran à cristaux liquides.

Pendant la phase de programmation les consoles permettent : l'écriture, la modification, l'effacement et le transfert d'un programme dans la mémoire de l'automate ou dans une mémoire EPROM.

Pendant la phase de réglage et d'exploitation elles permettent : de visualiser ou d'exécuter le programme pas à pas, de forcer ou de modifier les données (les entrées, les sorties, les bits internes, les registres de temporisation, les compteurs…).

Certaines consoles ne peuvent être utilisées que connectées à un automate car c'est ce dernier qui leurs fournit l'alimentation et la mémoire de travail, c'est les consoles de programmation On-line, avec ces consoles le programme introduit par l'utilisateur est directement mémorisé dans l'automate.

D'autres consoles peuvent fonctionner de manière autonome grâce à leurs mémoires interne et à leurs alimentations, c'est les consoles de programmation Off line, elles offrent un plus grand confort, le programme écrit de cette façon est appelé source, il est compilé par la console puis transféré dans la mémoire de l'automate. **[3]**

→Les boîtiers de tests

Les boîtiers de testes quand a eux sont destinées aux personnels d'entretien, ils permettent de visualiser le programme ou les valeurs des paramètres (affichage de la ligne de programme à contrôler, visualisation de l'état des entrées et des sorties…)

3.2.5. Les auxiliaires

Ils consistent principalement en :

- Un support mécanique (un rack) : l'automate se présentant alors sous forme d'un ensemble de cartes, d'une armoire, d'une grille et des fixations correspondantes.
- Un ventilateur : il est indispensable dans les châssis comportant de nombreux modules ou dans le cas où la température ambiante est susceptible de devenir assez élevée (plus de 40 °C).
- Un indicateurs d'état : il indique la présence de tension, l'exécution du programme (mode RUN), la charge de la batterie, le bon fonctionnement des coupleurs. [7]

4. Les langages normalisés :[6]

Un A.P.I. est programmé à l'aide de langages spécialisés, fournis par son constructeur et utilisables au travers d'une interface (un logiciel sur PC, un pupitre...). Ces langages peuvent être classés en 5 familles. Cependant, deux langages de la même famille et fournis par deux constructeurs différents ne sont pas forcément compatibles, ce qui est de nature à nuire à la portabilité des applications et à limiter la réutilisation du code. C'est pour cette raison que la commission électrotechnique internationale a entrepris un grand effort de normalisation visant à uniformiser les langages utilisés dans le domaine de la programmation des A.P.I., ce qui a donné naissance à la norme IEC (International Electrotechnical Commission) 61131-3 [IEC93]. Ce standard définit cinq langages correspondant aux familles de langages les plus utilisées pour la programmation des API.

Les familles peuvent être classées en 2grandes catégories :

- **Langages textuels:**

- ST « structured text » ou texte structuré

- IL « Instruction List » ou Liste d'instructions.

- **Langages graphiques:**

- SFC « Sequential Function Chart » ou GRAFCET

- LD « Ladder Diagram »ou schéma à relais

- FBD « Function Block Diagram »ou schéma par bloc.

4.1. Langages textuels

4.1.1. Liste d'instructions (IL)

Le langage IL est un langage textuel de bas niveau (proche du langage machine), qui utilise un jeu d'instruction simple, il trouve sa puissance dans les applications de petites tailles, et dans la création de sous programme ou procédure, car il permet un contrôle totale et une optimisation parfaite du code, par contre dans les grandes applications il est très difficile de programmer avec le IL, les programmes dans ce langage peuvent être traduit ou déduit des autres langages. Le IL a la même structure que l'assembleur, il utilise un ou plusieurs registres de travail. Les valeurs intermédiaires nécessaires pour

l'exécution d'une instruction donnée seront mémorisées dans ces registres le temps de leurs utilisations et il possède un jeu d'instruction assez riche pour décrire toutes les opérations arithmétiques et logiques, les opérations de comptage et temporisation, la comparaison et le transfert…Par exemple pour réaliser l'opération $h = w * \frac{x+y}{z}$ on utilise le code suivant :

```
LD x
LD y
+R
LD w
*R
LD z
/R
ST h
```

Figure.I.3. Exemple en langage LIST

4.1.2. Texte Structuré (ST) :

Le langage ST (Structured Text) est un langage de programmation textuel de haut niveau dédié aux applications d'automatisation, il est utilisé principalement pour décrire les procédures complexes et difficilement modélisables avec les langages graphiques, il peut aussi être utilisé en tant que sous programme avec d'autre langage de programmation. Il utilise les même énoncés que les langages de programmation de haut niveau (Pascal, C, C++…) comme: les assignations, les appels de fonction, les énoncés de contrôle (IF, THEN, ELSE, CASE) ou d'itération (FOR, WHILE, REPEAT) en plus des opérations arithmétiques et logiques.

Par exemple pour le calcul de l'écart entre deux points dans un plan cartésien, on exécute le programme suivant

```
FUNCTION  Ecart: REAL
VAR_INPUT
X1, X2, Y1, Y2 : REAL;
END_VAR
BEGIN
RESULT := SQRT((X1-X2)^2 + (Y1-Y2)^2);
END_FUNCTION
```

Figure.1.4. Exemple en langage structuré

4.2. Langages graphiques :

4.2.1. GRAFCET :

L'acronyme GRAFCET signifie: GRAphe Fonctionnel de Commande Etape Transition (SFC Sequential Function Chart). C'est une méthode de représentation graphique permettant de décrire le cahier de charge d'un automatisme. Il est adapté aux systèmes à évolution séquentielle, il est défini par un ensemble d'éléments graphiques de base traduisant le comportement de la partie commande vis-à-vis de ses entrées et de ses sorties. Un programme GRAFCET décrit un procédé comme une suite d'étapes, reliées entre elles par des transitions.

À chaque transition est associée une réceptivité, celle-ci est une condition logique qui doit être vraie pour franchir la transition et passer à l'étape suivante. Des actions sont associées aux étapes du programme. Le format graphique d'un programme GRAFCET est le suivant :

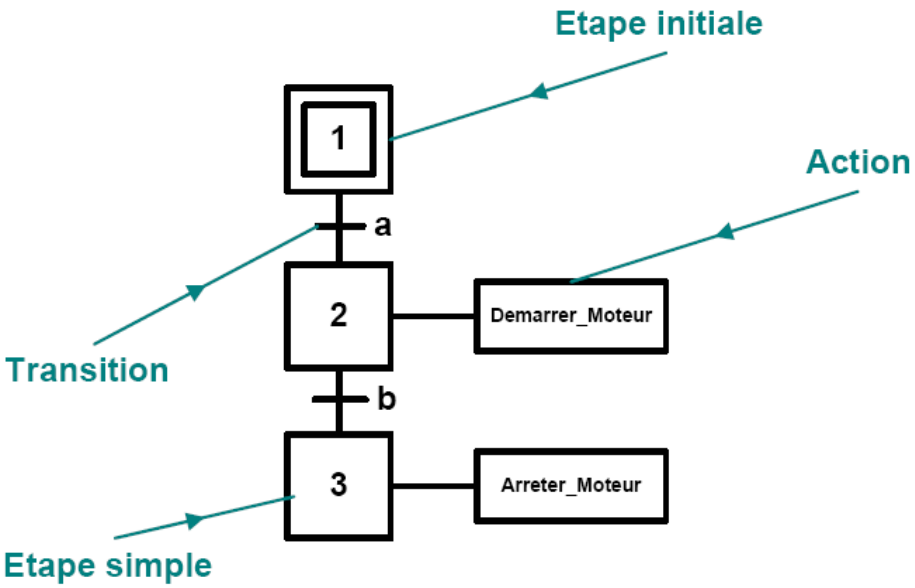


Figure I.5. Exemple en langage GRAFCET

Une étape représentée par un carré qui a un numéro identificateur et les actions associées sont indiquées dans un rectangle relié à la partie droite du carré; (l'étape initiale est représentée par un carré double).

Une liaison orientée représentée par une ligne, parcourue par défaut de haut en bas ou de gauche à droite.

Une transition entre deux étapes à qui est associé une réceptivité inscrite a ça droite, est représentée par une barre perpendiculaire aux liaisons orientées qui relient ces étapes.

4.2.2. LADDER DIAGRAM :

Le LD est une représentation graphique qui traduit directement des équations booléennes en un circuit électrique en combinant des contacts et des relais à l'aide des connexions horizontales et verticales, les contacts représentent les entrées (contact normalement ouvert, contact normalement fermé,…) et les relais représentent les sorties (relais directs, relais inversés,…), les diagrammes LD sont limités sur la gauche par une barre d'alimentation et par la masse sur la droite.

Par exemple pour réaliser la fonction logique

$$x = (a + b) * (\bar{c} + \bar{d} * e)$$

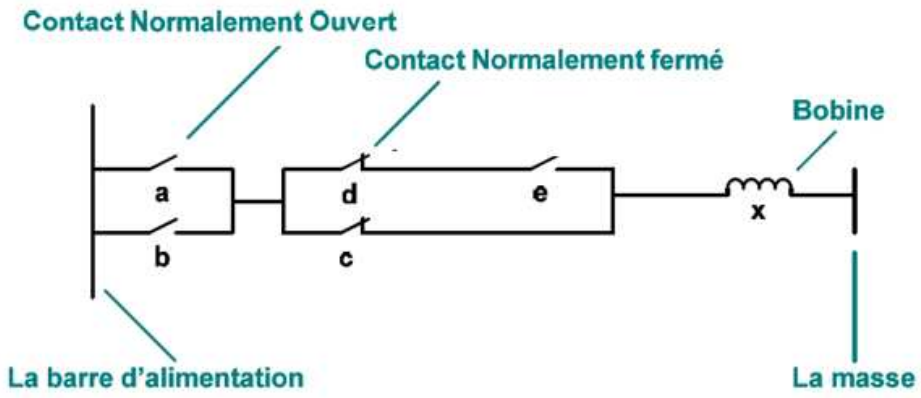


Figure I.6 : Exemple en langage LADDER

Le langage LD propose d'autre type de fonctions tel que les fonctions de comptages et de temporisations, les fonctions arithmétiques et logiques, les fonctions de comparaison et de transfert. Ces fonctions sans incluses dans tous les A.P.I.

4.2.3. Bloc de foncions(FBD)

C'est un langage graphique qui permet la construction d'équations complexes à partir des opérateurs standard, ou de blocs fonctionnels, il se compose de réseaux de fonctions préprogrammées ou non, représentées par des rectangles qui sont connectés entre eux par des lignes.

La programmation avec le FBD est très souple et facile à apprendre, la plupart des fonctions nécessaires (les fonctions arithmétique et logique, les fonctions de temporisation, des blocs, fonctionnels PID…) sont déjà disponibles dans la bibliothèque, il suffit juste de les connecter et bien paramétrer les entrées et les sorties, c'est-à-dire respecter le type des variables lors de la connexion.

Par exemple pour réaliser l'opération arithmétique $w = 20(\frac{x+y}{z})$ on aura besoin de trois blocs: un pour addition, un pour la multiplication et un autre pour la division.

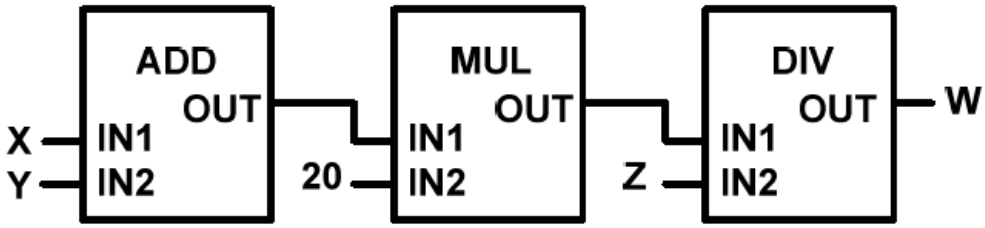


Figure I.7 : Exemple en langage FBD

5. La gamme Schneider Electric en automates programmables [8]

5.1. ZELIO LOGIC :

C'est un module logique programmable destiné à la réalisation de petits automatismes de moins 40 E/S pour des applications telles que :

- comptage de pièces
- commande de pompes et de compresseurs
- convoyeurs
- panneaux d'affichage
- volets roulants.

Sa programmation, très intuitive, s'effectue directement sur la face ou à l'aide du logiciel Zelio Soft. Existe en version sans afficheur ni touches.

Les principales caractéristiques du ZELIO sont :

- gamme compacte : 3 modèles monoblocs de 10, 12, 20 E/S (avec ou sans afficheur et touches) ;
- gamme modulaire : 2 bases de 10 ou 26 E/S extensibles jusqu'à 40 E/S par modules d'extension de 6, 10 ou 14 E/S et un module de communication Modbus ;
- alimentation : 100/240 VCA ou 24 VCC ;
- fixation sans accessoire avec nez modulaire de 45 mm ;
- blocs fonctions prêts à l'emploi ;
- logiciel de programmation sur PC.

5.2. TWIDO :

Version compacte ou modulaire partage des options, des extensions d'E/S et un logiciel de programmation communs. Application : installations simples et unitaires, machines répétitives et compactes.

Les principales caractéristiques sont :

- Applications standards de 10 à 100 E/S (maximum 252 E/S), 2 types de bases : compactes de 6, 10 ou 24 E/S et modulaires de 20 ou 40 E/S ;
- Extension d'E/S : 14 modules TOR et 4 analogiques ;
- Raccordement : borniers à vis ou à ressort, précâblage : connecteur HE10, Twido Fast, déport des E/S, maître AS-interface ;
- Options : mémoire, horodateur, communication (liaison série), réglage (afficheur)
- Compacité : 40 E/S pour 95x90x70 mm
- Fonctionnalités : compteurs rapides, sorties impulsionnelles, 1 port RS 485 multiprotocoles : Modbus Maître/esclave, ASCII, traitement rapide sur événements, type de donnés (mots, double mots, arithmétique flottante), jusqu'à 14 boucles PID…

5.3. TSX MICRO :

C'est un Automate développé pour satisfaire au mieux les exigences d'adaptabilité et de maintenabilité des machines. Sa modularité et sa compacité répondent de manière économique à l'automatisation aussi bien de machines simples à quelques dizaines d'E/S que de machines plus complexes à 480 E/S. Pour simplifier le câblage des machines, le TSX MICRO supporte la connexion du bus AS-i. il se programme à l'aide du logiciel PL7 Micro sous Windows.

Les principales caractéristiques sont:

- Jusqu'à 248 E/S TOR « in rack »;
- 96 E/S par déport TSX Nano ;
- 248 E/S sur bs AS ;

- Fonctions de comptage rapide (50kHz), analogique et régulation ;
- Sécurité machine ;
- Communication Uni-Telway, ASCII, Modbus, Fipway, FIPIO agent, modem.

5.4. PREMIUM :

Cette plate-forme d'automatismes, grâce à son architecture distribuée permet de répartir les cartes d'E/S et les fonctions métiers au plus près de l'application. Idéale pour les machines et installations modulaires et réparties, la gamme TSX Premium se décline :

- En format standard (TSX).
- EN format co-processeur pour PC.

L'intégration logicielle et matérielle des métiers permet de diminuer les temps de développement et de mise en route de l'installation. Enfin TSX Premium s'intègre à tous les niveaux de l'architecture de communication. Il se programme à l'aide des logiciels Unity Pro ou PL7.

Ses principales caractéristiques sont :

- Jusqu'à 2048 E/S TOR « in rack »;
- E/S à distance sur bus FIPIO et tiers ;
- 8 coupleurs AS-i, 248 E/S sur bus AS-I;
- Fonction de comptage rapide (1 MHz), analogique et régulation intégrée par configuration ;
- Sécurité machine ;
- Commande de mouvement multiaxe ;
- Communication Uni-Telway, ASCII, Modbus, Fipway,FIPIO, Ethernet;
- TCP-IP, Ethway;

5.5. QUANTUM :

La plate-forme d'automatismes TSX Quantum, grâce à son architecture modulaire s'adapte à l'ensemble des applications du process exigeant de la disponibilité, une grande puissance de calcul et un nombre important de d'E/S. Le TSX Quantum se programme à l'aide des logiciels Unity Pro ou Concept.

Ses principales caractéristiques :

- Jusqu'à 64000 E/S décentralisées sur plusieurs km ;
- Option de redondance sur alimentation, CPU et communications ;
- Fonctions e comptage, commande d'axe sur bus SERCOS ;
- Communication ASCII, Modbus plus, Ethernet TCP-IP, bus tiers.[cata] ;

6. Les différents modules de l'automate PREMIUM [9]

On s'intéresse à l'étude des différents modules de l'automate Premium. Dans les exemples de programmation en UNITY PRO, on choisira cet automate dans la configuration matérielle, on aura donc besoin de savoir quel module choisir.

6.1. Module d'alimentation

On prendra comme exemple pour illustrer ce module, les modules d'alimentation **TSX PSY…**, qui sont destinés à l'alimentation de chaque rack TSX RKY…et de ses modules. Le module d'alimentation est choisi en fonction du réseau de distribution (courant alternatif ou courant continu) et de la puissance nécessaire (modèle standard ou double format).On distingue les modules d'alimentation pour réseaux à courant continu et les modules pour réseaux à courant alternatif **[Annexe B.1.].**

Les modules peuvent être de format standard (TSX PSY 2600 et TSX PSY 1610) ou double format (TSX PSY 5500/3610/5520/8500) comme le montre la **Figure I.8**, le tableau **I.1** montre la composition générale de ces modules :

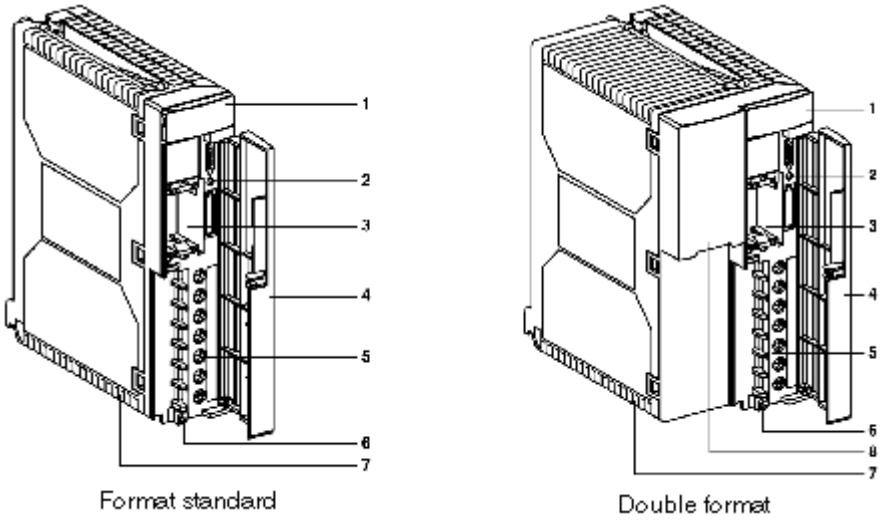


Figure I.8 : Dessin d'illustration des modules d'alimentation

Numéro	**Fonction**
1	Bloc de visualisation comprenant : -un voyant OK (vert), allumé si les tensions sont présentes et correctes -un voyant BAT (rouge), allumé lorsque la pile s'épuise ou est absente, -un voyant 24V (vert), allumé si le capteur de tension est présent. Ce voyant n'est présent que sur les modules d'alimentation à courant alternatif TSX PSY 2600/5500/8500.
2	Bouton RESET à pointe de crayon qui, lorsqu'il est actionné, déclenche une reprise à chaud de l'application.
3	Emplacement pour la pile permettant de sauvegarder la RAM interne du processeur.
4	Volet assurant la protection de la face avant du module
5	Bornier à vis pour le raccordement de : -réseau d'alimentation -contact relais alarme -alimentation capteur pour les alimentations à courant alternatif TSX PSY 2600/5500/8500

6	Orifice permettant le passage d'un collier de serrage des câbles
7	Fusible situé sous la protection du module : -tension 24VR sur l'alimentation à courant continu TSX PSY 3610 -tension primaire sur l'alimentation à courant continu TSX PSY 1610 **Remarque** : sur les modules TSX PSY 2600/5500/5520/8500, le fusible de protection de la tension primaire se trouve à l'intérieur du module et il n'est pas possible d'y accéder.
8	Sélecteur de tension 110/220, présent uniquement sur les alimentations à courant alternatif TSX PSY 5500/8500. A la livraison, le sélecteur est positionné sur 220.

Tableau I.1 : Composants d'un module d'alimentation.

6.2. Modules d'entrées/de sorties analogiques

Dans ce qui suit, nous allons expliciter les modules analogiques Premium. Ce sont des modules au format standard, qui occupent une seule position dans les racks TSX RKY•••. Ils peuvent s'implanter dans toutes les positions sur le rack à l'exception des deux premières (PS et 00) réservées respectivement au module d'alimentation du rack (TSX PSY•••) et au module processeur (TSX 57•••). Ils sont de deux types :

→***Entrées haut niveau tension / courant***, thermocouples et thermosondes. Les modules d'entrées offrent :

- 16 voies pour les TSX AEY 16•• ;
- 8 voies pour les TSX AEY 8•• ;
- 4 voies pour les TSX AEY 4••.

→***Sorties haut niveau tension / courant*** sur des voies isolées ou à point commun. Les modules de sorties offrent :

- 8 voies pour le TSX ASY 800 ;
- 4 voies pour le TSX ASY 410.

Ils sont équipés d'un connecteur Sub-D 25 points (TSX AEY 420/800/810 et TSX ASY 800), de deux connecteurs Sub-D 25 points (TSX AEY 1600/1614) ou d'un bornier à vis (TSX AEY 414 et TSX ASY 410).

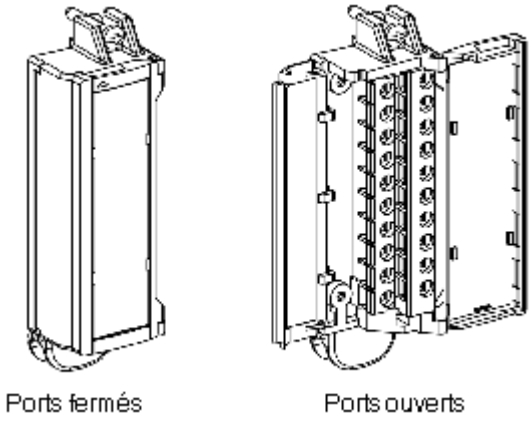


Figure I.9 : Connecteur sub-D à 25 points **Figure 1.10 :** Bornier à vis TSX BLY 01

6.2.1. Les différents éléments des modules analogiques à connecteur(s) Sub-D

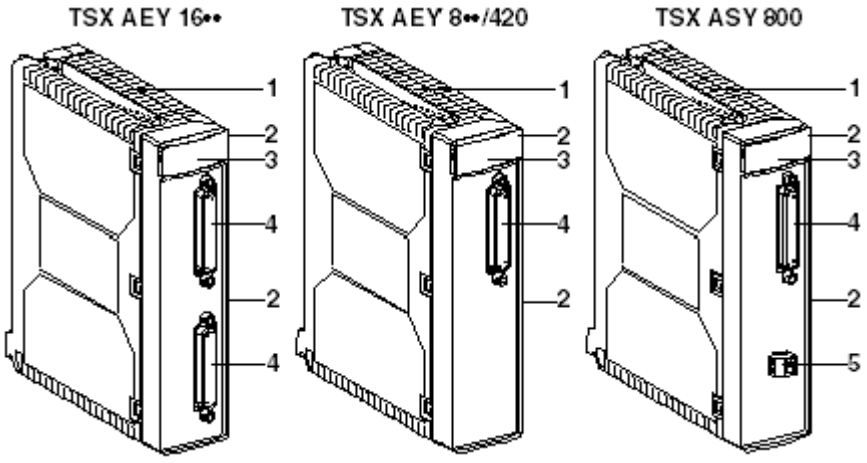


Figure I.11 : Les différents modules à connecteur(s) sub-D

Numéro	Descriptif
1	Corps rigide qui assure les fonctions de support et de protection de la carte électronique.
2	Marquages de référence du module (visible en face avant et sur le côté droit du module).
3	Marquages de référence du module (visible en face avant et sur le côté droit du module).
4	Connecteur Sub-D 25 points, pour le raccordement des capteurs ou des pré-actionneurs.
5	Bornier d'alimentation 24 VDC externe.

Tableau I.2 : Composants des modules d'entrée /sortie analogique.

6.2.2. Les différents éléments des modules analogiques à connecteur(s) Sub-D

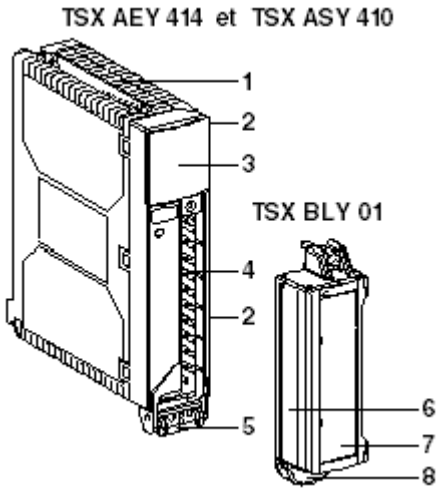


Figure I.12 : Les différents modules à bornier à vis

Numéro	Descriptif
1	Corps rigide qui assure les fonctions de support et de protection de la carte électronique.
2	Marquages de référence du module (visible en face avant et sur le côté droit du module).
3	Bloc de visualisation des modes de marche et des défauts.
4	Connecteur recevant le bornier à vis TSX BLY 01.
5	Codeur du module.
6	Bornier à vis (TSX BLY 01) débrochable, pour le raccordement des capteurs ou des pré-actionneurs.
7	Volet d'accès aux bornes à vis ; sert également de support pour l'étiquette de câblage du bornier et le marquage des voies.
8	Codeur du bornier.

Tableau I.3 : Composants des modules analogiques à bornier à vis

Un catalogue de ces modules est donné en annexe **[Annexe B.2.].**

6.2.3. Module d'entrées/de sorties TOR :

Les modules d'E/S TOR de la gamme Premium possèdent un format standard et occupent une seule position. Ils sont équipés d'un connecteur HE10 ou d'un bornier à vis (TSX BLY 01).

Pour les modules munis de sorties à connecteurs HE10, il existe une gamme de produits appelés TELEFAST 2 permettant le raccordement rapide des modules d'entrées/sorties TOR aux parties opératives.

Une large gamme d'entrées et de sorties TOR permet de répondre aux besoins rencontrés au niveau :

- fonctionnel : entrées/sorties continues ou alternatives, logique positive ou négative ;

- de la modularité : 8, 16, 32 ou 64 voies par module.

Les entrées reçoivent les signaux en provenance des capteurs et réalisent les fonctions suivantes :

- acquisition ;
- adaptation ;
- isolement galvanique ;
- filtrage ;
- protection contre les signaux parasites.

Les sorties réalisent les fonctions de mémorisation des ordres donnés par le processeur, pour permettre la commande des pré-actionneurs au travers de circuits de découplage et d'amplification.

6.2.3.1. Modules TOR avec raccordement par bornier à vis

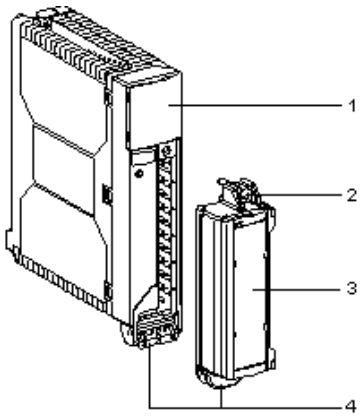


Figure 1.13 : Illustration d'un module d'E/S TOR à bornier à vis

Numéro	Descriptif
1	Bloc de visualisation et de diagnostic du module.
2	Bornier à vis débrochable pour raccordement direct des entrées/sorties aux capteurs et pré-actionneurs (référence : **TSX BLY 01**). Certains modules des sorties disposent de fusibles intégrés, accessibles de l'avant lorsque le bornier est enlevé.
3	Porte pivotante permettant l'accès aux vis du borner servant également de support à l'étiquette de repérage.
4	Support rotatif comprenant le dispositif de détrompage.

Tableau I.4 : Composants d'un module d'E/S TOR à bornier à vis.

6.2.3.2. Module d'E/S TOR avec raccordement par connecteur HE10

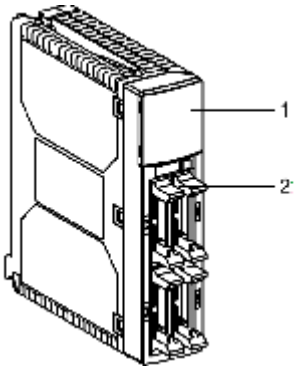


Figure I.14 : Illustration du module d'E/S TOR avec connecteur HE1O

Numéro	Descriptif
1	Bloc de visualisation et de diagnostic du module.
2	Connecteur **HE10**, protégé par un capot. Ils permettent le raccordement des entrées/sorties aux capteurs et pré-actionneurs soit directement soit par l'intermédiaire d'embases de raccordement TELEFAST 2.

Tableau I.5 : Composants module d'E/S TOR avec connecteur HE1O

Un catalogue de ces modules est donné en annexe **[Annexe B.3.].**

6.2.3.3. Module de déport de bus X :

Le bus de l'automate Premium permet de raccorder 8 racks de 12 emplacements (TSX RKY 12EX) ou 16 racks de 4, 6 ou 8 emplacements (TSX RKY 4EX/6EX/8EX) répartis sur une longueur de 100 mètres maximum.

Dans le cas d'applications nécessitant des distances plus élevées entre les racks, le module d'extension du bus X (TSX REY 200) permet d'augmenter de façon considérable cette distance tout en conservant les caractéristiques et la performance d'une station automate composée d'un seul segment de bus X sans module d'extension.

Le système se compose des éléments suivants :

- un module d'extension de bus X (TSX REY 200) appelé « maître » situé sur le rack ayant l'adresse 0 (rack hébergeant le processeur) et sur le segment principal du bus X. Ce module possède deux voies permettant d'étendre les deux segments du bus X à une distance maximum de 250 mètres,
- un ou deux modules TSX REY 200 modules appelés « esclave » et chacun situé sur un rack sur les segments de bus étendus,
- chacun des modules esclave est raccordé au module maître par un câble TSX CBRY 2500 équipé de connecteurs TSX CBRY K5.

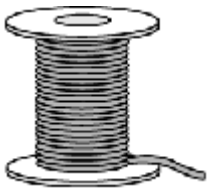


Figure I.15 : Câble TSX CBRY 2500

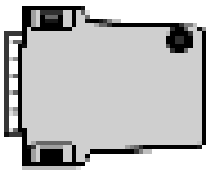


Figure I.16 : Connecteur TSX CBRY K

Description physique:

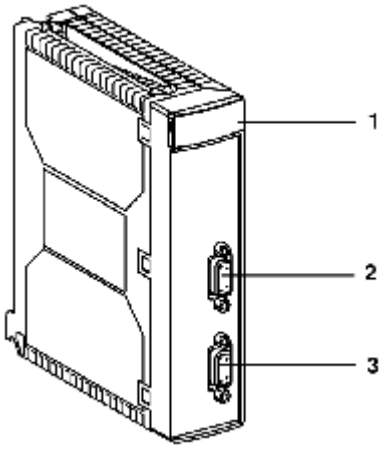


Figure I.17 : Schéma descriptif du module de déport

Libellé	Description
1	Bloc de visualisation composé de 6 voyants : • Voyant RUN : indique l'état de fonctionnement du module. • Voyant ERR : signale un défaut à l'intérieur du module. • Voyant I/O : signale un défaut extérieur au module. • Voyant MST : indique l'état de la fonction maître ou esclave du module. • Voyant CH0 : indique l'état de fonctionnement de la voie 0. • Voyant CH1 : indique l'état de fonctionnement de la voie 1.
2	Connecteur pour la liaison de la voie 0 du module.
3	Connecteur pour la liaison de la voie 1 du module.

Tableau I.6 : Composants d'un module de déport

Remarques :

- Le module maître s'installe obligatoirement sur le rack qui supporte le processeur (rack d'adresse 00) ; ce rack étant situé sur le segment de bus X principal, et dans une position libre de ce rack.
- Le module esclave s'installe sur l'un des racks du segment de bus déporté et dans une position quelconque de ce rack en dehors de la position dédiée au module d'alimentation.
- La configuration du module comme maître ou esclave est automatique :
    1. si le module est installé sur le rack d'adresse 0, il sera automatiquement déclaré comme maître.
    2. si le module est installé sur un rack d'adresse différente de 0, il sera automatiquement déclaré comme esclave.
- Si deux racks sont déclarés à l'adresse 0, le module maître doit être positionné sur le rack hébergeant les adresses de module « basses ».
- Si deux racks sont déclarés à l'adresse 0, le rack hébergeant les modules possédant les adresses « hautes » ne peut pas accueillir de module d'extension esclave.

Adresses modules « basses » :

- adresses 0 à 6 sur TSX RKY 8EX
- adresses 0 à 4 sur TSX RKY 6EX
- adresses 0 à 2 sur rack TSX RKY 4EX

Adresses modules « hautes » :

- adresses 8 à 14 sur rack TSX RKY 8EX
- adresses 8 à 12 sur rack TSX RKY 6EX
- adresses 8 à 10 sur rack TSX RKY 4EX

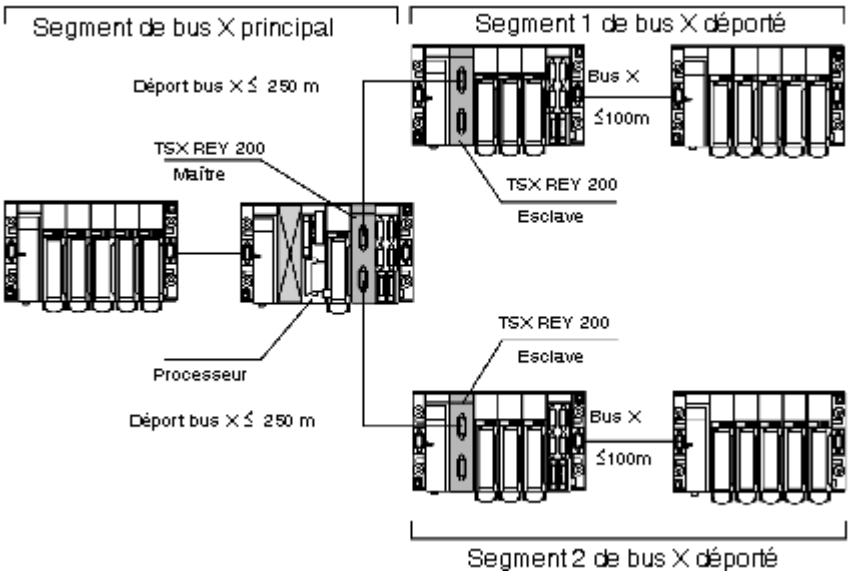


Figure I.18 : Station Premium avec module de déport

On voit bien que la distance maximale entre le processeur et les modules les plus éloignés peut être de 350 mètres. Cette distance de 350 mètres n'est possible que pour les modules d'entrées/sorties TOR simples. Les restrictions seront en fonction du type de module.

6.2.3.4. Module de comptage

On va prendre comme exemple les modules TSX CTY 2A, TSX CTY 4A et TSX CTY 2C, qui sont des modules de comptage standards. Ils ont utilisés pour compter les impulsions d'un capteur à une fréquence maximale de 40 KHz (CTY 2A/4A) ou de 1MHz (CTY 2C).

L'application de comptage permet d'effectuer un comptage rapide à l'aide de coupleurs, d'écrans Unity Pro et d'objets langage spécialisés.

Le contexte physique dans lequel le comptage va s'effectuer doit être défini (rack, alimentation électrique, processeur, modules ou périphériques, etc.) au cours de l'installation, puis la mise en œuvre logicielle doit être effectuée.

Les modules de comptage peuvent être installés dans tout emplacement disponible d'une configuration d'automates Premium.

Capteurs utilisés sur les voies :

- un capteur de proximité 2 ou 3 fils, de type PNP ou NPN. Si une sortie à contact mécanique est utilisée, il est nécessaire d'augmenter l'immunité de la voie, afin d'atténuer les rebonds de fermeture du contact ;
- un codeur de signal incrémental avec sorties différentielles 5 VCC (codeur avec émetteurs RS 422/485) ;
- un codeur de signal incrémental avec sortie 10 à 30 VCC (codeur Totem Pole) ;
- un codeur absolu à sorties série et interface RS 485 standard (TSX CTY 2C uniquement) ;

- un codeur absolu à sorties parallèles, à l'aide de l'adaptateur TELEFAST :

ABE-7CPA11 (TSX CTY 2C uniquement).

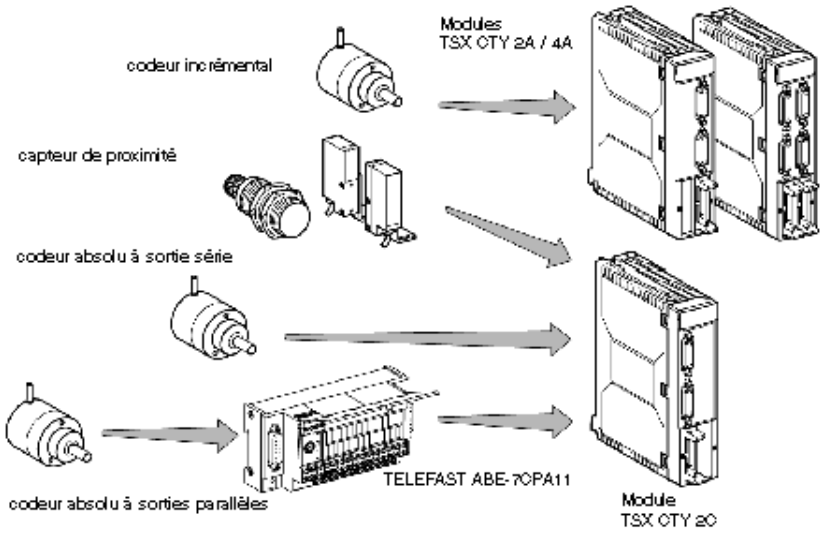


Figure I.19 : Capteurs utilisés pour les modules de comptage.

Description physique :

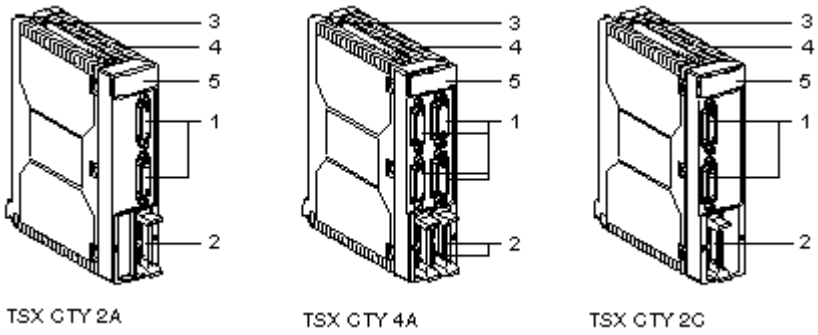


Figure I.20 : Illustration des modules TSX CTY 2A/4A 2C.

Repère	Description
1	Connecteur SUB D 15 broches standard permettant de raccorder : • le ou les capteurs de comptage associés aux voies 0 et 1 pour les modules TSX CTY 2A/2C et aux voies 0, 1, 2 et 3 pour le module TSX CTY 4A ; • l'alimentation du codeur lors de l'utilisation de ce type de capteur ; • le retour d'alimentation du codeur, qui sert à vérifier que ce dernier reçoit l'alimentation appropriée.
2	Connecteurs HE10 20 broches, utilisés pour chaque voie pour raccorder : • des entrées auxiliaires : o RAZ ou réglage sur la valeur prédéfinie ; o confirmation du comptage ; o capture. • des sorties auxiliaires ; • des alimentations externes : o alimentation entrée et sortie auxiliaire ; o alimentation d'autres capteurs.
3	Vis de fixation du module
4	Corps rigide, garantissant : • la gestion de la carte électromagnétique ; • le verrouillage du module dans son emplacement.
5	Voyants de diagnostic du module : • diagnostic de niveau module : o voyant RUN vert : indique le mode opératoire du module (module opérationnel) ; o voyant ERR rouge : indique l'état interne du module (erreur interne, module cassé) ; o voyant I/O rouge : indique une erreur externe ou un défaut applicatif. • diagnostic de niveau voie du module : o voyant CHx vert : indique le diagnostic de la voie : - voyant allumé : voie active ; - voyant clignotant : voie inactive ; - voyant éteint : voie non opérationnelle, non configurée ou configurée de manière erronée.

Tableau I.7 : Constituants des modules de comptage

Pour voir la mise en place des modules sur le rack, voir l'annexe **[Annexe B.4]**

7. Conclusion

Dans une conduite de processus, les informations provenant de capteurs sont acquises puis traitées par des programmes informatiques associés à des calculateurs (Automates). Les résultats des traitements permettent une action sur les systèmes au travers d'actionneurs agissant ainsi sur le processus physique **[10].**

L'étude des différents automates de la même gamme nous aide donc à mieux classer les produits d'automatisation qu'on aura à manipuler et, donc, proposer une solution optimale à un problème d'automatisation.

CHAPITRE II

UNITY PRO

1. Introduction :

Unity Pro est le logiciel commun de programmation, mise au point et exploitation des gammes d'automates Premium, Atrium, Quantum.

Il reprend toutes les valeurs d'usage reconnues des logiciels PL7 et Concept et propose un ensemble complet de nouvelles fonctionnalités pour plus de productivité et d'ouverture vers les autres logiciels.

Les cinq langages IEC6113-3 sont supportés en standard dans Unity Pro avec toutes les fonctions de mise au point, sur le simulateur ou directement en ligne avec l'automate.

Il permet de structurer une application pour les plates-formes Atrium, Premium et Quantum en modules fonctionnels composés de :

- Sections (code programme).
- Tables d'animation.
- Ecrans d'exploitation.

2. Création d'un projet sous UNITY PRO :

2.1. Configuration matérielle

2.1.1. Choix du processeur

Il peut être simple ou double format, il occupe respectivement une ou deux adresses. Il gèrel'ensemble d'une station automate constituée de :

Modules d'entrées/de sorties TOR ;

- Modules d'entrées/de sorties analogiques ;
- Modules métiers (comptage, commande d'axes, commande pas à pas, communication...) qui peuvent être répartis sur un ou plusieurs racks connectés au bus X .

Le nombre d'entrées/sorties pouvant être gérés diffère d'un processeur à un autre, donc il faut savoir le nombre d'entrées/sorties dont on a besoin avant de choisir notre processeur **[ANNEXE C1].**

Le choix du processeur est la première étape à réaliser pour créer une application, pour ce faire il faut :

- Choisir la plate-forme : Premium ou Quantum (non interchangeable)
- Choisir le type de processeur

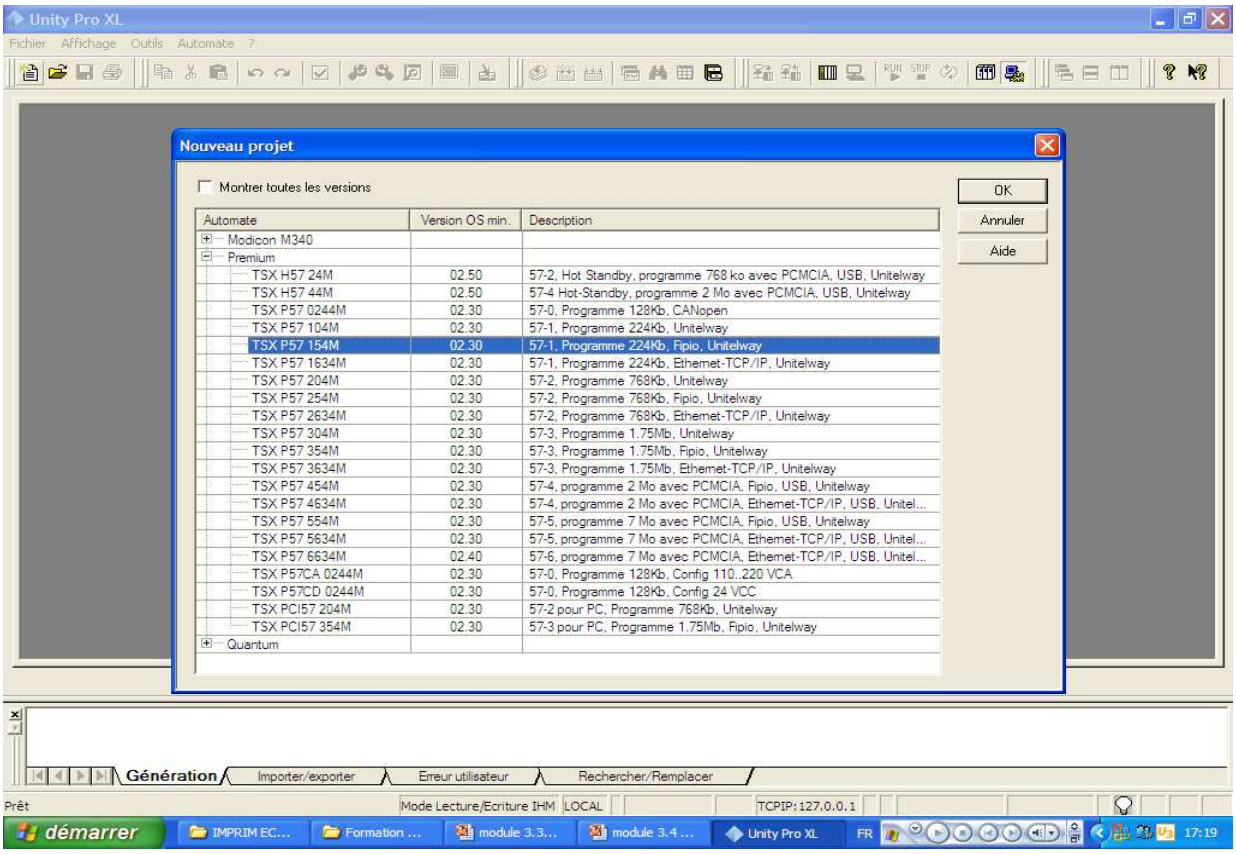


Figure II.1 : Choix du processeur

Remarque : Pour une station Premium, le processeur occupe l'emplacement 0 et il peut occuper l'emplacement 1 si une alimentation double format est configurée.

On peut remplacer un processeur grâce à l'éditeur de configuration, si un remplacement n'est pas autorisé, un message d'erreur sera affiché. Le nouveau processeur doit obligatoirement appartenir à la même famille d'automate que le processeur précédemment configuré.Si certains modules d'entrée/sortie précédemment configurés ne sont plus pris en

charge par le nouveau processeur, des messages d'erreur s'affichent lors de l'analyse du projet. Il est possible de remédier à ces incompatibilités.

Cette opération ne peut s'effectuer qu'en mode local (automate non connecté), la figure suivante montre comment faire pour le remplacement.

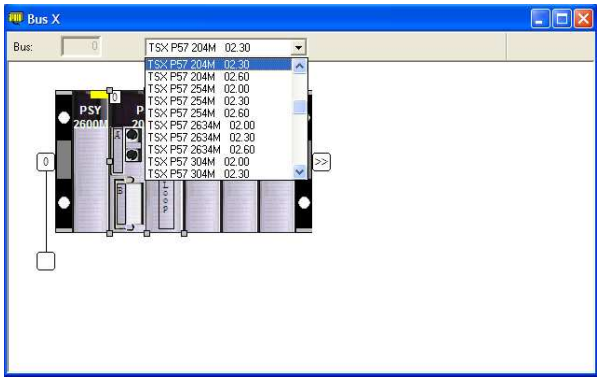


Figure II.2 : Liste des processeurs pour un remplacement.

Type de processeurs			**TSX P57 103M**	**TSX P57 153M**	**TSX P57 203M**	**TSX P57 2623M**	**TSX P57 253M**	**TSX P57 2823M**
Configuration maximale	Nb de racks	4/6/8 Emplacements	4		16			
		12 emplacements	2		8			
	Nb d'emplacements maximal pour modules		32		128			
Fonctions	Nb maximal	E/S TOR	512		1024			
		E/S analogiques	24		80			
		Voies de régulation	-		10 (jusqu'à 30 boucles simples)			
		Voies métiers	8		24			
	Connexions intégrées	Ethernet TCP/IP	-			1	-	1
		Fipio gestionnaire	-	1 (63 agents)	-		1 (127 agents)	
		Liaison série	1 liaison avec 2 connecteurs (TER et AUX) 19.2 Kbit/s					
	Nb max de connexions	Réseaux	1		1	1, aucun si Ethernet intégré utilisé	1	1, aucun si Ethernet intégré utilisé
		Bus AS-i	2		4			
		Bus CANopen	1	-	1			
		Bus InterBus ou Profibus DP	-		1, aucun si CANopen utilisé			
Mémoires	Capacité maximale	Sans carte PCMCIA(Kmots)	32, programme et données		48, programme et données		64, programme et données	
		Avec carte PCMCIA(Kmots)	64, programme 32, données		160,programme 48, données		160, programme 64, données	
		Stockage de données (Kmots)	128		2688			
	Taille max des zones objets	Bits internes localisés (%Mi) (bits)	4096		8132			
		Données internes Localisées (Kmots)	30.5 pour mots internes %M•i 32 pour mots constants %K•i					

Tableau II.1 : Exemple d'un automate premium avec les principales caractéristiques

2.1.2. Configuration d'un rack

Lors de la création d'un projet, un rack par défaut est sélectionné. Son adresse est la suivante :

- 0 pour un automate de la famille Premium/Atrium ou Modicon M340 ;
- 1 pour un automate de la famille Quantum.

Ce rack contient le type de processeur sélectionné lors de la création du projet. Il est possible de remplacer ce processeur par un processeur compatible. Le nombre de racks gérés par le processeur diffère selon le processeur choisi, le tableau **[ANNEXE C2]** montre le nombre de rack gérés pour différents processeurs.

On peut ajouter un rack en faisant glisser un des racks proposés dans le catalogue matériel (navigateur de l'éditeur de bus) vers les zones « 1 » ou « 2 », ou bien en double cliquant sur une de ces zones, comme montré sur la figure suivante:

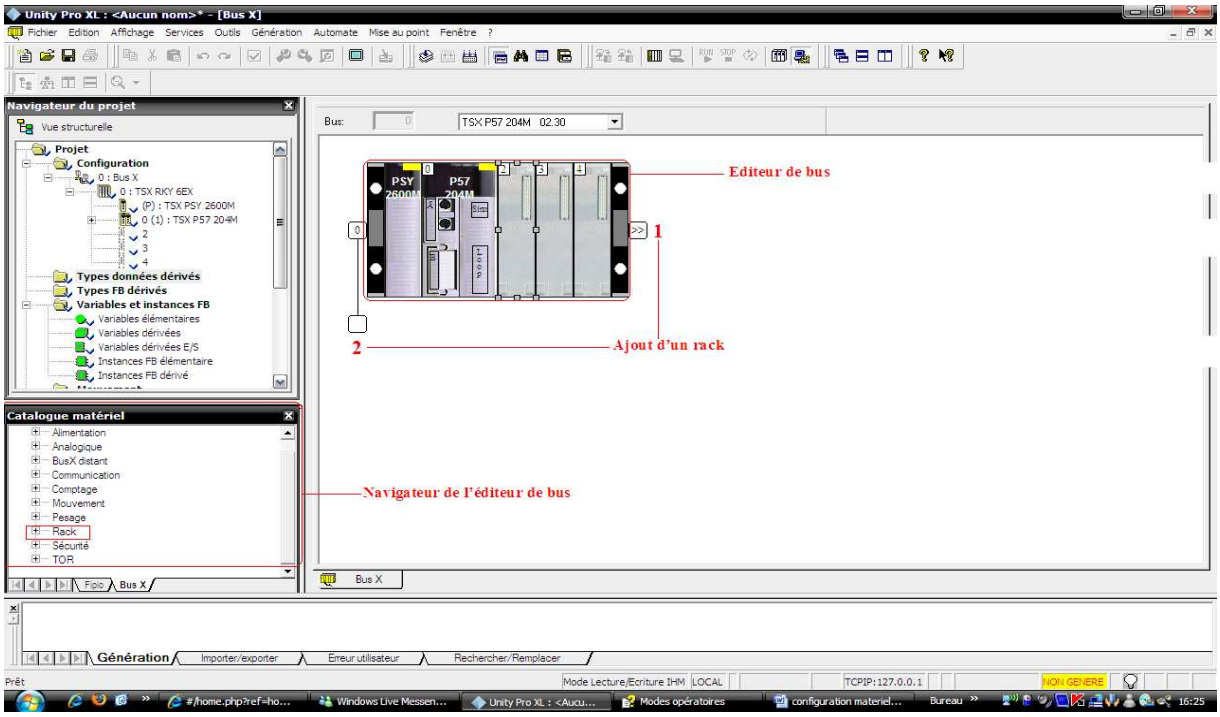


Figure II.3 : Ajout d'un rack.

Remarque :

En ajoutant un rack dans la zone« 1 », Il aura la même adresse(ici 0) et les modules seront numérotés à partir de la première puissance de 2 qui vient après le numéro du dernier module. Dans la figure précédente, le dernier module ayant l'emplacement « 4 », le nouveau rack commencera avec l'emplacement « 8 ».

En ajoutant un rack dans la zone « 2 », il aura une adresse différente (ici 1) et les modules seront numérotés à partir de 0.

2.1.3. Configuration des différents modules :

2.1.3.1. Module d'alimentation :

Pour une station Premium, le module d'alimentation est configuré par défaut. Il doit occuper la position la plus à gauche du rack. Cette position ne dispose pas d'adresse.

Un module d'alimentation double format occupe aussi la position d'adresse 0 (occupée habituellement par le module processeur), dans ce cas le module processeur doit être configuré à la position d'adresse 1.

On peut changer d'alimentation en en faisant glisser à partir du navigateur de l'éditeur de bus (ou catalogue matériel)

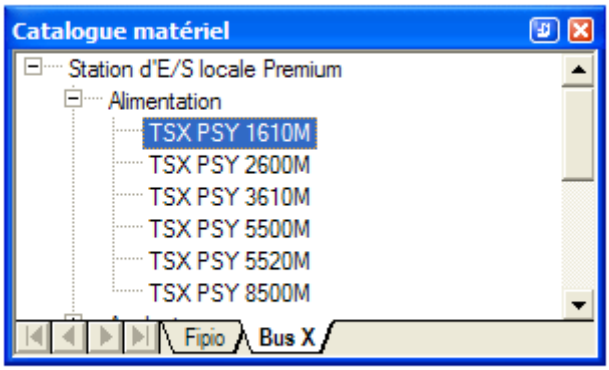


Figure II.4 : Choix de l'alimentation

Bilan de consommation :

Un bilan de consommation est établi sur :

- Le module d'alimentation d'un rack
- Chaque module (processeur, modules d'E/S), dépendent du module d'alimentation du rack.

Ce bilan est présenté sous forme de barographe ou chaque couleur à une signification particulière, elle signale pour la tension correspondante:

- le débit de courant en cours: couleur verte,
- la quantité de courant encore disponible: couleur blanche,
- une surcharge de courant: couleur rouge, lors du dépassement un message est affiché.
- et la puissance totale (même code de couleur),

Le bilan de consommation *du module d'alimentation* montre la quantité de courant débitée par l'alimentation pour chaque tension qu'elle fournit, ainsi que la puissance totale.

Lorsqu'on ajoute ou retire un module, le bilan est ajusté à l'ouverture de la fenêtre Bilan de l'Alimentation et des E/S.

On accède au bilan de consommation par le menu contextuel en sélectionnant le module d'alimentation puis en choisissant l'onglet *Alimentation*.

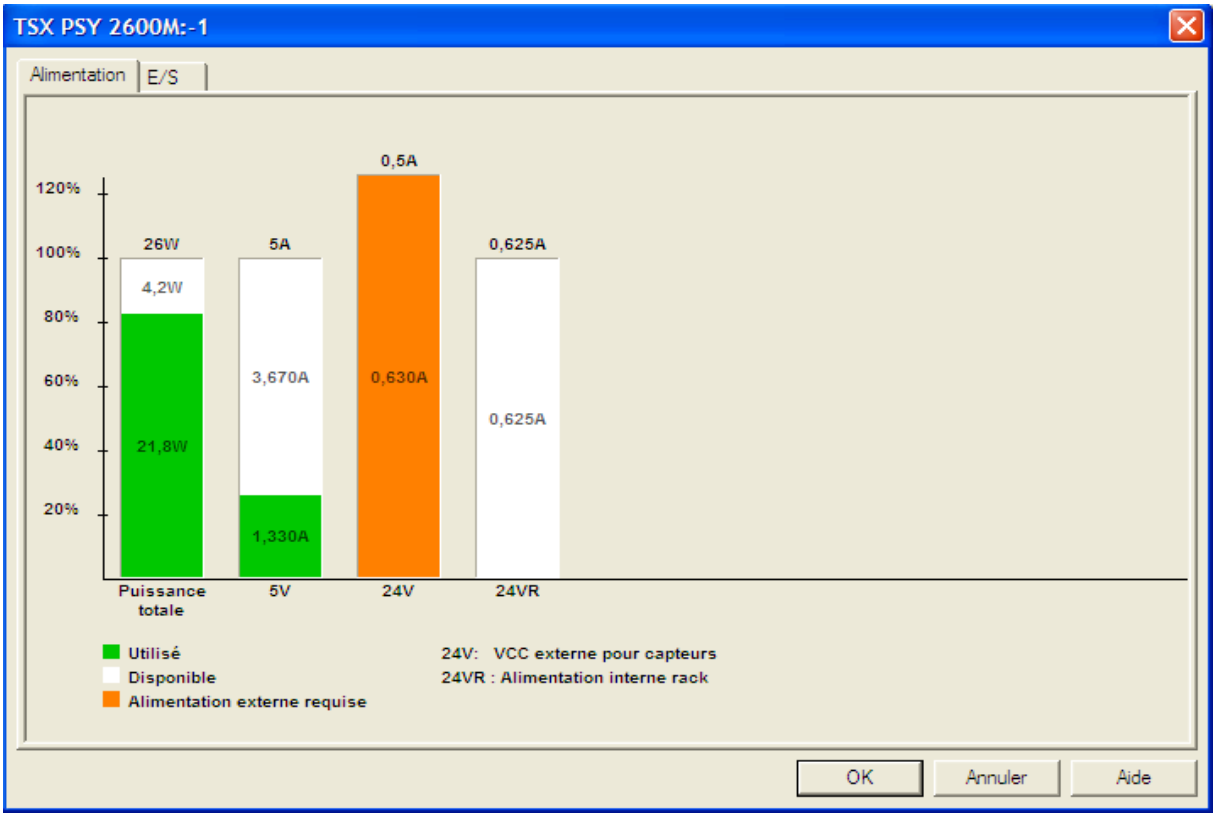


Figure II.5 : Bilan de consommation de l'alimentation

Remarque : Le bilan de consommation pour les autres modules montre la quantité de courant débitée dans le module pour chaque tension qu'il utilise, ainsi que la puissance totale.

2.1.3.2. Les autres modules :

Le catalogue matériel offre la possibilité de choisir un module et de le faire glisser dans l'un des racks de l'éditeur de bus.

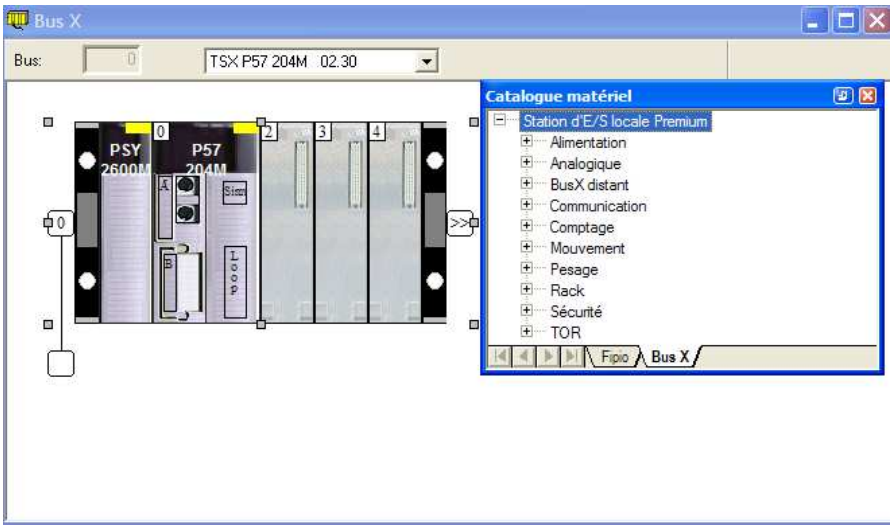


Figure II.6 : Modules proposés par le catalogue matériel

Pour chaque type de module, (analogique, communication, comptage…) plusieurs références sont proposées. Il suffit de déployer en sélectionnant le petit +.

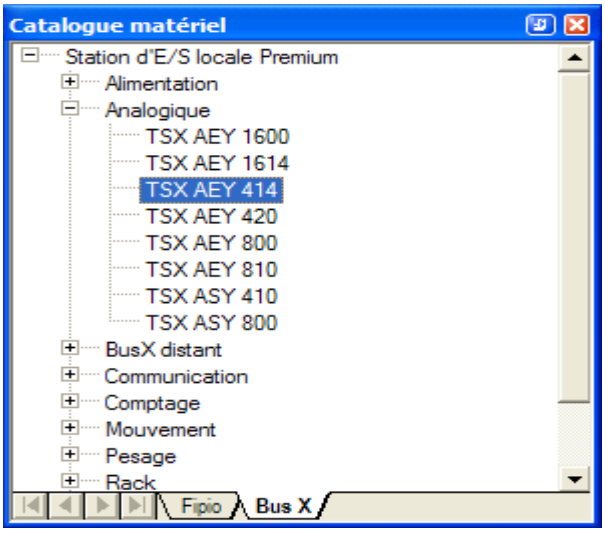


Figure II.7 :Sélection d'un module

En double-cliquant sur le module choisi, on obtient une fenêtre qui nous donne la description du module ainsi que l'onglet « Objet d'E/S ». Cet onglet permet de gérer les objets d'entrée et de sortie d'un module, d'un équipement sur un bus de terrain ou encore les objets mémoire et système de l'automate.

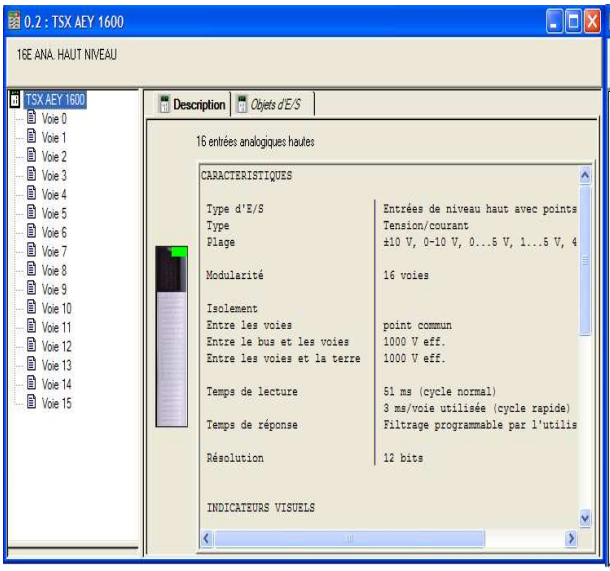


Figure II.8 : Description

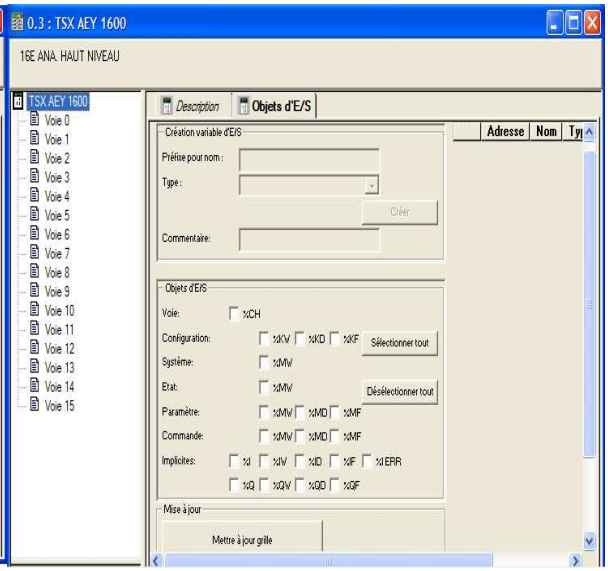


Figure II.9 : Objet d'E/S

Pour chaque voie, on peut avoir l'écran de configuration, Cet écran permet d'afficher et de modifier les paramètres en mode local, (automate non connecté). On peut alors :

- Affecter un nom à al voie (ça revient à nommer une variable d'entrée ou de sortie) dans le champ « symbole ».
- Choisir la gamme d'entrée ou de sortie (+/- 10 V, 4-20mA…)
- Effectuer la mise à l'échelle de la variable

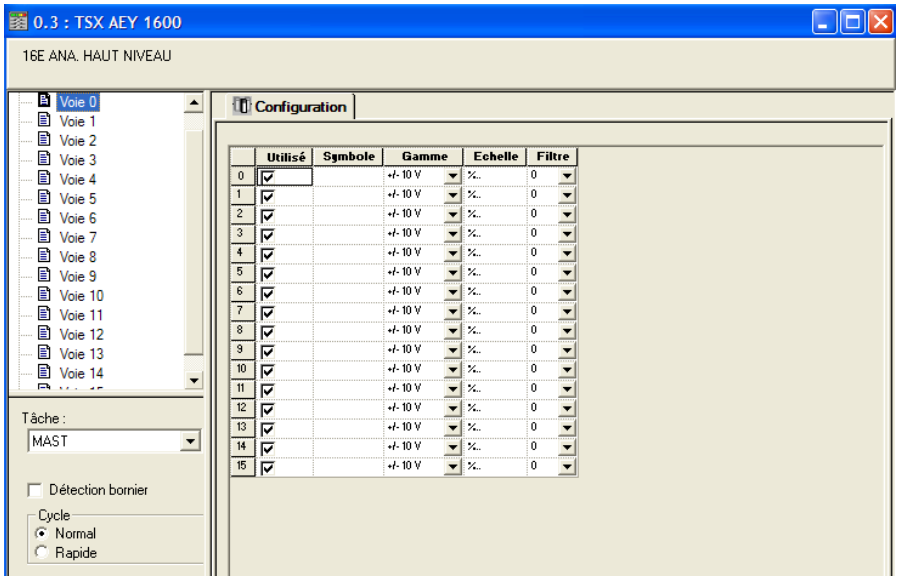


Figure II.10 : Ecran de configuration d'une voie

2.1.4. Configuration d'un réseau :

2.1.4.1. Créer le réseau logique.

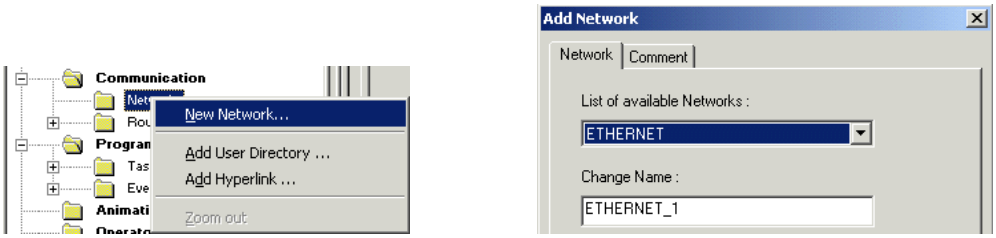


Figure II.11 : Création d'un réseau

- Ajouter un nouveau réseau (clic droit sur le dossier Réseaux du navigateur d'application) **(1)**
- Choisir le type de réseau à créer (Ethernet, Modbus+..) et saisir son nom **(2) [ANNEXE A]**
- Saisir un commentaire (si nécessaire)

2.1.4.2. Configurer le réseau logique.

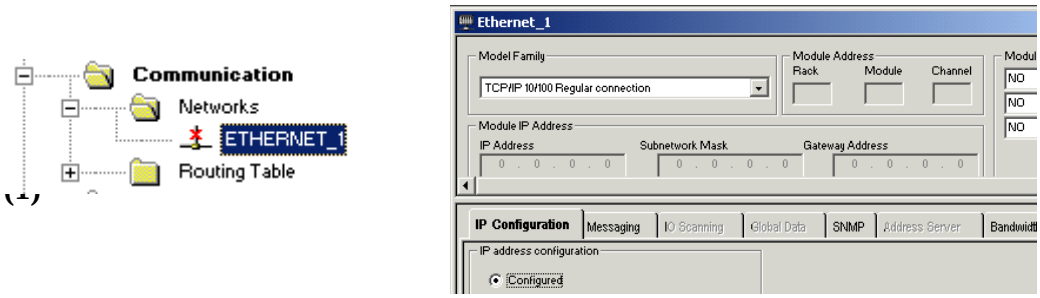


Figure II.12 : Création d'un réseau

- Activer le réseau logique à configurer
- Configurer le réseau logique : global data, I/O scanning, …

2.1.4.3. Définir le module de communication ou la carte PCMCIA :

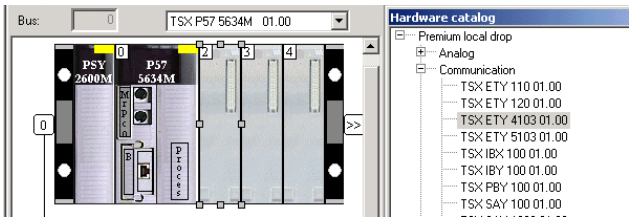


Figure 2.13 : Choix du module de communication

- Définir le module de communication (glisser-déplacer depuis le catalogue matériel)
- Ou définir la carte PCMCIA (double clic sur la position de la carte et ajout du sous-module)

2.1.4.4. Associer le module ou la carte PCMCIA au réseau logique.

- Ouvrir le module de communication
- Sélectionner la voie
- Associer le module au réseau logique.

2.1.4.5. Configuration du module de communication :

Ces modules peuvent être simples ou doubles, en choisissant le module « communication », on a la liste des modules compatibles avec le processeur utilisé. Un module de communication non compatible ne pouvant pas être sélectionné.

Après avoir ajouté le module, on double click sur lui et on aura la fenêtre « configuration » affichée.

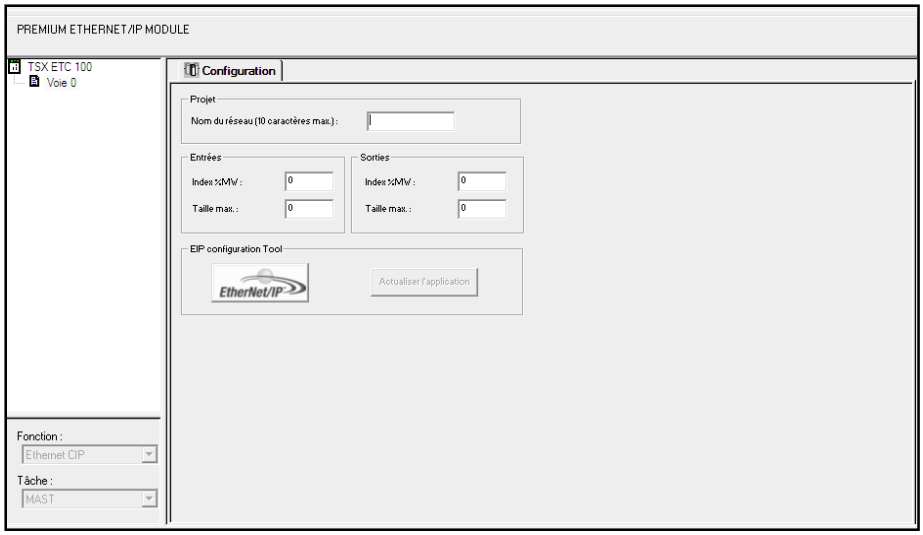


Figure II.14 : Configuration du module de communication.

2.2. Edition du programme :

Un programme s'exécute en trois étapes, à savoir, lecture de données puis tratement du programme et en fin écriture des sorties. Il peut être ***cyclique*** ou ***periodique***, dans le premier cas il entame le prochain cycle juste après la mise à jour des sortie, alors que pour le deuxième il ne l'entame qu'après un temps fixe (periode) même si la mise à jour des sorties a été faite.

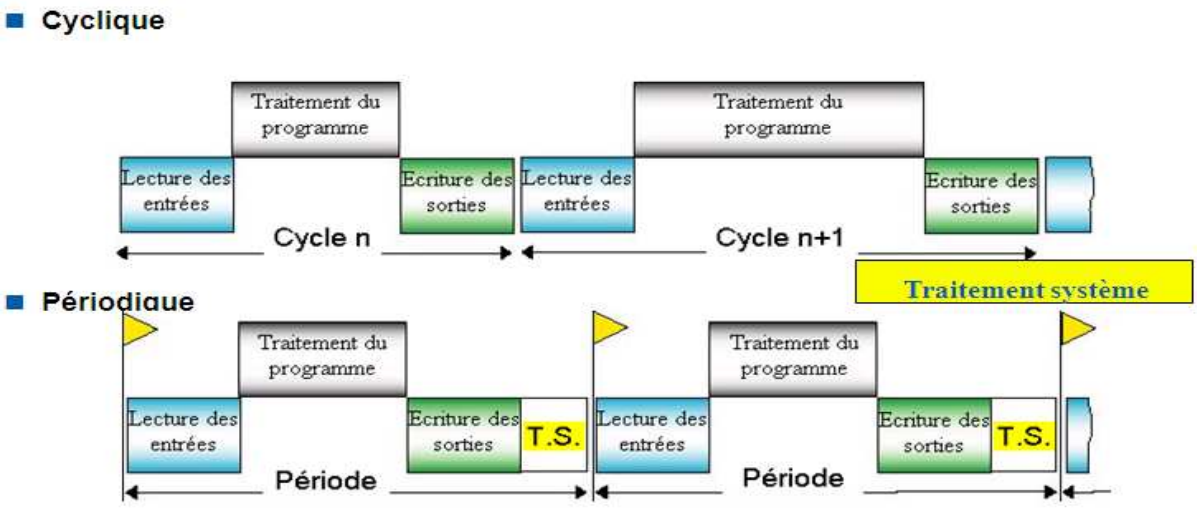


Figure II.15 : Etapes d'exécution d'un programme (cyclique et périodique)

Un programme peut être :

- Mono-tâche : Ne comprenant que la tâche principale (MAST)
- Multi-tâche : Comprenant les tâches MAST + tâches rapides (FAST) + tâches événementielles (EVT) + tâches auxiliaires (AUX).

Ces tâches diffèrent de part leur priorité d'exécution, la figure suivante montre l'ordre de priorité.

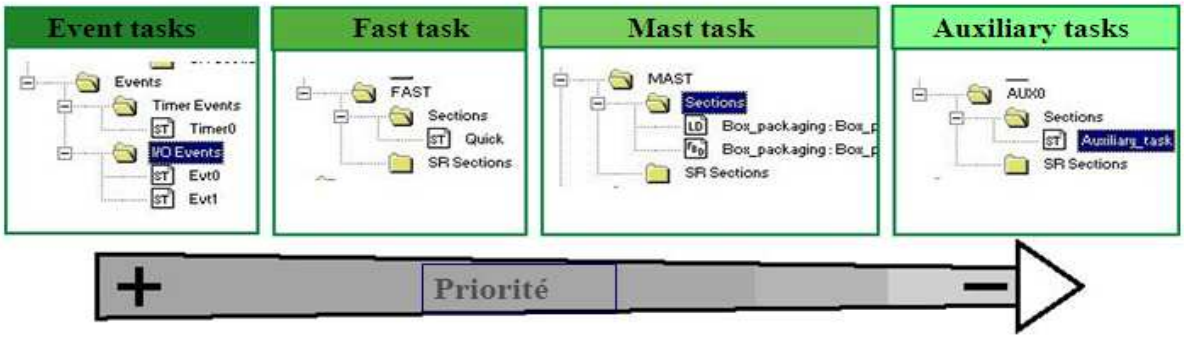


Figure II.16 : Ordre de priorité des différentes tâches.

2.2.1. Les différentes tâches :

2.2.1.1. La tâche MAST :

La tâche « maître » représente le programme principal. Elle est obligatoire quelle que soit la structure adoptée mono-tâche ou multitâche donc elle est crée par défaut.

Le programme de la tâche maître est constitué de plusieurs modules de programmes appelés sections et sous-programmes, qui sont liées à une tâche. Une même section ou sous-programme ne peut pas appartenir simultanément à plusieurs tâches.

Les appels aux sous-programmes s'effectuent dans les sections ou depuis un autre sous-programme (8 niveaux d'imbrications maximum).

L'exécution de la tâche maître peut être choisie (en configuration) : cyclique ou périodique.

2.2.1.2. Création d'une nouvelle tâche :

Dans le cas d'un programme multitâches on peut ajouter une tâche depuis le navigateur de projet, on a le choix entre une tâche FAST et AUX (disponible que dans quelques processeurs).

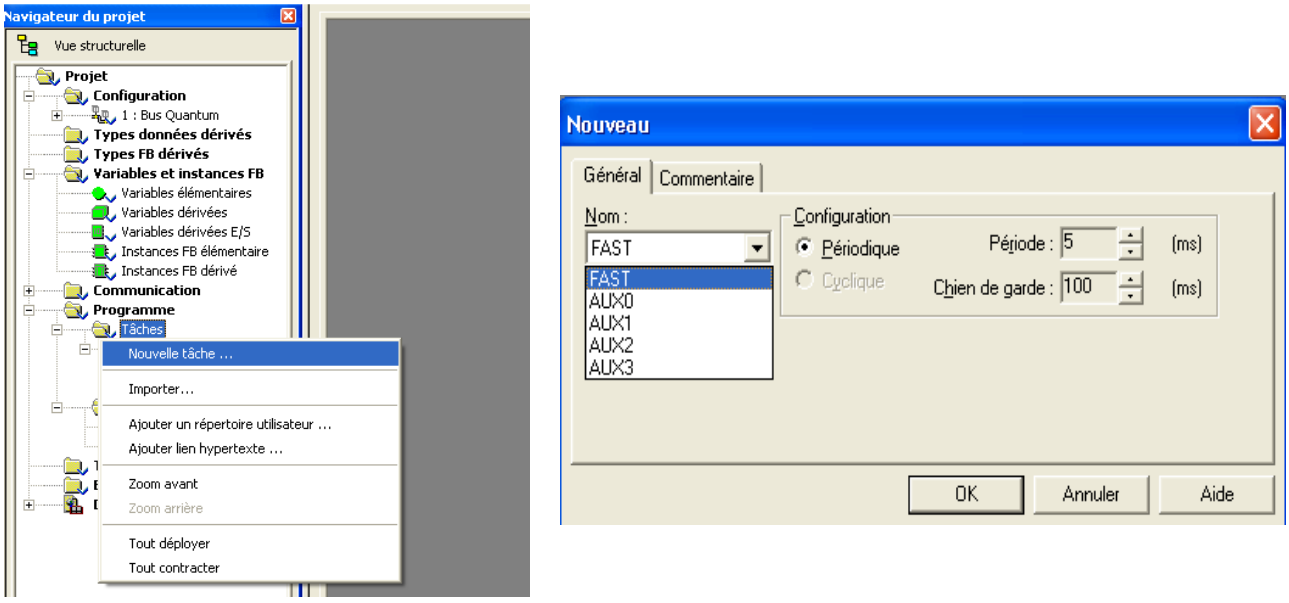


Figure II.17 : Création d'une nouvelle tâche.

2.2.1.2.1. Tâche FAST :

C’est une tâche plus prioritaire que la tâche maître MAST et périodique avec une période configurable de 1 à 255 ms. Comme pour la tâche MAST, le programme associé se compose de sections et de sous-programmes.

Le contrôle de la tâche ainsi que la visualisation des temps d’exécution se fait avec des bits systèmes associés.

Exemple : le bit système %S11 est positionné à 1 et l’application est déclarée en défaut bloquant pour l’automate en cas de débordement (chien de garde).

2.2.1.2.2. Tâches auxiliaires AUX :

Une tâche auxiliaire est utilisée pour les tâches de traitement plus lentes. On peut programmer jusqu’à 4 tâches auxiliaires maximum (AUX0 à AUX3) sur Premium TSX P57 5•• et Quantum 140 CPU 6••••

Elle est composée de sections et des sous-programmes programmés en LD, FBD, IL, ST**.** L’exécution est périodique (de 10 ms à 2,55 s)

2.2.1.2.3. Tâche événementielle « EVT E/S» et « TIMER »:

Elle est utilisée pour réduire le temps de réponse du programme aux événements des modules d'entrée/sortie et aux temporisateurs d'événement, elle contient une seule section programmée en LD, FBD, IL, ST.

EVTi : événements provenant des modules d'entrée/sortie

TIMERi : événements provenant des temporisateurs d'événements (fonction ITCNTRL).

2.2.2. Choix du langage de programmation :

Dans le navigateur de projet on fait un clic droit sur MAST et on choisit « nouvelle section », on a la possibilité de choisir l’un des cinq langages de programmation

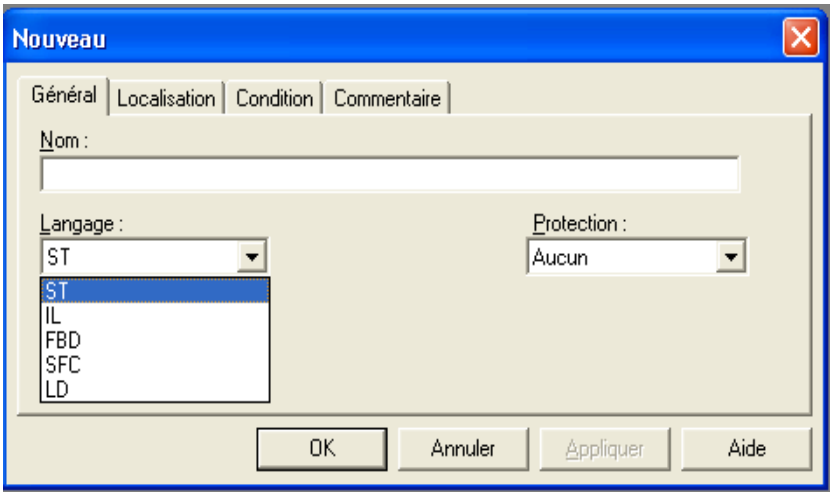


Figure II.18 : choix du langage.

2.2.2.1. LADDER DIAGRAM:

Le Ladder est équivalent à un schéma à relais.Il est composé de contacts, fonctions et bobines reliés à 2 barres verticales. Chaque réseau est composé d ’objets graphiques correspondant aux contacts, liaisons, bobines, blocs opérations, blocs fonctions EFs/EFBS/DFBs, saut, appel de sous programmes.

Coté gauche on trouve une barre de potentiel qui correspond à la phase (conducteur L), seuls les objets LD (contacts, bobines) connectés à cette source d ’alimentation sont exécutés par le programme LD. La barre de potentiel droite correspond au conducteur neutre n ’est pas visible. Cependant toutes les sorties de bobines et FFB sont logiquement connectées.

Le langage de programmation LD est orienté cellules, c’est-à-dire un seul objet est placé dans chaque cellule**.** Une grille divise la fenêtre en lignes et colonnes.

L'ordre de traitement des objets individuels d'une section LD est déterminé par le flux de données d'une section. Les réseaux associés à la barre d'alimentation gauche sont traités du haut vers le bas (connexion à la barre d'alimentation gauche). Les réseaux de la section sont indépendants des autres et sont traités dans l'ordre de placement.

Le jeu d’instruction est donné en annexe **[ANNEXE C3.2]**

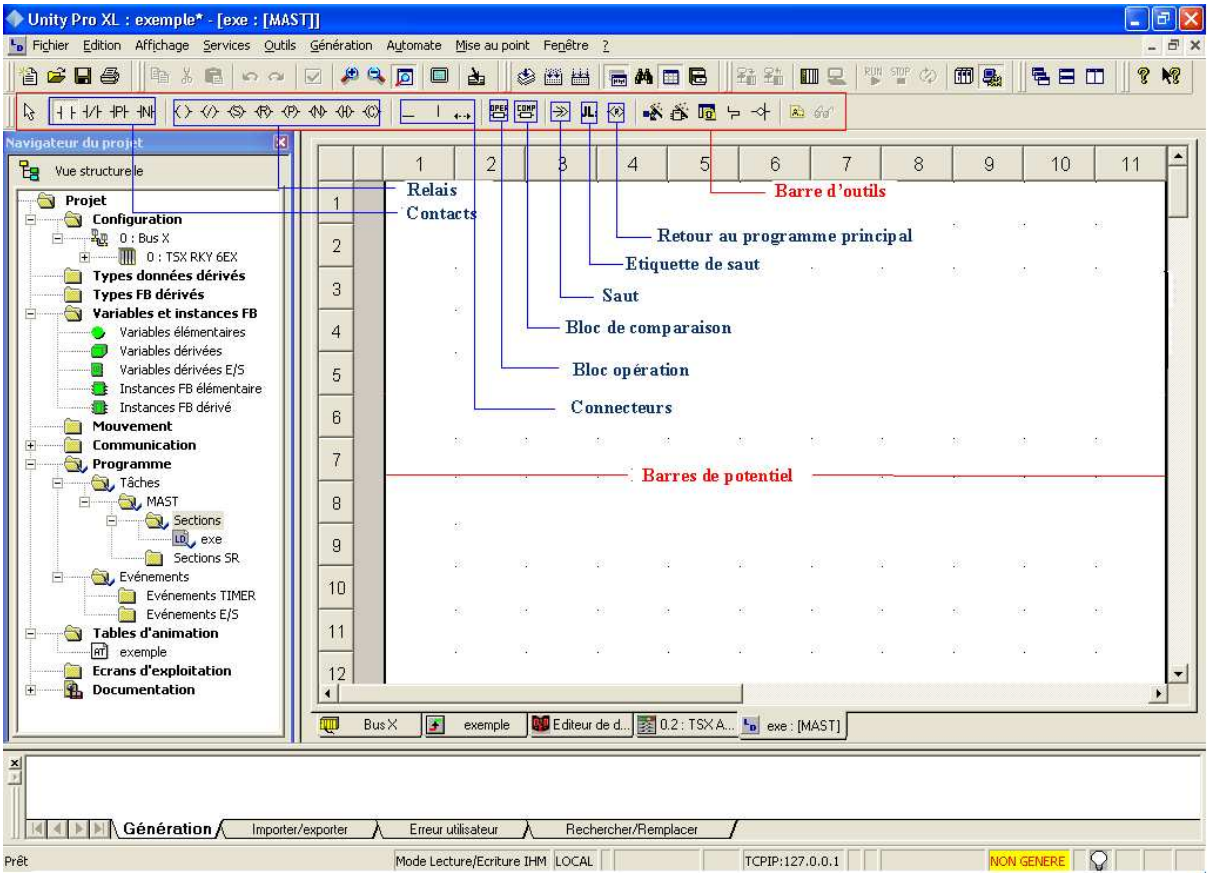


Figure II.19 : Editeur de programme LADDER.

2.2.2.2. LITTERAL STRUCTURE (ST) :

Le langage littéral structuré (ST) est un langage évolué de type algorithmique particulièrement adapté à la programmation des fonctions arithmétiques complexes, manipulations de données. Il permet la réalisation de programmes par écriture de lignes de programmation, constituées de caractères alphanumériques.

Une section de programme littéral est organisée en phrases. Une phrase littérale est l'équivalent d'un réseau de contacts en langage contacts.

On a accès directement aux boucles conditionnelles à travers la barre d'outils, on clique sur l'une d'elles et elle est automatiquement ajoutée dans notre programme. Il nous suffit de remplir les vides avec des instructions se terminant par un « ; ».

On peut aussi importer des fonctions (EF) et des blocs de fonctions (EFB, DFB) grâce à l'assistant FFB.

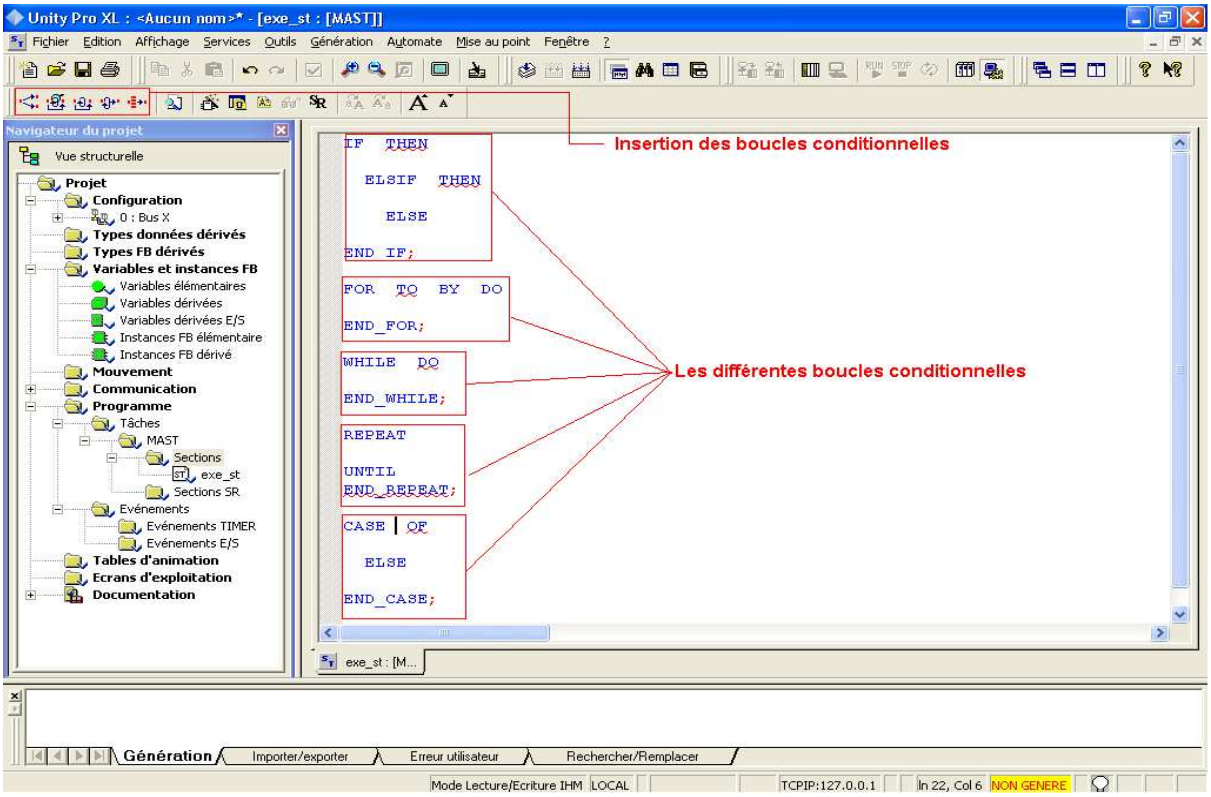


Figure II.20 : Editeur de programme ST.

Les instructions doivent être terminées par des points-virgules. Une ligne peut contenir plusieurs instructions (séparées par des points-virgules).Un seul point-virgule représente une instruction vide. Une ligne est limitée à 300 caractères. Il est possible d'utiliser des sauts de ligne dans les instructions (instructions d'affectation à plusieurs lignes).Des étiquettes, symboles et commentaires peuvent être librement placés dans la section. (les commentaires peuvent être saisis à tout endroit où les espaces sont autorisés).

Une vérification de la syntaxe et de la sémantique a lieu directement après la saisie des instructions d'affectation. Le résultat de la vérification est indiqué par différentes couleurs de texte.

Les sections comportant des erreurs de syntaxe ou de sémantique peuvent également être enregistrées.

2.2.2.3. Instruction List « IL »

Instruction List (*IL*) est un langage booléen proche du langage machine, il est compatible avec la norme IEC 61131-3. Il est difficilement lisible pour un utilisateur.

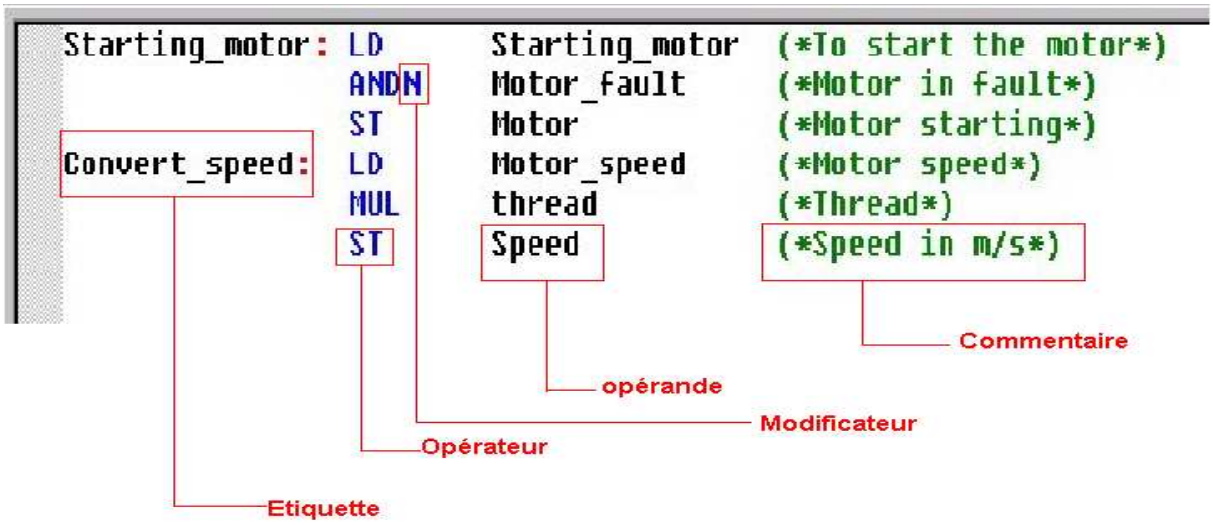


Figure II.21 : Programme en IL.

Chaque instruction commence dans une nouvelle ligne et est composée de :

- *Commentaire (optionnel)* : Informations supplémentaires sur la ligne
- *Opérateur* : définit la nature de l'instruction. Les différents opérateurs sont :
  - Assignment (*LD, ST, S, R*)
  - Logique (*AND, OR, XOR, NOT*)
  - Arithmétique (*ADD, SUB, MUL, DIV, MOD*)
  - Comparaison (*GT, GE, EQ, NE, LE, LT*)
  - Structure (*JMP, RET, «) »*)

- Appel de fonction (*CAL function_name*)

➢ *Modificateur* : influence l'exécution de l'opérateur, les différents modificateurs sont :

- N : invertit la valeur de l'opérande bit par bit.
- C : l'instruction associée n'est exécuté que si le résultat est vrai.
- CN : l'instruction associée n'est exécuté que si le résultat est faux.
- ( ) : sont utilisées pour pouvoir combiner les différentes instructions.

➢ *Opérande* : l'objet sur lequel agit l'opérateur, il peut être une adresse directe, une valeur, une variable, un DDT, un élément d'une DDT, une sortie d'une EFB ou bien un appel d'une EFB/DFB.

➢ *Etiquette (optionnel)* : localise une séquence dans le programme.

Un jeu d'instruction est donné en annexe **[ANNEXE C3.1]**

2.2.2.4. Functional Block Diagram « FBD »:

Le « Function Block Diagram » est un éditeur graphique orienté « flux de données », il est conforme au standard IEC 61131-3.Ce langage est particulièrement adapté aux applications de commande de processus, il est constitué de blocs élémentaires, fonctions dérivées, structures.…

Une section programmée en FBD comporte l'équivalent d'une grille présentant 30 colonnes de 23 lignes.

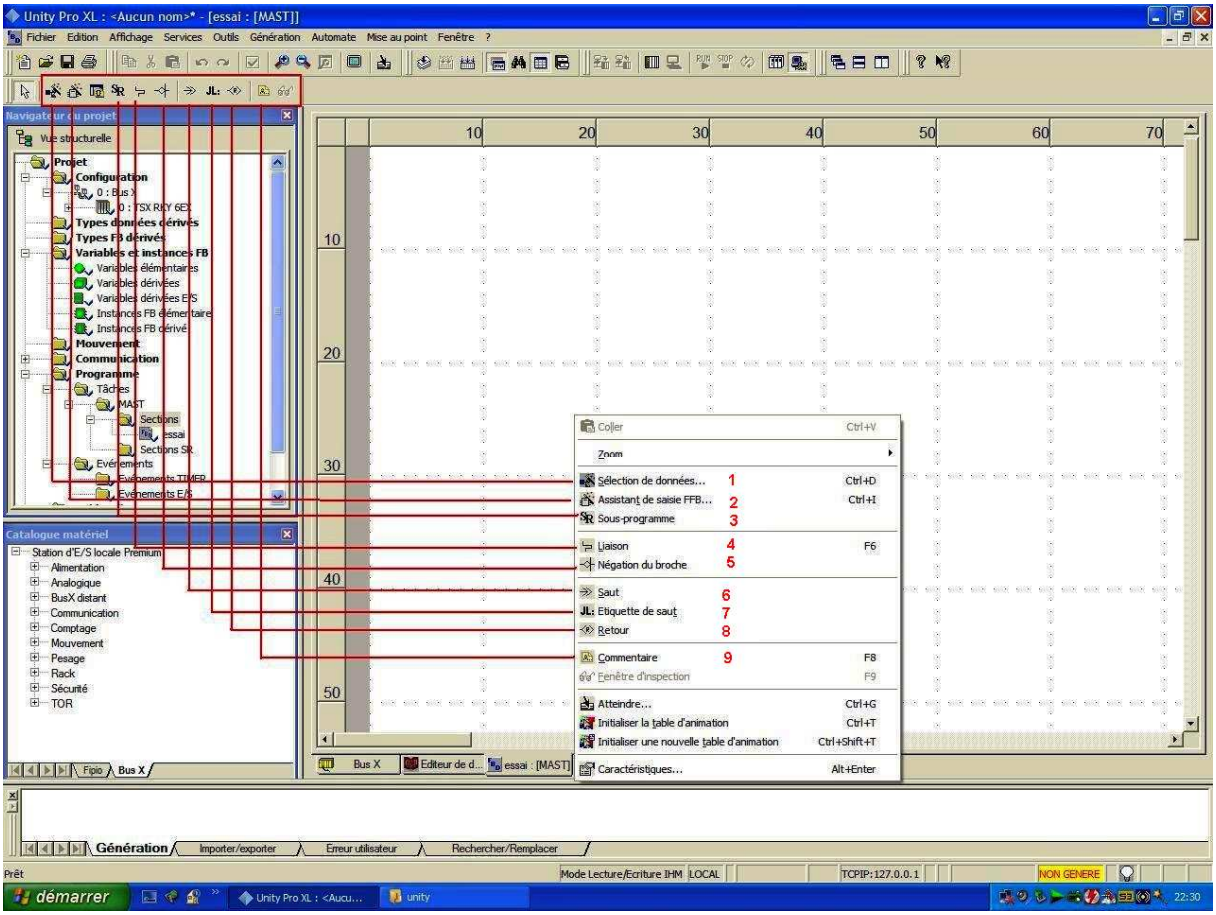


Figure II.22 : Editeur de Programme FBD.

Comme on peut le voir sur la figure, l'éditeur DFB propose de nombreuses fonctionnalités accessibles soit dans la barre d'outils, ou bien avec un clic droit sur la fenêtre principale. On a :

1) Sélection de données : Pour choisir une variable déjà crée dans la bibliothèque ou bien définir une nouvelle variable.

2) Assistant saisie FFB : Pour entrer un nouveau bloc.

3) Sous programme : Pour créer un sous programme.

4) Liaison : Pour relier les différents blocs

5) Négation de branche

6) Saut :

7) Etiquette de saut

8) Retour :

9) Commentaire.

Les objets du langage de programmation FBD offrent des aides permettant de structurer une section en un ensemble de :

- Blocs élémentaires EFs, Les blocs fonctions élémentaires EFBs,
- Blocs fonctions dérivés DFBs
- Procédures
- Eléments de contrôle

Ces objets regroupés sous l'abréviation générique FFB peuvent être liés entre eux par :

- Des liaisons
- Des paramètres réels
- Des commentaires peuvent être ajoutés pour insérer une logique étendue sous forme de macros.

Les Différentes fonctions FFB peuvent être importées grâce au gestionnaire de bibliothèque.

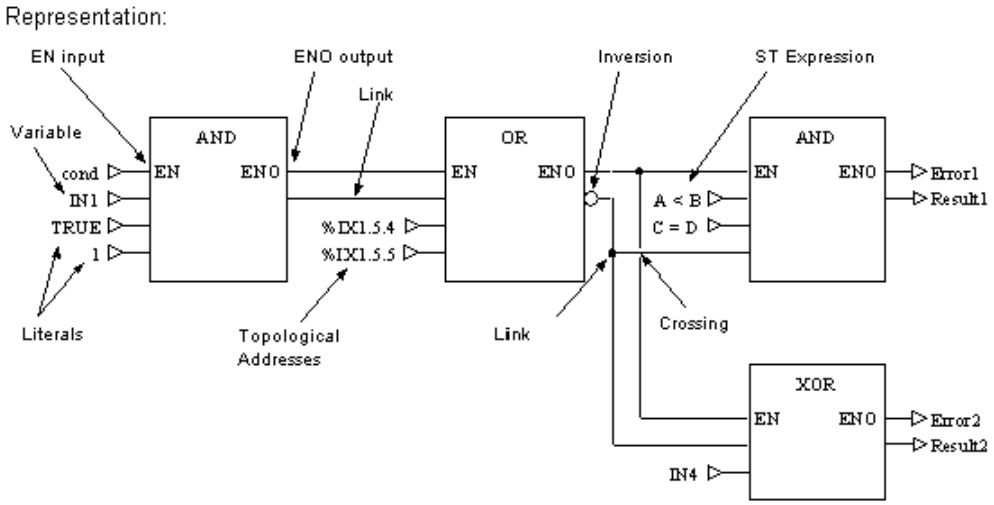


Figure II.23 : Exemple d'un schéma FBD.

Comme le montre la figure précédente, les entrées des différents blocs peuvent être des constantes booléennes et analogiques, des variables, et des expressions logiques programmées en « ST ».

2.2.2.5. Sequantial Functional Chart (SFC) :

2.2.2.5.1. Présentation :

Un diagramme fonctionnel en séquence conforme à CEI se compose dans Unity Pro de sections SFC (niveau supérieur), de sections de transition et de sections d'actions

Ces sections SFC ne sont admises que dans la tâche maître du projet. Dans d'autres tâches ou DFB, les sections SFC ne peuvent pas être utilisées.

Chaque section SFC contient exactement un réseau SFC (séquence) dans le jeton unique. Les jetons multiples d'une section SFC peuvent contenir un ou plusieurs réseaux SFC indépendants les uns des autres.

Une section SFC contient les objets de création de programme suivants : **(figure II.24)**

- étape
- macro-étape (séquence de sous-étapes imbriquées)
- transition (condition de transition)
- saut
- liaison
- divergence en OU
- convergence en OU
- divergence en ET
- convergence en ET

La logique de la section peut être commentée par des objets texte.Une section SFC est une "machine d'états", c.-à-d. que l'état est déterminé par les étapes actives et les transitions renvoient le comportement de commutation/modification entre les états. Les étapes et transitions sont reliées les unes aux autres par des liaisons dirigées. Deux étapes consécutives ne peuvent jamais être directement reliées ; elles sont toujours séparées par une transition. Les évolutions des états actifs de signaux se déroulent le long des liaisons dirigées, et sont déclenchées par la commutation d'une transition. Le déroulement d'une séquence va dans le sens des liaisons dirigées et se déroule de la partie inférieure de l'étape précédente à la partie supérieure de l'étape suivante. Les divergences sont traitées de gauche à droite.

Chaque étape peut compter zéro ou plusieurs actions. A chaque transition est associée une condition de transition.

La dernière transition de la séquence est toujours reliée à une autre étape de la séquence (par une liaison graphique ou un symbole de "saut") de manière à obtenir une boucle fermée. Les séquences d'étapes se déroulent donc de façon cyclique. **[ANNEXE C.3.3]**

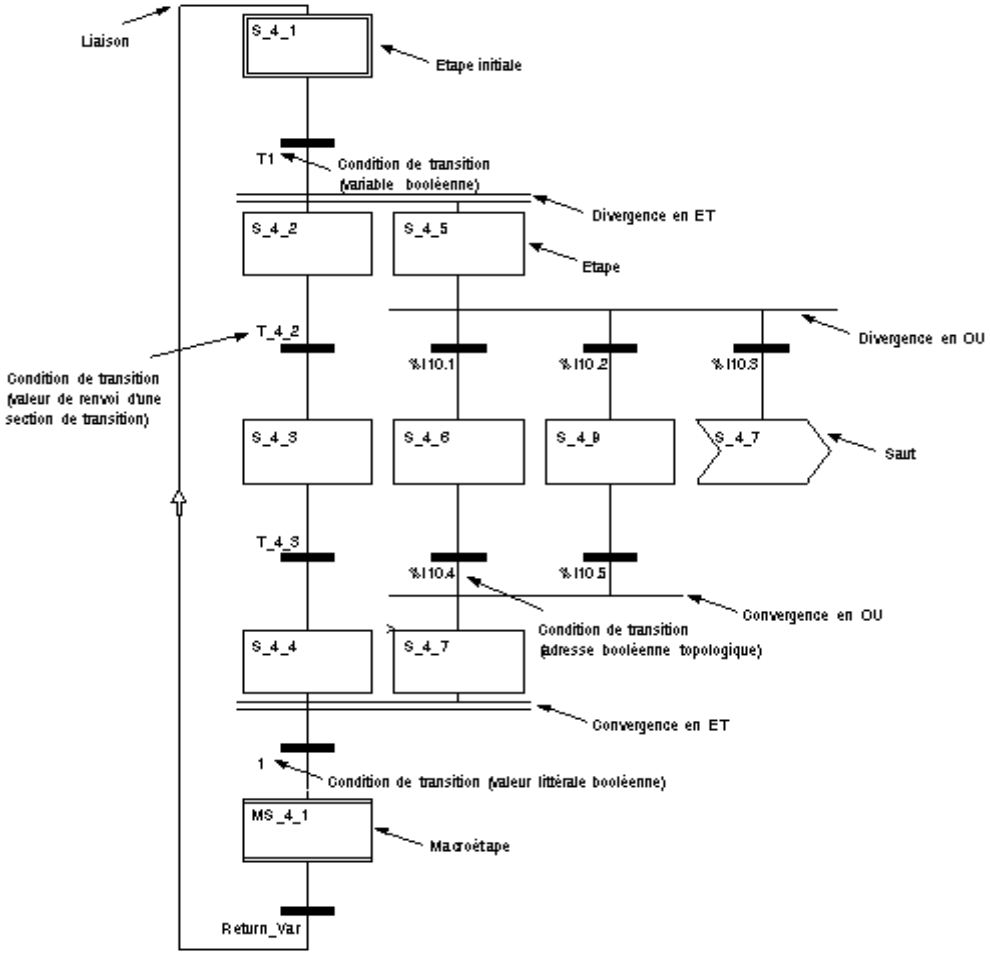


Figure II.24 : Représentation d'une section SFC

Le comportement d'un réseau SFC dépend largement du nombre de jetons choisis, c.-à-d. du nombre d'étapes actives.

Un comportement univoque est possible en utilisant un seul jeton (single token). Les divergences en ET comportant un jeton actif (étape) par branche sont considérées comme des jetons uniques. Ceci correspond à une séquence d'étapes selon la norme CEI 61131-3.

Une séquence d'étapes comportant un maximum d'étapes actives (jetons multiples) définies par l'utilisateur augmente le niveau de liberté. Les limitations relatives à l'obligation d'unicité et du non-blocage sont à cet effet levées et doivent être assurées par l'utilisateur. Les séquences d'étape à jetons multiples ne sont pas conformes à la norme CEI 61131-3.

2.2.3. Variables

2.2.3.1. Définitions

Une variable est une entité mémoire de type BOOL, WORD, DWORD, etc., dont le contenu peut être modifié par le programme au cours de l'exécution.

Une **variable « affectée »** est une variable affectée à un module d'E/S ou associée à une référence mémoire. Par exemple, la variable Pression_eau est associée au mot mémoire %MW102. Pression_eau est considérée comme localisée.

Une **variable « non affectée »** est une variable non affectée à l'E/S ou non associée à une référence mémoire (impossible de déterminer sa position en mémoire). Une variable qui n'a pas d'adresse affectée est dite non affectée.

Une **variable publique** est une variable disponible avec certains blocs fonctions. Ces variables transfèrent des valeurs statistiques (valeurs qui ne sont pas influencées par le processus) au bloc fonction. Elles sont utilisées pour régler les paramètres du bloc fonction.

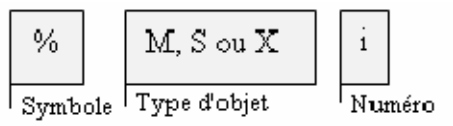
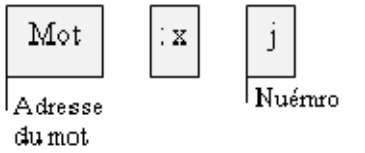
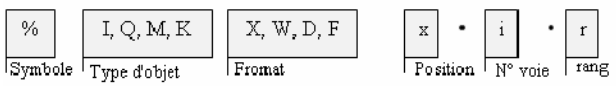
Une **variable privée** est une variable utilisée par certains blocs fonctions. Ces variables ne sont pas accessibles par le programme d'application

I/ODDT est l'abréviation de « Input/Output Derived Data Type » (type de données dérivées d'entée/sortie). L'élément I/ODDT désigne un type de données structurées représentant un module ou une voie d'un module d'automate. Chaque module expert de l'application possède ses propres I/ODDT.

Les **constantes** sont des variables de type INT, DINT ou REAL affectées dans le champ constant (%K) ou des variables utilisées dans l'adressage direct (%KW, %KD ou %KF). Le contenu de ces constantes ne peut pas être modifié par le programme pendant l'exécution.

2.2.3.2. Adressage

L'adressage diffère selon que notre variable est un bit ou un mot, se trouve dans un module ou bien est interne.

Règles d'adressage	**Syntaxe**
Objets bits : 	**Type d'objet :** M : bits internes S : bits système X : étapes (Grafcet) **Numéro :** dépend de la configuration
Bits extrait de mots : 	Numéro : 0 à 15 rang du bit dans le mot.
Objets de modules d'entrées/sorties du TSX37 : 	**Type d'objet :** I : image de l'entrée Q : image de la sortie M :variable interne K : constante interne **Format :** X : booléen W : simple longueur 16bits D : double longueur 32bits F : flottant 32bits

	Position module : 0 à 10 (TSX 37-22) **N° voie :** 0 à 31 ou MOD **Rang :** 0 à 127 ou rang
Objets de modules d'entrées/sorties du rack :  	**Type d'objet :** I : image de l'entrée Q : image de la sortie M : variable interne K : constante interne **Format :** X : booléen W : simple longueur 16bits D : double longueur 32bits F : flottant 32bits **Rack :** 0 à 7 **Position module :** 0 à 14 (position dans le rack) **N° voie :** 0 à 127 ou MOD **Rang :** 0 à 127 ou rang

Tableau II.2 : Adressage

2.2.3.3. Editeur de données :

On y accède depuis le navigateur de projet en cliquant sur « Variables et instances FB »

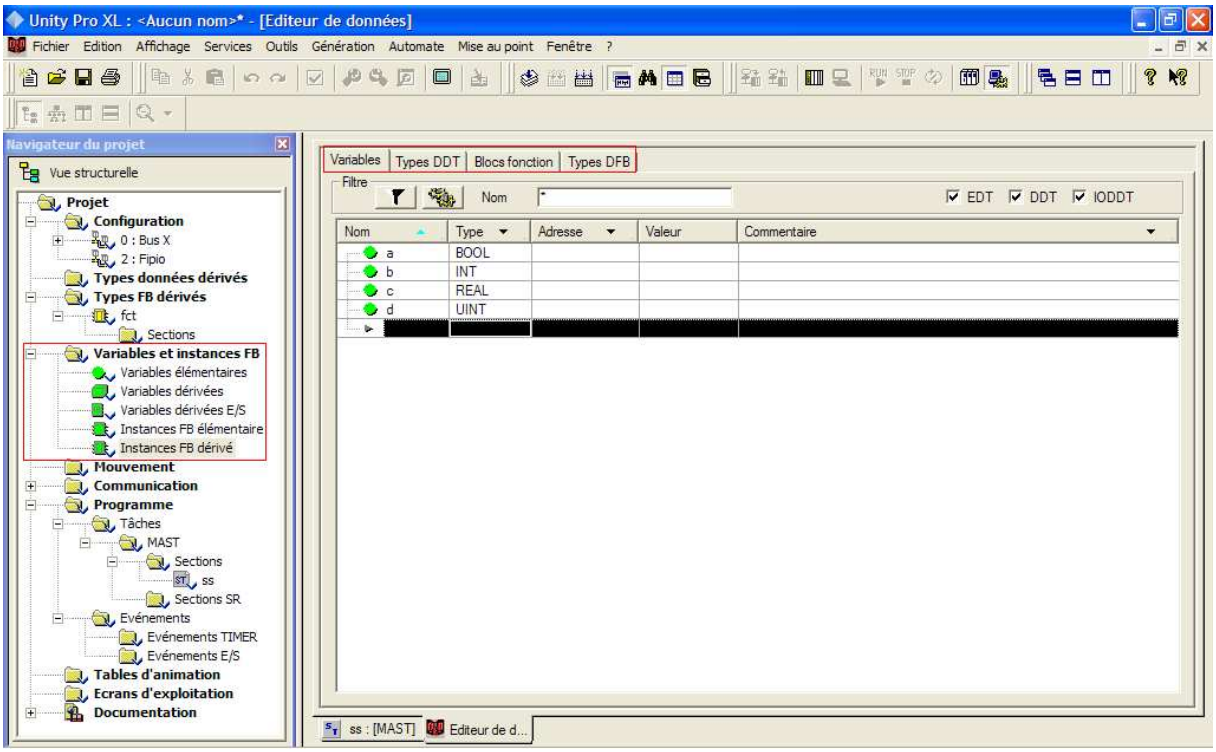


Figure II.25 : Editeur de données.

2.2.3.3.1. Onglet « Variables »

L'éditeur comprend quatre onglets, l'onglet « variables » permet de définir les différentes variables utilisés dans le programme, leurs adresses si elles sont affectées, leurs types, des noms et des commentaires.

Les variables peuvent être de différents types :

- **BOOL** : doit être activé (ON) (1) ou désactivé (OFF) (0)
- **WORD** : représente une "chaîne de 16 bits", ce qui signifie que la longueur des données est de 16 bits
- **INT** : représente une valeur d'entier. La plage des valeurs s'étend de -32 768 à 32 767

- **UINT** : représente une valeur d'entier non signé. La plage des valeurs s'étend de 0 à 65535

- **REAL** : représente une valeur à virgule flottante. La plage des valeurs s'étend de 8,43e-37 à 3,36e+38

2.2.3.3.2. Onglet « types DDT »

Un type de données dérivé (DDT) correspond : soit à un tableau, soit à une structure concernant les données d'entrées/sorties. Dans ce cas, le type n'est pas créé par l'utilisateur mais fourni par le constructeur (IODDT),

2.2.3.3.3. Onglet « Bloc fonctions »

Permet de créer des instances de types « Blocs de Fonction Elémentaires (EFB) », les EFB sont des blocs prédéfinis qu'on peut trouver dans la bibliothèque (par exemple : les compteurs, les temporisations…). Cet onglet permet de choisir un de ces blocs pour le programme et définir ses variables d'entrée/sortie.

2.2.3.3.4. Onglet « types DFB »

Cet onglet permet de créer des instances de bloc de fonctions dérivées, ce sont des blocs crées par l'utilisateur. On défniit les entrées/sorties qui peuvent êtres locales ou globales et on définit son rôle en la programmant on duble cliquans sur « section », on peututiliser les 4 langages de programmations (IL, LD, FBD et ST).

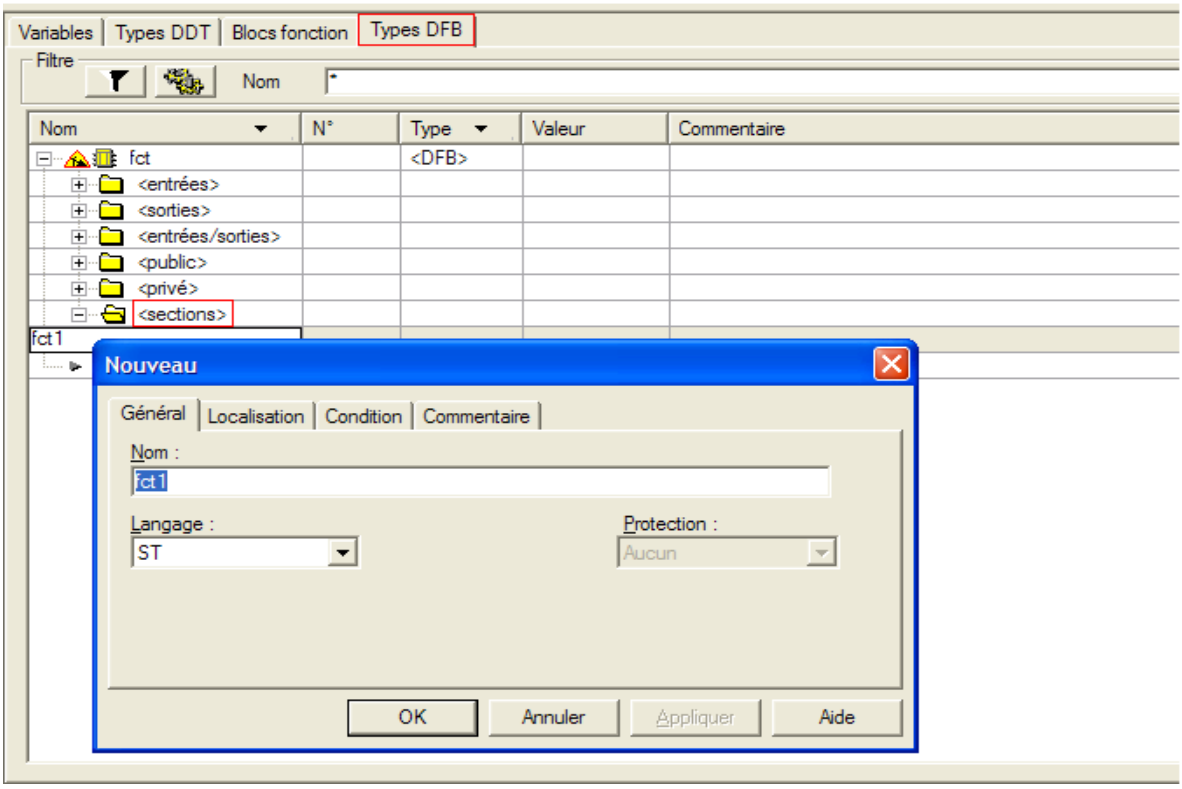


Figure II.25 : Création d'un bloc DFB.

La DFB crée ne peut être utilisée que dans le projet courant.Pour qu'elle puisse être utilisé dans n'importe quel projet sur UNITY PRO il suffit de la placer dans la bibliothèque comme le montre la figure suivante :

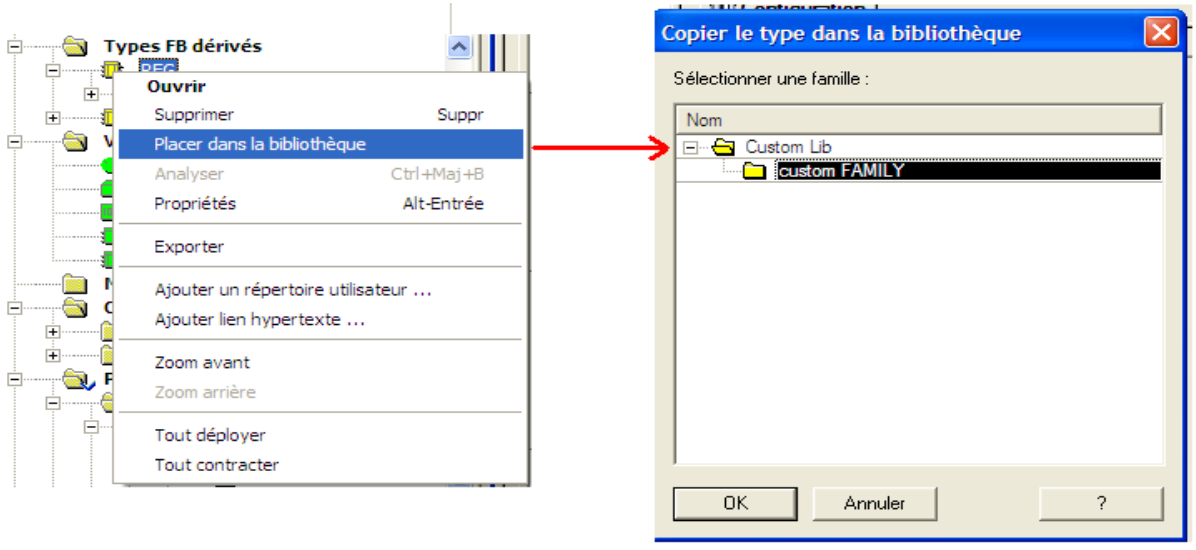


Figure II.26 : Ajout de la DFB dans une bibliothèque.

2.2.3.4. Gestionnaire de bibliothèque :

Il contient tous les objets disponibles pour développer un projet d'automatisation:

- EFs (Fonctions Elémentaires) :
- EFBs (Blocs fonctions Elémentaires)
- DFBs (Blocs fonctions dérivés)
- DDTs (Types de Données dérivés)

Il fournit un ensemble de fonctionnalités permettant de modifier le contenu de la bibliothèque. Il exécute les transferts entre la bibliothèque et le projet.

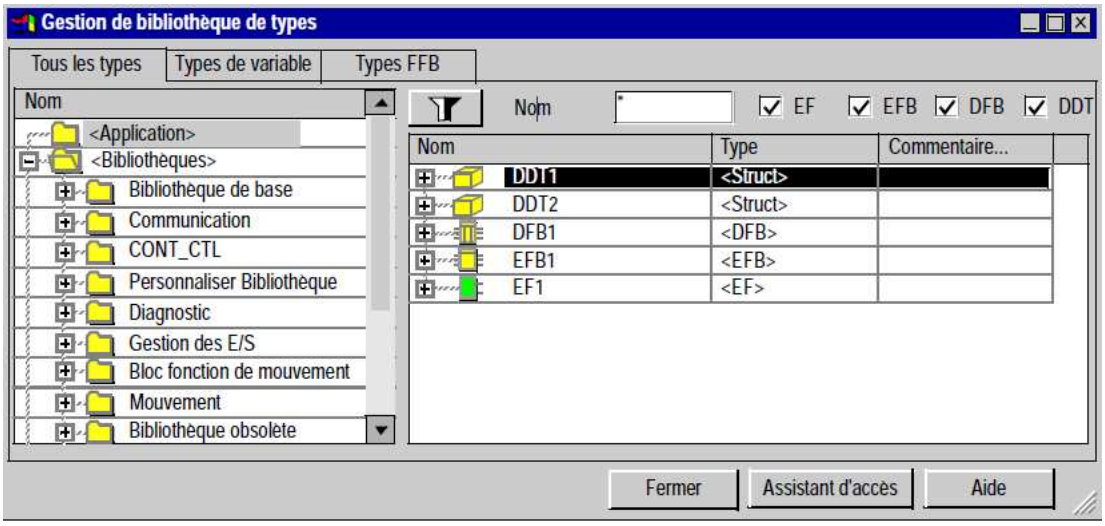


Figure II.27 : gestionnaire des bibliothèques.

On a un large choix de bibliothèques, qui sont :**Bibliothèque de base** : avec les groupes Tableaux, CLC_INT, Comparaison, Date et heure, Logique, Mathématiques, Statistiques, Chaînes, Temporisateurs et compteurs, Type à Type.

- **Communication** : avec le groupe Etendu contenant des FFB de communication
- **CONT_CTL** : avec les groupes Conditionnement, Automate, Mathématiques, Mesures, Traitement de sortie, Gestion des consignes

- **Diagnostics** : avec le groupe Diagnostics contenant 14 FFB de diagnostic
- **Gestion E/S** : avec les groupes Configuration E/S analogique, Mise à l'échelle E/S analogique, Echange explicite, E/S immédiate, Configuration E/S Quantum
- **Mouvement** : avec les groupes Commande d'axe, CAM, MMF Start
- **Bib Obsolète** : avec les groupes CLC, CLC_PRO, Extensions/Compatibilité
- **Système** : avec les groupes Evénements, Gestion SFC, Horloge Système

2.2.3.5.Assistant FFB:

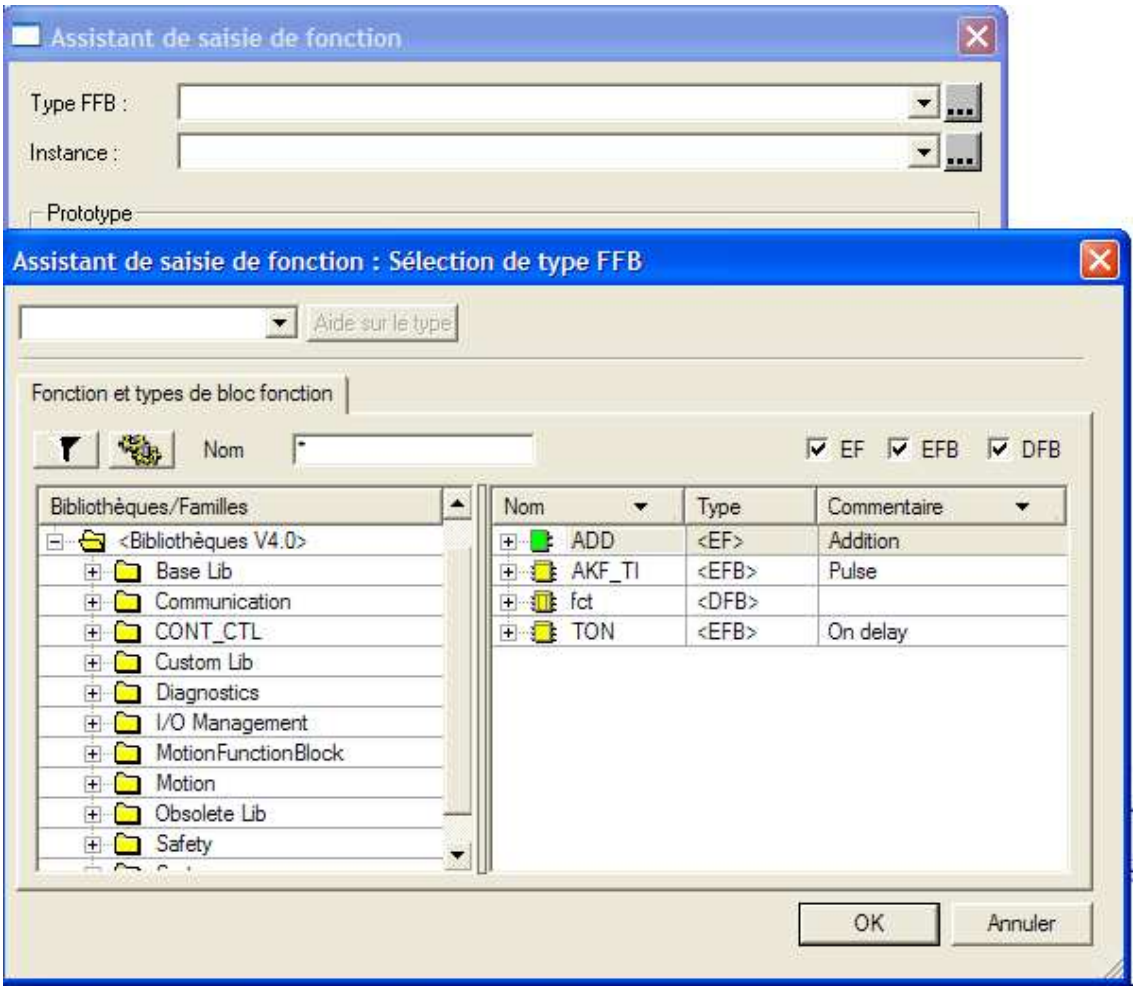


Figure II.28 : Assistant FFB.

Il permet d'ajouter un FFB dans un programme quelque soit le langage de programmation utilisé, il n'y a que la représentation de ce bloc diffère. On y accède depuis la barre d'outils de chaque éditeur de programme ou bien en faisant un clic droit au milieu de cet éditeur.

On peut donc ajouter des Bloc EFB ou des fonctions EF qui sont prédéfinies et disponibles dans la bibliothèque, dans ce cas il faut juste affecter des variables aux entrées/sorties. On peut aussi créer un bloc DFB en définissant nos entrées/sorties et en programmant la fonction de ce bloc.

En choisissant un bloc, il est automatiquement ajouté dans le programme, il reste à définir les variables d'entrées/sorties. Cetteaffectation diffère d'un langage à un autre.

2.3. Simulation avec Unity Pro :

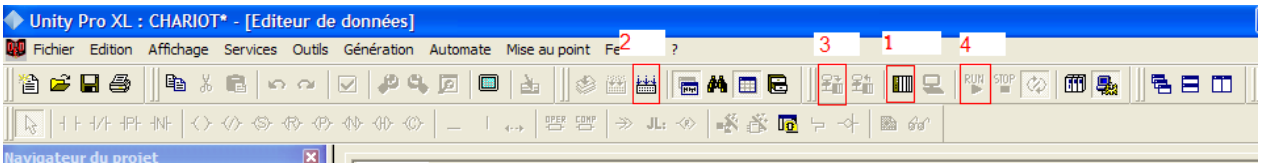


Figure II.29 : Simulation du programme.

Pour exécuter le programme il faut :

1) Connecter l'automate

2) Générer le projet

3) S'il n'y a pas d'erreurs détectées, charger le programme.

4) Exécuter le programme.

S'il n'y a pas d'automate connecté on peut visualiser l'exécution soit avec la table d'animation, soit avec les écrans de visualisation.

2.3.1. Table d'animation

On y accède depuis le navigateur de projet, elle permet de :

Modifier les variables internes grâce à l'onglet ***« modification »***, soit en en inscrivant la valeur dans le champ correspondant soit en utilisant la mise à 1/0 (pour les booléens)

Forcer l'état d'une variable affectée à un module d'entrées grâce à l'onglet ***« Forcer »***.et ce en inscrivant la valeur ou en utilisant aussi la mise à 1 ou mise à 0 comme le montre la figure suivante :

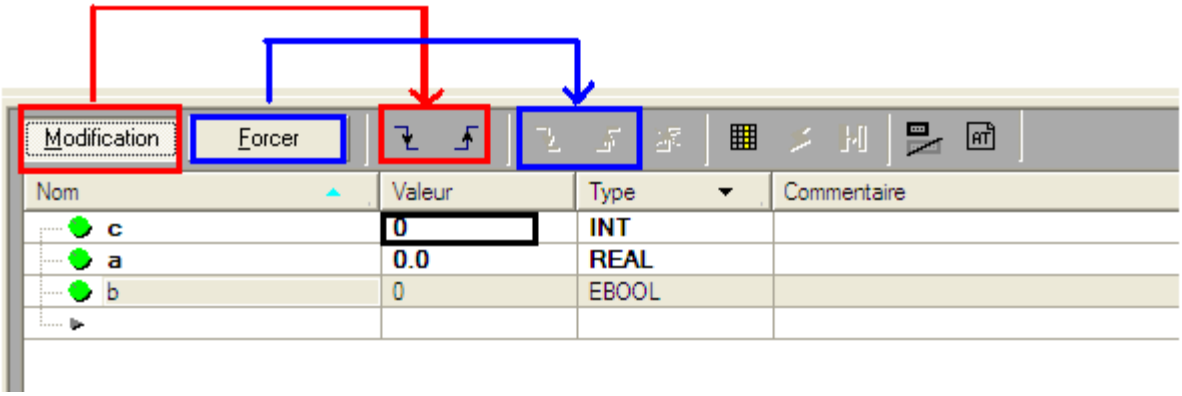


Figure II.30 Table d'animation

2.3.2. Ecran d'exploitation :

2.3.2.1. Présentation :

Les écrans d'exploitation intégrés sont destinés à faciliter l'exploitation d'un procédé automatisé. Ils utilisent dans le logiciel Unity Pro :

- Le navigateur projet qui permet de naviguer dans les écrans et lancer les différents outils (l'éditeur graphique, l'éditeur de variables, l'éditeur de messages, ...),
- L'éditeur graphique qui permet de créer ou modifier les écrans. En mode connecté, il permet également de visualiser les écrans animés et de conduire le procédé,

- La bibliothèque d'objets qui présente des objets constructeurs et permet de les insérer dans les écrans. Elle permet aussi de créer ses propres objets et de les insérer dans une famille de la bibliothèque.

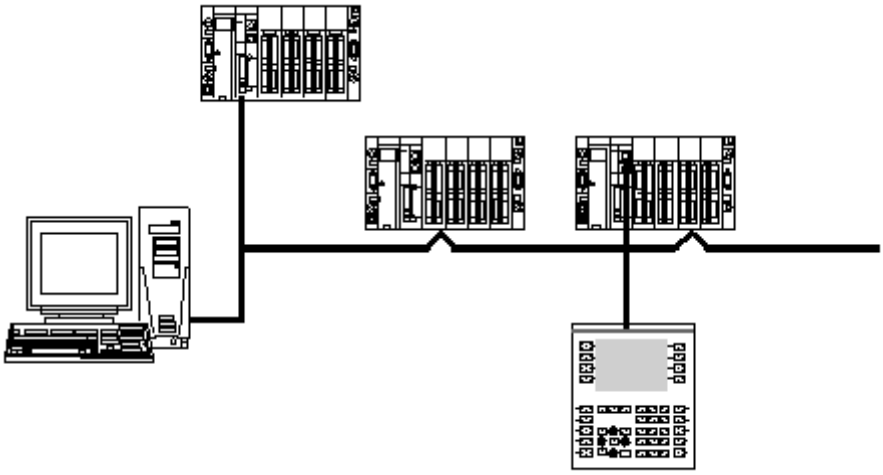


Figure II.31 : Exemple de structure d'automatisme qui utilise des écrans d'exploitation.

Dans cette structure, on trouve :

- L'automate qui contient le projet d'automatisme qui gère le procédé.
- Le terminal qui contient le projet d'automatisme avec les écrans d'exploitation. Il est connecté aux automates par la liaison console ou par un réseau.
- Les écrans d'exploitation visualisent le procédé et peuvent être commandés par le clavier du terminal, la souris ou un pupitre de commande connecté aux automates.

2.3.2.2. Création d'un écran d'exploitation :

Pour accéder aux écrans d'exploitations, il faut Visualiser le projet selon la vue structurelle (Affichage→Vue Structurelle), puis déployer le dossier écran d'exploitation. Si on veut ouvrir un nouvel écran il faut choisir « Nouvel écran » dans le menu contextuel du dossier écran d'exploitation.

Les objets qui peuvent être créés dans un écran graphique sont de 4 types :

- les objets standards : ligne, rectangle, ellipse, courbe, polyligne, texte. Ils peuvent être statiques ou dynamiques (ayant une variable associée modifiant leur affichage).
- les images : fichiers bitmap avec l'extension BMP ou JPG,
- les objets de pilotage (ou de commande) : bouton, case à cocher, champ de saisie, compteur, curseur, objet d'échange explicite, bouton de navigation écran…Ils sont activés par une action de la souris (ou du clavier). En fonction de l'attribut qui a été fixé, ces objets agissent sur leurs variables associées.
- les objets composés : ensemble d'objets des 3 types précédents, créé par l'utilisateur ou en provenance de la bibliothèque d'objets.

La bibliothèque d'objets présente les objets constructeurs et permet de les insérer dans les écrans d'exploitation. Les objets sont classés dans des familles. La bibliothèque permet aussi de créer ses propres objets en les insérant dans une famille de la bibliothèque.

La bibliothèque s'ouvre à partir de la commande Outils→Bibliothèque des écrans d'exploitation.

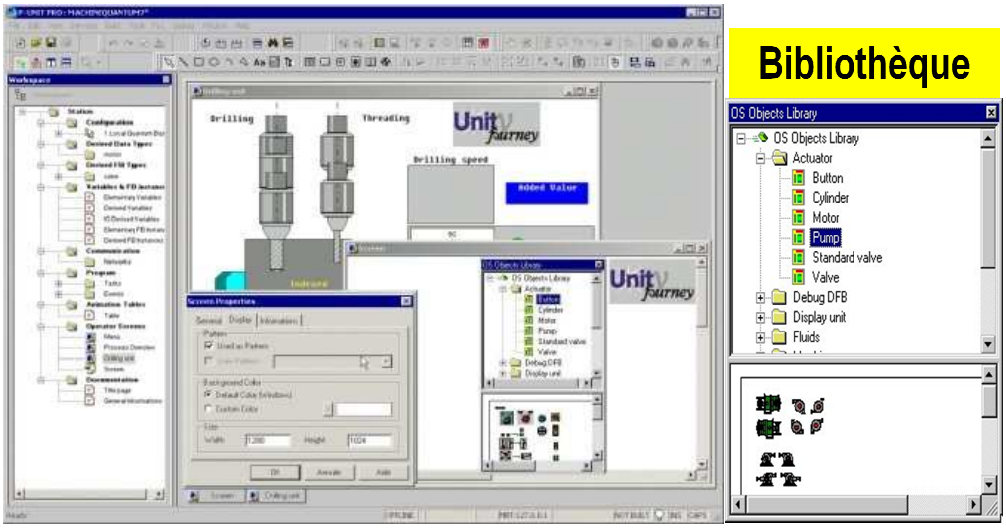


Figure II.32 : Ecran d'exploitation

2.4. Exemples d'applications :

2.4.1. Simulation d'un chariot :

2.4.1.1. Cahier de charge

Le système est défini par un chariot qui doit transporter une marchandise d'un point A à un point B.

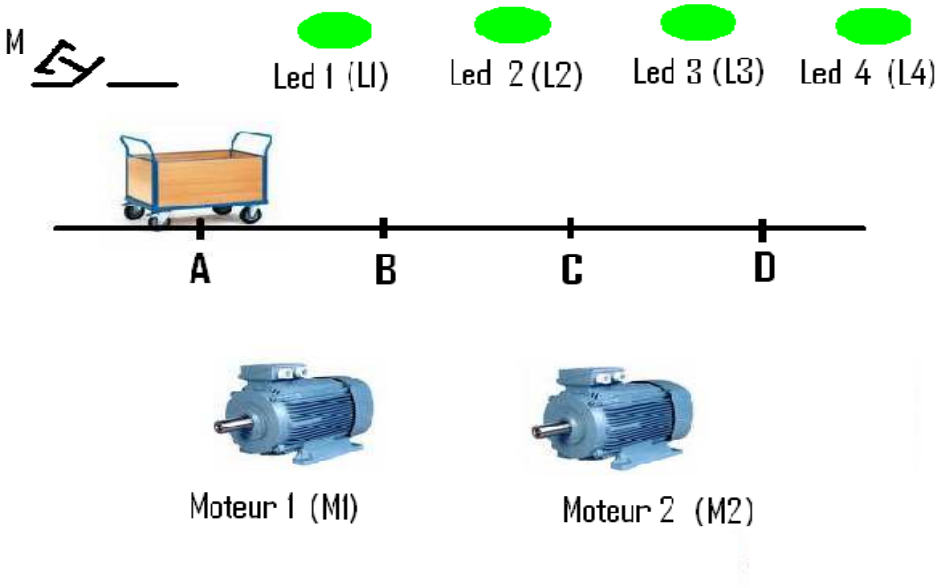


Figure II.33 : Représentation du chariot

Le système comporte :

- 4 microcontacts A, B, C et D.
- Un bouton poussoir M qui permet de démarrer le chariot donc le cycle de fonctionnement.
- 2 moteurs d'entrainement du chariot M1 et M2
- 4 leds de visualisation L1, L2, L3 et L4.

Quand le chariot est au point A et qu'on appuie sur le bouton M, M1 est démarré et L1 est allumée.

Arrivé au point B, M2 démarre, L1 s'éteint et L2 s'allume. Au point C, M1 s'arrête, L2 s'éteint et L3 s'allume.

Arrivé au point D, M2 s'arrête, L3 s'éteint, L4 s'allume et on lance une temporisation de 20 mn. Terminée, on démarre M1 et on allume L3 jusqu'au point C.

Arrivé au point C, on allume L2 et on démarre M2.

Arrivé à B, on arrête M1, on allume L1 jusqu'à A.

Arrivé à A, on éteint tout et on ne fait plus rien jusqu'au lancement d'un nouveau cycle par M.

2.4.1.2. Réalisation matérielle :

La réalisation matérielle, du réseau de PETRI, à base de bascules RS est la suivante :

$$X_0 : \begin{cases} S_0 = X_6 A \\ R_0 = X_0 Am \end{cases} ; X_1 : \begin{cases} S_1 = X_0 Am \\ R_1 = X_1 B \end{cases} ; X_2 : \begin{cases} S_2 = X_1 B + X_5 C \\ R_2 = X_2\left(C\overline{S} + BS\right) \end{cases} ; X_3 : \begin{cases} S_3 = X_2 C\overline{S} \\ R_3 = X_3 D \end{cases} ; X_4 :$$

$$\begin{cases} S_4 = X_3 D \\ R_4 = X_4 T \end{cases} ; X_5 : \begin{cases} S_5 = X_4 T \\ R_5 = X_5 C \end{cases} ; X_6 : \begin{cases} S_6 = X_2 BS \\ R_6 = X_6 A \end{cases} .$$

Où :

A, B, C, D : Sont des contacts électriques.

T : représente la fin de la temporisation.

S : représente la variable sens.

Les équations de sorties sont :

$$M_1 = X_1 + X_2 + X_5; M_1 = X_2 + X_3 + X_6;\ L_1 = X_1 + X_6;\ L_2 = X_2;\ L_3 = X_3 + X_5;\ L_4 = X_4;\ L_T = X_4.$$

Avec :

M_1, M_2 : Moteurs.

L_1,L_2, L_3, L_4, L_T : voyants.

La réalisation matérielle, du réseau de PETRI, à base de relais électriques est la suivante :

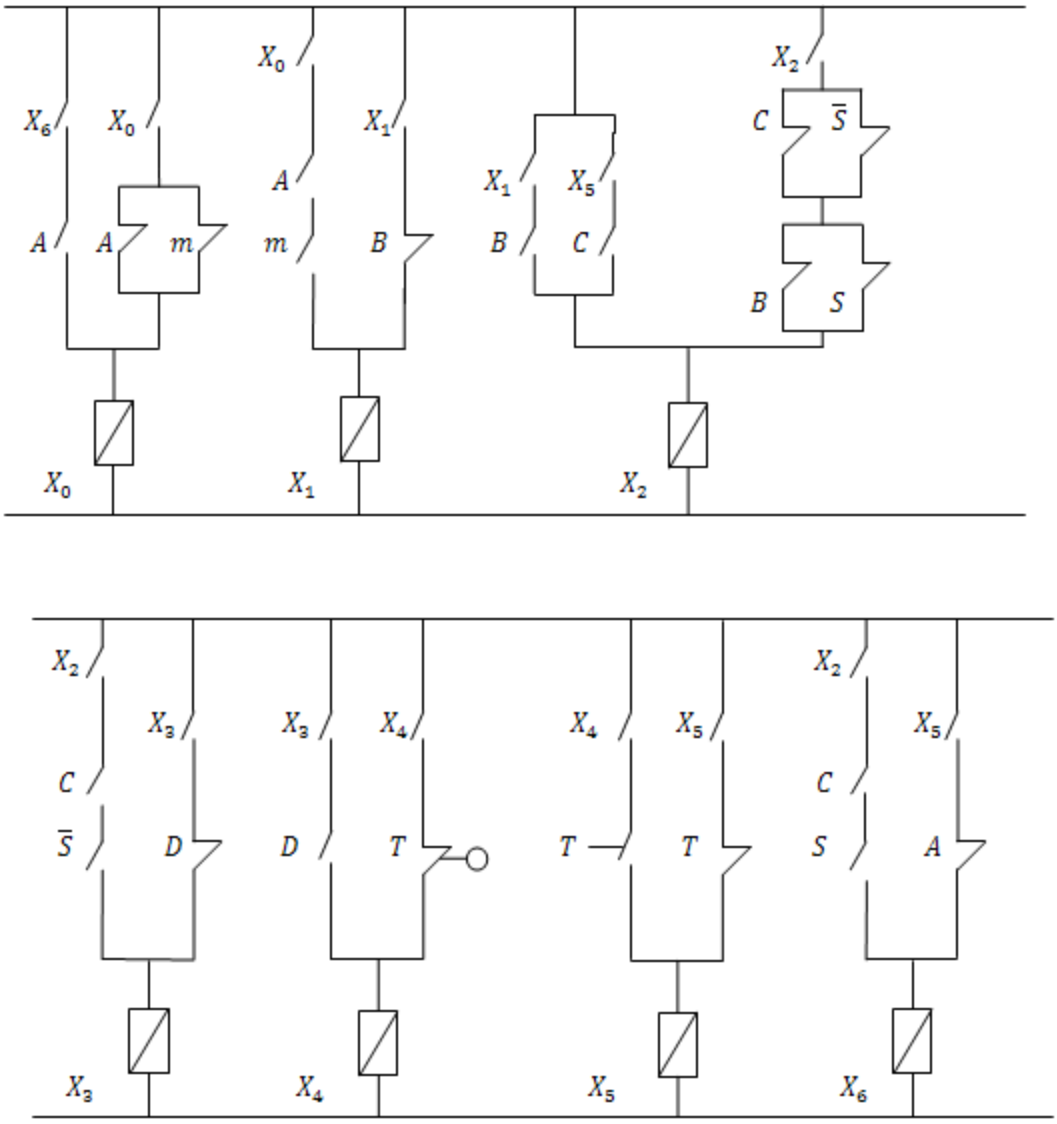


Figure II.34 : Réalisation materielle de l'exemple du chariot

2.4.1.3. Grafcet :

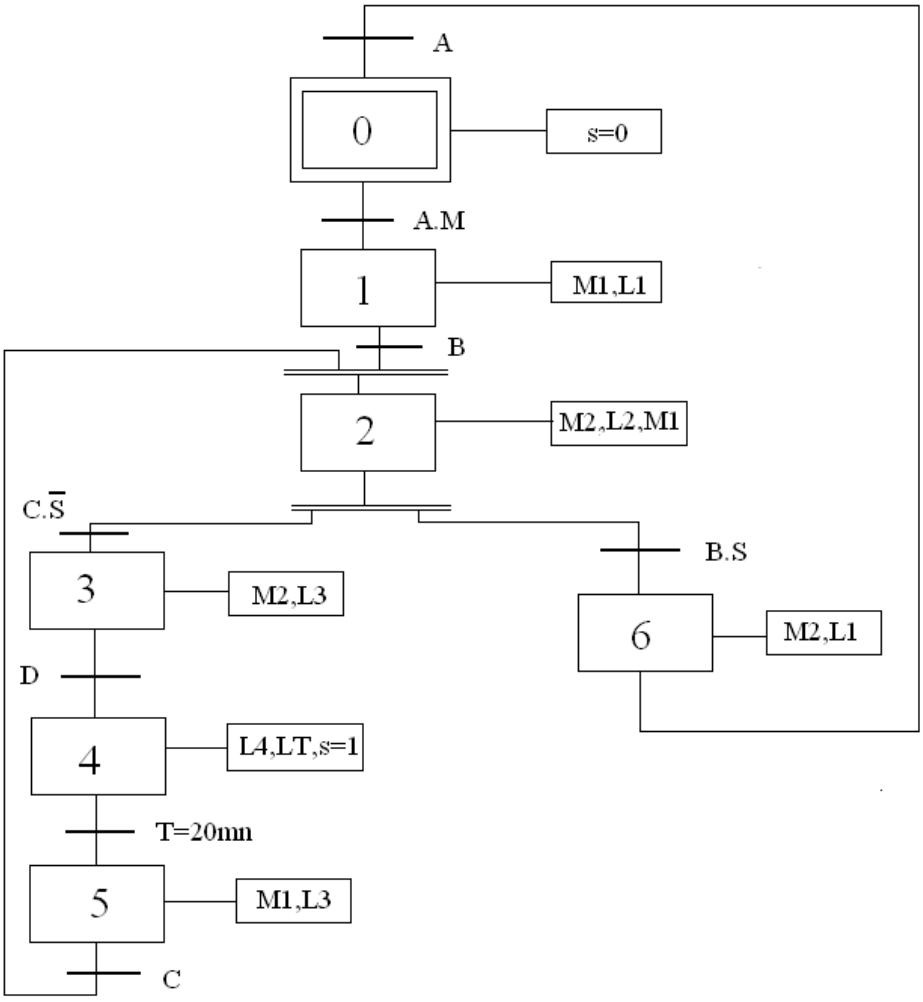


Figure II.35 : Grafcet de l'exemple du chariot

2.4.1.4. Programmation avec LADDER :

Il suffit de retranscrire la réalisation matérielle en langage (LD). Pour ce faire on a besoin des contacts, des bobines et d'un bloc de temporisation. On utilise des « mémento (Mi) » pour chaque « état (Xi) »,ces mémento permettent la sauvegarde de l'état.

On a utilisé le bloc de tempo « TON » importé de la bibliothèque, elle est enclenchée par « M4 ». La sortie « FBI_ .Q » ne se met à 1 qu'après l'écoulement du temps T2. La figure suivante montre la mise à 1 de l'état X4 et fait intervenir la plupart des instructions. Le programme intégral est donné en annexe. **[ANNEXE C.4]**

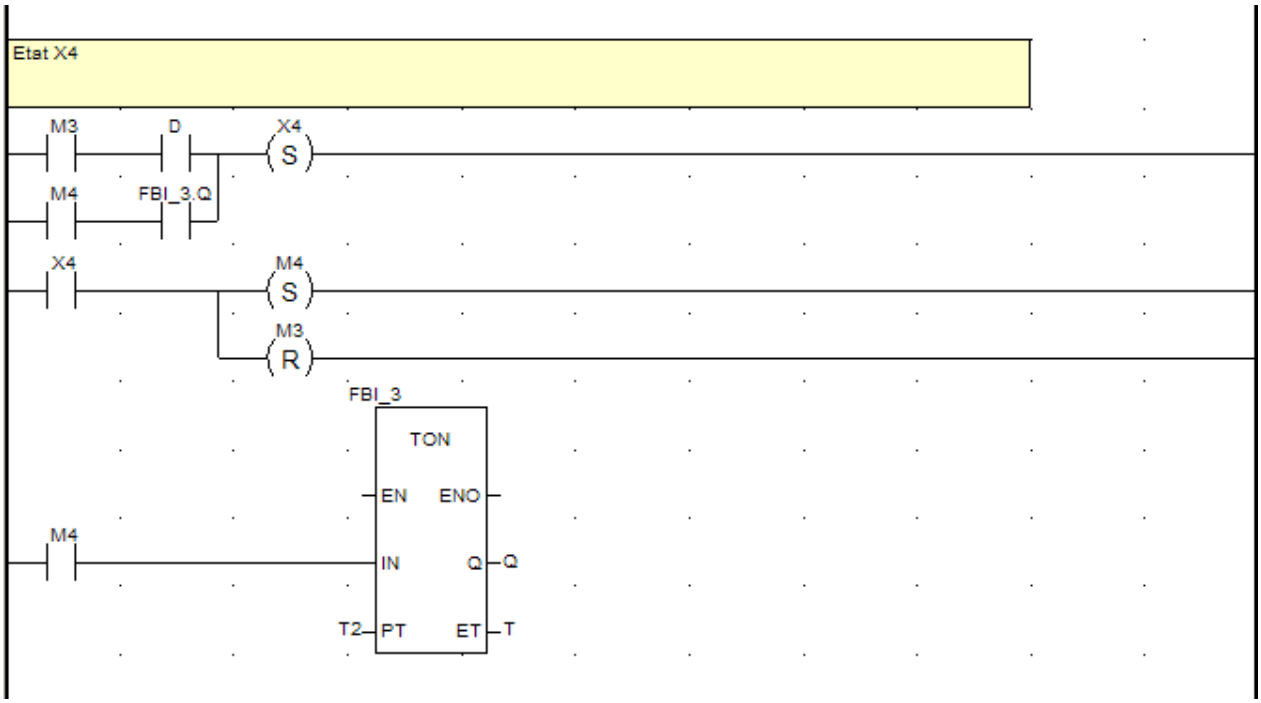


Figure II.36 : Programme du chariot.

2.4.1.5. Simulation

On modifie ou (force pour les variables périphériques) les variables selon le cahier de charge et on constate le changement dans les sorties affectées à un module de sortie.

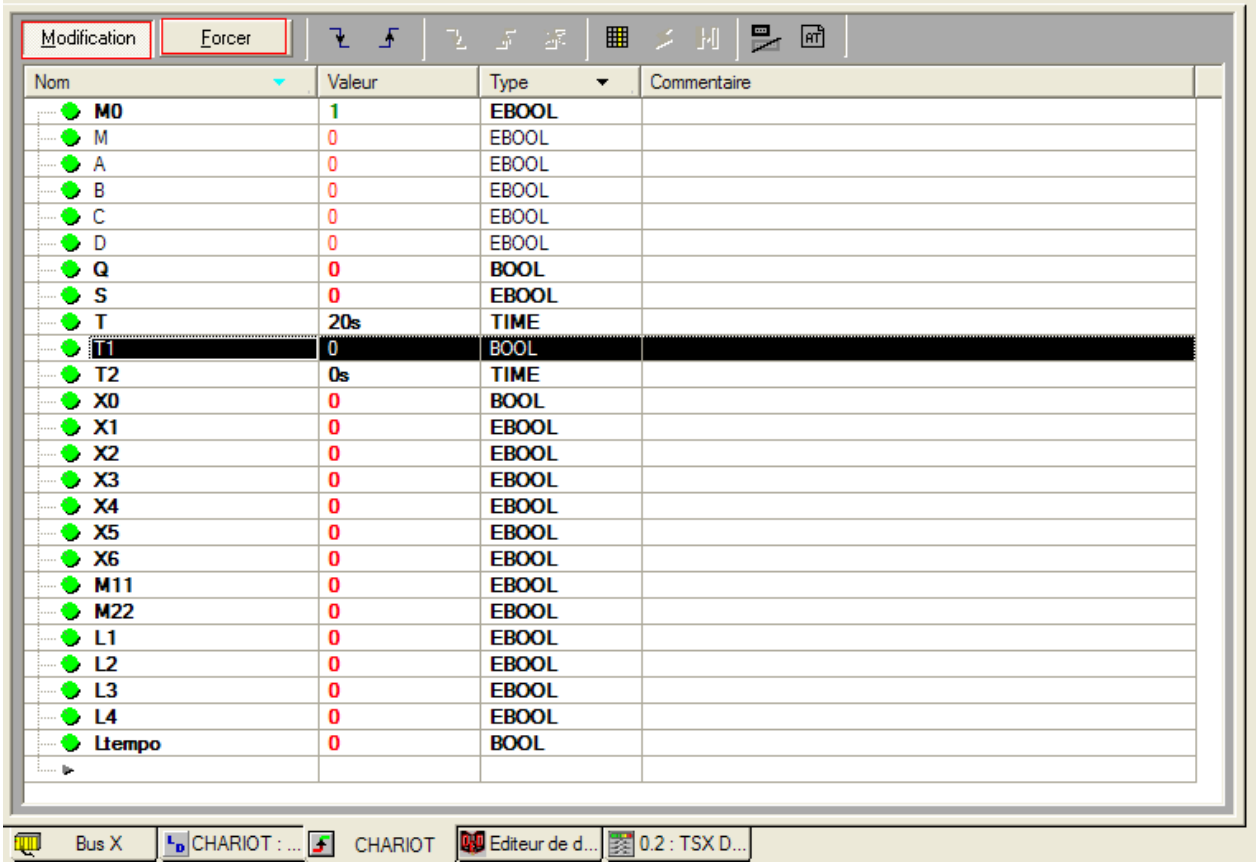


Figure II.37 : Simulation du programme du chariot.

2.4.2. Simulation d'un circuit RC

2.4.2.1. Position du problème

Soit à simuler le circuit suivant :

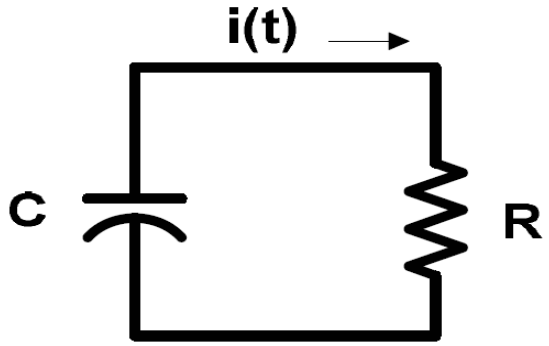


Figure II.38 : Circuit RC

On a : $0 = V_R + V_c$ $II.1$

Et : $I = C\frac{dV_c}{dt}$ $II.2$

On obtient alors l'équation différentielle de premier ordre suivante :

$$RC\frac{dV_c}{dt} + V_c = 0 \quad II.3$$

On l'écrit sous la forme : $\dot{V_c} = \left(-\frac{1}{RC}\right) * V_c = f(V_c)$ $II.4$

La méthode utilisée est celle RK45 :

$$V_{i+1} = V_i + \frac{h}{6}(k_1 + 2k_2\,2k_3 + k_4)k_1 = f(t_i, v_i) \quad II.5$$

Avec $k_2 = h.f\left(t_i + \frac{h}{2}, v_i + \frac{hk_1}{2}\right)$ $II.6$

$$k_3 = h.f\left(t_i + \frac{h}{2}, v_i + \frac{hk_2}{2}\right) \quad II.7$$

$$k_4 = kh.f(t_i + h, v_i + hk_3) \quad II.8$$

Pour la simulation on prendra :

C=1 uF, R=10 KΩ et h=0.01 sec

$$V_{c0} = 10\,V(condensateurinitialementchargé)$$

2.4.2.2. Programmation avec « IL »

Pour la programmation, nous avons utilisé une DFB pour programmer la fonction « f ». On appelle cette fonction pour le calcul des différents « Ki ». Le programme est donné en annexe.

2.4.2.3. Simulation

On utilise la table d'animation suivante en modifiant les paramètres des différentes entrées :

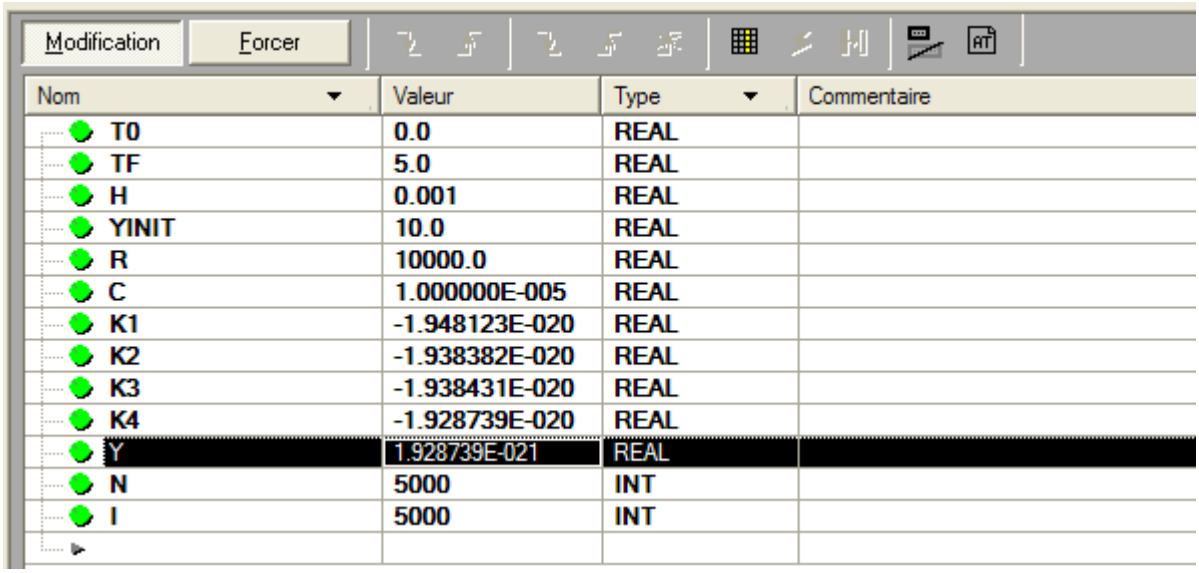


Figure II.39 : Simulation du circuit RC.

On voit bien qu'à la fin de la simulation la valeur de la tension au bord du condensateur qui est représentée par « y » tend vers zéro. Ceci correspond à la décharge du condensateur

2.4.3. Fabrication d'un mélange de produit

2.4.3.1. Cahier de charge :

Le système comporte :

-Un réservoir

-5 robinets : R1 et R2 pour la recette1, R3 et R4 pour la recette 2, et R pour la vidange

-4 zones de saisie pour entrer les valeurs des niveaux bas « L », intermédiaire « H1 » et haut « H2 » ainsi que la valeur de la temporisation

-4 boutons de commande : 2 pour choisir la recette, un pour le démarrage du processus « m », et un dernier pour forcer la vidange -un mélangeur

Dès que la recette est choisie et que le bouton m est activé, le 1er robinet (R1 pour la recette 1 et R3 pour la recette 2) sera ouvert jusqu'au niveau H1 puis le 2ème robinet s'ouvre jusqu'au niveau H2. A ce moment, une temporisation est déclenchée et le mélangeur démarre, après la temporisation, la vidange ce déclenche automatiquement jusqu'au niveau L. On peut arrêter le système à tout moment, et forcer la vidange aussi à tout moment.

2.4.3.2. Programmation avec ST :

On a utilisé l'instruction « case » pour le choix des recettes, les boucles « IF » pour simuler le fonctionnement et on a fait appel à un bloc de temporisation. Le programme est donné en annexe. **[ANNEXE C.4.1]**

2.4.3.3. Simulation :

Pour ce faire, on utilisé un écran d'exploitation, on a importé les différents objets cités dans le cahier de charges de la bibliothèque d'objet (figure)

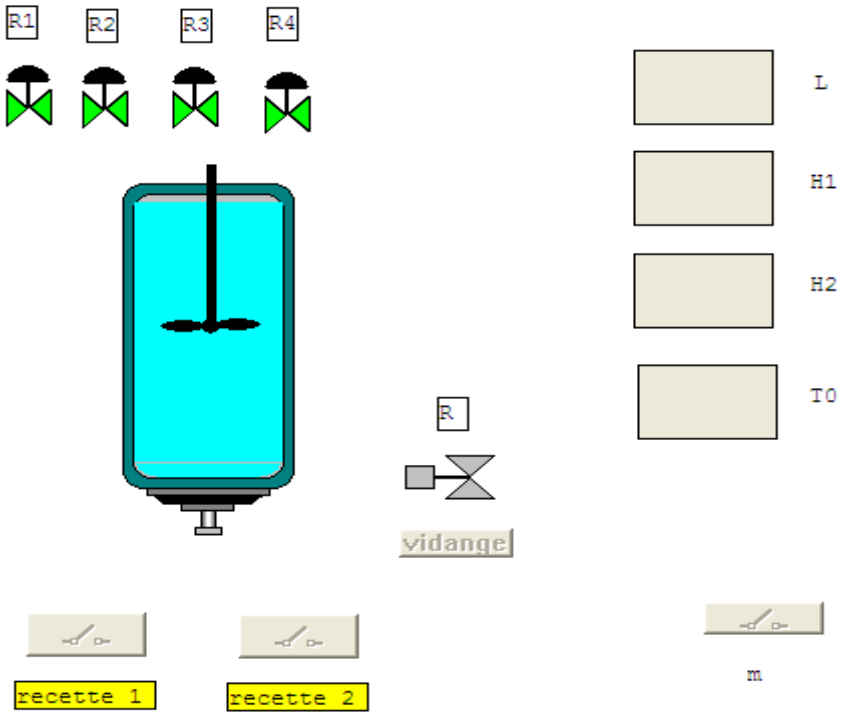


Figure II.40 : Objets constituant l'écran de l'exemple

Pour que les objets de pilotage ou de commande soient exécutables (les affichages par exemple), il faut appuyer sur l'icône « Valider l'écriture des variables ». De là, on peut simuler le système suivant le cahier de charge. Les différentes étapes de la simulation sont :

1ère étape : remplissage entre le niveau bas et le niveau intermédiaire (Figure C5)

2ème étape : remplissage entre le niveau H1 et H2 (Figure C6)

3ème étape : temporisation et démarrage du mélangeur (Figure C7)

4ème étape : Vidange ((Figure C8)

[ANNEXE C.4.3]

2.4.4. Exemple d'un SFC

2.4.4.1. Cahier de charge

Les éléments constituant l'application sont :

- 2 boutons de commande GD et DG
- 3 voyants lumineux

Le but est d'allumer les voyants un après l'autre de gauche à droite si l'on appui sur le bouton GD et de droite à gauche si l'on appui sur le bouton DG.

2.4.4.2. Programmation avec SFC

La transition entre les voyant étant très rapide, nous avons posé cette variable de transition comme un temporisation « T » qui a été programmé en LADDER. Le programme est donné en annexe. **[ANNEXE C.4.4]**

2.4.4.3. Simulation avec un écran d'exploitation

On a crée deux boutons pour les directions « gauche droite » et « droite gauche », il y a aussi des voyants de visualisations.

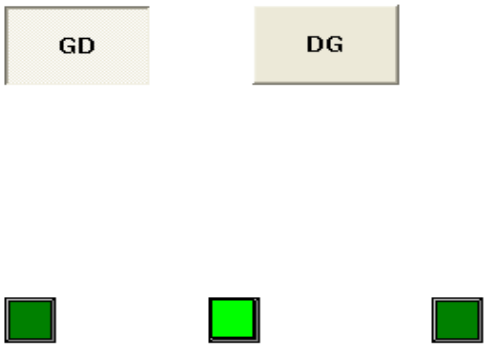


Figure. II.41 Ecran d'exploitation de l'application SFC

3. Conclusion

Intégrant la totalité des langages et les outils nécessaires pour une programmation complète et souple, la gamme des logiciels UNITY PRO offre une facilité d'utilisation et de compréhension.

La possibilité de créer des blocs de fonctions utilisateurs « DFB » et de les intégrer à la bibliothèque est un avantage de taille. On peut ainsi élaborer nos propres fonctions et les utiliser dans n'importe quel projet

CHAPITRE III

VIJEO DESIGNER

1. Introduction

Vijeo Designer est un logiciel de création de projets IHM (interface homme-machine), développé par Schneider Electric Industries SAS.

On peut exécuter des applications utilisateur (les projets IHM créés dans Vijeo Designer) sur un large éventail d'ordinateurs, de plates-formes et d'environnements.

On peut créer des écrans très simples mais aussi très complexes, intégrant des graphiques et des animations répondant à des besoins. En outre, l'approche unique de Vijeo Designer quant à la conception et l'implémentation d'interfaces homme-machine réduit les tâches de programmation au minimum.

Vijeo Designer propose des fonctions d'animation polyvalentes qui permettent de créer des écrans en un tour de main. Il offre également un large éventail de composants polyvalents, rapidement configurables et compatibles avec l'interface graphique conviviale de Vijeo Designer.

Il permet de configurer rapidement des connexions à plusieurs équipements. Il prend également en charge une large gamme de pilotes d'équipement qu'on peut utiliser pour transmettre des données sans aucune programmation.

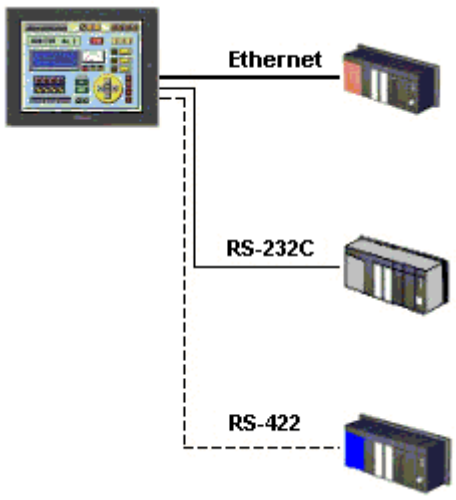


Figure III.1 : Connexion de Vijeo Designer à plusieurs automates

2. Types d'écrans :

On peut créer des écrans standards pour les opérations et le contrôle en combinant divers composants tels que des objets, des objets graphiques et des animations. Les écrans types que nous pouvons créer sont des écrans opérationnels, de contrôle, de fenêtre popup et des écrans alarmes

2.1. Les écrans opérationnels :

Les écrans opérationnels fonctionnent avec les périphériques. Dans Vijeo-Designer, on Ces écrans contiennent des commutateurs, des voyants et des claviers.

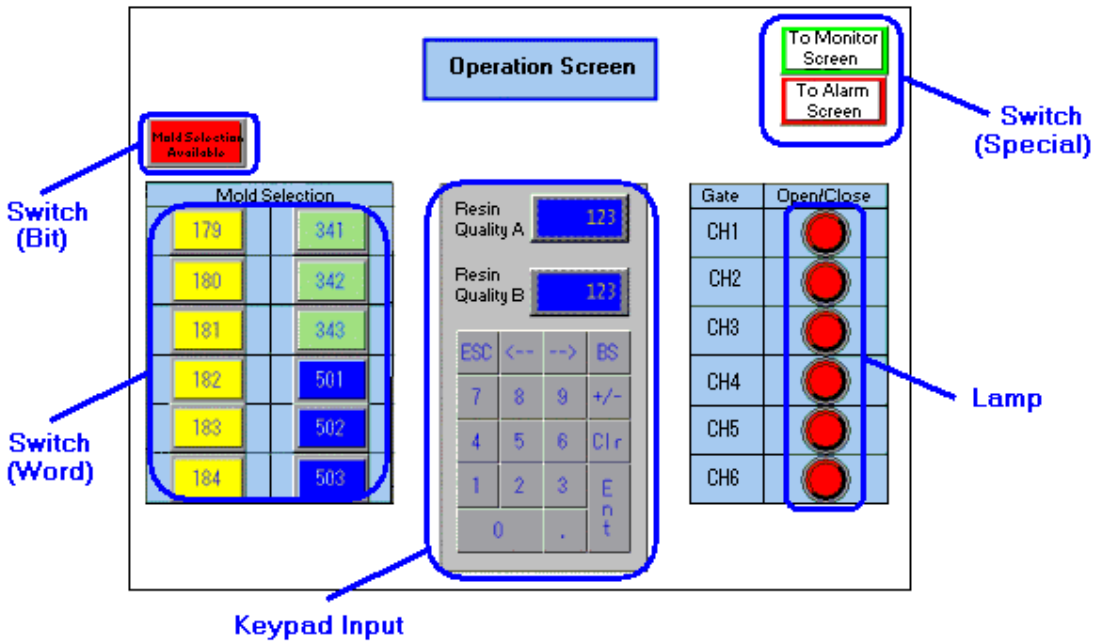


Figure III.2 : écran opérationnel.

2.1.1. Ecrans de contrôle :

Les écrans de contrôle affichent des données. Dans ce type d'écran, on peut afficher sous forme de valeurs numériques ou de graphiques, la température, la pression ainsi que d'autres mesures enregistrées dans les adresses de périphérique. Ces écrans de contrôle se composent ainsi de:

- Affichages de données, affichages de la date/heure et affichages de texte
- Courbes
- Courbes de tendance
- Les affichages de messages et d'images

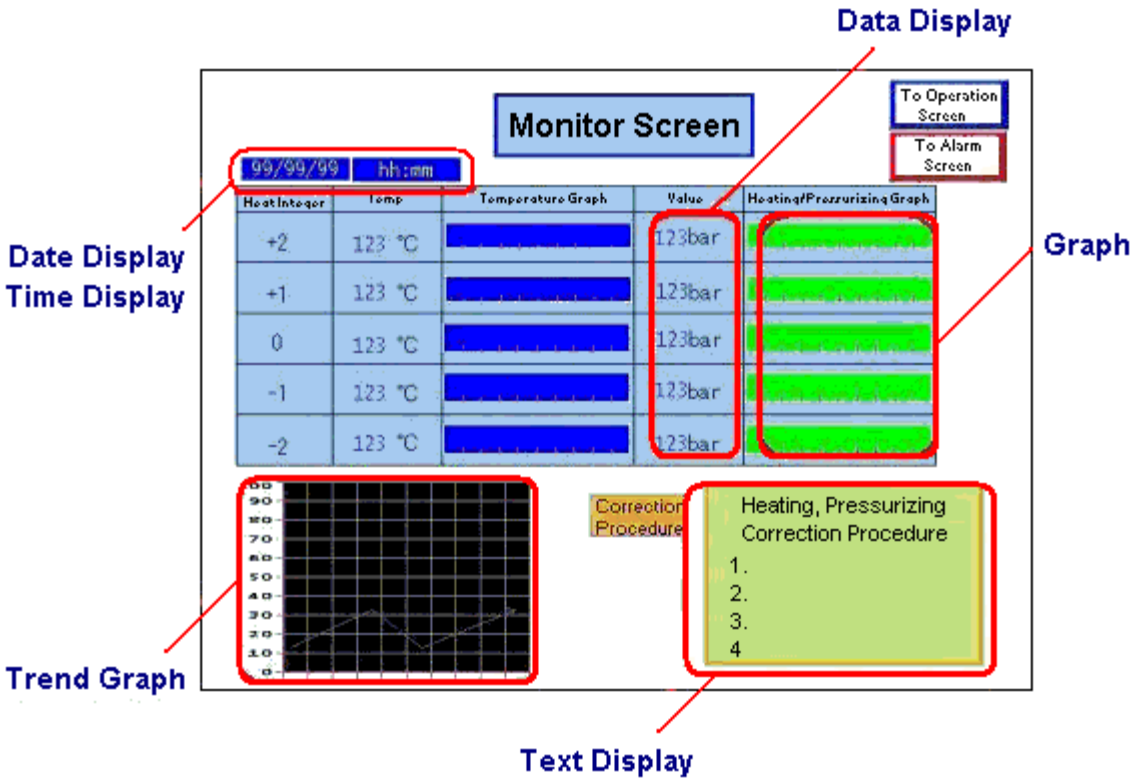


Figure III.3 écran de contrôle.

2.1.2. Ecrans de fenêtres popup :

On peut afficher des fenêtres popup à l'aide d'un interrupteur ou d'un script. Les fenêtres popup s'affichent au-dessus de l'écran actif (voir l'exemple ci-dessous).

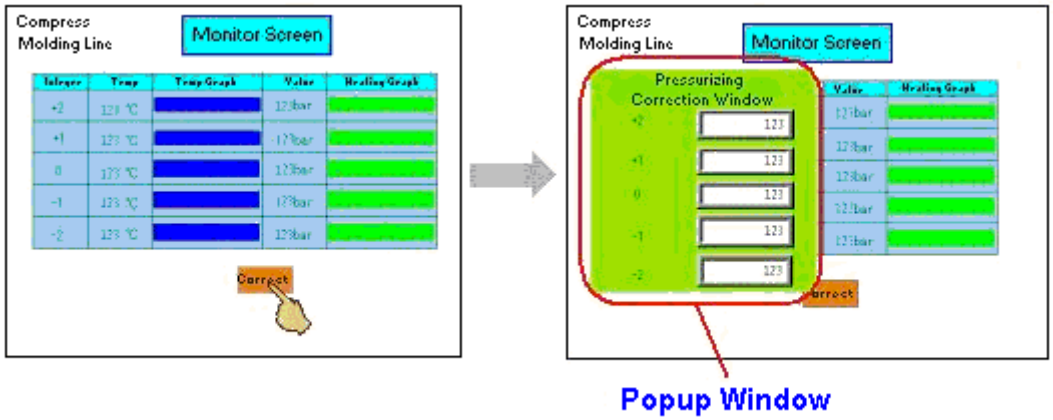


Figure III.4 : écran de fenêtres popup

2.1.3. Ecrans alarmes

Il faut utiliser l'objet Résumé d'alarme pour créer facilement des écrans alarmes. On utilise l'objet de commutateur pour créer des boutons qui gèrent les opérations d'alarmes comme la sélection, l'acquittement et l'effacement des alarmes.

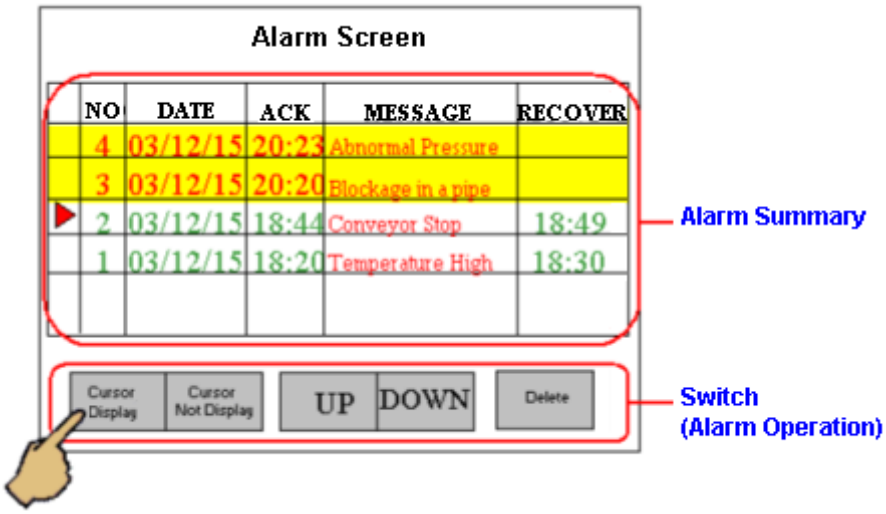


Figure III.5 : écran alarmes.

3. Types d'objets

Les objets, les dessins et l'animation sont les fonctionnalités essentielles de création d'écrans dans Vijeo Designer. Elles permettent de créer différents types d'écran, tels que des écrans opérationnels, de contrôle, de fenêtre popup et des écrans alarmes. Le tableau en **[ANNEXE2]** répertorie les objets disponibles dans Vijeo Designer.

4. Les variables

Les variables sont des espaces nommés de la mémoire qui stockent des données. Une variable peut être considérée comme un contenant de valeurs de données.

Pour communiquer avec les automates et les autres équipements connectés à la machine cible, on peut créer une variable et affecter une adresse de périphérique.

La variable de Vijeo-Designer, qui est affectée à une adresse de périphérique, est mise à jour lors de chaque modification des données au niveau de l'équipement. Si on associe les adresses de périphérique à des variables, Vijeo-Designer peut lire et afficher les données provenant de l'équipement.

Lorsque plusieurs équipements sont connectés à la machine cible, on peut afficher les données d'adresse de périphérique provenant de différents équipements dans le même écran.

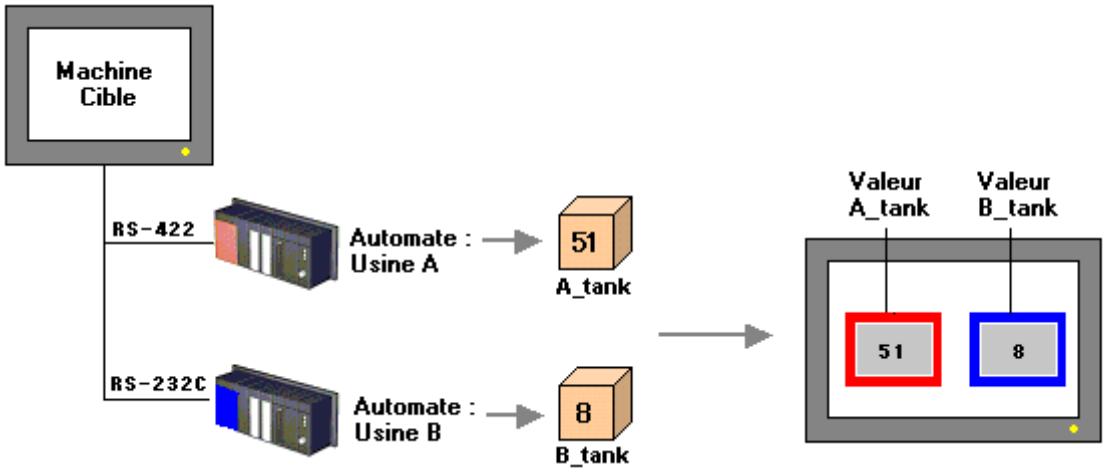


Figure III.6 : Plusieurs équipements pour une seule machine

Les variables internes ne sont pas associées à un équipement, et sont utilisées uniquement pour les opérations internes de Vijeo-Designer. Contrairement aux variables externes qui sont associées à un équipement.

Il existe six types de variable Vijeo-Designer : TOR (booléen), entier, flottant, chaîne, entier de bloc, et flottant de bloc. Vijeo-Designer propose également un autre type de variable, appelée Structure, c'est-à-dire un dossier qui contient plusieurs variables. **[ANNEXE D.3]**

5. Création d'un projet

Cette section décrit l'assistant qui s'affiche lorsque Vijeo-Designer est lancé pour la première fois. L'assistant guide pour créer un projet dans Vijeo-Designer.

Figure III.7 : Assistant de création d'un nouveau projet

Ensuite, dans la boite de dialogue, il faut configurer les champs suivants :

- Nom du projet
- Description ou commentaire (Ce champ est limité à 255 caractères.)

- Type : Il faut indiquez si le projet dispose d'une ou de plusieurs cibles. Si plusieurs, indiquez le nombre.

- -Mot de passe du projet (si nécessaire).

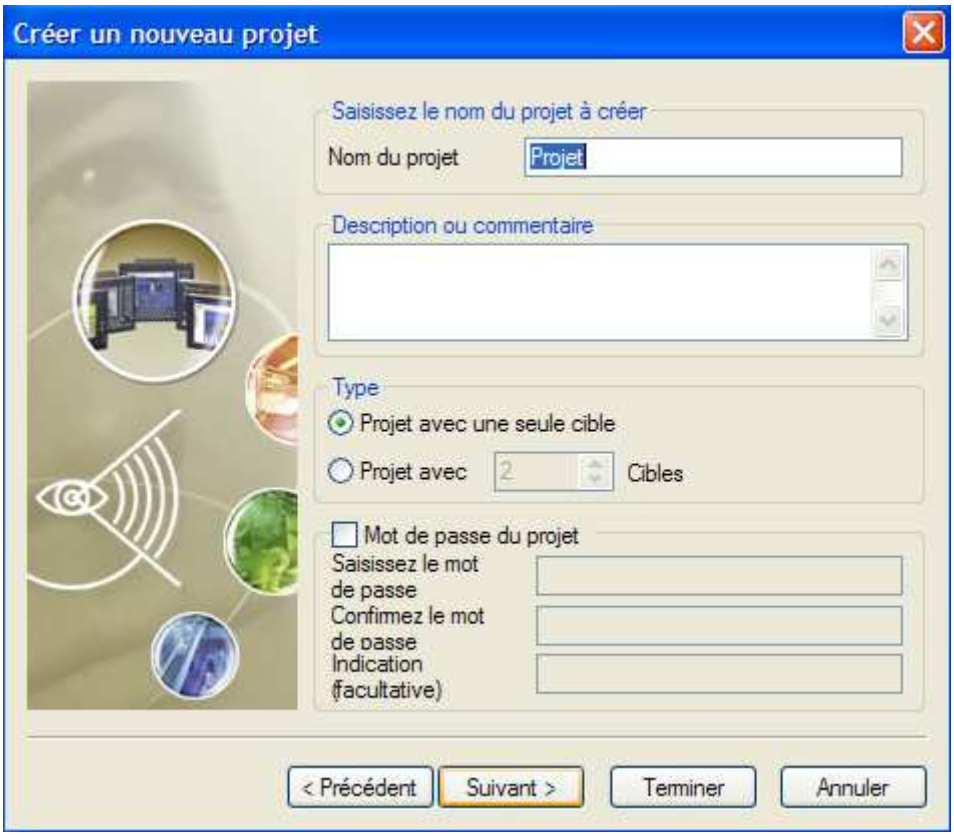


Figure III.8 : Configuration du projet

5.1. Configuration de la cible.

Il faut choisir, le nom, le type et le modèle de la cible. Vijeo Designer prend en charge un bon nombre d'écran tactiles.**[ANNEXE D.1]**

Figure III.9 : Choix de la cible.

Après avoir choisi la cible, il faut saisir l'adresse IP de la cible

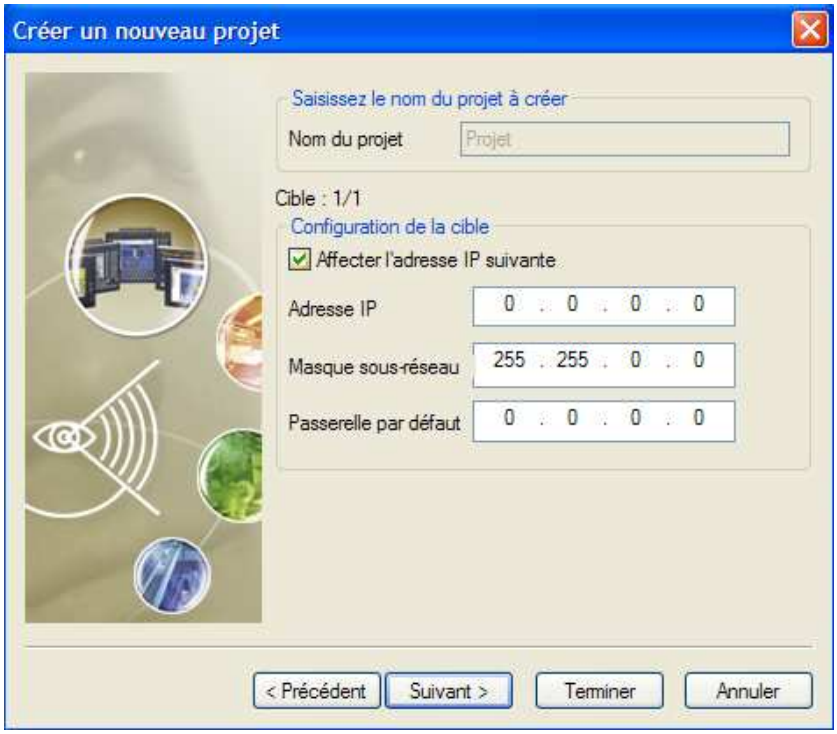


Figure III.10 : Saisie de l'adresse IP de la cible.

5.2. Configuration du pilote de communication

Pour ajouter un équipement. Il faut cliquer sur Ajouter pour ouvrir la boîte de dialogue Nouveau pilote. Puis sélectionner un équipement.

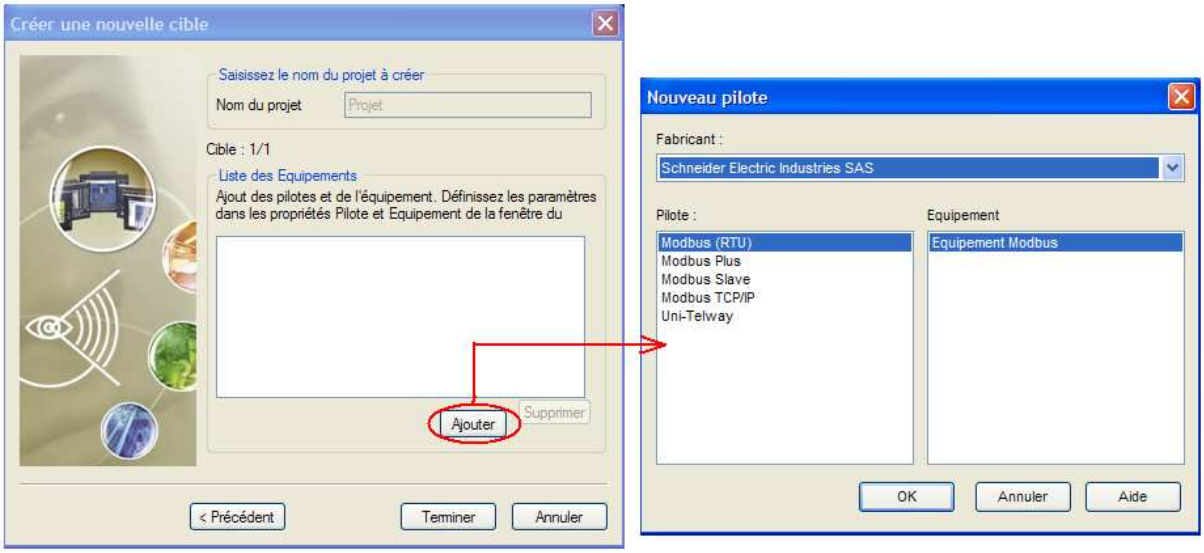


Figure III.11 : Choix du pilote de communication

Pour communiquer avec des équipements, il faut les connecter au port série (RS-232C/RS-422), au port Ethernet ou au module/à la carte de communication de la machine cible, puis ajouter un pilote. Vijeo-Designer utilise des pilotes pour activer les communications avec l'équipement. Cela évite de créer des programmes de communication complexes.

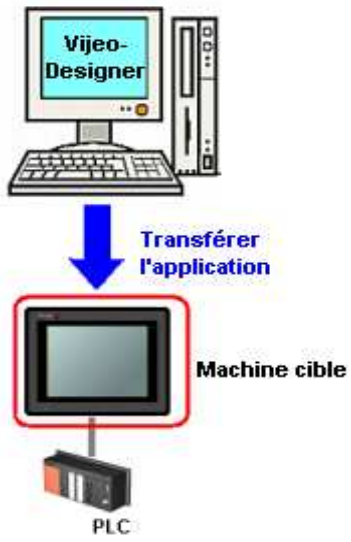


Figure III.12 : Machine cible.

Remarque : Sur la machine cible, il faut installer Vijeo Designer Runtime où l'on va exécuter l'application conçue dans l'éditeur Vijeo Designer. L'éditeur Vijeo designer et Vijeo Designer Runtime doivent donc être compatibles.

En général, on choisit le pilote Modbus TCP/IP, dans ce cas la figure suivante sera affichée. Il faut alors configurer l'équipement en spécifiant l'adresse IP de l'automate programmable qui sera en communication avec l'HMI. On coche aussi la case « syntaxe IEC61131 » afin que les adresses soient conformes à cette norme

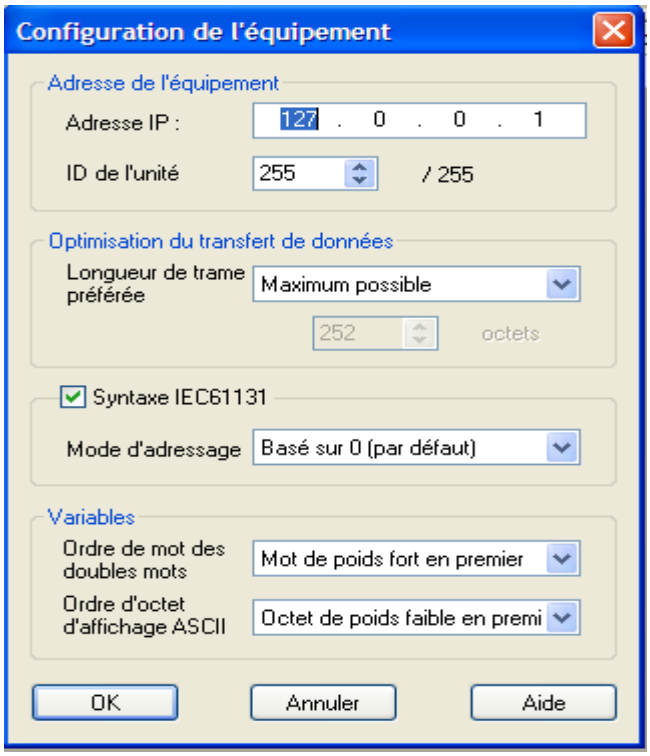


Figure III.13 : Configuration de l'équipement Modbus TCP/IP.

Remarque : Vigeo designer peut communiquer avec le simulateur de « UNITY PRO ». Ainsi on pourra créer des projets en mode simulation et les visualiser à l'aide de Vijeo designer. Pour ce faire, on choisit le pilote « modbus TCP-IP » et on met l'adresse du simulateur « 127.0.0.1 » dans la configuration de l'équipement (figure ci-dessus)

5.3. Création des écrans :

5.3.1. Dessin de graphique

L'Editeur Vijeo-Designer utilise deux types d'objets graphiques : des outils de dessin et des objets. La Bibliothèque d'objets Vijeo-Designer est une bibliothèque d'images prêtes à l'emploi représentant des équipements industriels et d'autres objets matériels.

Figure III.14 : Objets graphiques.

On constate deux types d'objets graphiques : Outils de formes et Objets configurables.

5.3.1.1. Les outils et formes

Ce sont des formes géométriques, pouvant être animées grâce à des variables. On définit leur animation en double cliquant sur l'objet concerné.

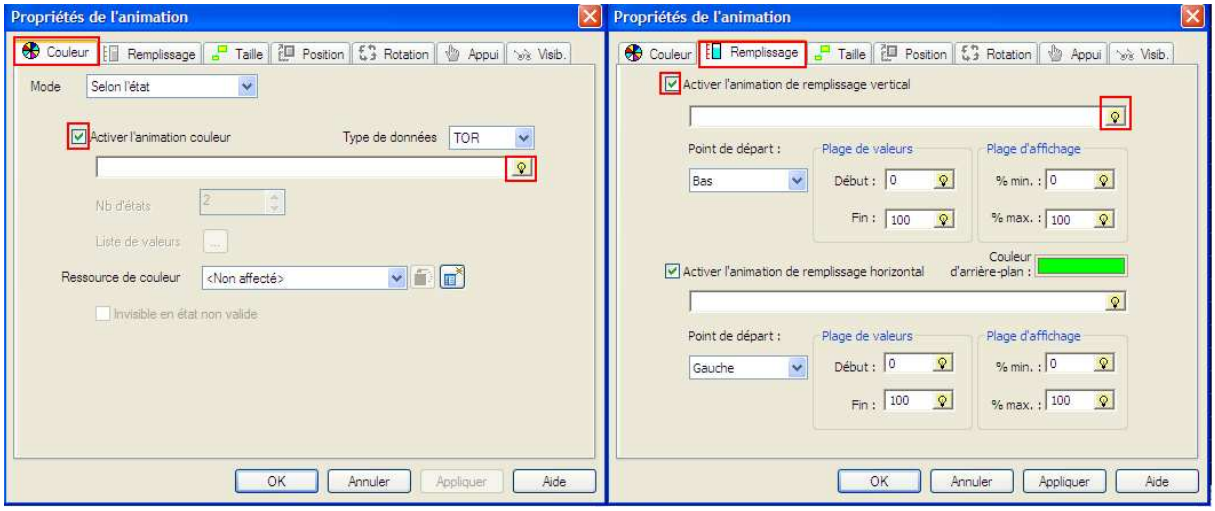


Figure III.15 : Animation d'un outil de forme.

Comme on le voit sur la figure précédente, il y a plusieurs onglets. On peut les configurer en activant l'animation et en leur affectant une variable qui définit leurs variations. Vijeo Designer prend en charge les huit types d'animation suivants :

- **Animation de la couleur :** Modifier la couleur d'un objet.
- **Animation de remplissage :** Afficher graphiquement les variations de niveau.
- **Animation de la taille :** Afficher les variations de volume.
- **Animation Position :** Déplacer un objet verticalement et horizontalement
- **Animation Rotation :** Faire pivoter un objet.
- **Animation sur appui :** Utiliser un objet comme un commutateur.
- **Animation de la valeur :** Afficher ou saisir des données.
- **Animation Visibilité :** Afficher/masquer des objets

5.3.1.2. Les objets configurables

5.3.1.2.1. Les commutateurs

Vijeo-Designer fournit deux types de commutateurs que l'on peut créer à l'aide des outils qui se trouvent dans la barre d'outils Objets graphiques :

- Le commutateur simple
- Le bouton radios
- Le sélecteur circulaire
- Le sélecteur vertical/horizontal :

Tous ces commutateurs peuvent être configurés en double cliquant sur eux, on peut alors définir la fonction, la couleur selon l'action, la visibilité et les variables associées.

Un tableau des principales fonctionnalités des commutateurs est donné en annexe

5.3.1.2.2. Configuration des alarmes :

Dans Vijeo-Designer, on peut indiquer les alarmes à l'utilisateur sous divers formats d'affichage :

5.3.1.2.2.1. Affichage du résumé des alarmes :

On peut afficher une liste d'alarmes sur un écran à l'aide du résumé des alarmes. Les alarmes peuvent avoir quatre états : actif, acquitté (ACQ), non acquitté (NONACQ), et retombée. Le résumé d'alarmes peut afficher des alarmes qui sont dans ces quatre états. On peut imprimer ces messages d'alarme ou les enregistrer dans un fichier .csv.

5.3.1.2.2.2. Bannière d'alarme :

Les messages de bannière d'alarme défilent de la droite à la gauche de l'écran. On peut afficher les messages de bannière d'alarme à l'aide de l'affichage global et de l'affichage local. L'affichage global permet d'afficher des messages d'alarme sur tous les écrans. L'affichage local permet d'afficher des messages d'alarme uniquement sur un écran spécifique. On peut utiliser l'affichage global et l'affichage local en même temps.

Pour configurer une bannière d'alarme globale, il faut créer un groupe d'alarme puis créer des variables et configurer les messages d'alarmes, les groupes d'alarmes et définir d'autres paramètres d'alarmes, puis finalement activer la bannière d'alarme dans les propriétés de la cible. Pour la bannière d'alarme locale, il faudra faire glisser un objet de bannière d'alarme de la bibliothèque d'objet et le déposer dans l'écran.

5.3.1.2.2.3. Configuration du son :

Il est possible de signaler les alarmes aux opérateurs d'écran en configurant la machine cible pour qu'elle émette un son.

Pour configurer le son, il faut :

- Dans l'onglet Variables de la fenêtre du navigateur, cliquer deux fois sur une variable pour afficher la boîte de dialogue des propriétés de la variable.
- Sélectionner l'onglet Alarme et cochez la case Alarme.
- Cochez la case Son.

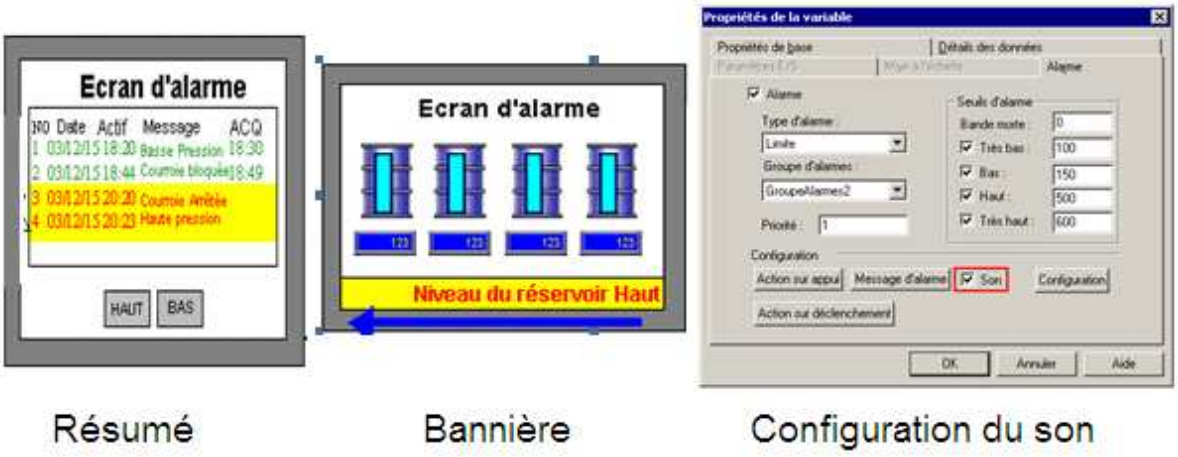


Figure III.16 : Configuration d'une alarme

5.3.1.2.3. Courbes de tendances

On utilise les courbes de tendances pour tracer les variations d'une variable donnée qu'elle soit locale ou externe (provenant d'un équipement).

On commence par configurer les paramètres de la grille de la courbe, à savoir,

- Le nombre d'échantillons affichés : dépend de la taille de la grille.
- Défilement : représente la valeur de l'échantillon duquel il commence à tracer
- Le pas : Dépend du défilement et du nombre d'échantillon
- La plage : définit la plage de la variable et celle de l'affichage de celle-ci
- Echelle des temps et des données : définit le nombre de divisions de la grille pour chaque courbe

Après ça, on affecte à la voie 1 la variable qu'on veut tracer. Si on veut tracer d'autres variables dans la même courbe, il suffit d'activer les autres voies. On peut aussi définir la couleur pour chaque variable

Remarque : pour pouvoir tracer une variable, il faut qu'elle appartienne à un groupe de journalisation. On peut configurer cela dans les propriétés de la variable.

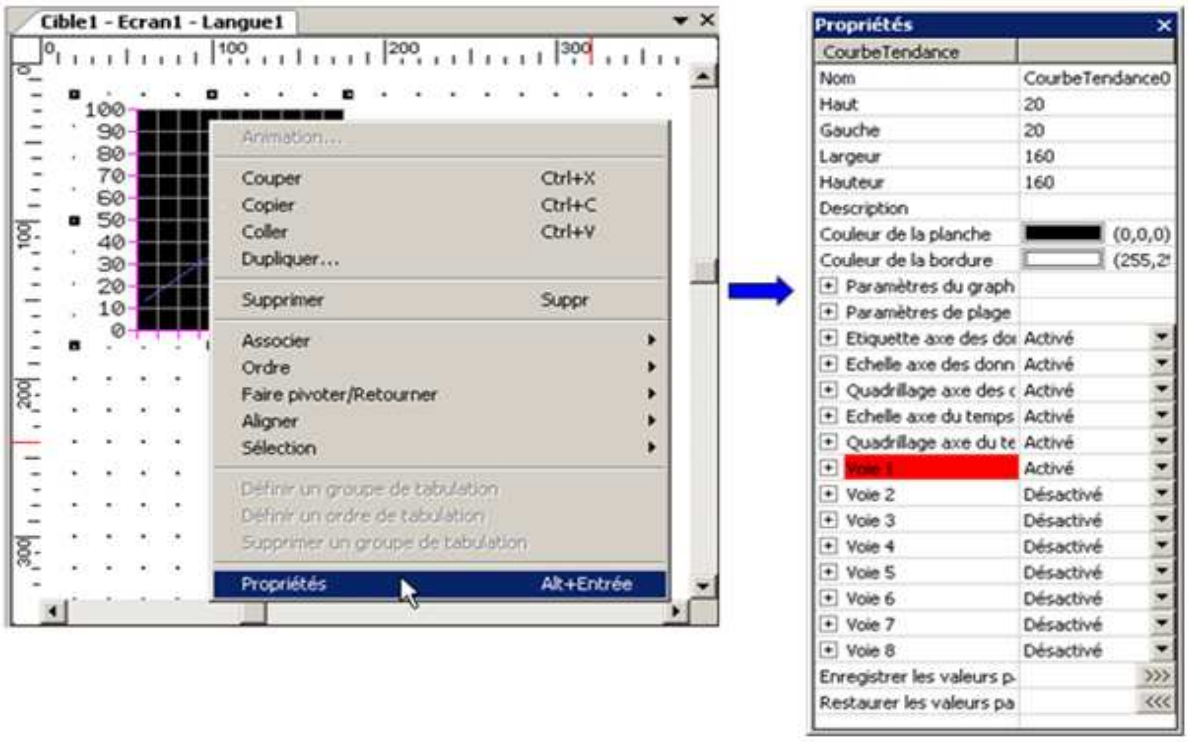


Figure III.17 : Configuration d'une courbe de tendance.

5.3.1.4. Bibliothèque d'objet :

La Bibliothèque d'objets est une hiérarchie de dossiers servant à stocker dessins, animations, écrans, scripts, groupes d'alarmes, objets graphiques, objets de bibliothèques d'images et objets de bibliothèque d'objets. Lorsqu'on stocke un objet graphique, la bibliothèque d'objets stocke également les variables associées à cet objet.

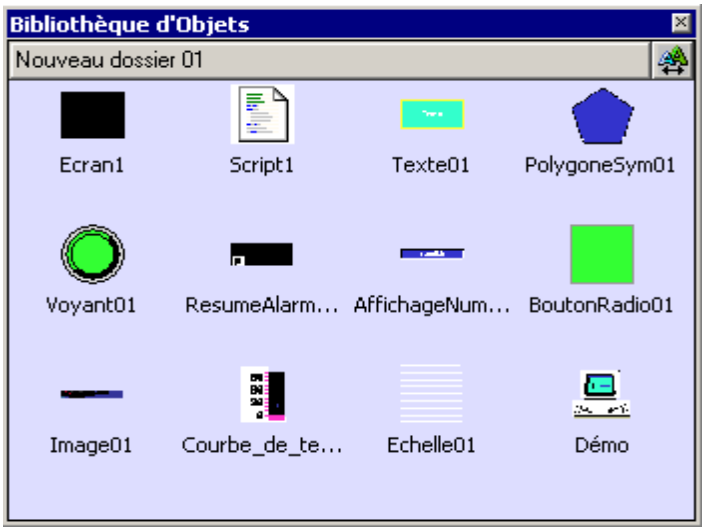


Figure III.18 : Bibliothèque d'objets.

5.4. Création de variables :

Comme cité auparavant, les variables peuvent être de six types et elles peuvent être internes ou externes.

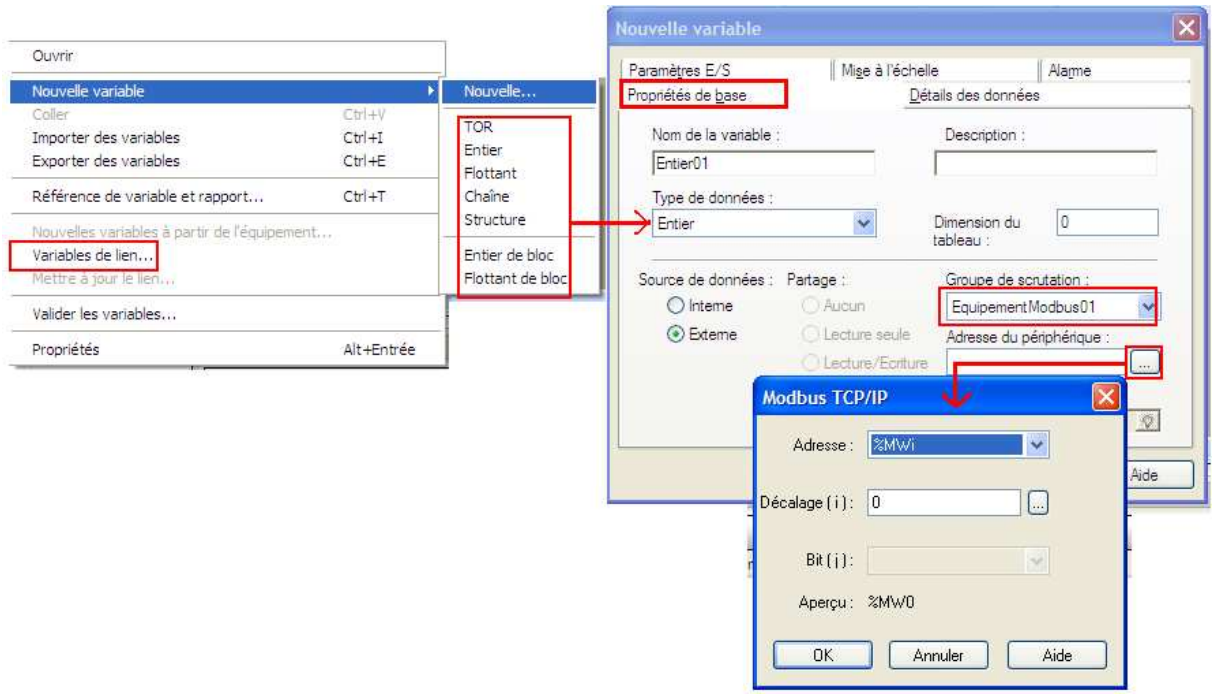


Figure III.19 : Création d'une variable.

Comme on le voit on peut directement apporter une table de mnémonique d'un programme UNITY PRO et ce en cliquant sur « variable de lien ».

Sinon on clique sur « nouvelle variable » on aura la possibilité de créer une variable interne ou externe en spécifiant son adresse dans l'équipement (l'automate par exemple)

L'onglet ***« alarme »*** permet d'ajouter la variable à un groupe d'alarme et définir des actions quand celle-ci arrive

L'onglet ***«détails des données »*** permet de définir la plage de variation de la variable et d'ajouter celle-ci à un groupe de journalisation.

L'onglet mise à l'échelle permet la mise à l'échelle de la variable.

5.5. Les scripts :

Un script Vijeo-Designer contient des instructions écrites par des utilisateurs expérimentés afin de programmer le type de réaction de la machine cible face aux événements en temps réel, tels que : une pression sur un appui, une modification d'écran ou un changement de valeur. La principale raison de l'utilisation de scripts Vijeo-Designer s'explique par le fait que la programmation n'est pas possible à l'aide des objets ou d'une animation. Une autre raison justifiant le recours aux scripts est que vous pouvez utiliser des variables de source externe dans les opérations de script afin d'ajouter une dimension de programmation susceptible de manquer dans le programme de l'équipement.

Les scripts Vijeo-Designer sont basés sur le langage de programmation Java. On peut utiliser une partie des méthodes et classes Java, en plus des méthodes spécifiques à Vijeo-Designer. En dépit des différences entre les scripts Vijeo-Designer et Java, on peut, dans une large mesure, utiliser les opérateurs et les objets comme lorsque on programme dans Java.

Les scripts sont composés de deux parties : le déclenchement et les instructions codées.

- Le déclanchement : Le déclenchement définit le moment où le script est exécuté. Les déclenchements varient en fonction du type de script.

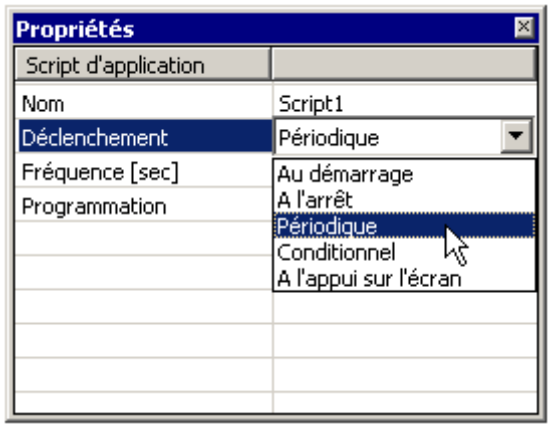


Figure III.20 : Déclanchement du script.

6. Exemple d'application :

Afin de montrer une partie des fonctionnalités du logiciel présentées auparavant, nous nous proposons d'étudier un système ayant le cahier de charge suivant.

6.1. Cahier de charge

On désire maintenir le niveau d'un réservoir entre un certain niveau 2 et un niveau max et ce comme suivant :

Si le niveau est inférieur à un niveau 1 alors la vanne 1 est ouverte seule et le remplissage du réservoir commence

A partir d'un certain niveau 2 la vanne 2 est ouverte et donc le remplissage continue avec une fréquence moins importante que la première.

Quand le niveau atteint le niveau max on ferme la vanne 1 tout en maintenant la vanne 2 ouverte, le réservoir commence alors à se vider.

Arrivé au niveau 2 la vanne 1 est ré-ouverte et le réservoir se remplit lentement de nouveau. Ainsi le niveau sera maintenu entre niveau 2 et niveau max.

6.2. Outils Vijeo Designer utilisés :

Afin de créer cet écran de visualisation on procède par les étapes suivantes :

6.2.1. Création des écrans :

Cette étape est réalisée à l'aide des outils de dessin, que l'on trouve dans la barre d'outils et des objets graphique que l'on trouve dans la bibliothèque d'objets.

L'écran **« MAST »** est l'écran principal, il comporte les objets suivant :

- Un réservoir
- Trois affichages numériques (les valeurs des niveaux 1,2 et max)
- Deux vannes, une pour le remplissage du réservoir et l'autre pour la vidange.

- Deux voyants lumineux pour différentes signalisations.

- Deux commutateurs « RUN » et « STOP » pour démarrer et arrêter le système

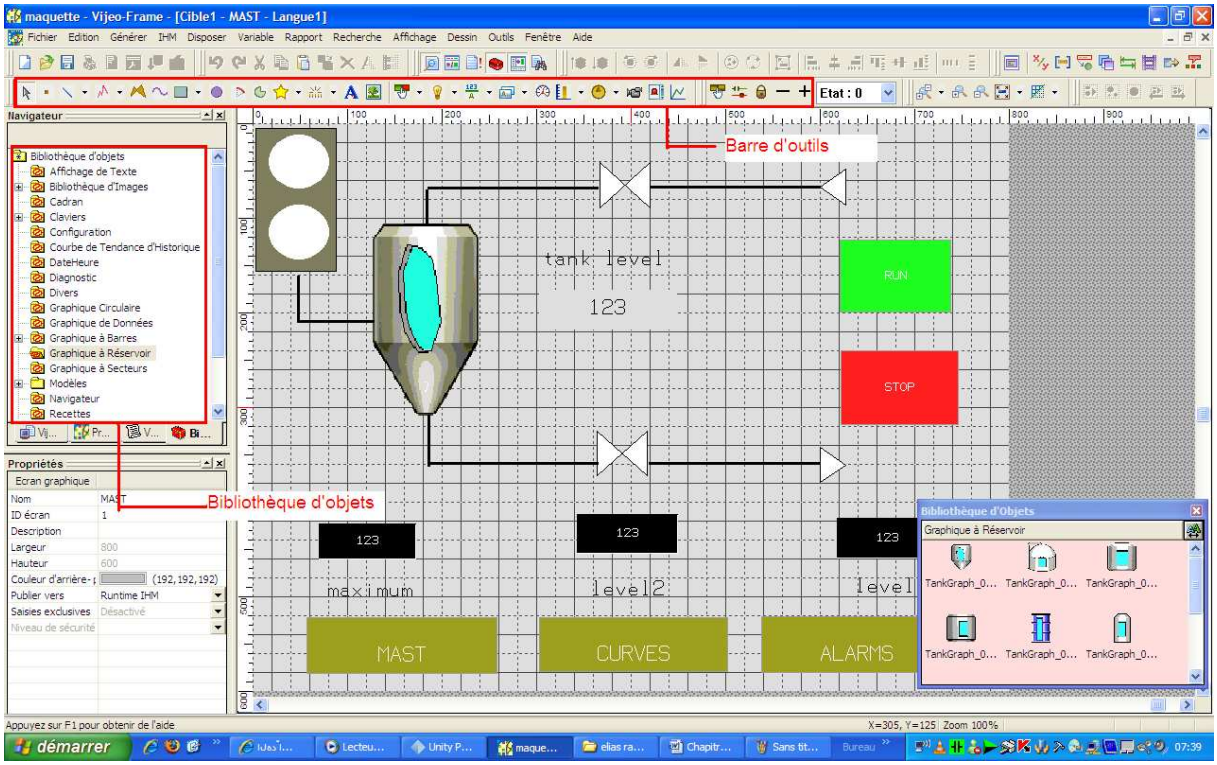


Figure III.21 : Création de l'écran1.

L'écran «**CURVES »** est une courbe de tendance qu'on fait glisser depuis la barre d'outils, on affecte à cette courbe variable « tank_level ».

Remarque : La variable « tank_level » doit être affectée à un groupe de journalisation, Ceci se fait en allant dans les propriétés de la variable.

L'écran **« ALARMS »** est un écran d'alarme, on le fait glisser depuis la barre d'outils. On peut modifier ses propriétés, telles que le groupe d'alarme, les colonnes du tableau…

Les alarmes seront donc affichées sur ce tableau, Ces alarmes sont affectées à des variables. Dans notre exemple, on a crée un variable « var5 » qui se met à 1 quand il y a un dépassement du niveau « Nmax ». On configure alors cette variable comme suit :

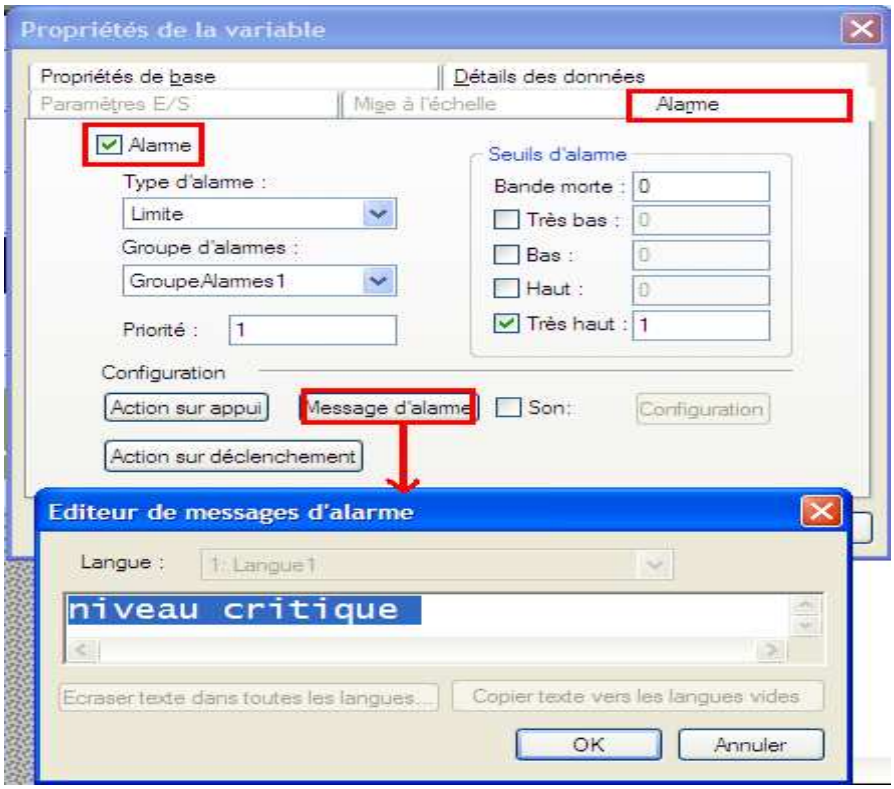


Figure III.22 : Configuration d'une alarme

Pour passer d'un écran à un autre, nous avons crée les commutateurs « MAST », « CURVES » et « ALARMS » dans les trois écrans. On les configure comme suit :

On va dans l'onglet « Général », on choisit la fonction « A l'appui » et l'opération « Ecran ». En fin, on choisit l'écran correspondant au bouton (dans la figure c'est l'écran MAST).

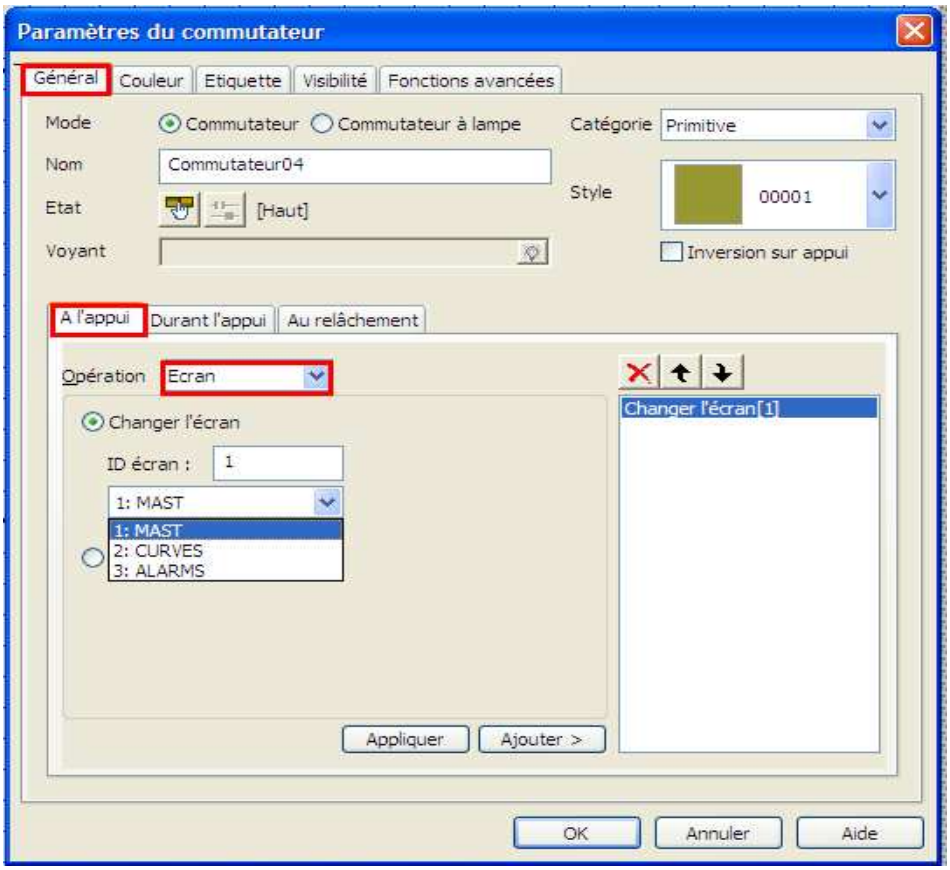


Figure III.23 : Commutateur de changement d'écrans.

6.2.2. Création des variables :

Dans cet exemple toutes les variables sont internes mais, en réalité, il devrait y avoir des variables externes liées aux équipements. Si c'est le cas on devra préciser l'adresse de celles-ci dans la mémoire.

Les variables configurées sont :

- Démarrage (TOR) : pour le démarrage de la simulation
- Arrêt (TOR) : Pour l'arrêt
- Level1, Level2, maximum (Entier) : variables des différents niveaux dont on a besoin. Ils font figure de détecteurs
- Tank_level (entier) : Niveau du liquide
- Var, var1, var2, var3, var4, var5 : variables intermédiaires

Remarque : Après avoir crée ces différentes variables, on attribue aux différents objets de notre écran ces variables là. Par exemple la variable « tank-level » est affectée à la fenêtre d'affichage du niveau du réservoir.

6.2.3. Elaboration des scripts :

En réalité, on devrait utiliser cette maquette afin de contrôler le système à distance et ce à partir d'un écran de visualisation. On utilise alors les mêmes variables que celles définies dans l'automate programmable

Dans notre exemple, le programme n'ayant pas été implémenté sur un écran de type XBT et l'automate n'étant pas branché, on a du utilisé des scripts pour pouvoir simuler le fonctionnement de ce système.

Nous avons utilisé pour cette maquette six scripts, il faut choisir « nouveau script » après avoir fait un clic droit sur « scripts d'applications ». Les scRipts sont donnés en **«ANNEXE D.4 »**

Remarque : Pour pouvoir simuler le processus et surtout le débit de remplissage et de vidange du réservoir, on modifie simplement ***la fréquence*** d'exécution du script correspondant dans l'onglet « *propriétés* ».Ainsi :

→Le script de remplissage entre les niveaux 1 et 2 (Script1) a une fréquence de 1sec car seule la vanne 1 est activée et on sait qu'elle a un fort débit.

→Le script de remplissage entre les niveaux 2 et max (Script2) a une fréquence de 3sec car les 2vannes sont activées. Le remplissage est alors le plus lent.

→Le script de vidange entre les niveaux max et 2 (Script3) a une fréquence de 2sec car seule la vanne 2 est activée et on sait qu'elle a un débit moins important que la vanne 1.

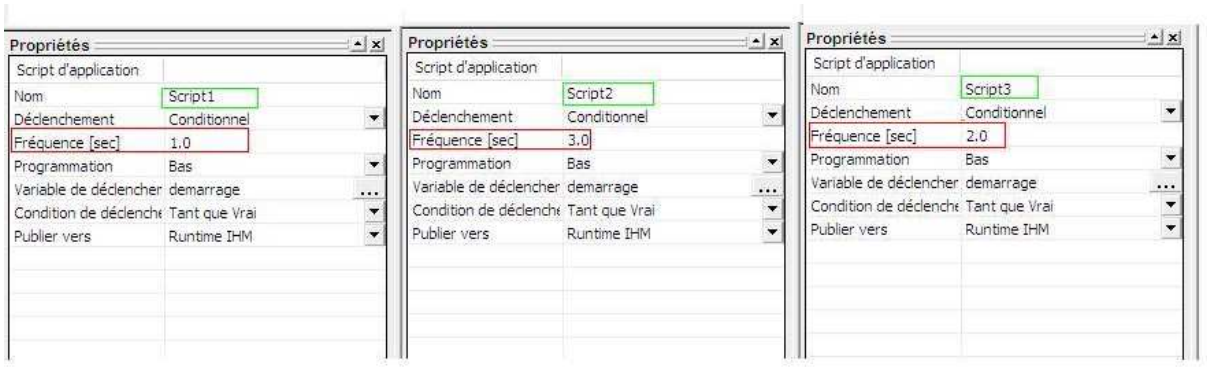


Figure III.24 : Changement de la fréquence d'exécution d'un script.

6.2.4. Exécution de la simulation

Pour démarrer la simulation, il faut valider le projet, le générer puis choisir « simulation du périphérique » dans le menu contextuel.

Le programme sera alors compilé et des erreurs seront affichées si on en commet. S'il n'y en a pas, les fenêtres suivantes seront affichées :

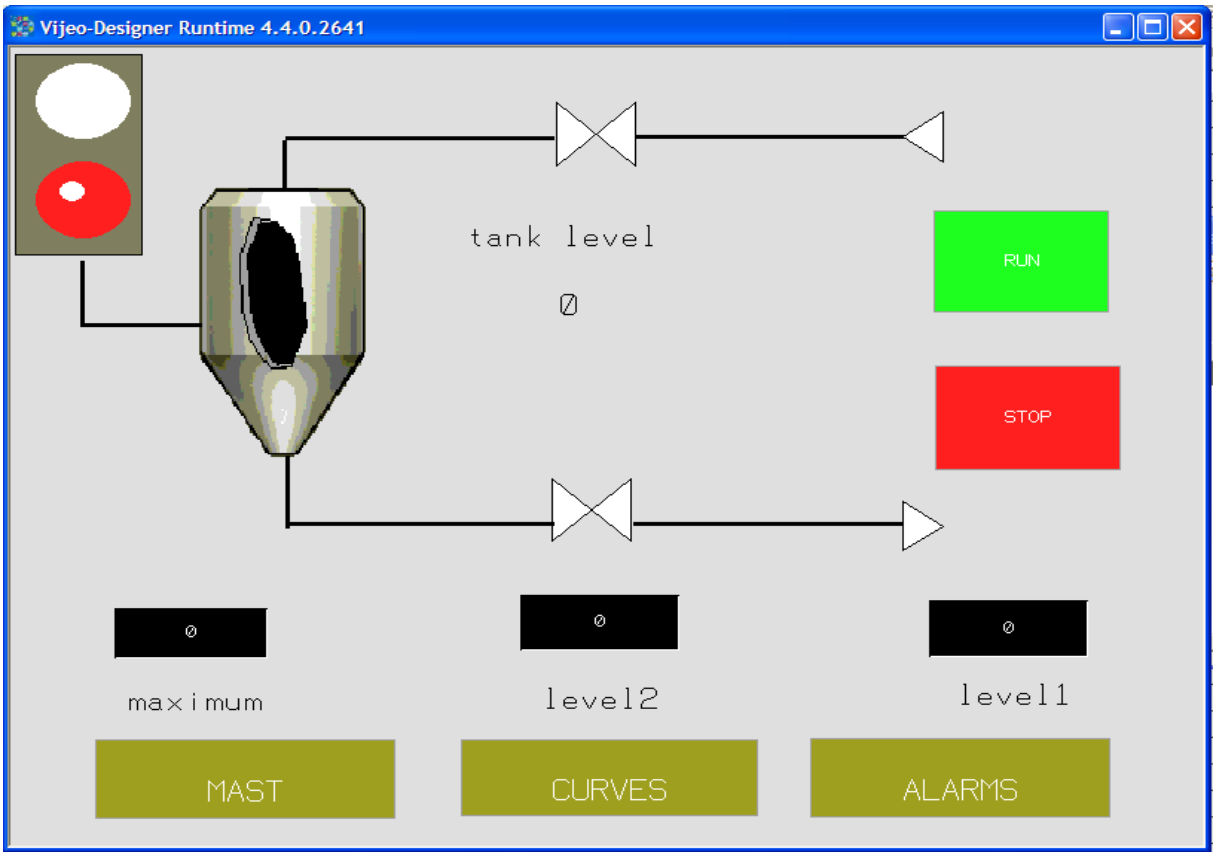


Figure III.25 : Ecran de simulation de l'exemple.

Pour saisir les valeurs dans les affichages numériques, il apparaît un clavier

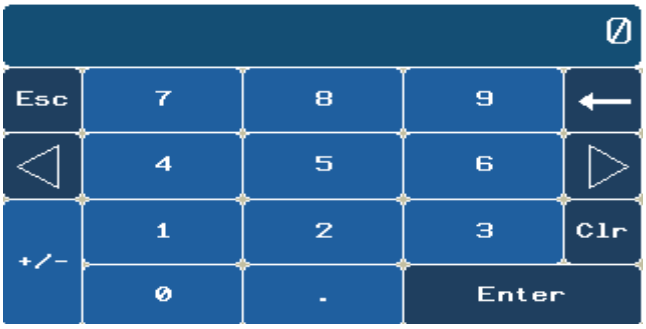


Figure III.26 : Clavier pour les affichages numériques.

Les valeurs affichées sur les écrans correspondent à des pourcentages de la valeur maximale du niveau de liquide que peut contenir ce réservoir.

Pour démarrer le processus, il faut appuyer sur le bouton RUN, on remarquera alors 3 étapes :

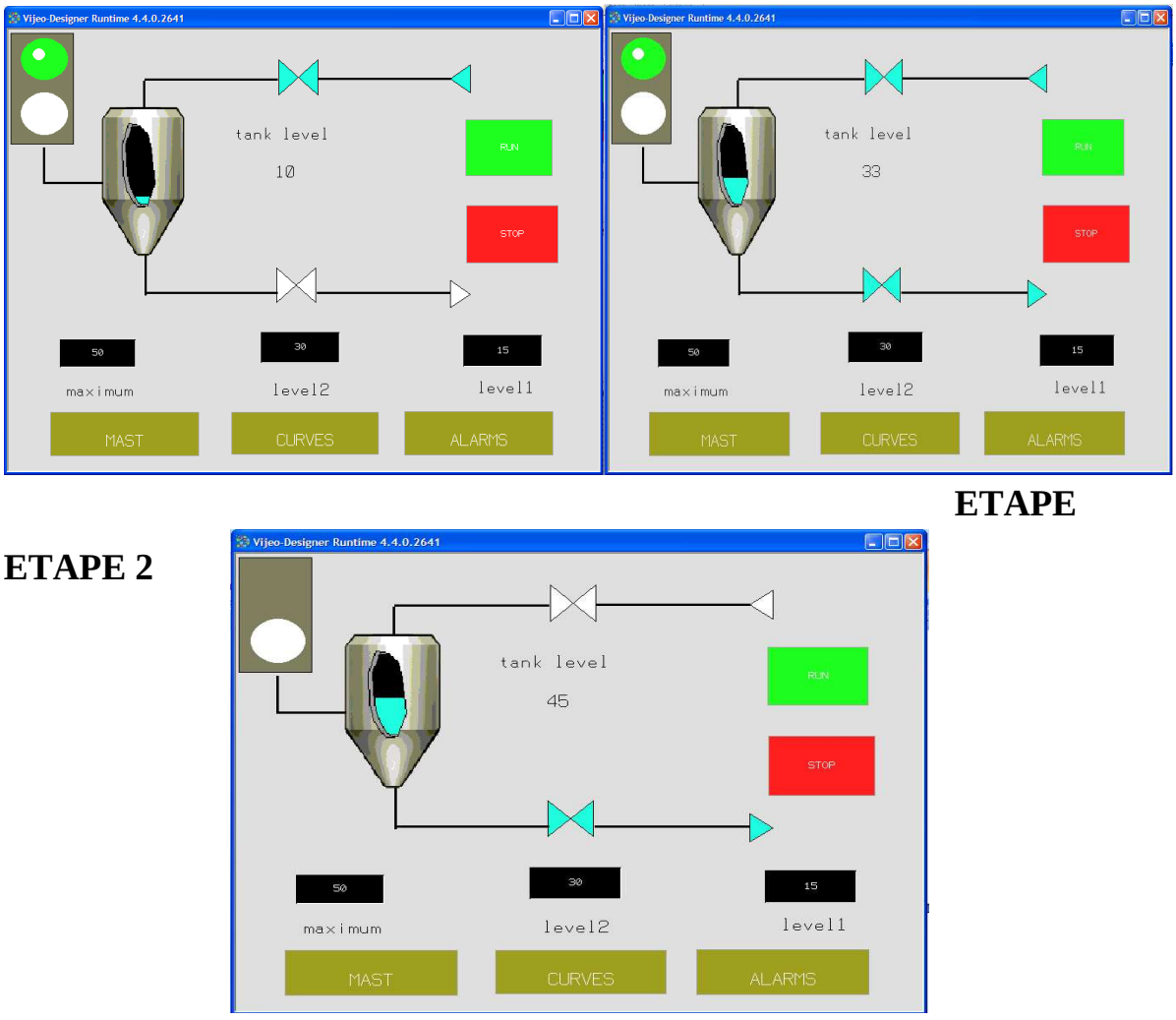


Figure III.27 : Les étapes du processus.

- **Etape1 :** Remplissage du réservoir entre les niveaux 0 et (N1=15%) avec un rapide débit correspondant à la vanne1.

- **Etape2 :** Remplissage du réservoir entre les niveaux (N1=15%) et (Nmax=50%) avec le plus faible débit car les 2 vannes sont ouvertes.
- **Etape3 :** Vidange du réservoir entre les niveaux (Nmax=50%) et (N2=30%) avec un débit inferieur à celui du remplissage.

Pour arrêter le système, il faut appuyer sur le bouton STOP

6.2.4.1. Signalisation des voyants

Voyant haut : Permet de savoir si le réservoir se remplit ou bien se vide.

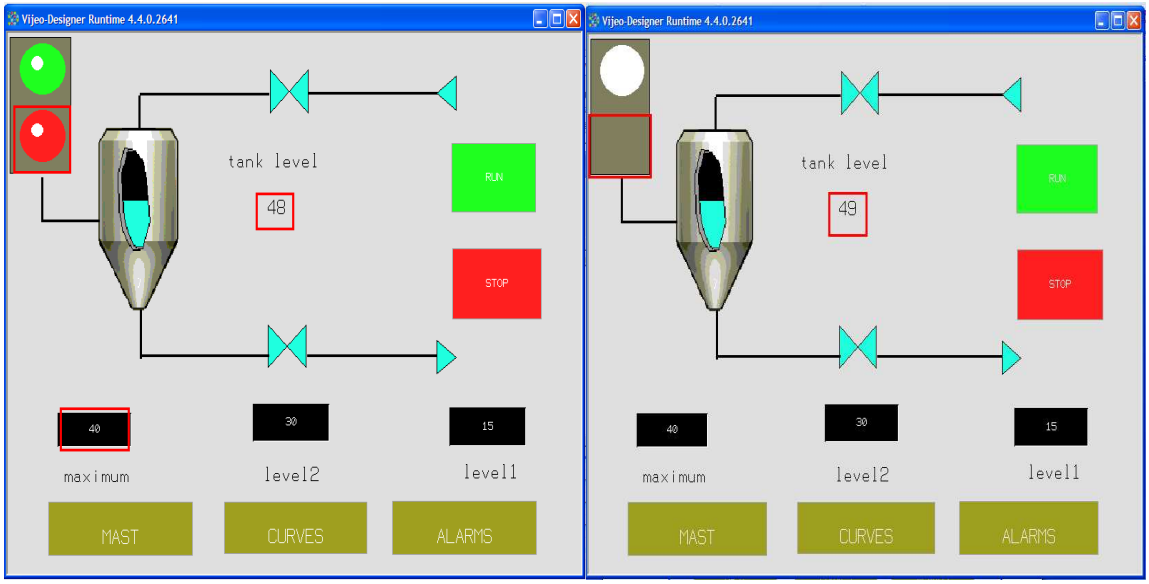
→Le voyant **« vert constant »** désigne que le réservoir se remplit (Etapes 1 et 2)

→Le voyant **« vert clignotant »** désigne que le réservoir se vide (Etape 3)

Voyant bas : Permet de savoir si le système est en arrêt et fait office d'alarme.

→ Le voyant **« rouge clignotant »** veut dire que le système est en arrêt, On le voit dès qu'on appui sur le bouton « STOP »

→ Le voyant **« rouge constant »** veut dire qu'il y a eu dépassement du niveau max du réservoir (50% dans notre exemple). Dans notre exemple, on peut la simuler en changeant la valeur de (Level-max) en une valeur inférieure à celle affichée par l'indicateur du réservoir.



DEPASSEMENT **ARRET**

Figure III.28 : Signalisation du voyant rouge.

Comme nous le montre la figure ci-dessus, en changeant la valeur du niveau max à (Level max=40% par exemple) alors que le niveau du réservoir a atteint une valeur supérieure (48%) le voyant bas vire au « rouge constant ».

Cependant, le réservoir continuera à se remplir. Il faudrait donc que l'on appui sur le bouton « STOP » dès qu'on remarque que le voyant est rouge. Le système sera alors à l'arrêt est le voyant bas deviendra « rouge clignotant » (Dans notre exemple on a arrêté le système une seconde après, Level=49%).

6.2.4.2. Affichages des courbes

En appuyant sur le bouton « CURVES » on voit la courbe défiler, on remarque les différentes étapes de notre processus dans la figure suivante.

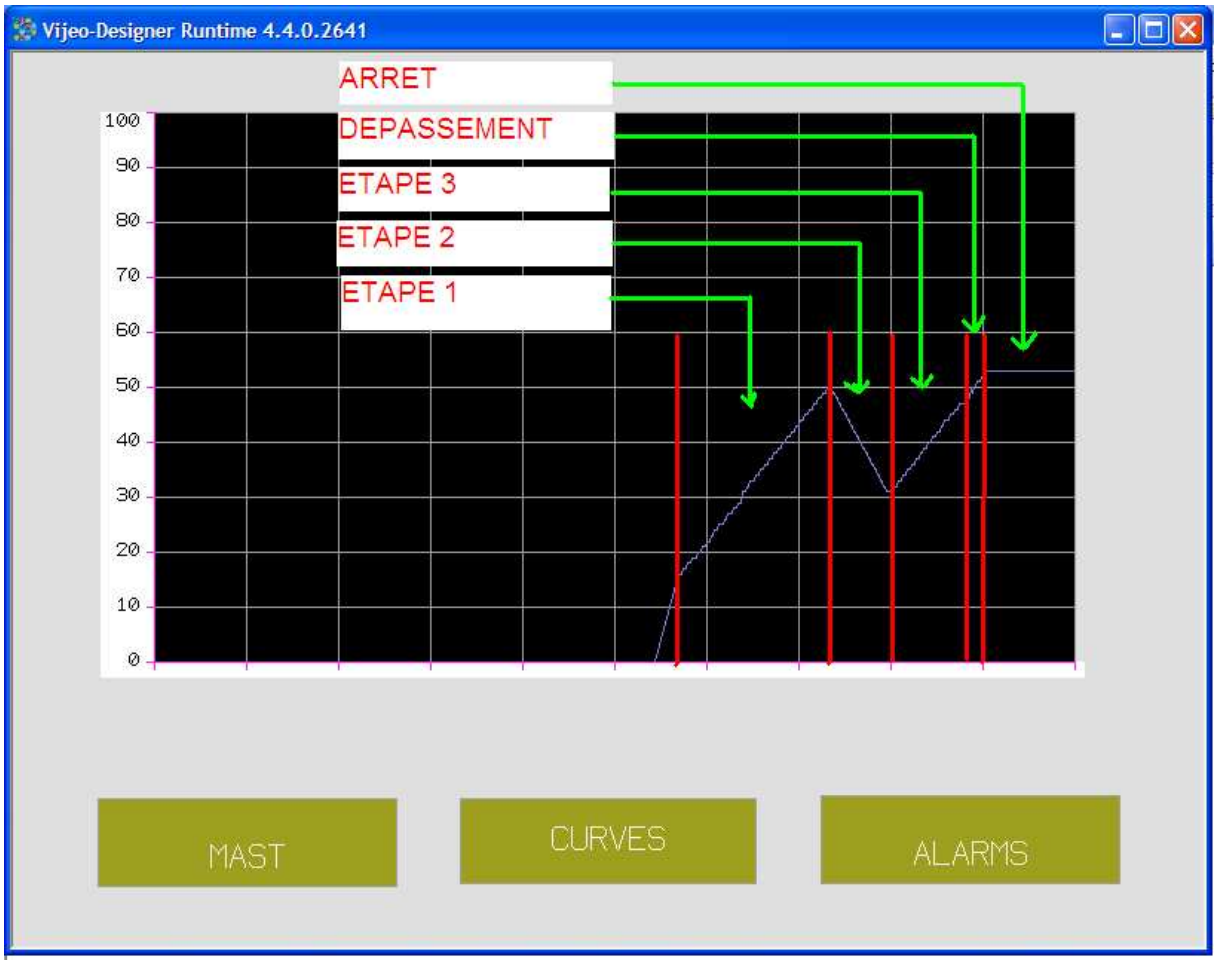


Figure III.29 : Courbe de tendance.

6.2.4.3. Ecrans d'alarmes :

Si le niveau « tank-level » dépasse le niveau « Nmax » le message d'erreur suivant apparait. Ce message étant configuré dans la variable à laquelle on a affectée l'alarme

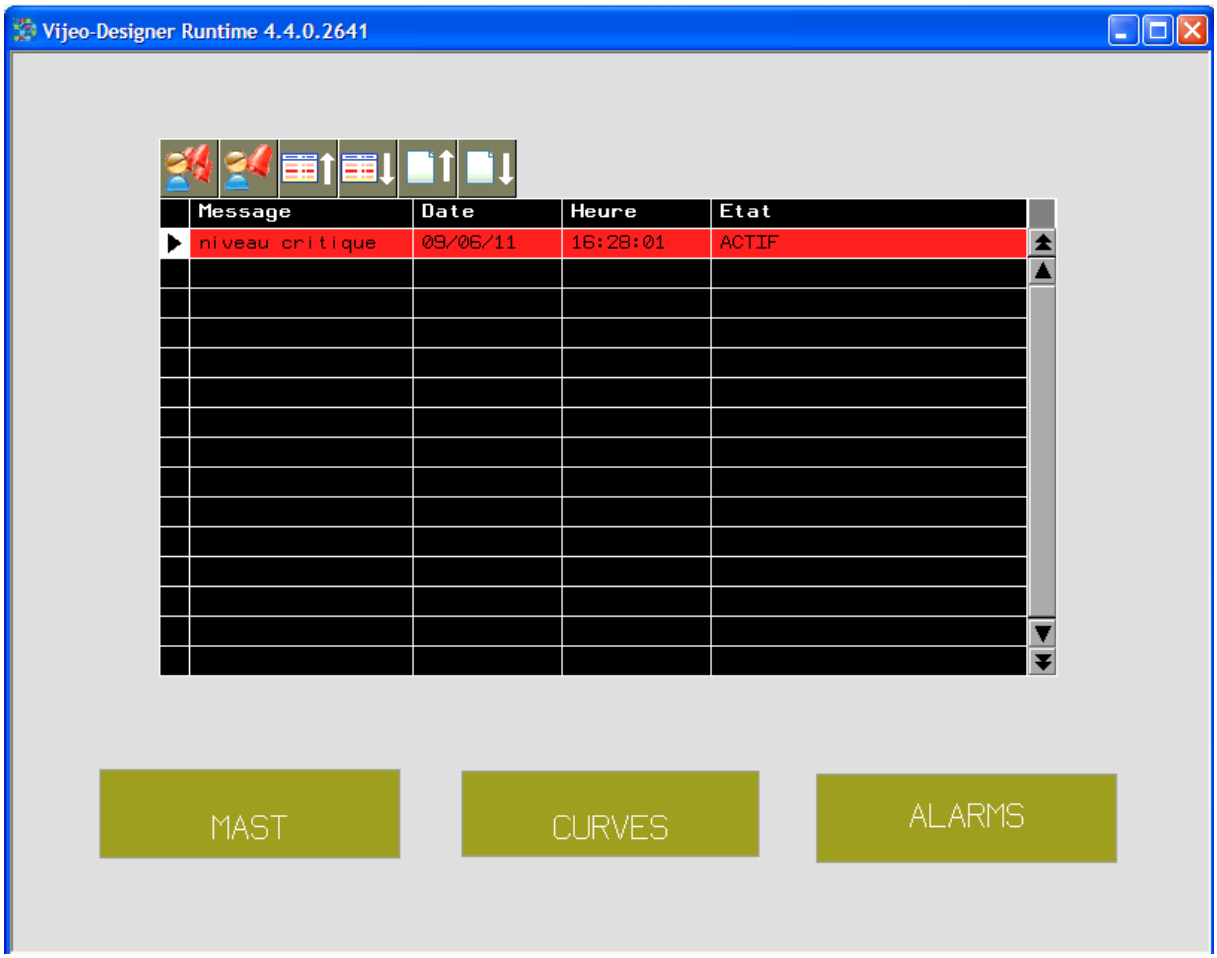


Figure III.30 : Signalisation d'alarme.

Remarque : Dans le cas d'une implémentation du programme sur un écran XBT, On aura juste à remplacer les variables internes par des variables externes. Ces variables sont celles qu'on utilise dans le programme implémenté dans l'automate programmable et qui sert à commander le remplissage de ce réservoir.

Dans ce même cas, on utilisera les scripts d'application juste pour la création d'éventuelles erreurs et non pour la simulation du fonctionnement du processus. Ce dernier étant défini dans le programme de l'automate.

7. Conclusion

Vigeo designer est un logiciel très souple, facile d'utilisation, il permet de créer des écrans de visualisation afin de superviser un processus donné. Nous avons essayé de montrer au mieux fonctionnalités offertes à travers l'exemple d'application, toutefois, nous n'avons utilisé que des variables internes. Nous proposerons, donc, de visualiser des variables externes provenant de l'automate au prochain chapitre.

CHAPITRE IV

APPLICATIONS

1. Simulation numérique

1.1. Introduction [11]

Il existe plusieurs méthodes de résolution exacte des équations différentielles. Ces méthodes s'adaptent par des équations différentielles particulières (Lagrange, Bernoulli…).

On rencontre fréquemment des équations qui ne sont pas de type continu. En général, la façon de procéder est d'utiliser une méthode numérique qui donne une valeur approchée de la solution.

Le problème classique qu'on rencontre est le problème de la valeur initiale.

$y'=f(y,t) \quad t \in [t_0,t_f] \quad y(t_0)$=condition initiale

En général, les méthodes numériques pour résoudre ces problèmes portent l'hypothèse que l'on peut découper l'intervalle en n parties égales.

$h = \frac{t_f - t_0}{n}$ h est appelé le *pas d'intégration.*

On calcule ensuite $y(t_0+h)$, $y(t_0+2h)$, $y(t_0+3h)$,…, $y(t_0+nh)$.

Toutes les méthodes numériques sont établies à partir du développement de Taylor d'une fonction y(t) au voisinage d'un point.

$$y(t_0+h)=y(t)+hy'(t)+\frac{h^2}{2!}y''(t)+\ldots+\frac{h^n}{n!}y^{(n)}(t)+\theta^{n+1}$$

Cette application a pour but de montrer comment utilisé Unity Pro pour faire de la simulation numérique.

Il existe différentes méthodes de calcul numériques, les plus utilisées sont les celles de Runge-Kutta. Elles reposent sur le principe d'itération. C'est-à-dire que la sortie à l'instant i dépend de celle de l'instant i-1.

Dans ce qui suit on utilisera la méthode de Runge-Kutta d'ordre quatre, la RK45.

1.2. Runge-Kutta d'ordre 4 [12]

Considérons le problème suivant :

Pour $t \in [t_0,t_f]$

$y'=f(t,y)$ avec $y(t_0)=y_0$

Pour un pas de discrétisation h, l'algorithme du calcul numérique est donné par les équations :

$$y_{n+1}=y_n+\frac{h}{6}(K_1+2K_2+2K_3+K_4) \qquad \text{IV.1}$$

Où :

$K_1=f(t_n,y_n)$

$K_2=f(t_n+\frac{h}{2},y_n+\frac{K_1}{2})$

$K_3=f(t_n+\frac{h}{2},y_n+\frac{K_2}{2})$

$K_4=f(t_n+h,y_n+K_3)$

Pour un système multivariable, par exemple 2 variables, le problème devient :

$$\begin{cases} y' = f_1(t, y, w) \\ w' = f_2(t, y, w) \end{cases} \qquad (IV.2)$$

Pour un pas de discrétisation h, l'algorithme du calcul numérique est donné par les équations :

$$\begin{cases} y_{n+1} = y_n + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \\ w_{n+1} = w_n + \frac{h}{6}(V_1 + 2V_2 + 2V_3 + V_4) \end{cases} \qquad (IV.3.)$$

Où

$K_1=f_1(t_n,y_n)$ $\qquad$ $V_1=f_2(t_n,y_n)$

$K_2=f_1(t_n+\frac{h}{2},y_n+h\frac{K_1}{2},w_n+h\frac{V_1}{2})$ $\qquad$ $V_2=f_2(t_n+\frac{h}{2},y_n+h\frac{K_1}{2},w_n+h\frac{V_1}{2})$

$K_3=f_1(t_n+\frac{h}{2},y_n+h\frac{K_2}{2},w_n+h\frac{V_2}{2})$ $\qquad$ $V_3=f_2(t_n+\frac{h}{2},y_n+h\frac{K_2}{2},w_n+h\frac{V_2}{2})$

$K_4=f_1(t_n+h,y_n+hK_3,w_n+hV_3)$ $\qquad$ $V_4=f_2(t_n+h,y_n+hK_3,w_n+hV_3)$

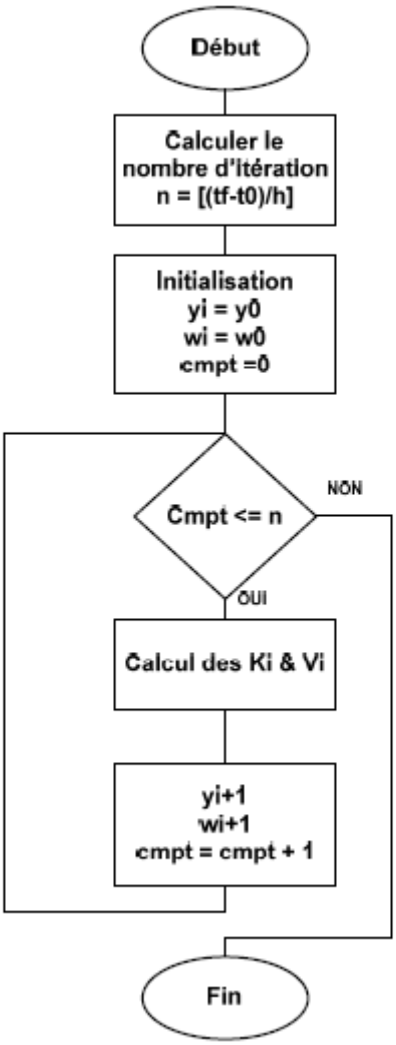


Figure IV.1 : Organigramme du calcul numérique.

1.2.1. Simulation d'un système du premier, second et troisième ordre

Prenons par exemple les équations différentielles suivantes pour les trois ordres :

$$\dot{y} + a_1 y + a_2 = a_3 u(t) \qquad \text{(IV.4)}$$

$$\ddot{y} + a_1 \dot{y} + a_2 y + a_3 = a_4 u(t) \qquad \text{(IV.5)}$$

$$\dddot{y} + a_1 \ddot{y} + a_2 \dot{y} + a_3 y + a_4 = a_5 u(t) \qquad \text{(IV.6)}$$

Pour simuler ces systèmes, nous avons programmé une DFB (**ANNEXE E.1.1**). Dont les entrées sont :

-n : l'ordre choisi (1,2 ou3).

-Les paramètres de a_1 jusqu'à a_5. Si l'on choisit un ordre inférieur à 3, on doit annuler les paramètres dont on n'a pas besoin.

-Les valeurs initiales y(0), y'(0) et y''(0). On annule les valeurs dont on n'a pas besoin.

-Le pas d'intégration h

-Et la commande u

L'unique sortie est le signal y.

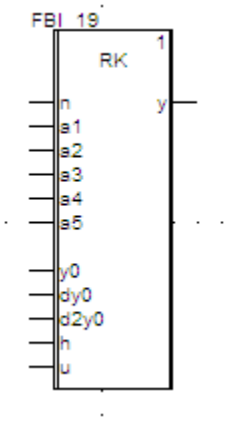


Figure IV.2 : Bloc DFB pour la simulation de systèmes linéaires

1.2.2. Simulation d'un système non linéaire

On prendra pour exemple le système régi par l'équation différentielle non linéaire suivante :

$\ddot{x} + 3t\dot{x} + 3t^2 x = u^2$ On suppose les conditions initiales nulles.

La DFB utilisée pour la simulation aura pour entrée u(t) et pour sortie x(t). h=0.02

Remarque : Les programmes se trouvent en annexe.(**ANNEXE E.1.2**)

1.3. Conclusion

Dans cette partie, nous avons réalisé des blocs DFB qui servent à simuler des systèmes linéaires d'ordre 1, 2 ou 3 et un système non linéaire. Nous avons utilisé pour cela, la méthode de Runge-Kutta d'ordre4. Les résultats obtenus lors des essais sont pratiquement identiques au cas réel. Donc on pourra utiliser ces blocs pour d'autres applications.

2. Régulation PID :

2.1. Introduction :

Le régulateur standard le plus utilisé dans l'industrie est le régulateur PID (Proportionnel Intégral Dérivé), car il permet de régler à l'aide de ses trois paramètres les performances (amortissement, temps de réponse) d'une régulation d'un processus modélisé par un deuxième ordre.

Nombreux sont les systèmes physiques qui, même en étant complexes, ont un comportement voisin de celui d'un deuxième ordre, dans une certaine échelle de temps. Par conséquent, le régulateur PID est bien adapté à la plupart des processus de type industriel et est relativement robuste par rapport aux variations des paramètres du procédé, quand on n'est pas trop exigeant pour les performances de la boucle fermée par rapport à celles de la boucle ouverte (par exemple, accélération très importante de la réponse ou augmentation très importante de l'amortissement en boucle fermée). **[13]**

2.2. Création de la DFB « PID »

Nous allons créer un bloc DFB d'un régulateur PID, le régulateur ayant la forme de la figure suivante :

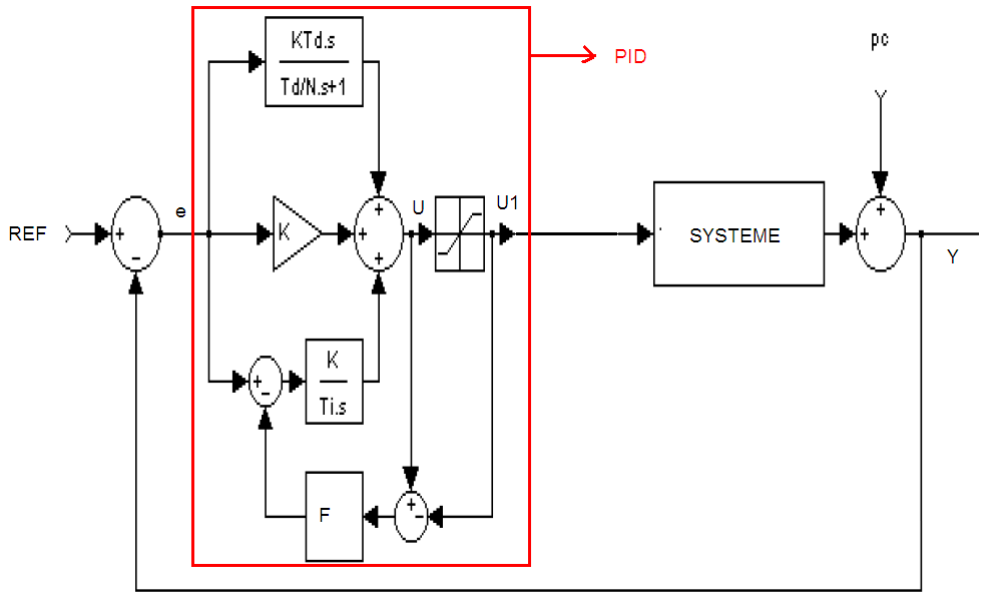


Figure IV.3 : Schémas bloc avec un PID contenant un anti wind-up **[14]**

Notre régulateur, en plus de contenir les trois actions (proportionnelle, intégrale et dérivée), contient un « Anti-windup ». Ceci s'explique par le fait que la commande résultant du régulateur est limitée.

L'anti-windup permet alors au régulateur de revenir plus rapidement à la zone linéaire quand il atteint la saturation et ce en retranchant un terme dans la partie intégrale

Ce terme est égal à $F * (U1 - U)$

$U1$: Commande après passage par l'élément de saturation

U : Commande avant passage par l'élément de saturation

Pour la fonction F nous avons choisi de prendre la forme proposée par « *STROM* » et « *RUNDQUIST* ». Ainsi F sera égal à : **[15]**

$$F = \frac{T_i}{K_p\sqrt{T_i T_d}}$$

Le programme se trouve en annexe **[ANNEXE E.2]**

2.3. Exemple de commande d'un four électrique

2.3.1. Présentation du four [7]

1 - Afficheur digital + clavier

2 - Entrée du signal de commande

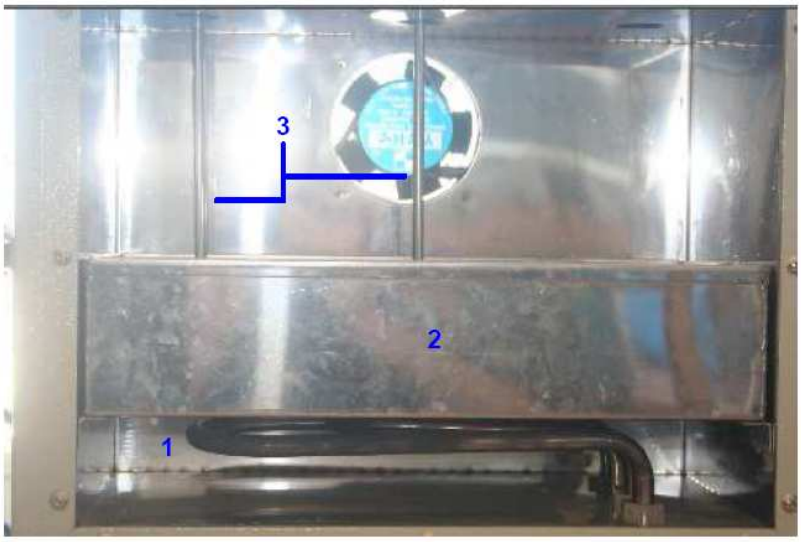
3 - Manuel / Auto

4 - Temperature/ Humidité

5 - On / Off

Figure IV.4. Description du four

Ce four fonctionne avec une alimentation alternative de 220V, il a deux modes de fonctionnement le mode manuel et le mode automatique.



1 - Résistance

2 - Banc

3 - Capteurs de Température

Figure IV.5. Contenu du four.

Le four est constitué d'une résistance dont la tension d'alimentation est contrôlée par un triac, d'un banc où on peut mettre le liquide et de deux capteurs de température. L'un d'eux est relié à l'afficheur digital, l'autre au circuit de transduction.

Le capteur de température utilisé est l'AD590. C'est un capteur à réponse rapide, il produit un courant proportionnel à la température absolue $1\mu A/°K$.

Voici quelques de ces caractéristiques :

- Courant linéaire de sortie:
- Etendue de mesure : de −55°C à +150°C.
- Linéarité excellent: ±0.3°C au dessus de toutes les amplitudes (Aucun circuit de linéarisation n'est nécessaire).
- Etendue de tension d'alimentation: de 4 V à 30 V.

2.3.2. Identification du système [7]

Le modèle mathématique du four étant inconnu, On a utilisé le résultat de l'identification de la réponse indicielle en boucle ouverte pour différents points de fonctionnement, car le four est un système fortement non linéaire (les modèles ne seront valables qu'autour des points de fonctionnement).

Le modèle choisi est celui de Broîda, qui est de la forme : $F(s) = \frac{k}{1+T*s} e^{-\tau s}$

On relève donc la réponse du système et, à partir du graphe obtenu, on identifie les paramètres du modèle via les équations suivantes :

$$\begin{cases} K = \dfrac{\Delta Y_\infty}{\Delta E_\infty} \\ T = 5.5(t_2 - t_1) \\ \tau = 2.8t_2 - 1.8t_1 \end{cases}$$

Où :

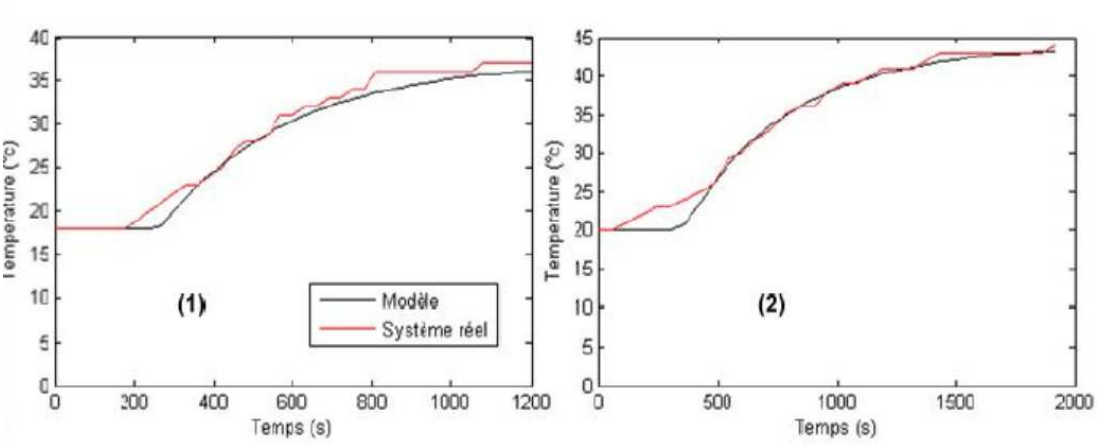
t_1 : Correspond au temps où la sortie $Y = 0.28\, Y_\infty$

t_2 : Correspond au temps où la sortie $Y = 0.4\, Y_\infty$

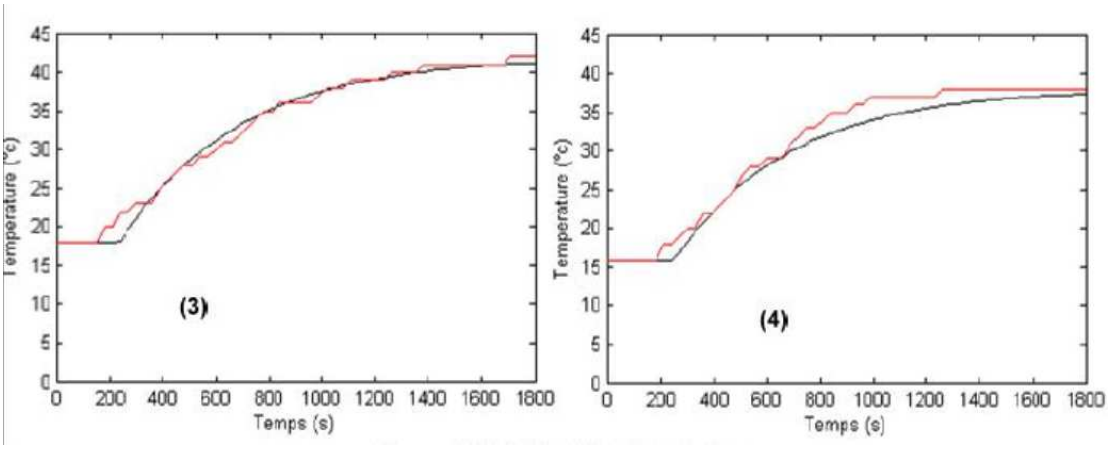
On a choisi quatre points de fonctionnement et les résultats sont montrés dans le tableau suivant :

Point de fonctionnement	Température initiale	Modèle identifié
30	18	$\frac{6.4}{1+324*s}e^{-268s}$
35	20	$\frac{6.8571}{1+451*s}e^{-846s}$
45	18	$\frac{5.33}{1+452*s}e^{-238s}$
55	16	$\frac{4}{1+436*s}e^{-262s}$

Tableau IV.1. Modèle de Broida pour différents points de fonctionnements



1er Point : 30° 2ème point : 35°



3ème Point : 40° 4ème point : 45°

Figure IV.6. Identification du système

On remarque que ces modèles décrivent bien le système réel autour des points de fonctionnement, les réponses montrent que le système est très lent. Nous allons donc introduire notre régulateur PID afin de diminuer le temps de réponse.

2.3.3. Détermination des paramètres PID

Pour cela et, aussi, pour pouvoir simuler la fonction de transfert nous avons utilisé la relation : $e^{-\tau s} = 1 + \tau * s$

Les fonctions de transferts s'écrivent alors sous la forme :

$$F(s) = \frac{k}{(1 + T_1 * s) * (1 + T_2 * s)}$$

Le régulateur PID s'écrit :

$$R(s) = K_p(1 + T_d * s + \frac{1}{T_i * s}) \cong K_p \frac{(1 + T_i * s) * (1 + T_d * s)}{T_i * s}$$

La fonction de transfert en boucle ouverte s'écrit alors :

$$F_{BO}(s) = K_p \frac{(1 + T_i * s) * (1 + T_d * s)}{T_i * s} * \frac{k}{(1 + T_1 * s) * (1 + T_2 * s)}$$

Il suffit donc de prendre

$T_i = T_2$ et $T_d = T_1$

La fonction de transfert en boucle fermée devient alors :

$$F_{BF}(s) = \frac{1}{\left(1 + \frac{T_2}{k * K_p} * s\right)}$$

Le système résultant se comporte comme un premier ordre dont la constante de temps apparente est :

$$T_a = \frac{T_2}{k * K_p}$$

Le temps de réponse étant :

$$T_r = 3 * T_a = \frac{3 * T_2}{k * K_p}$$

On règle donc Kp pour obtenir le temps de réponse désiré. **[16]**

Les paramètres des différentes fonctions de transfert sont calculés selon la méthode énoncée ci-dessus, et montrés dans le tableau suivant :

Modèle identifié	Ti	Td	Kp
$\frac{6.4}{1 + 324 * s} e^{-268s}$	324	268	1
$\frac{6.8571}{1 + 451 * s} e^{-846s}$	846	451	1
$\frac{5.33}{1 + 452 * s} e^{-238s}$	452	238	1
$\frac{4}{1 + 436 * s} e^{-262s}$	436	262	1

Tableau IV.2 Paramètres du régulateur PID pour les différents points de fonctionnement

2.3.4. Programmation sous UNITY PRO :

On utilise les blocs RK2 et PID crées précédemment en introduisant les valeurs des paramètres PID du tableau ci-dessus. Pour les paramètres du RK2 il faut écrire le système sous la forme suivante : $\ddot{y} + a * \dot{y} + b * y + c = d * u$

Dans notre cas cette équation s'écrit :

$$\ddot{y} + \frac{T_1 + T_2}{T_1 * T_2} * \dot{y} + \frac{1}{T_1 * T_2} * y = \frac{k}{T_1 * T_2} * u$$

On en tire donc les valeurs des paramètres du système :

$$a = \frac{T_1 + T_2}{T_1 * T_2} \quad ; \quad b = \frac{1}{T_1 * T_2} \quad ; \quad c = 0 \quad ; \quad d = \frac{k}{T_1 * T_2}$$

Les valeurs des paramètres pour les différents points de fonctionnement sont données dans le tableau suivant :

Modèle identifié	A	B	D
$\frac{6.4}{1 + 324 * s} e^{-268s}$	0.006818	0.000001152	0.00007306
$\frac{6.8571}{1 + 451 * s} e^{-846s}$	0.003399	0.000002621	0.00001737
$\frac{5.33}{1 + 452 * s} e^{-238s}$	0.006414	0.000009296	0.000049546
$\frac{4}{1 + 436 * s} e^{-262s}$	0.00611	0.000008754	0.000035016

Tableau IV.3 : Paramètres du bloc RK2 pour la régulation du four

Pour la programmation, on a utilisé le langage DFB et nous avons pris des plages correspondant aux différents points de fonctionnement. **[E3]**

Normalement, le signal de commande résultant du boc PID doit être une entrée du système réel, mais comme nous n'avons pas pu manipuler le four réellement nous l'avons injecté dans la fonction de transfert correspondant à la plage qui contient la référence.

2.3.5. Simulation avec Vijeo Designer:

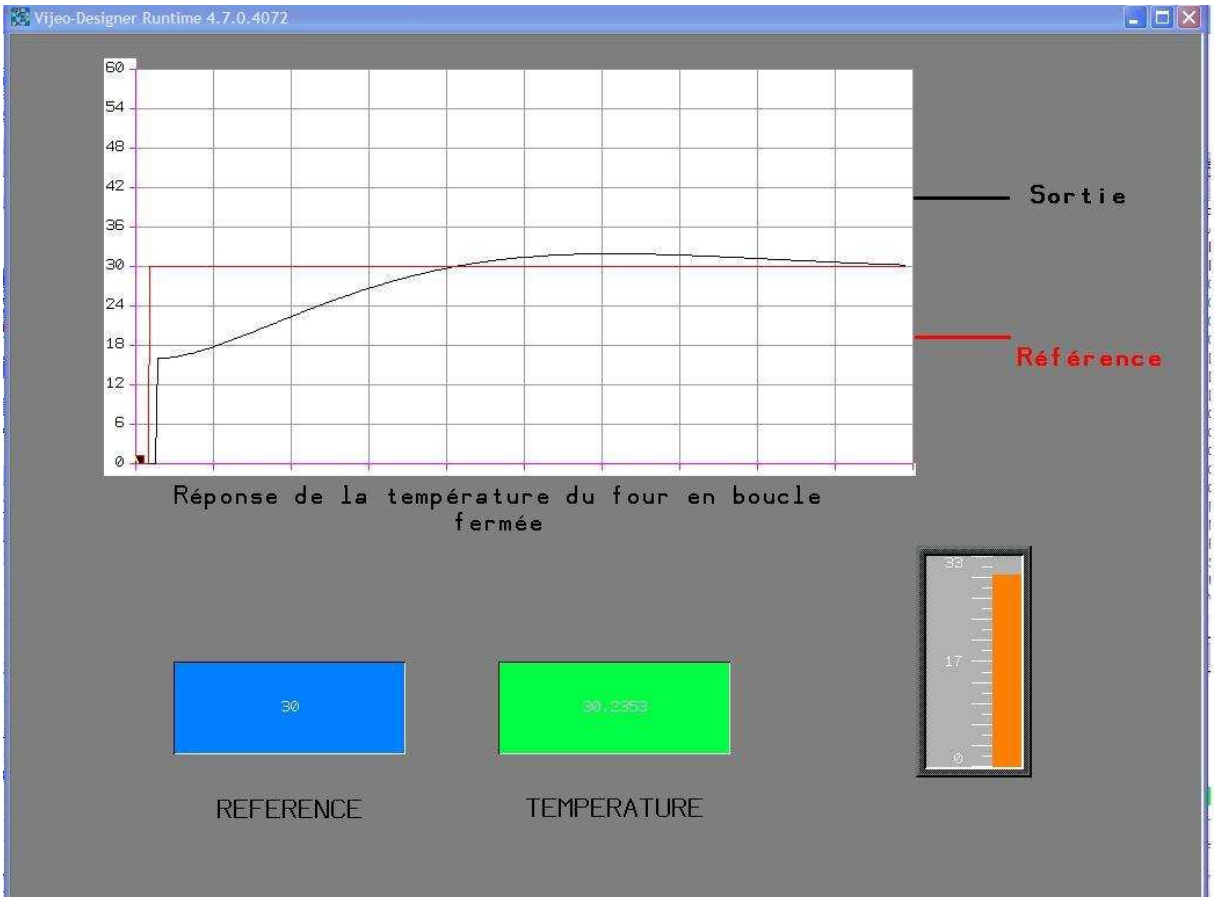


Figure IV.7 : Simulation de la réponse du four à une référence de 30°

Nous voyons bien d'après la figure précédente que la sortie suis bien la référence, on a mis un curseur qui se met au rouge quand il y a dépassement de 10% de la référence. Ce qui n'est pas arrivé. De plus la réponse est plus rapide qu'en boucle ouverte.

2.4. Conclusion :

Le bloc de simulation numérique nous a permis de simuler le four. Cela nous a, aussi, permis de tester l'efficacité du régulateur PID avec anti-windup qu'on a crée. En effet, on constate que la régulation est parfaitement établie et que la commande U ne se sature pas (elle reste dans la plage linéaire)

3. Identification de systèmes dynamiques

3.1. Introduction

Dans le domaine de l'automatique, il est important de disposer d'un modèle adéquat de l'installation à commander dans le but de mettre au point un régulateur performant. Un tel modèle peut s'obtenir par modélisation physique; on parle alors de modèle de connaissance. Cependant, il est souvent difficile, voire impossible, de développer un tel modèle physique dans la pratique industrielle. Pour pallier à cette difficulté, l'automaticien a appris à développer des modèles de représentation qui décrivent le comportement entrée-sortie du processus. On appelle ainsi identification de systèmes dynamiques l'étape de modélisation correspondante.

Il existe plusieurs méthodes d'identification. Elles sont divisées en deux grandes catégories : les méthodes de base (dites approches classiques), ce sont des méthodes graphiques et les méthodes numériques (approches modernes) qui peuvent être récursives ou non.

Dans ce qui suit, nous avons utilisé pour l'identification, la méthode des « Moindres Carrés récursifs » (MCR).

3.2. Méthode des moindres carrés récursifs [11]

C'est un algorithme d'adaptation paramétrique récursif permettant de minimiser le critère quadratique suivant :

$$J = \frac{1}{2}\sum_{i=1}^{t} \varepsilon^2(i)$$

Voici le résumé de cet algorithme :

$$\begin{cases} \hat{\theta}(t+1) = \hat{\theta}(t) + F(t+1)\Phi(t)\,\varepsilon^0(t+1) \\ F^{-1}(t+1) = F^{-1}(t) + \Phi(t)\Phi^T(t) \\ \varepsilon^0(t+1) = y(t+1) - \hat{\theta}^T(t)\Phi(t) \end{cases}$$

Où

- $\hat{\theta}$(t+1) est le vecteur des paramètres estimés à l'instant présent.
- $\hat{\theta}$(t) est le vecteur des paramètres estimés à l'instant précédent.
- F est le gain d'adaptation décroissant. Pour la programmation, nous avons utilisé le lemme d'inversion de matrice qui donnera cette expression :

$$F(t+1) = F(t) - \frac{F(t)\Phi(t)\Phi^T(t)F(t)}{1+\Phi^T(t)F(t)\Phi(t)}$$

- Φ (t) est le vecteur de mesure défini par :

$$\Phi^T(t) = [-y(t) \quad -y(t-1) \quad ... \quad -y(t-n+1) \quad u(t) \quad u(t-1) \quad ... \quad u(t-m+1)]$$

- ε°(t+1) est l'erreur de prédiction a priori

On prendra pour l'initialisation de F la matrice 1000*I. Le programme des MCR se trouve en annexe. **[ANNEXE E.4]**

Pour obtenir un modèle consistant, il est important d'exciter le processus avec toutes les fréquences de sa plage de fonctionnement. Le signal d'entrée appliqué doit donc être riche en fréquences (posséder un large spectre). En général on applique un signal séquences binaires pseudo aléatoires (SBPA). **[11]**

3.3. Séquences Binaires Pseudo Aléatoires (SBPA) [11]

La SBPA est un signal déterministe qui possède deux niveaux logiques « 0 » et « 1 ». D'une manière analogique, la SBPA peut être générée en utilisant un registre à décalage à N cellules rétro-alimentées à travers une fonction booléenne qui est la somme modulo 2.

Dans notre application, nous avons programmé, pour générer une SBPA, un registre à 5 cellules. Le programme se trouvant en annexe. **[ANNEXE E.5]**

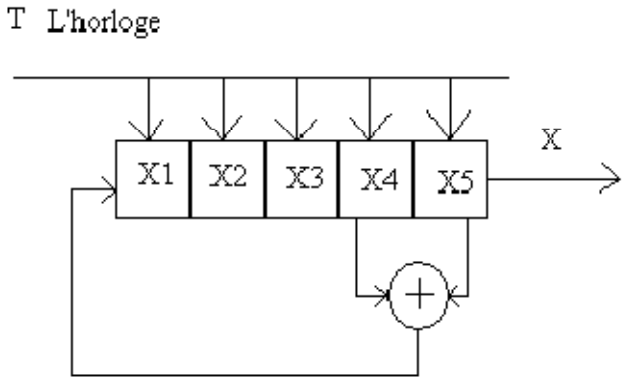


Figure IV.8 : Représentation d'un registre à décalage à 5 cellules

Le X représente le signal de la SBPA, sa génération se déroulera de la façon suivante :

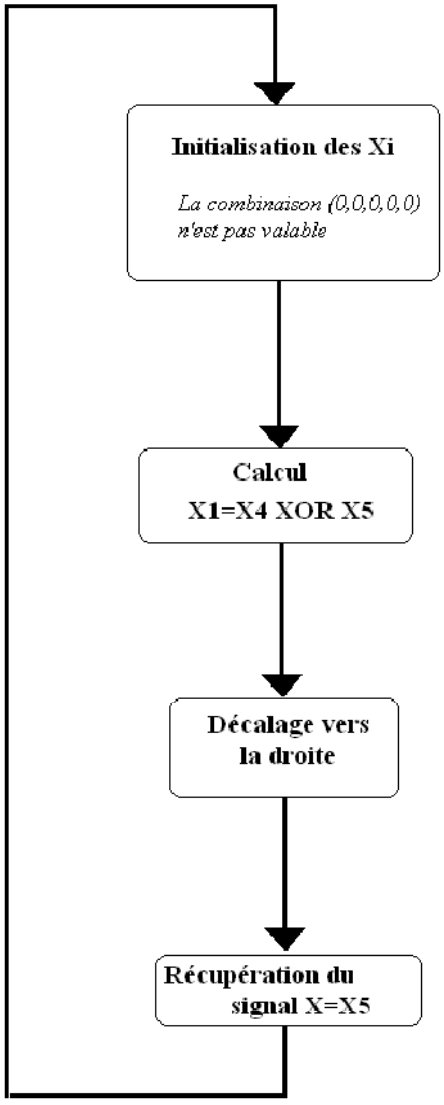


Figure IV.9 : Organnigramme de la génération de la SBPA

3.4. Exemple

Nous avons établi dans un projet Unity, une section MAST en FBD pour tester le bloc MCR programmé. Dans cette section on trouve :

-Le bloc des MCR qui a comme entrée le signal de sortie y et la commande u et comme sortie le vecteur de paramètre θ.

-Un bloc simulant une SBPA, dont la sortie sera l'entrée commune du bloc MCR, du système réel et du modèle présumé.

-Deux blocs pour simuler des systèmes de type ARX, l'un d'eux simulera le système réel (on fixe les paramètres) et l'autre simulera le modèle dont les paramètres sont les sortie du bloc MCR.

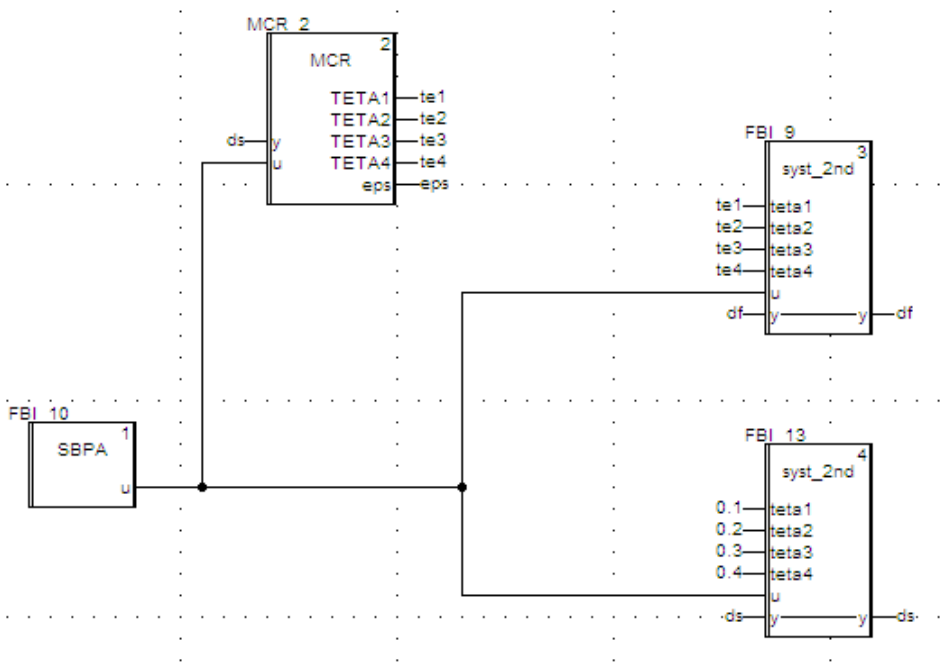


Figure IV.10 : Section FBD testant le bloc MCR.

Et d'après la figure 4.11, on voit bien que les deux sorties sont pratiquement identiques.

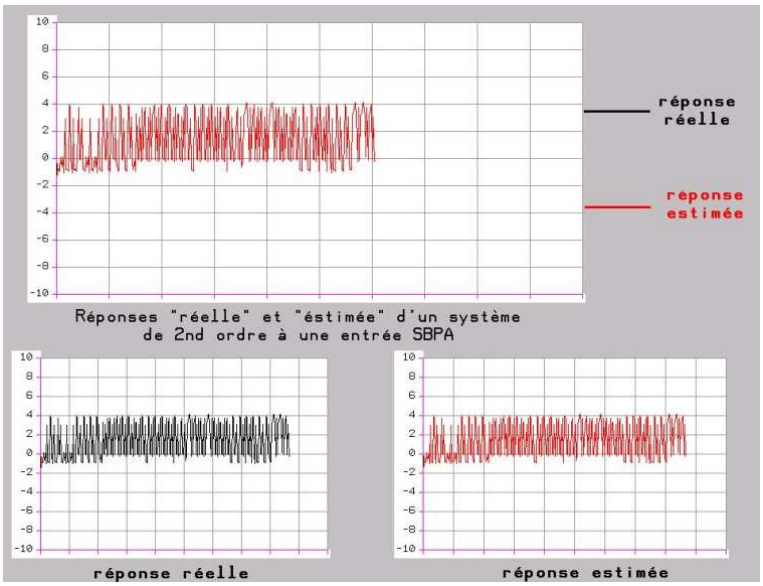


Figure 4.11 : Graphe Vijeo Designer illustrant la superposition de la sortie réelle et de la sortie du modèle.

3.5. Conclusion

L'identification est une étape importante à franchir avant l'automatisation de tout procédé industriel. Elle permet de caractériser de façon quantitative le fonctionnement des systèmes étudiés.

Nous avons présenté et développer dans ce chapitre la méthode d'identification des MCR, qui permet d'élaborer un modèle mathématique à partir de mesures que l'on peut acquérir sur le processus à commander. **[17]**

4. Autorégulation (P.I.D. autoajustable)

4.1. Introduction

Dans ce qui a été présenté, on a supposé que le modèle était connu, or ce n'est pas le cas en pratique, dans la mesure où les caractéristiques du système peuvent varier dans le temps. Donc, on doit procéder à l'estimation récursive de ces paramètres de façon à pallier ces fluctuations.

L'estimation des paramètres sera faite à partir de la mesure en continu de l'entrée et de la sortie du processus à réguler, ceci en utilisant par exemple la méthode des moindres carrés récursive.

4.2. Représentation discrète d'un régulateur [17]

La transformée de Laplace d'un régulateur P.I.D. s'écrit :

$$F(p) = k(1 + \frac{1}{T_i p} + T_d p)$$

On peut obtenir une représentation discrète de plusieurs façons.

L'une d'elles consisterait par exemple a approcher l'action intégrale par la méthode des trapèzes.

Dans ce qui suit, nous allons déterminer la transmittance discrète du P.I.D. numérique en calculant séparément les transmitances échantillonnées de la partie proportionnelle et intégrale de la partie dérivée.

La transformée en z^{-1} de la partie P.I du régulateur, pour une période d'échantillonnage T_0 s'écrira :

$$H_1(z^{-1}) = k[1 + \frac{T_0}{T_i}\frac{z^{-1}}{1-z^{-1}}]$$

$$H_1(z^{-1}) = \frac{\alpha_0 + \alpha_1 z^{-1}}{1-z^{-1}}$$

En posant : $A_1(z^{-1}) = \alpha_0 + \alpha_1 z^{-1}$

Et $I(z^{-1}) = 1 - z^{-1}$

La transmittance puisée de la partie P.I. du régulateur s'écrira :

$$H_1(z^{-1}) = \frac{A_1(z^{-1})}{I(z^{-1})}$$

En supposant la régularité des signaux mis en jeu, ce qui reviendrait à supposer que leurs dérivées restent constantes entre deux instants d'échantillonnage consécutifs, il vient, pour la partie dérivée :

$$H_2(z^{-1}) = \frac{k\,T_d(1 - e^{-T_0/\alpha T_d})(1 - z^{-1})}{1 - e^{-T_0/\alpha T_d}\ z^{-1}}$$

$$H_2(z^{-1}) = \frac{\beta(1 - z^{-1})}{1 + \gamma\, z^{-1}}$$

En posant : $B_1(z^{-1}) = \beta\, I(z^{-1})$

Et $C_1(z^{-1}) = 1 + \gamma\, z^{-1}$.

La transmittance échantillonnée, du régulateur P.I.D s'écrira alors :

$$H(z^{-1}) = \frac{A_1(z^{-1})}{I(z^{-1})} + \frac{B_1(z^{-1})}{C_1(z^{-1})}$$

Les relations existantes entre les paramètres du P.I.D. continu et ceux du P.I.D discret sont mentionnées dans le tableau 1.

Paramètres du P.I.D. discret	Paramètres du P.I.D. continu
$\alpha_0 = k$ $\alpha_1 = k(T_0/T_i - 1)$ $\beta = k\,T_d(1 - e^{-\frac{T_0}{\alpha T_d}})/T_0$ $\gamma = -e^{-\frac{T_0}{\alpha T_d}}$	$k = \alpha_0$ $T_i = T_0\,\alpha_0/(\alpha_0 + \alpha_1)$ $T_d = \beta T_0/(k(1+\gamma))$

Tableau IV.4 : Relation entre les paramètres du P.I.D. continu et discret

La correspondance en sens inverse se fera si le coefficient γ est compris entre -1 et zéro.

En posant :

$$S = A_1 C_1 + B_1 I$$

Et

$$R = I \times C_1$$

L'équation caractérisant la dynamique du régulateur s'écrira :

$$R(z^{-1})u(t) = S(z^{-1})[y_r(t) - y(t)]$$

4.3. Réglage du P.I.D. discret [17]

Considérons le processus représenté par l'équation suivante :

$$y(t) = \frac{b_1 z^{-1} + b_2 z^{-2}}{1 + a_1\, z^{-1} + a_2\, z^{-2}}\; u(t)$$

Déterminons les paramètres du régulateur PID de façon à ce que l'équation caractéristique du système en boucle fermée, corresponde à :

$$D(\, z^{-1}) = 1 + d_1\, z^{-1} + d_2\, z^{-2} + d_3\, z^{-3} + d_4\, z^{-4}$$

En identifiant les pôles du système en boucle fermée aux zéros de l'équation caractéristique désirée, il vient :

$$A(\, z^{-1})R\,(z^{-1}) + \, z^{-1}\, B(\, z^{-1})\, S(\, z^{-1}) = D(\, z^{-1})$$

On posera

$$S = s_0 + s_1 z^{-1} + s_2 z^{-2}$$

La résolution du système d'équation ci-dessous obtenu par identification des coefficients des puissances de z^{-1} donnera les valeurs des paramètres S_0, S_1, S_2 et γ

$$\begin{cases} d_1 = b_1 s_0 + \gamma + a_1 - 1 \\ d_2 = b_2 s_0 + b_1 s_1 + (a_1 - 1)\gamma + a_2 - a_1 \\ d_3 = b_2 s_1 + b_1\, s_2 + (a_2 - a_1)\gamma - a_2 \\ d_4 = b_2 s_2 - a_2 \gamma \end{cases}$$

$$s_0 = \frac{1}{det}\big[b_1(a_1 - a_2)\big(d_1 b_1 + b_2(d_1 - a_1)\big) - {b_2}^2\big(1 + 2d_2 - a_2 + a_1(a_2 - 1)\big) - {b_1}^2(a_2 + d_4) + b_1 b_2(a_1 + d_3)\big]$$

$$s_1 = \frac{1}{det}\{{b_1}^2(a_2 - a_1 - d_2) + b_1 b_2[d_2(a_1 - a_2)^2 + d_3(a_1 - 1) + d_1 a_2] + d_4 {b_1}^2(1 - a_1) + {b_2}^2(d_1(a_2 - a_1) - a_1(a_2 - a_1 + 1) - d_3)\}$$

$$s_2 = \{-{b_1}^2[a_2(d_3 + a_2 + d_4) + d_4 a_1] + b_1 b_2[a_2(b_2 - a_2) - a_1 + d_4(a_1 - 1)] - {b_2}^2[a_2(d_1 - a_1 + 1) + d_4]\}$$

$$\gamma = \frac{1}{det}\{{b_1}^2(d_4 b_1 - b_2 d_3 - b_2 a_2) + b_1 {b_2}^2(d_2 - a_2 + a_1) - {b_2}^3((d_1 - a_1 + 1)\}$$

Les paramètres α_0, α_1 et β s'obtiendront à partir de l'expression du polynôme S(z^{-1}).

$$S = A_1 C_1 + B_1 I$$

Avec

$$A_1 = \alpha_0 + \alpha_1 z^{-1}$$

$$C_1 = 1 + \gamma z^{-1}$$

$$B_1 = \beta I$$

$$I = 1 - z^{-1}$$

$$S = (\alpha_0 + \beta) + (\gamma\alpha_0 + \alpha_1 - 2\beta)z^{-1} + (\gamma\alpha_1 + \beta)z^{-2}$$

$$\Delta = 1 + 2\gamma + \gamma^2$$

$$\alpha_0 = \frac{1}{\Delta}(s_0 + 2s_0\gamma + s_1\gamma - s_2)$$

$$\alpha_1 = \frac{1}{\Delta}(s_1 + 2s_2 - s_0\gamma + s_2\gamma)$$

$$\beta = \frac{1}{\Delta}(s_2 - \gamma s_1 + \gamma^2 s_0)$$

$$\begin{cases} K = \alpha_0 \\ T_i = h\alpha_0/(\alpha_0 + \alpha_1) \\ T_d = \beta h/(\beta\alpha_0(1 + \gamma)) \end{cases}$$

On peut calculer les paramètres d en fonction des pôles échantillonnés désirés, dont la valeur est comprise entre 0 et 1, comme suit :

$$\begin{cases} d_1 = -\dfrac{P_1P_2(P_3+P_4)+P_3P_4(P_1+P_2)}{P_1P_2P_3P_4} \\ d_2 = \dfrac{P_1P_2+P_3P_4+(P_1+P_2)(P_3+P_4)}{P_1P_2P_3P_4} \\ d_3 = \dfrac{P_1+P_2+P_3+P_4}{P_1P_2P_3P_4} \\ d_4 = \dfrac{1}{P_1P_2P_3P_4} \end{cases}$$

4.4. Algorithme du P.I.D. discret autoajustable

L'algorithme du P.I.D. discret autoajustable peut être décrit comme suit :

Données : Les pôles p1, p2, p3 et p4.

Etape 0 : Initialisation.

Valeurs initiales des paramètres du modèle.

Etape 1 : Estimation

Mise à jour des paramètres du modèle du second ordre relatif au processus

Etape 2 : Détermination des paramètres de la loi de contrôle.

La résolution du système d'équations

Etape 3 : Signal de commande.

Détermination du signal de commande.

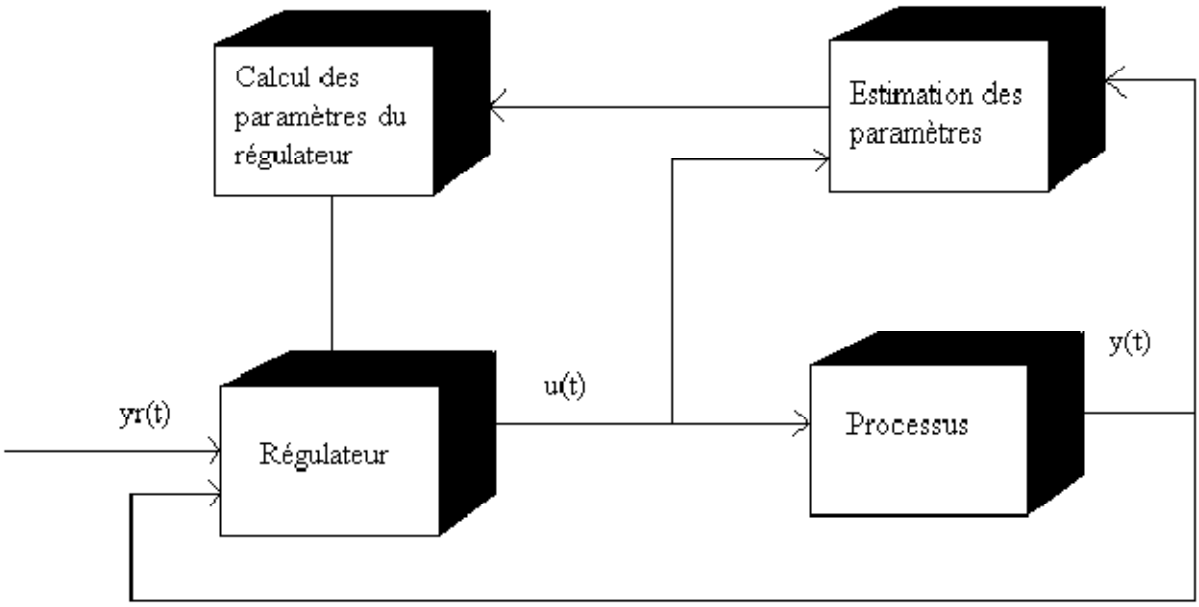


Figure IV.12 : Schéma-bloc de l'algorithme du P.I.D. autoajustable

4.5. Application sur Unity Pro

Nous avons programmé le P.I.D autoajustable sous logiciel Unity Pro sous forme d'une D .F.B (bloc fonction dérivé), et nous l'avons testé sur un système linéaire simulé grâce au bloc Runge-Kutta déjà programmé (voir annexe)

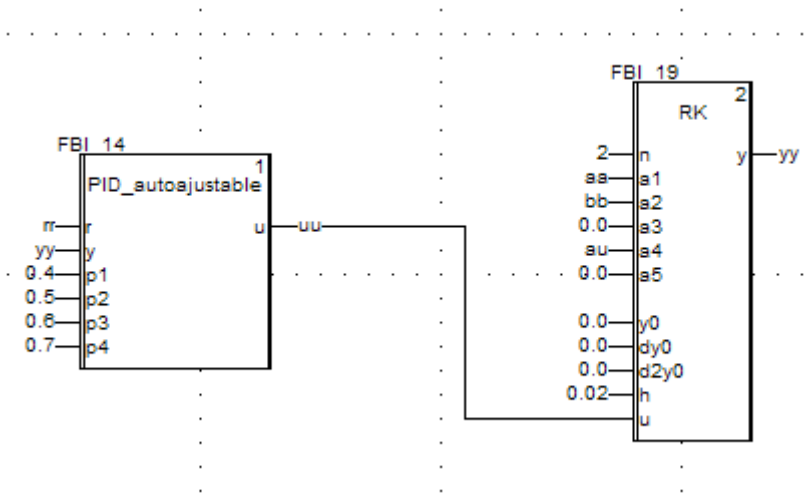


Figure IV.13 : Section d'essai du P.I.D. autoajustable

A l'intérieur du bloc P.I.D. autoajustable, il y a les blocs d'identification (MCR) et de régulation simple (P.I.D.), ainsi qu'un bloc servant à calculer les paramètres du régulateur, dont le programme se trouve en annexe .

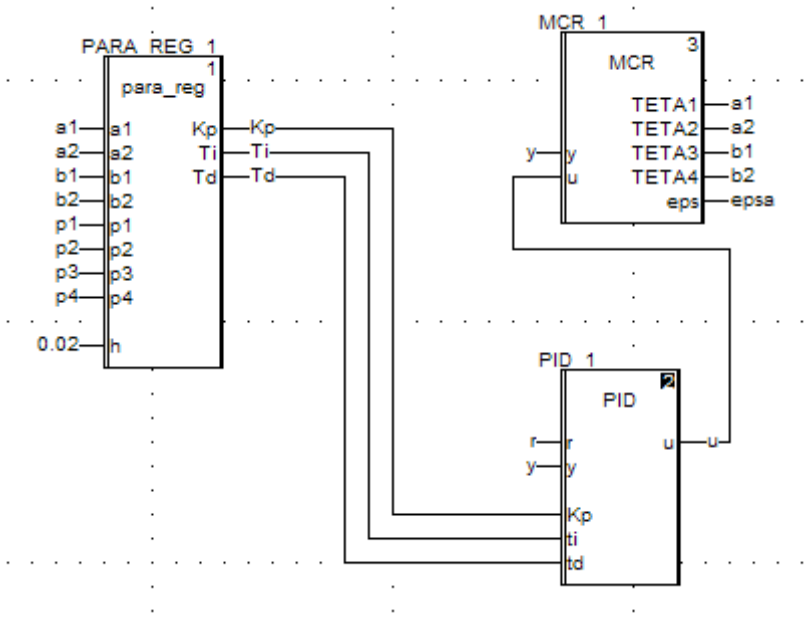


Figure 4.14. Blocs constituant le bloc P.I.D. autoajustable

Pour simuler cette application, nous avons utilisé une table d'animation qui est un outil de Unity Pro où l'on peut changer en temps réel les paramètres du système et même son ordre afin de simuler les fluctuations mais aussi la référence en se gardant de dépasser une certaine valeur vue que la commande est limitée par l'antiwindup. On peut également visualiser la sortie pour voir si elle suit bien la référence.

Nom	Valeur	Type	Commentaire
aa		REAL	
bb		REAL	
au		REAL	
rr		REAL	
yy		REAL	

Figure IV.15 : Table d'animation

Nous avons essayé plusieurs possibilités et la sortie converge en général avec un dépassement de 15% et une erreur de l'ordre de 10^{-6}.

4.6. Conclusion

Pour maintenir les performances du système de commande développé dans les paragraphes précédents, et ceci en présence de perturbations affectant la dynamique du système ou de variations des caractéristiques du processus à réguler, il s'avère utile de modifier en ligne les paramètres du régulateur P.I.D. numérique de façon à pallier l'effet de toutes ces nuisances.

C'est ce que nous avons essayé de réaliser sous logiciel Unity Pro. Et les résultats étaient satisfaisants.

CONCLUSION GENERALE

CONCLUSION GENERALE :

Un ingénieur par définition, doit être capable de résoudre des problèmes que rencontre l'industrie, mais également de répondre à ses besoins. Ceci, avec optimisation et performance. La société « Schneider Electric Algérie » qui nous a accueillis durant notre stage de fin d'étude, avait besoin d'un bloc de régulation auto-ajustable pour les systèmes dynamiques qu'elle rencontre dans ses projets. Ceci fut donc, l'objectif de notre mémoire.

Dans ce rapport, nous avons établi quatre grandes parties essentielles à la réalisation de notre objectif. Tout d'abord, un chapitre sur les automates programmables industriels, où nous avons pris connaissance de leurs architecture, leur fonctionnement, leur environnement … Mais aussi de la gamme « Télémécanique » avec ses différents modules et fonctions.

Dans la seconde partie, nous avons appris à utiliser le logiciel de programmation d'automates « Unity Pro ». Ce logiciel admet cinq langages de programmation. Pour bien illustrer ces langages, nous avons programmé des exemples pour chaque d'eux. Les simulations de ces exemples ont parfaitement suivi les cahiers de charge que nous nous sommes imposé.

Pour pouvoir réaliser des graphiques et visualiser les résultats obtenus avec le logiciel « Unity Pro », nous avons du utiliser le logiciel « Vijeo Designer », qui est un logiciel de programmation d'interfaces Homme-Machine. Vers la fin du chapitre trois, un exemple illustre bien plusieurs options de ce logiciel.

Toutes ces parties ainsi que plusieurs cours étudiés durant notre cursus (MCR, Runge-Kutta, PID…) ont servi à réaliser notre objectif. Le bloc de régulation auto-ajustable a bien fonctionné en simulation et ce malgré les fluctuations des systèmes qu'on lui a appliqués.

Il serait intéressant, de réaliser lors d'un projet futur, une optimisation de ce bloc et de faire des essais par la suite sur un système réel afin de pouvoir l'utiliser dans l'industrie.

BIBLIOGRAPHIE

BIBLIOGRAPHIE

[1] http://st-div.web.cern.ch/st-div/workshop/ST98WS/controls/Dblanc.pdf

[2] M. BERTRAND, **« Automates programmables industriels »**, Techniques de l'ingenieur,Vol. S 8 015.

[3] A.ABRICHE et S.BELKAS **« Réalisation et gestion d'un prototype de station de pompage à base d'automates programmables industriels SIEMENS »** PFE à l'Ecole Nationale Polytechnique

[4] G. MICHEL, **« Les A.P.I Architecture et application des automates programmables industriels »**, Edition DUNOD, 1987

[5]Jack HUGH **« Automating Manufacturing Systems with PLC's** (1993-2005)

[6] P.JARGOT, **« Langages de programmation pour API. Norme IEC 1131-3 »**, Techniques de l'ingénieur, Vol. S 8 030.

[7] A. ACHAB et F.MEFTOUT **« Simulation et Commande des Systèmes Continus et Discontinus en temps réel par Automates Programmables ».** PFE à l'Ecole Nationale Polytechnique.

[8] Catalogue Telemecanique 2005

[9]Catalogue telemecanique juillet 2004 **« plate forme d'automatisme Modicon Premium-unity pro & pl7 »**

[10] P.Dumas, **« Informatique industrielle »**, édition Dunod 2001

[11] Cours d' **« identifications des processus »** 4ème année Automatique. Ecole Nationale Polytechnique

[12] Cours de **« Modélisation Analytique et Simulation »** 4ème année Automatique. Ecole Nationale Polytechnique

[13] Alina BESANÇON-VODA **« Régulateurs PID analogiques et numériques »**
R 7 416 Techniques de l'ingénieur

[14] : L.RUNDQUIST**., "anti-reset windup for PID controllers",** pre-prints of the 11th IFAC wold congress, Vol 8, pp 146-151, 1990.

[15] K.J .STROM, , et L.RUNDQUIST, **"Integrator windup and How to avoid it",**
proceeding of the America control conference, Pittsbourgh,
pp 1693-1698,1989.

[16] Marcel NOUGARET **principes généraux de corrections R7 405 Techniques d'ingénieurs.**

[17] K.NAJIM et G.MURATET **« Pratique numérique des processus industriels.** Edition MASSON 1983.

ملخص

العمل المقدم في هذه المذكرة مبني على دراسة المسير الصناعي وبرمجته. صب أهتمامنا على العائلة "تيليميكانيك" وأيضا على البرمجيات "يونيتي برو" و "فيجيو ديزاينر" حيث الهدف كان برمجة "PID أوتوماتيكي" و تصوير النتيجة على HMI,

كلمات مفتاحية:يونيتي برو, فيجيو ديزاينر, المسيرات, PID أوتوماتيكي, تيليميكانيك

Résumé :

Le travail présenté dans ce mémoire est basé sur l'étude des automates et de leur programmation. Nous nous sommes intéressés à la gamme Télémécanique et les logiciels Unity pro et Vijeo Designer. La finalité étant de programmer un bloc PID auto ajustable ainsi que le simuler et visualiser le résultat sur une HMI.

Mots clés : Unity Pro, Vijeo Designer, Automates programmables industriels, MCR, PID auto ajustable, Télémecanique.

Abstract :

The work presented in this paper is based on the study of controllers and their programming. We were interested in the range Télémécanique and software Unity Pro and Vijeo Designer. The aim was to set a self adjustable PID block, simulate it and visualize the result on an HMI.

Keywords: Unity Pro Vijeo Designer, industrial PLCs, MCR, PID auto adjustable Télémécanique

ANNEXE A

RESEAUX DE TERRAINS

A.1. Introduction

l'API est le cœur du système automatisé, il a donc besoin de communiquer avec les différents organes le constituant (capteurs, pré-actionneurs….)

En premier lieu, on a voulu transmettre le signal « analogique » émis par le capteur, donc les ingénieurs ont essayé d'utiliser des variations de tensions pour transmettre l'information du capteur à travers un fil sur une distance donnée. Mais ce n'était pas fiable parce qu'un simple changement de longueur ou des résistances des fils peut changer la valeur de la mesure.

Dans les années 60 est apparue une solution pour transmettre les données analogiques du capteur sans perte d'information, c'est la ***boucle 4-20mA*.** Elle est devenue alors le standard en ce temps là et a été utilisé jusqu'à la dernière décennie

Depuis la dernière décennie et l'arrivée des technologies numériques dans l'industrie, il y a eu un changement dans la conception et la réalisation des circuits électriques

L'échange d'information étant devenu numérique, il fallait concevoir des liaisons par réseaux entrainant l'utilisation de connecteurs et de connexions préfabriqués. Cela a facilité considérablement la réalisation et la maintenance des équipements électriques.

A.2. Topologie :

C'est l'arrangement physique du réseau, on distingue généralement les variantes suivantes :

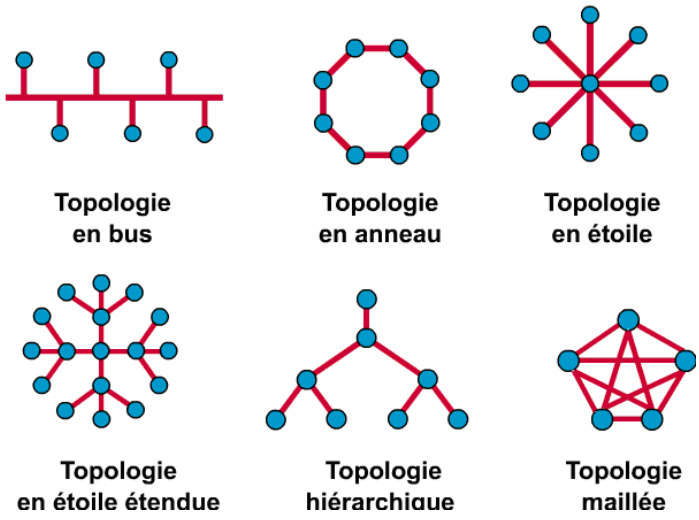


Figure.A.1. Topologie d'un réseau

En bus *:*

C'est l'une des organisations les plus simples. Tous les éléments sont reliés à une même ligne de transmission par l'intermédiaire de câbles. Cette topologie est facile à mettre en œuvre, la défaillance d'un nœud ou d'un élément ne perturbe pas le fonctionnement des autres organes. Les réseaux de niveau machine ou capteurs utilisent cette méthode, ils sont d'ailleurs appelés « bus ». La typologie bus se met en œuvre soit par chaînage des équipements les uns avec les autres, soit par connexion via un boitier de raccordement (TAP) au câble principal

En étoile *:*

Cette topologie a l'avantage d'être très souple en matière de gestion et de dépannage. Un équipement intermédiaire, tel qu'n répéteur ou un commutateur, relie les stations finales. La défaillance d'un nœud ne perturbe pas le fonctionnement global du réseau, mais ce n'est pas le cas de l'équipement intermédiaire qui constitue un point unique de défaillance.

En anneaux :

Reprend la topologie de l'étoile en apportant une meilleure disponibilité du réseau.

En réseau :

Maillé et assez peu utilisé dans l'industrie et présente l'inconvénient d'un nombre important de liaisons.

A.3. Protocole :

Un protocole de communication est une spécification de plusieurs règles pour un type de communication particulier. Initialement, on nommait protocole ce qui est utilisé pour communiquer sur une même couche d'abstraction entre deux équipements différents. Par extension de langage, on utilise parfois ce mot aujourd'hui pour désigner les règles de communication entre 2 couches sur un même équipement.

A.4. Norme OSI :

Le modèle OSI (Open System Interconnexion) a été crée par l'ISO (organisation internationale de normalisation), Il a permis d'obtenir une base commune à la description de tout réseau informatique. Ce modèle est composé de 7couches respectant les principes suivants :

-Chaque couche supporte un protocole indépendamment des autres couches

-Chaque couche procure les services à la couche immédiatement supérieure

-Chaque couche requiert les services de la couche immédiatement inférieure

Les fonctions des différentes couches sont représentées dans le tableau suivant

N :	***Couche***	***Fonction de la couche***	***Exemples***
7	***Application***	-Interface avec l'utilisateur -Parvient les requêtes à la couche présentation	HTTP,SMTP ,POP3 , Modbus
6	***Présentation***	-Définit la manière de représentation des données -Convertit les données pour qu'elles puissent être interprétés par tous les systèmes	HTML,XML
5	***Session***	-Assure les communications et les liaisons correctes entre les systèmes. -Définit l'ouverture des sessions sur les équipements du réseau	ISO8327,NETBIOS
4	***Transport***	-Permet une communication de bout en bout -Gère la segmentation et le réassemblage des données, le contrôle du flux ainsi que la détection d'erreurs et la reprise sur erreur	TCP, UDP, RTP
3	***Réseau***	-S'occupe de l'acheminement de paquets (datagrammes)à travers le réseau	IP, ICMP, IPX, WDS
2	***Liaison***	-permet une liaison sans erreurs à partir du	ARCnet, PPP,

		support physique	Ethernet, Token ring
1	***Physique***	-Définit les protocoles d'échange de bits et les aspects électriques, mécaniques et fonctionnels de l'accès au réseau	CSMA, RS232, 10Base-T , ADSL

Tableau A.1. Modèle OSI

A.5.Etude de quelques réseaux :

Dans cette partie nous allons étudier les réseaux qu'on pourrait, éventuellement utiliser, dans les automates programmables Télémécanique.

A.5.1. Boucle 4-20mA

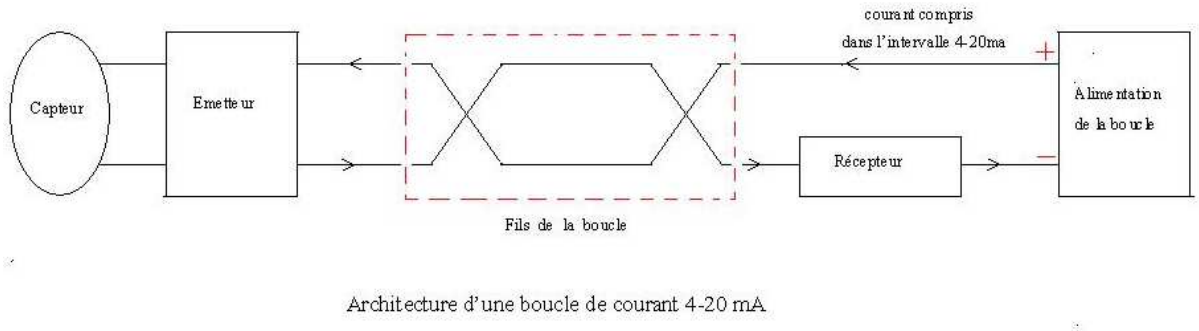
La boucle est composée de :

- Un émetteur : Il est composé d'un capteur qui va mesurer les grandeurs physiques et d'un émetteur de courant 4-20 mA. L'émetteur convertit la valeur mesurée par le capteur en un courant compris dans l'intervalle 4-20 mA (4ma correspond à la plus petite valeur, 20mA à la plus grande e 0mA à un défaut)

- Une alimentation : On utilise en générale des alimentations 24V pour alimenter les émetteurs a l'aide des deux fils de la boucle, il reçoit un courant entre 0-4mA donc il doit consommer moins de 4mA. On peut trouver des alimentations 12V

- Les fils : Deux fils relient tous les composants ensemble. Ces doivent être de faible résistance, avoir bonne proteés parction contre la foudre, être insensibles aux tensions induites par un relais et ayant une seule mise à la masse.

- Le récepteur : Il peut être un afficheur digital ou une table d'enregistrement par exemple, Il se comporte comme une charge résistive. Il peut y avoir plus d'un récepteur dans la boucle tant qu'il y a assez de tension pour alimenter la boucle, il faut juste prendre ça en considération pour fournir l'alimentation adéquate. A titre d'exemple, si on avait 3 récepteurs avec une résistance d'entrée de 250ohms ces 3 résistances consommeraient au maximum 15V (pour un courant de 20mA).Il faudrait alors que l'alimentation fournisse 15V plus la tension nécessaire au fonctionnement du récepteur



Architecture d'une boucle de courant 4-20 mA

Figure A. Boucle 4-20mA

A.5.2.Réseau TELWAY :

Le réseau TELWAY assure la communication entre les automates programmables TSX 7.

La connexion des automates au réseau s'effectue par le coupleur réseau TSX MPT 104.

Ces coupleurs peuvent être implantés dans n'importe quel emplacement (0 à 7) des bases automate. Le nombre dé-coupleurs dépend du type d'automate.

A.5.2.1. Caractéristiques :

Structure	Nature	Réseau local inter automates
	Topologie	Bus avec dérivations passives
	Méthode d'accès	Gestion par maître flottant, liaison série type BSC (Binary Synchronous Communication)
Transmission	Mode	Modulation d'amplitude
	Débit binaire	19,2 K bits/s
	Fréquence porteuse	150 kHz
	Médium	Paire torsadée blindée
Configuration	Nombre de stations	16 maximums
	Longueur du réseau	2000 m maximum hors dérivation
	Dérivations	30 m maximum entre bus et station
	Multi réseau	Possibilité d'interconnexion de 127 réseaux
Services	COM	Base de données distribuée : 64 mots de 16 bits pour l'ensemble des stations (0 ou 4 mots par station)
	UNI-TE	Requêtes point à point avec compte-rendu 32 octets maximum

	Surveillance	Etat du réseau accessible par les terminaux FTX 517 connectés à un réseau ETHWAY/ FIPWAY équipés du logiciel NETDIAG

Tableau A.2. Caractéristiques du réseau TELWAY

A.5.2.2. Performances :

Les temps de transfert des différents messages véhiculés sur le réseau TELWAY sont des temps garantis pour une configuration réseau donnée.

Les différents paramètres sont :

-Le type d'échange : échange des variables communes seul, échange de messages point à point seul ou les 2 types simultanément.

-Le nombre de stations connectées au réseau.

-Le temps de cycle de l'automate composant chaque station.

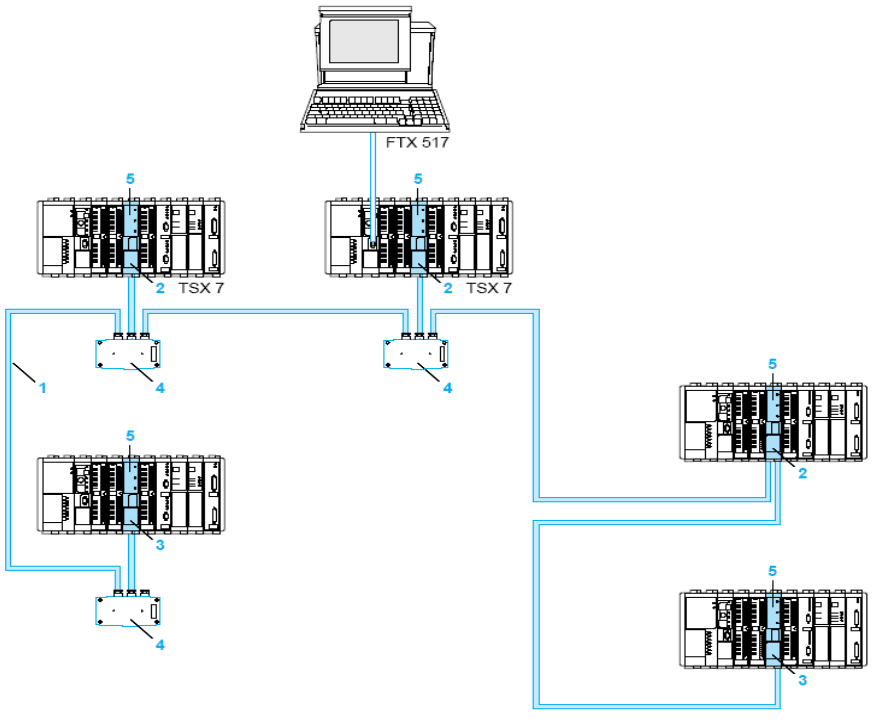


Figure.A.1. Câblage d'un réseau TELWAY

1 TSX CBE 00 : Câble réseau, paire torsadée blindée.

2 TSX MPT 60 : Bornier pour station intermédiaire, inclut le codage d'adresse de la station.

3 TSX MPT 61 : Bornier pour station d'extrémité, inclut le codage d'adresse de la station et l'adaptation de fin de ligne.

4 TSX MPT 50 : Boîtier de dérivation en "T".

5 TSX MPT 104 : Coupleur réseau pour automate TSX/PMX 47/67/87/107.

A.5.3. Ethernet :

Le réseau Ethernet a été crée aux années 1970 par Digital Intel Xerox, il est standardisé en 1985 avec la norme IEEE 802.3

En 1998 fut standardisé le « Fast Ethernet » associé à la technologie « switching technology » et au mode full duplex. De là on a commencé a l'utiliser comme réseau de terrain.

Ethernet vise essentiellement les applications de

- coordination entre automates programmables.

- Supervision locale ou centralisée.

- Communication avec l'informatique de gestion de production.

- Communication avec des entrées/sorties distantes.

Deux profils de communication sont supportés par les coupleurs réseaux ETHERNET

- Le profil ETHWAY reprenant tous les mécanismes de l'architecture de communication X-WAY:

- Système d'adressage X-WAY.

- Messagerie UNI-TE.
- Base de données distribuées (mots communs).

Remarque : ce profil n'est plus utilisé.

- **Le profil TCP/IP** sur ETHERNET permettant la communication en :

- Messagerie UNI-TE avec l'ensemble de l'architecture X-WAY.
- Messagerie Modbus.

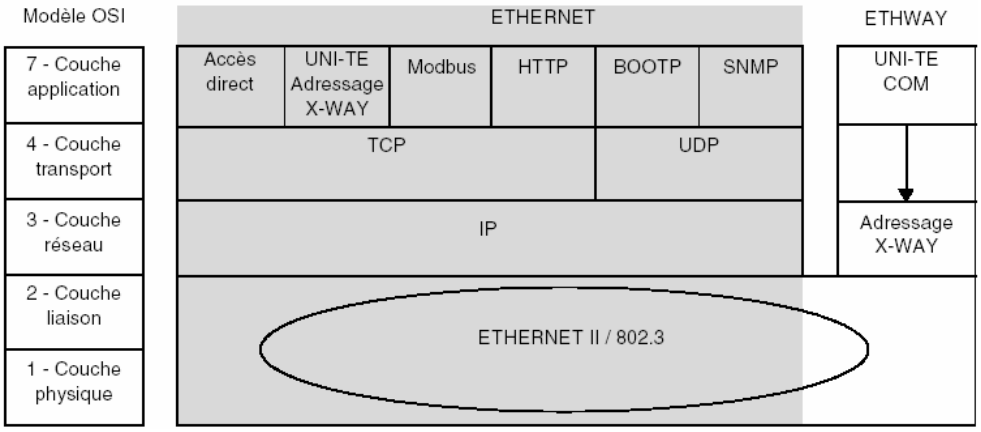


Figure A.2. Figure A.3. Service et protocoles associés à Ethernet

Pour Schneider Electric le choix des services et protocoles associés au réseau Ethernet est fait sur Modbus comme messagerie industrielle , TCP /IP comme pile de réseau et le support physique dédié à l'Ethernet.

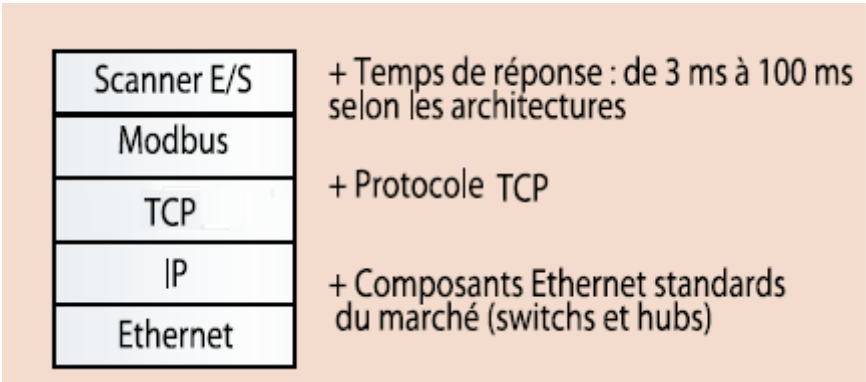


Figure A.3. Service et protocoles associés au Ethernet pour Schneider Electric

A.5.3. Ethernet TCP/IP:

Le mode client permet d'envoyer une requête sur un canal virtuel de communication. Chaque canal virtuel correspond à une connexion TCP sur un équipement.

Le mode serveur pour des automates clients Modbus leur permet d'accéder à la base de données et de surveiller les équipements

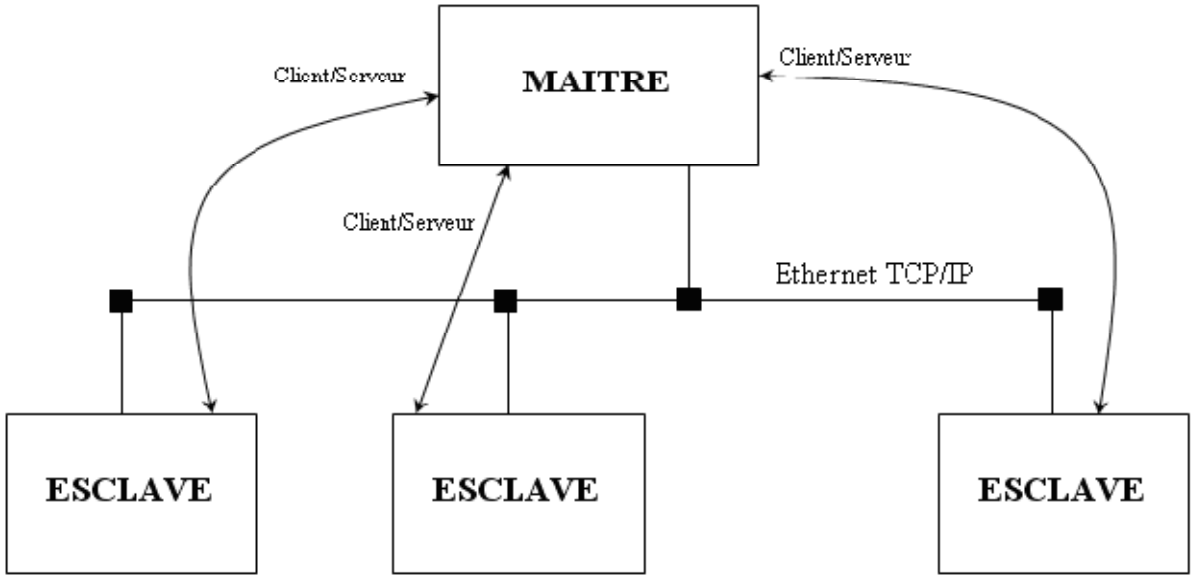


Figure A.4. Communication avec Ethernet TCP/IP

Principe d'encapsulation

Lorsque l'application Modbus envoie des données à l'aide de TCP/IP les données traversent de haut en bas chaque couche jusqu'à aboutir au support physique où elles sont alors émises sous forme de suite de bits. L'encapsulation illustré ci-dessous consiste pour chaque couche à ajouter de l'information aux données en les commencent par des en-têtes

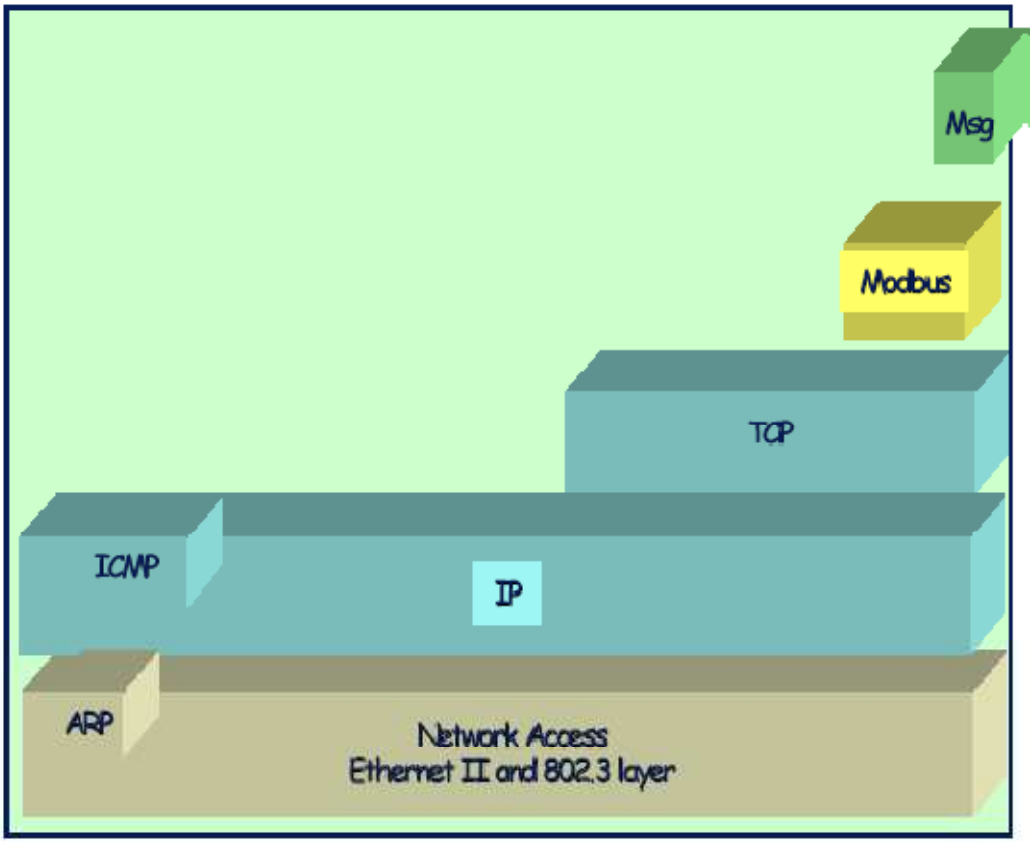
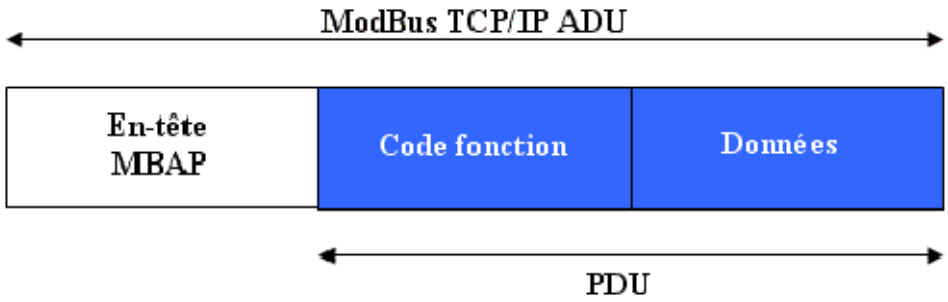


Figure A.5. Principe d'encapsulation

Le protocole Modbus sur TCP/IP

Le protocole Modbus sur TCP/IP est défini comme suit



ADU : Application Data Unit

PDU : Protocol Data Unit

Toutes les requêtes et réponses de Modbus sont conçues de telle manière que le destinataire puisse vérifier qu'un message est fini.

L'information de longueur est diffusée dans l'en-tête de MBAP composé de 7 octets pour permettre au destinataire d'identifier les frontières des messages même si le message a été coupé en plusieurs paquets pour la transmission.

Donc, Un même coupleur peut communiquer avec un équipement distant en mode client (par exemple un automate Quantum) et un autre équipement distant en mode serveur (par exemple un PC superviseur).

Par exemple l'automate Premium 1 est client vis à vis de l'automate Quantum. Il ouvre la connexion TCP-IP et émet des messages Modbus vers le Quantum.

L'automate Premium 1 est serveur vis à vis Premium 2. Le Premium 2 ouvre une connexion

TCP/IP et émet des messages Modbus vers le TSX Premium1

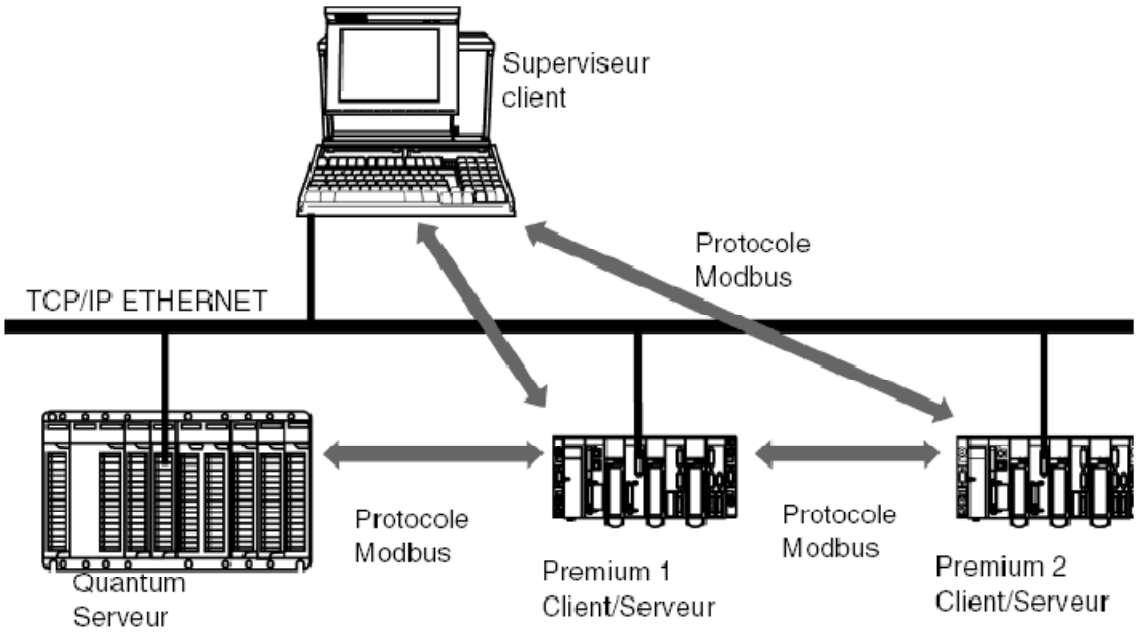


Figure A.5.Communication via Ethernet TCP/IP et Modbus

ANNEXE B

CATALOGUE DES MODULES

B.1. Catalogue des alimentations

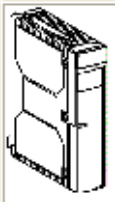
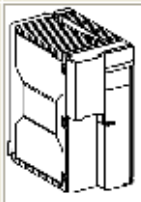
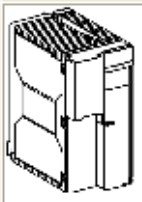
Références	TSX PSY 2600	TSX PSY 5500	TSX PSY 8500
			
Caractéristiques d'entrées			
Tensions nominales	100...240 VCA	100...120 VCA / 200...240 VCA	100...120 VCA / 200...240 VCA
Valeurs limites	85...264 VCA	85...140 VCA / 190...264 VCA	85...140 VCA / 190...264 VCA
Fréquence limite	47...63 Hz	47...63 Hz	47...63 Hz
Durée micro-coupures secteur acceptée	inférieure ou égale à 10 ms	inférieure ou égale à 10 ms	inférieure ou égale à 10 ms
Puissance apparente	50 VA	150 VA	150 VA
Courant nominal d'entrée	0,5 A à 100 V 0,3 A à 240 V	1,7 A à 100 V 0,5 A à 240 V	1,7 A à 100 V 0,5 A à 240 V
Caractéristiques de sorties			
Puissance totale	26 W	50 W	80 W
Tensions de sortie	5 V, 24 VR (1) 24 VC (2)	5 V, 24 VR (1) 24 VC (2)	5 V, 24 VC (2)
Courant nominal 5 V	5 A	7 A	15 A
Courant nominal 24 VR	0,6 A	0,8 A	non fourni
Courant nominal 24 VC	0,5 A	0,8 A	1,6 A
Fonctions auxiliaires			
Relais alarme	oui (1 contact à fermeture, libre de potentiel sur bornier		
Visualisation	oui, par voyant en face avant		
Pile de sauvegarde	oui (surveillance état par voyant en face avant du module)		
Conformité aux normes	IEC 1131-2		

Tableau B.1 Catalogue des alimentations pour réseaux à courant alternatif

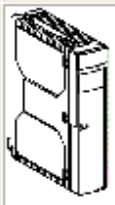
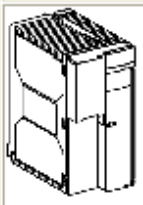
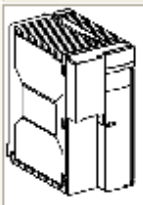
Références	TSX PSY 1610	TSX PSY 3610	TSX PSY 5520
			
Caractéristiques d'entrées			
Tensions nominales	24 VCC non isolée	24 VCC non isolée	24...48 VCC isolée
Valeurs limites	19,2...30 VCC	19,2...30 VCC	19,2...60 VCC
Durée micro-coupures secteur acceptée	inférieure ou égale à 1 ms	inférieure ou égale à 1 ms	inférieure ou égale à 1 ms
Courant nominal d'entrée	≤ 1,5 A	≤ 2,7 A	≤ 3 A/24 V 1,5 A/48 V
Caractéristiques de sorties			
Puissance totale	26 W	50 W	80 W
Tensions de sortie	5 V, 24 VR (1)	5 V, 24 VR (1)	5 V, 24 VR (1)
Courant nominal 5 V	5 A	7 A	7 A
Courant nominal 24	0,6 A	0,8 A	0,8 A
Fonctions auxiliaires			
Relais alarme	oui (1 contact à fermeture, libre de potentiel sur bornier		
Visualisation	oui, par voyant en face avant		
Pile de sauvegarde	oui (surveillance état par voyant en face avant du module)		
Conformité aux normes	IEC 1131-2		

Tableau B.2. Catalogue des alimentations pour réseaux à courant continu

B.2 Le catalogue des modules d'entrées analogiques

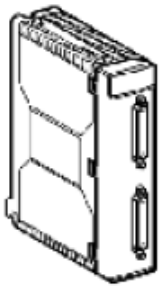
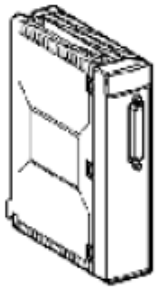
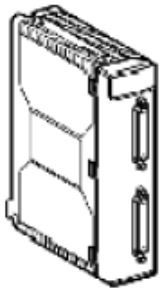
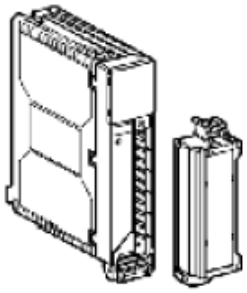
Type de module	Entrées					
						
Nombre de voies	16	8		4	16	4
Plage	+/- 10 V 0 à 10 V 0 à 5 V 1 à 5 V 0 à 20 mA 4 à 4 20 mA				+/- 80 mV Thermocouple	+/- 10 V 0 à 10 V +/- 5 V 0 à 5 V 1 à 5 V 0 à 20 mA 4 à 20 mA -13 à +63 mV 0 à 400 ohms 0 à 3 850 ohms Thermosonde Thermocouple
Courant consommé à 24 VR	0 mA					
Courant consommé à 5 V	270 mA (typ.) 380 mA (max.)		475 mA (typ.) 630 mA (max.)	500 mA (typ.) 800 mA (max.)	300 mA (typ.) 400 mA (max.)	660 mA (typ.) 940 mA (max.)
Mode de tension partagée des voies	Partagé		+/- 200 VCC	Partagé	+/- 100 VCC	+/- 200 VCC
Résolution	12 bits		16 bits			
Connexions	2 x Sub-D 25 broches	1 x Sub-D 25 broches			2 x Sub-D 25 broches	Bornier à vis à 20 broches
TELEFAST 2 dédié	ABE-7CPA 02 ABE-7CPA 03	ABE-7CPA 02 ABE-7CPA 03	ABE-7CPA 02 ABE-7CPA 31	ABE-7CPA 02 ABE-7CPA 03 ABE-7CPA 21	ABE-7CPA 12	-
Référence•• TSX	AEY 1600	AEY 800	AEY 810	AEY 420	AEY 1614	AEY 414

Tableau B.3.Le catalogue des modules d'entrées analogiques

-

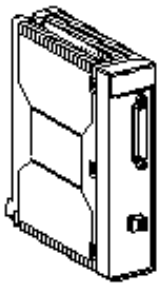
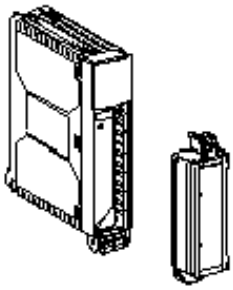
Type de module	Sorties analogiques	
		
Nombre de voies	8	4
Plage	+/- 10 V 0 à 20 mA 4 à 20 mA	
Courant consommé à 24 VR	300 mA (typ.) (1) 455 mA (max.)	0 mA
Courant consommé à 5 V	200 mA (typ.) 300 mA (max.)	990 mA (typ.) (2) 1 220 mA (max.) (2)
Mode de tension partagée des voies	Partagé	Isolement de 1 500 Veff
Résolution	14 bits en tension 13 bits en courant	11 bits signe +
Connexions	1 x 25 broches Sub-D Bornier à vis à 2 broches	Bornier à vis à 20 broches
TELEFAST 2 dédié	ABE-7CPA 02	ABE-7CPA 21
Référence•• TSX	ASY 800	ASY 410
Légende :		
(1)	Uniquement dans le cas d'utilisation d'une alimentation interne 24 V (0 mA dans le cas d'une alimentation externe)	
(2)	+20 mA par voie active.	

Tableau B.4. Le catalogue des modules de sorties analogiques :

B.3. Catalogue des modules d'entrées/de sorties TOR

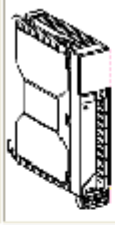
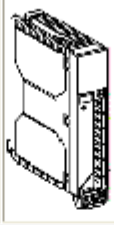
Type de module	Entrées avec raccordement par bornier à vis						
Illustration	Module d'entées TOR 			Module d'entrées TOR 			
Nombre de voies	8 entrées	16 entrées					
Gamme	24 VCC		48 VCC	24 VCA 24 VCC	48 VCA	100..120 VCA	200..240 VCA
Isolement	Entrées isolées						
Conformité CEI 1131-2	Type 2 (1)						
Logique	Positive			Négative	-		
Compatibilité DDP	DDP 2 fils DC et 3 fils PNP, normes CEI 947-5-2			DDP 2 fils DC et 3 fils NPN, normes CEI 947-5-2			
				DDP 2 fils AC, normes CEI 947-5-2			
Filtrage	Intégré 4 ms			Intégré, Réseau 50 ou 60 Hz			
Raccordements	Bornier à vis						
Référence TSX••	**DEY 08D2**	**DEY 16D2**	**DEY 16D3**	**DEY 16A2**	**DEY 16A3**	**DEY 16A4**	**DEY 16A5**
Légende :							
(1) Pour le module **TSX DEY 16A2**, conformité type 2 uniquement en 24 VCA.							

Tableau. B.5. Catalogue des modules d'entrées TOR (à bornier a vis)

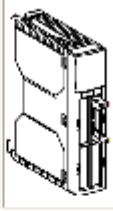
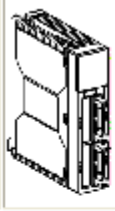
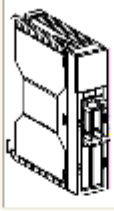
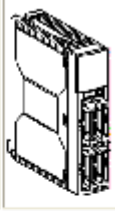
Type de module	Entrées avec raccordement par connecteur **HE10**			
Illustration	Module d'entrées TOR 	Mod. E TOR 	Mod. E TOR 	Mod. E TOR 
Nombre de voies	16 entrées rapides	32 entrées		64 entrées
Gamme	24 VCC		48 VCC	24 VCC
Isolement	Entrées isolées			
Conformité CEI 1131-2	Type 1		Type 2	Type 1
Logique	Positive			
Compatibilité DDP	DDP 2 fils DDP 3 fils PNP			
Filtrage Filtrage programmable Mémorisation d'état Evénement	0,1..7,5 ms par pas de 0,5 oui oui oui	Fixe 4 ms		
Raccordements	Connecteurs **HE10**			
Référence TSX••	**DEY 16FK**	**DEY 32D2K**	**DEY 32D3K**	**DEY 64D2K**

Tableau B.6. Catalogue des modules d'entrées TOR (connecteur HE10)

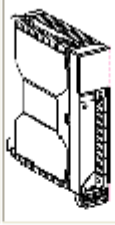
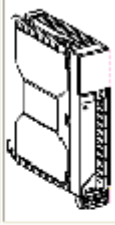
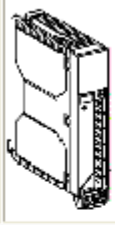
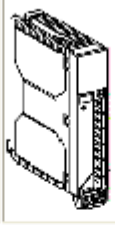
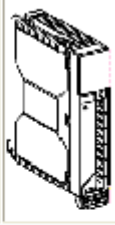
Type de module	Sorties à transistors avec raccordement par bornier à vis				
Illustration	Module de sorties TOR 	Module de sorties TOR 	Module de sorties TOR 	Module de sorties TOR 	Module de sorties TOR 
Nombre de voies	8 sorties			16 sorties	
Gamme	24 VCC		48 VCC	24 VCC	48 VCC
Isolement	Sorties isolées				
Courant	0,5 A	2 A	1 A	0,5 A	0,25 A
Conformité CEI 1131-2	Oui				
Protection	Sorties protégées contre le court-circuits et les surcharges avec réarmement automatique ou commandé avec circuit de démagnétisation rapide des électro-aimants.				
Repli	Repli configurable des sorties Surveillance permanente de la commande des sorties et mise à 0 des sorties si détection d'un défaut interne.				
Logique	Positive				
Temps réponse	1 ms	0,2 ms	0,3 ms	1 ms	1 ms
Raccordements	Bornier à vis				
Référence TSX••	**DSY 08T2**	**DSY 08T22**	**DSY 08T31**	**DSY 16T2**	**DSY 16T3**

Tableau B.7. Catalogue des modules de sortie à transistors TOR (bornier à vis)

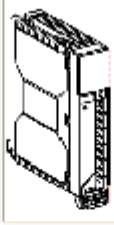
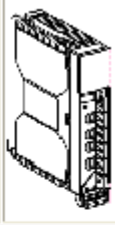
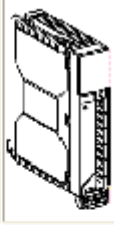
Type de module	Sorties à relais avec raccordement par bornier à vis			
Illustration	Module TOR 	Module de sorties TOR 		Module TOR 
Nombre de voies	8 sorties			16 sorties
Gamme	12..24 VCC ou 24..240 VCA	24..130 VCC	24..48 VCC ou 24..240 VCA	12..24 VCC ou 24..240 VCA
Isolement	Sorties isolées entre contact et terre			
Courant	3 A	5 A		3 A
Conformité CEI 1131-2	Oui			
Protection	Pas de protection	Protection par fusibles interchangeables. Mise à 0 des sorties sur détection de défaut, réarmement après remplacement du fusible.		Pas de protection
Repli	Repli configurable des sorties.			
Déverrouillage bornier	Dispositif de coupure automatique des sorties lors du déverrouillage bornier.			
Logique	Positive/négative			
Raccordements	Bornier à vis			
Référence TSX••	**DSY 08R5**	**DSY 08R4D**	**DSY 08R5A**	**DSY 16R5**

Tableau B.8. Catalogue des modules de sortie à relais TOR (bornier à vis)

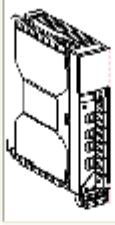
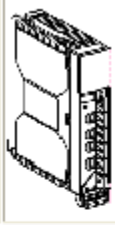
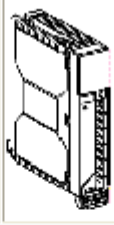
Type de module	Sorties à triac avec raccordement par bornier à vis		
Illustration	Module de sorties TOR 	Module de sorties TOR 	Module de sorties TOR 
Nombre de voies	8 sorties	16 sorties	
Gamme	48..240 VCA		24..120 VCA
Isolement	Sorties isolées		
Courant	2 A	1 A	
Conformité CEI 1131-2	Oui		
Protection	Protection par fusibles interchangeables.		Sorties non protégées contre les court-circuits ou surcharges. Protection 'anti-feu' par fusibles non interchangeables.
Repli	Repli configurable des sorties.		
Déverrouillage bornier	Dispositif de coupure automatique des sorties lors du déverrouillage bornier.		
Raccordements	Bornier à vis		
Référence TSX••	**DSY 08S5**	**DSY 16S5**	**DSY 16S4**

Tableau B.9. Catalogue des modules de sortie à triac TOR (bornier à vis)

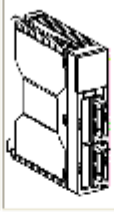
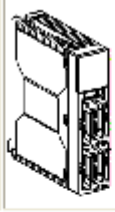
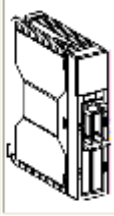
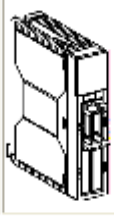
Type de module	Sorties statiques avec raccordement par connecteur **HE10**	
Illustration	Module de sorties TOR 	Module de sorties TOR 
Nombre de voies	32 sorties	64 sorties
Gamme	24 VCC	
Isolement	Sorties isolées	
Courant	0,1 A	
Conformité CEI 1131-2	Oui	
Protection	Sorties protégées contre les court-circuits et les surcharges avec réarmement automatique ou commandé.	
Repli	Repli configurable des sorties surveillance permanente de la commande des sorties et mise à 0 des sorties si détection d'un défaut interne.	
Logique	Positive	
Raccordements	Connecteur HE 10	
Référence TSX••	**DSY 32T2K**	**DSY 64T2K**

Tableau B.10. Catalogue des modules de sortie à triac TOR (connecteur H1O)

	Type de module	Sorties statiques avec raccordement par connecteur HE 10	
	Illustration	Module mixte E/S TOR 	Module mixte E/S TOR 
	Nombre de voies	16 entrées rapides 12 sorties	16 entrées rapides 16 sorties réflexes
Entrées	Gamme	24 VCC	
	Isolement	Entrées isolées	
	Conformité CEI 1131-2	Type 1	
	Logique	Positive	
	Compatibilité DDP	DDP 2 fils	
	Filtrage programmable	Oui (0,1..7,5 ms par pas de 0,5)	
	Mémorisation d'état	Oui	
	Evénement	Oui	

Sorties	Gamme	24 VCC	
	Isolement	Sorties isolées	
	Courant	0,5 A	
	Conformité CEI 1131-2	Oui	
	Protection	Sorties protégées contre les court-circuits et les surcharges avec réarmement automatique ou commandé avec circuit de démagnétisation rapide des électro-aimants.	
	Repli	Repli configurable des sorties. Surveillance permanente de la commande des sorties et mise à 0 des sorties si détection d'un défaut interne.	
	Logique	Positive	
	Temps réponse	0,6 ms	
	Raccordements	Connecteurs HE10	
	Référence TSX••	DMY 28FK	DMY 28RFK

Tableau B.11. Catalogue d'entées sorties mixtes

B.4. Mise en place des modules dans le rack

Tous les modules d'E/S TOR Premium possèdent un format standard

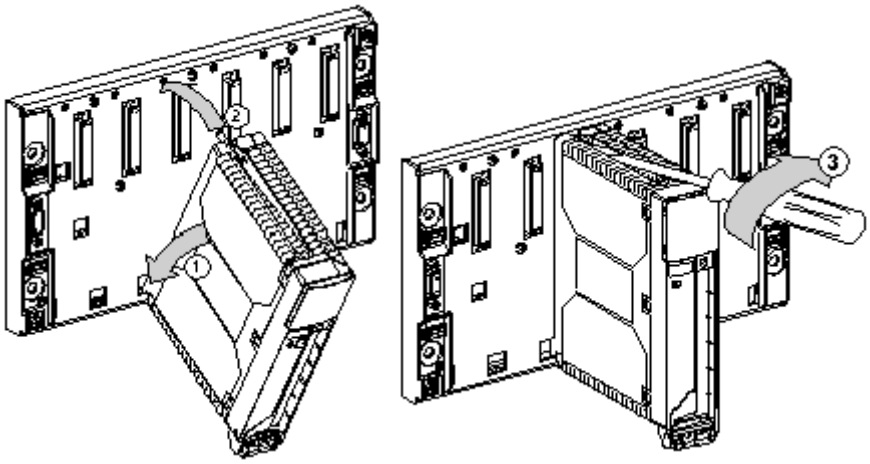


Figure .B.1. Mise en place d'un module TOR

1. Positionnement des deux ergots situés à l'arrière du module (dans la partie inférieure) dans les emplacements correspondants du rack.

2. Ramener le module vers le haut afin de l'enclencher sur le connecteur du rack.

3. Serrage des vis de fixation de la partie supérieure du module de façon à solidariser le module avec le rack (couple de serrage : 2,0 N.m). Si cette vis n'est pas serrée, le module ne peut pas rester en position dans le rack.

Remarque : On peut procéder au montage et au démontage des modules lorsque l'alimentation capteurs et pré-actionneurs est coupée et le bornier déconnecté.

ANNEXE C

UNITY PRO

C.1. Configuration des processeurs PREMIUM

Type TSX	Format physique	Nombre d'E/S TOR maximum par rack	Taille mémoire maximum			Liaison Fipio maître intégrée	Liaison Ethernet intégrée
			RAM interne	PCMCIA			
				Données	Programme		
P57 0244 (1)	Simple	256	96 K8	96 K8	128 K8	-	-
P57 104	Simple	512	96 K8	96 K8	224 K8	-	-
P57 1634	Double	512	96 K8	96 K8	224 K8	-	X
P57 154	Simple	512	96 K8	96 K8	224 K8	X	-
P57 204	Double	1024	160 K8	160 K8	768 K8	-	-
P57 254	Double	1024	192 K8	192 K8	768 K8	X	-
P57 2634	Double	1024	160 K8	160 K8	768 K8	-	X
P57 304	Double	1024	192 K8	192 K8	1 792 K8	-	-
P57 354	Double	1024	224 K8	224 K8	1 792 K8	X	-
P57 3634	Double	1024	192 K8	192 K8	1 792 K8	-	X
P57 454	Double	2048	320 K8	440 K8	2048 K8	X	-
P57 4634	Double	2048	320 K8	440 K8	2 048 K8	-	X
P57 554	Double	2048	1024 K8	1024 K8	7168 K8	X	-
P57 5634	Double	2048	1024 K8	1024 K8	7168 K8	-	X
P57 6634	Double	2048	640 K8	896 K8	4096 K8	-	X
H57 24M	Double	1024	192 K8	192 K8	768 K8	-	X
H57 44M	Double	2048	440 K8	440 K8	2048 K8	-	X

Tableau C.1. Configuration des processeurs PREMIUM

C .2. Exemples de processeurs avec nombre de racks gérés :

Type de processeurs	Nombre de racks gérés
Pour tous les automates M340 Version 01.00.	1 rack
Pour BMX P34 1000 Version 02.00	2 racks
Pour BMX P34 20X0	4 racks

Tableau C .2. Nombre de rack gérés par les processeurs M340

Type de processeurs	Nombre de racks gérés
TSX 57 0244	1 rack
TSX 57 1x4	Jusqu'à 4 racks
TSX P57 204 TSX PCI 57 204 TSX P57 254 TSX P57 2634 TSX P57 304 TSX P57 354 TSX P57 3634 TSX P57 454 / TSX PCI 57 354 TSX P57 4634 TSX P57 554 TSX P57 5634	Jusqu'à 16 racks

Tableau C .3. Nombre de rack gérés par les processeurs PREMIUM

Type de processeurs	Nombre de racks gérés
140 CPU 311-10	Ne dépend pas du type de processeur
140 CPU 434-12A	
140 CPU 534-14A	
140 CPU 651-50\60\60S	
140 CPU 671-60\60S	

Tableau C .4. Nombre de rack gérés par les processeur M340

C.3. Jeux d'instructions :

C.3.1. Liste d'instruction (IL) :

Modificateur	Applicable sur les opérandes du type de données	Description
N	BOOL, BYTE, WORD, DWORD	Le modificateur N est utilisé pour inverser bit à bit la valeur d'un opérande.
C	BOOL	Le modificateur C est utilisé pour exécuter l'instruction associée, si la valeur de l'accu est 1 (TRUE).
CN	BOOL	Si le modificateur C est combiné avec le modificateur N, l'instruction associée est exécutée seulement si la

		valeur de l'accumulateur est un 0 booléen (FALSE).
(	Toutes	Le modificateur Parenthèse gauche (est utilisé pour repousser l'évaluation de l'opérande, jusqu'à ce que l'opérateur Parenthèse droite) apparaisse. Le nombre d'opérations Parenthèse droite doit être égal au nombre de modificateurs Parenthèse gauche. Il est possible d'imbriquer les parenthèses.

Tableau. C.5. Modificateurs :

Opérateur	**Modificateur**	**Signification**	**Opérandes**	**Description**
LD	N (uniquement pour les opérandes du type de données BOOL, BYTE, WORD ou DWORD)	Charge la valeur de l'opérande dans l'accumulateur	Valeur littérale, variable, adresse directe avec type de données quelconque	Avec LD, la valeur d'un opérande est chargée dans l'accumulateur. Les données de l'accumulateur s'adaptent automatiquement au type de données de l'opérande. Cela s'applique également aux types de données dérivés.
ST	N (uniquement pour les	Enregistre la valeur de l'accumulateur dans	Variable, adresse directe avec type de	Avec ST, la valeur actuelle de l'accumulateur est enregistrée dans l'opérande. Le type de données de l'opérande doit correspondre au type

	opérandes du type de données BOOL, BYTE, WORD ou DWORD)	l'opérande.	données au choix	de données de l'accumulateur.

Opérateur	**Modificateur**	**Signification**	**Opérandes**	**Description**
S	-	Définit l'opérande sur 1 si le contenu de l'accumulateur est 1.	Variable, adresse directe du type de données BOOL	S permet de définir l'opérande sur 1 lorsque le contenu de l'accumulateur actuel est un 1 booléen.
R	-	Définit l'opérande sur 0 si le contenu de l'accumulateur est 1.	Variable, adresse directe du type de données BOOL	R permet de définir l'opérande sur 0 lorsque le contenu actuel de l'accumulateur est un 1 booléen.
AND	N, N(, (	ET logique	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD ou DWORD	L'opérateur AND établit une liaison ET logique entre le contenu de l'accumulateur et l'opérande. Pour les types de données BYTE, WORD et DWORD, la liaison est effectuée par bit.

OR	N, N(, (	OU logique	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD ou DWORD	L'opérateur OR établit une liaison OU logique entre le contenu de l'accumulateur et l'opérande. Pour les types de données BYTE, WORD et DWORD, la liaison est effectuée par bit.
XOR	N, N(, (	OU exclusif logique	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD ou DWORD	L'opérateur XOR établit une liaison OU exclusif logique entre le contenu de l'accumulateur et l'opérande. Si plus de deux opérandes sont reliés, le résultat de l'opération est à l'état 1 pour un nombre impair d'états 1 et à l'état 0 pour un nombre pair d'états 1. Pour les types de données BYTE, WORD et DWORD, la liaison est effectuée par bit.
NOT	-	Négation logique (complément)	Contenu d'accumulateur du type de données BOOL, BYTE, WORD ou DWORD	NOT permet d'inverser le contenu de l'accumulateur par bit.

Tableau C.6. Opérateur Logiques :

Opérateur	**Modificateur**	**Signification**	**Opérandes**	**Description**
ADD	(	Addition	Valeur littérale, variable, adresse directe du type de données INT, DINT, UINT, UDINT, REAL ou TIME	ADD permet d'ajouter la valeur de l'opérande à la valeur du contenu de l'accumulateur.
SUB	(	Soustraction	Valeur littérale, variable, adresse directe du type de données INT, DINT, UINT, UDINT, REAL ou TIME	SUB permet de retirer la valeur de l'opérande du contenu de l'accumulateur.
MUL	(	Multiplication	Valeur littérale, variable, adresse directe du type de données INT, DINT, UINT, UDINT ou REAL	MUL permet de multiplier le contenu de l'accumulateur par la valeur de l'opérande. .
DIV	(	Division	Valeur littérale, variable, adresse directe du type de données INT, DINT, UINT, UDINT ou REAL	DIV permet de diviser le contenu de l'accumulateur par la valeur de l'opérande.
MOD	(	Division modulo	Valeur littérale, variable, adresse directe du type de	MOD permet de diviser la valeur du premier opérande par la valeur du deuxième et

			données INT, DINT, UINT ou UDINT	de sortir le reste de la division (modulo) comme résultat.

Tableau C.7. Opérateurs arithmétiques :

Opérateur	Modificateur	Signification	Opérandes	Description
GT	(	Comparaison : >	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	GT permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le contenu de l'accumulateur est supérieur au contenu de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur est inférieur ou égal au contenu de l'opérande, le résultat est un 0 booléen.
GE	(	Comparaison : >=	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD, DWORD, STRING, INT,	GE permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le contenu de l'accumulateur est supérieur/égal au contenu de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur est inférieur au

			DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	contenu de l'opérande, le résultat est un 0 booléen.
EQ	(	Comparaison : =	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	EQ permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le contenu de l'accumulateur est égal à celui de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur n'est pas égal à celui de l'opérande, le résultat est un 0 booléen.
NE	(	Comparaison : <>	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	NE permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le contenu de l'accumulateur n'est pas égal à celui de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur est égal à celui de l'opérande, le résultat est un 0 booléen.
LE	(	Comparaison : <=	Valeur littérale, variable, adresse directe du type	LE permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le

			de données BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	contenu de l'accumulateur est inférieur ou égal à celui de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur est supérieur à celui de l'opérande, le résultat est un 0 booléen.
LT	(	Comparaison : <	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	LT permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le contenu de l'accumulateur est inférieur à celui de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur est supérieur ou égal à celui de l'opérande, le résultat est un 0 booléen.

Tableau C.8.Opérateurs de comparaison

Opérateur	**Modificateur**	**Signification**	**Opérandes**	**Description**
CAL	C, CN (uniquement si le contenu de l'accumulateur est du type de données BOOL)	Appel d'un bloc fonction, d'un DFB ou d'un sous-programme	Nom d'instance d'un bloc fonction, d'un DFB ou d'un sous-programme	CAL permet d'appeler un bloc fonction, un DFB ou un sous-programme avec ou sans conditions.
NOM_DE_FONCTION	-	Exécution d'une fonction	Valeur littérale, variable, adresse directe (le type de données dépend de la fonction)	Le nom de fonction vous permet d'exécuter une fonction.
NOM_DE_PROCEDURE	-	Exécution d'une procédure	Valeur littérale, variable, adresse directe (le type de données dépend de la procédure)	Le nom de procédure vous permet d'exécuter une procédure.

Tableau C.9. Opérateurs d'appel

Opérateur	Modificateur	Signification	Opérandes	Description
JMP	C, CN (uniquement si le contenu de l'accumulateur est du type de données BOOL)	Saut vers l'étiquette	ETIQUETTE	JMP permet d'effectuer un saut avec ou sans conditions vers une étiquette.
RET	C, CN (uniquement si le contenu de l'accumulateur est du type de données BOOL)	Retour dans l'unité organisationnelle supérieure suivante du programme	-	Des opérateurs RETURN peuvent être utilisés dans des blocs de fonction dérivés DFB et des SR (sous-programmes). Des opérateurs RETURN ne peuvent pas être utilisés dans le programme principal. Dans un DFB, un opérateur RETURN force le retour au programme qui a appelé le DFB. Le reste de la section de DFB contenant l'opérateur RETURN n'est pas exécuté. Les sections suivantes du DFB ne sont pas exécutées. Le programme qui a appelé

				le DFB sera exécuté après retour du DFB. Si le DFB est appelé par un autre DFB, le DFB appelant sera exécuté après le retour. Dans un SR, un opérateur RETURN force le retour au programme qui a appelé le SR. Le reste du SR contenant l'opérateur RETURN n'est pas exécuté. Le programme qui a appelé le SR seraexécuté après retour du SR.
)	-	Traitement d'opérations placées en attente	-	La parenthèse droite) permet de lancer l'édition de l'opérateur en attente. Le nombre d'opérations Parenthèse droite doit être égal au nombre de modificateurs Parenthèse gauche. Il est possible d'imbriquer les parenthèses.

Tableau C.10. Opérateurs de structuration

C.3.2. LADDER DIAGRAM (LD):

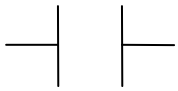
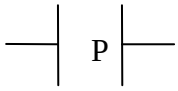
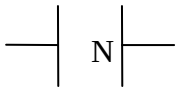
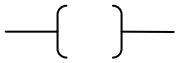
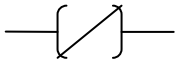
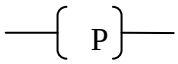
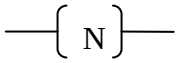
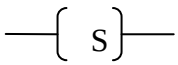
Désignation	**Représentation**	**Description**
A fermeture		Dans le cas de contacts à fermeture, l'état de la liaison de gauche est transféré vers la liaison de droite si l'état du paramètre booléen réel associé (indiqué par xxx) est ON. Sinon, l'état de la liaison de droite est OFF.
A ouverture		Dans le cas de contacts à ouverture, l'état de la liaison de gauche est transféré vers la liaison de droite si l'état du paramètre booléen réel approprié (indiqué par xxx) est OFF. Sinon, l'état de la liaison de droite est OFF.
Contact de détection de transitions positives		Dans le cas de contacts de détection de transitions positives, la liaison de droite est ON pour un cycle de programme, si un passage de OFF à ON du paramètre réel booléen (xxx) associé a lieu et qu'en même temps, l'état de la liaison de gauche est ON. Sinon, l'état de la liaison de droite est 0.
Contact de détection de transitions négatives		Dans le cas de contacts de détection de transitions négatives, la liaison de droite est ON pour un cycle de programme, si un passage de ON à OFF du paramètre réel booléen (xxx) associé a lieu et qu'en même temps, l'état de la liaison de gauche est ON. Sinon, l'état de la liaison de droite est 0.

Tableau C.11. Contacts :

Désignation	**Représentation**	**Description**
Bobine		Dans le cas de bobines, l'état de la liaison de gauche est transféré vers le paramètre booléen réel associé (indiqué par xxx) et la liaison de droite.
bobine inverse		Dans le cas de bobines inverses, l'état de la liaison de gauche est copié sur la liaison de droite. L'état inversé de la liaison de gauche est copié vers le paramètre booléen réel associé (indiqué par xxx). Si la liaison de gauche est OFF, alors la liaison de droite sera également OFF et le paramètre booléen réel associé sera ON.
Bobine de détection de transitions positives		Dans le cas de bobines de détection de transitions positives, l'état de la liaison de gauche est copié sur la liaison de droite. Le paramètre réel associé du type de données EBOOL (indiqué par xxx) est 1 pour un cycle de programme, si un passage de 0 à 1 de la liaison gauche est effectué.
Bobine de détection de transitions négatives		Dans le cas de bobines de détection de transitions négatives, l'état de la liaison de gauche copié sur la liaison de droite. Le paramètre réel booléen associé (indiqué par xxx) est 1 pour un cycle de programme, si un passage de 1 à 0 de la liaison gauche est effectué.
Bobine d'enclenchement		Avec une bobine d'enclenchement, l'état de la liaison de gauche est copié sur la liaison de droite. Le paramètre booléen réel associé (indiqué par xxx) est défini sur ON si l'état de la liaison de gauche est ON, sinon il reste inchangé. Le paramètre booléen réel associé peut être réinitialisé via la bobine de réinitialisation.
Bobine de		Avec une bobine de réinitialisation, l'état de la liaison

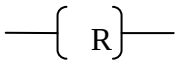
réinitialisation		de gauche est copié sur la liaison de droite. Le paramètre booléen réel associé (indiqué par xxx) est défini sur OFF si l'état de la liaison de gauche est ON, sinon il reste inchangé. Le paramètre booléen réel associé peut être enclenché via la bobine d'enclenchement.
Bobine d'arrêt		Avec des bobines d'arrêt, si le statut de la liaison de gauche est 1, l'exécution du programme est arrêtée immédiatement. (Avec des bobines d'arrêt, l'état de la liaison de gauche n'est pas copié sur la liaison de droite.)
Bobine d'appel		Avec des bobines d'appel, l'état de la liaison de gauche est copié vers la liaison de droite. Si l'état de la liaison de gauche est ON alors le sous-programme associé (indiqué par xxx) est appelé. Le sous-programme à appeler doit se trouver dans la même tâche que la section LD appelante. Il est possible d'appeler des sous-programmes au sein de sous-programmes. Les sous-programmes sont un complément de la norme CEI 61131-3 et doivent être activés de manière explicite. Dans les sections d'actions SFC, les bobines d'appel (appels de sous-programmes) ne sont autorisés que si le mode Multitoken a été activé.

Tableau C.12. Bobines

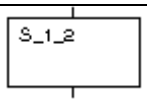
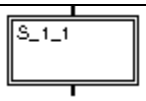
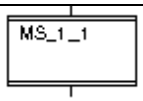
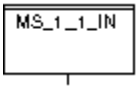
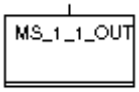
C.3. 3. FBD :

Désignation	Représentation	Description
Jump		Si l'état de la liaison gauche est 1, un saut est exécuté jusqu'à l'étiquette (dans la section courante). Pour générer un saut conditionnel, l'objet saut est lié à une sortie FFB booléenne. Pour générer un saut inconditionnel, la valeur 1 est affectée à l'objet saut via la fonction AND.
Libellé	LABEL:	Les repères (destinations de saut) sont représentés comme du texte avec deux-points à la fin. Le texte est limité à 32 caractères et doit être unique dans l'ensemble de la section. Le texte doit respecter les conventions de nommage générales. Les étiquettes de saut ne peuvent être placées qu'entre les deux premiers points de trame sur la marge gauche de la section. Note : Les étiquettes de saut ne doivent "couper" aucun réseau, c'est-à-dire qu'une ligne imaginaire entre l'étiquette de saut et la marge droite de la section ne doit être coupée par aucun objet. Cela est également valable pour les liaisons.
Return		Des objets RETURN ne peuvent pas être utilisés dans le programme principal. Dans un DFB, un objet RETURN force le retour au programme qui a appelé le DFB.

		Le reste de la section de DFB contenant l'objet RETURN n'est pas exécuté. Les sections suivantes du DFB ne sont pas exécutées. Le programme qui a appelé le DFB sera exécuté après retour du DFB. Si le DFB est appelé par un autre DFB, le DFB appelant sera exécuté après le retour. Dans un SR, un objet RETURN force le retour au programme qui a appelé le SR. Le reste du SR contenant l'objet RETURN n'est pas exécuté. Le programme qui a appelé le SR sera exécuté après retour du SR.

Tableau C.13. Instruction FDB

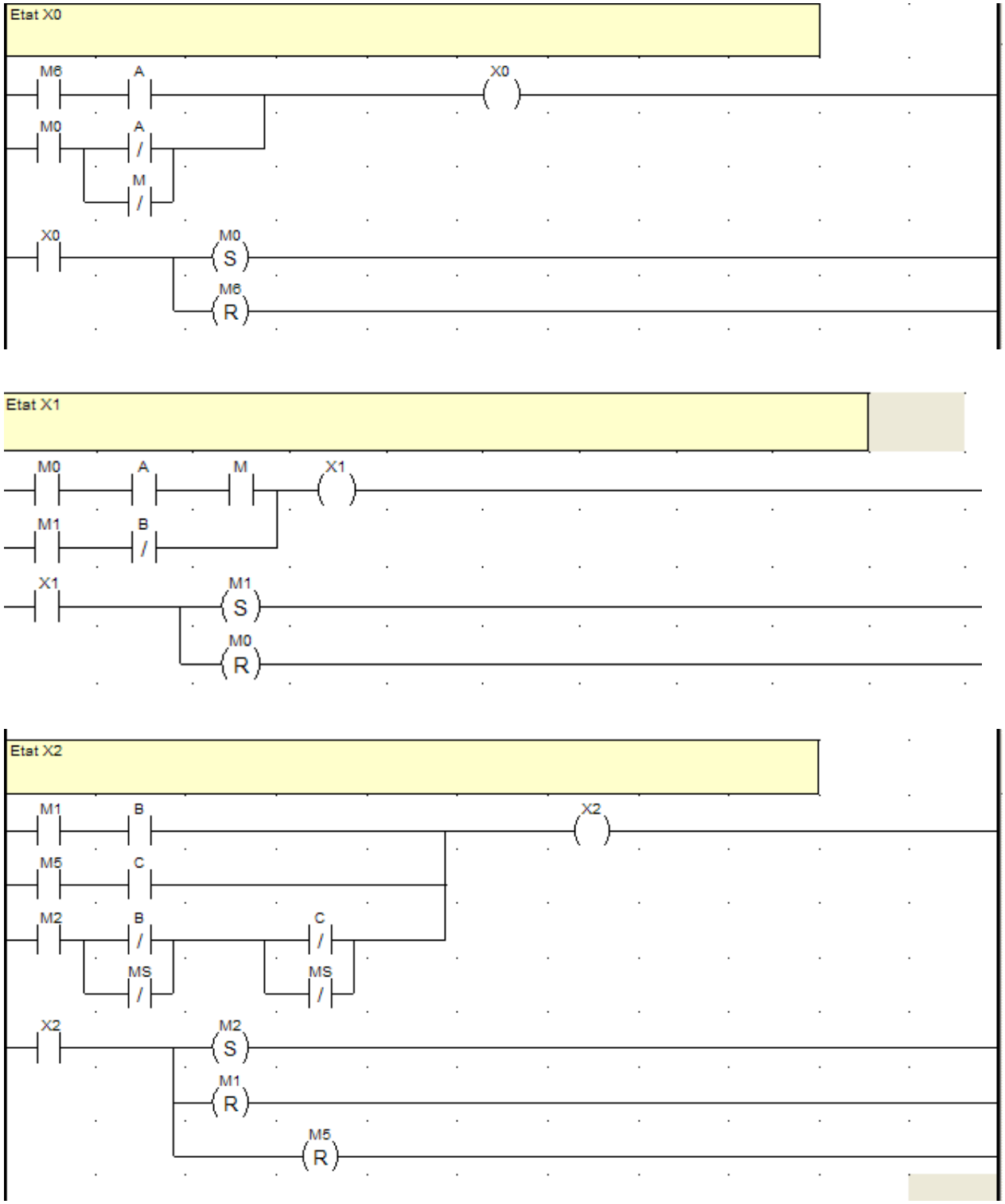
C.3.4. SFC

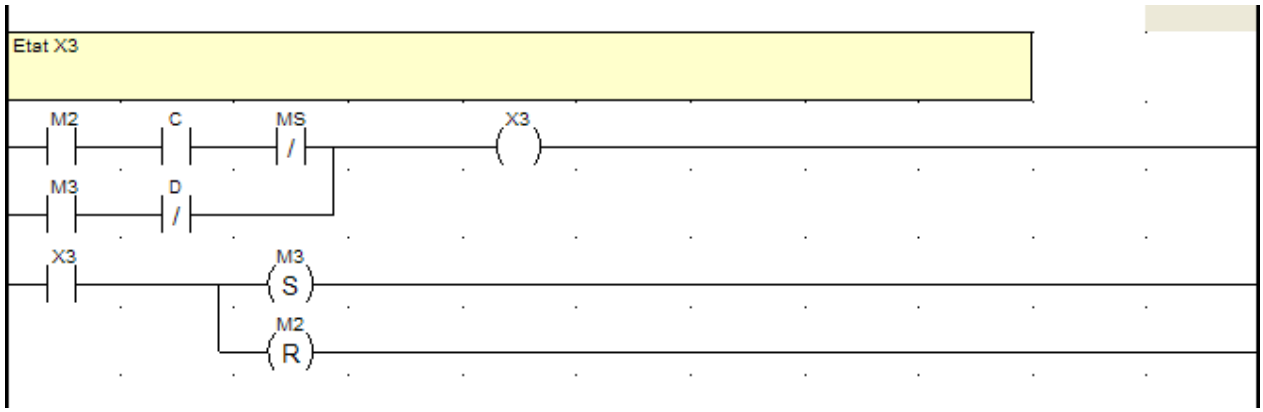
Type	Représentation	Description
Etape « normale »		Une étape devient active lorsque l'étape précédente devient inactive (un temps de retard éventuellement défini doit s'être écoulé) et la transition située en amont est vraie. Une étape devient généralement inactive lorsque le temps de retard éventuellement défini s'est écoulé et que la transition située en aval est vraie. Pour les convergences en ET, toutes les étapes précédentes doivent être vraies Chaque étape compte zéro ou plusieurs actions. Les étapes sans action sont considérées comme des étapes d'attente.
Etape initiale		L'état initial d'une séquence est caractérisé par l'étape initiale. A l'issue de l'initialisation du projet ou de la séquence, l'étape initiale est active. Généralement, aucune action n'est affectée aux étapes initiales. Pour les jetons uniques (conformes à la norme CEI 61131-3), seule une étape initiale est admise par séquence. Pour les jetons multiples un nombre d'étapes initiales pouvant être défini (de 0 à 100) est possible.
Macroétape		servent à appeler une macro-section et ainsi à établir une structure hiérarchique des commandes d'enchaînement.
Etape d'entrée		Chaque macro-section commence par une étape d'entrée.
Etape de sortie		Chaque macro-section se termine par une étape de sortie.

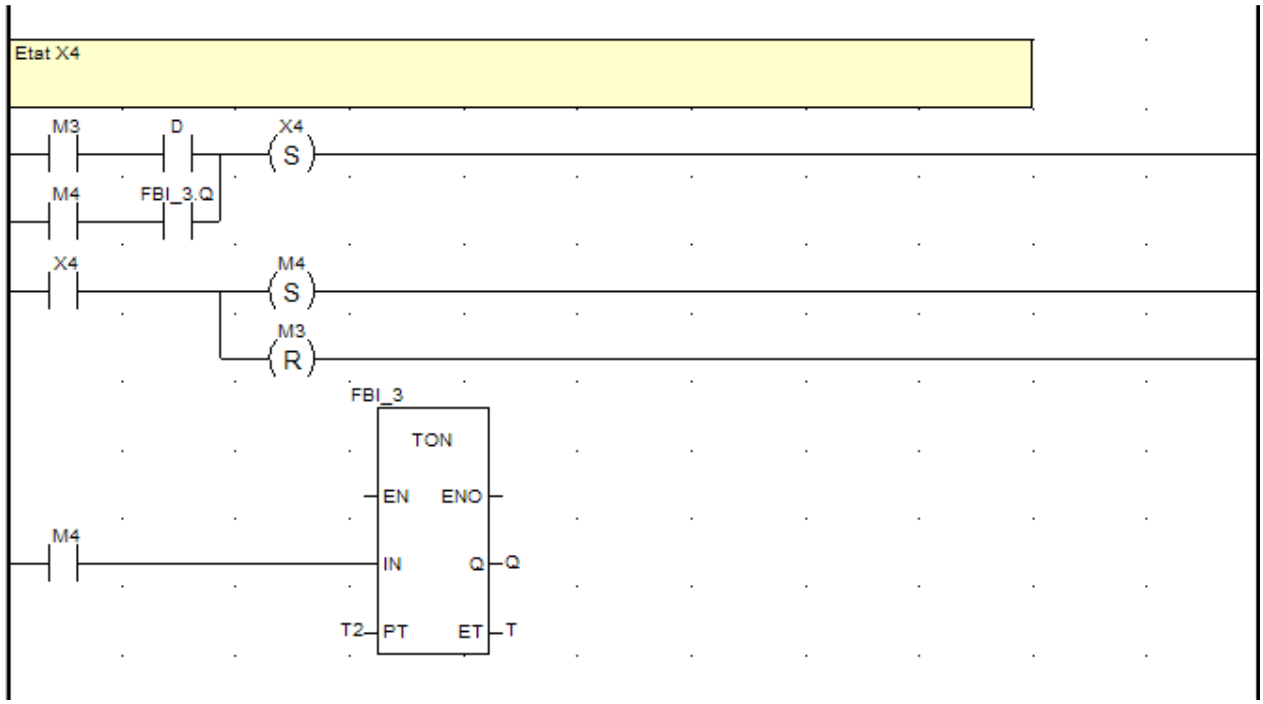
En plus de ces fonctions de contrôles, on peut importer toutes les fonctions et les blocs de fonctions de la bibliothèque.

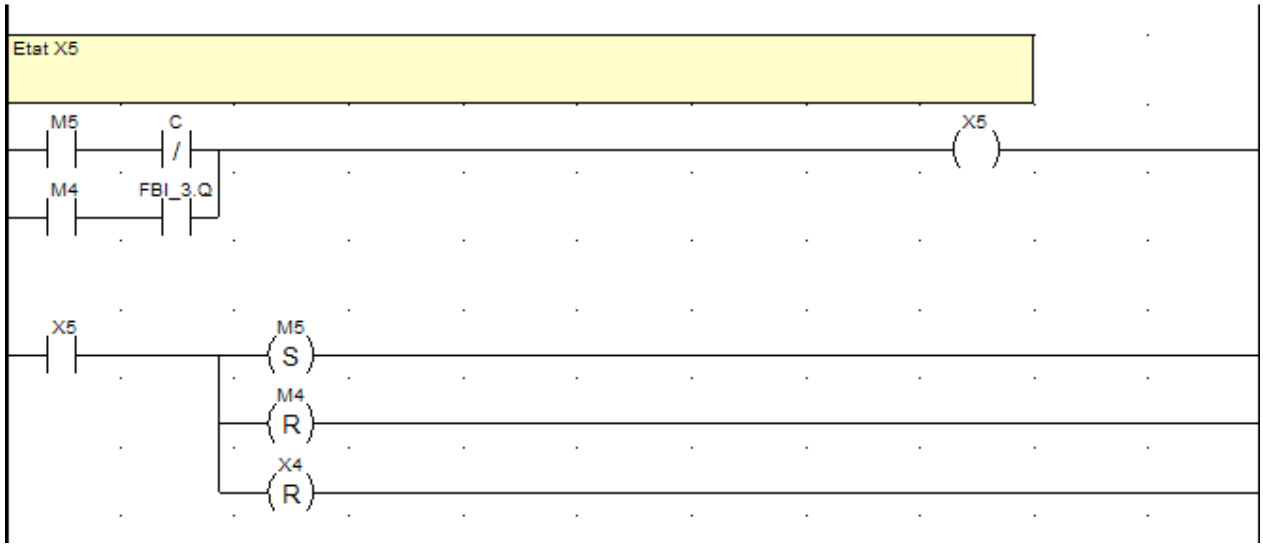
C.4. Programmes des exemples d'application de UNITY PRO

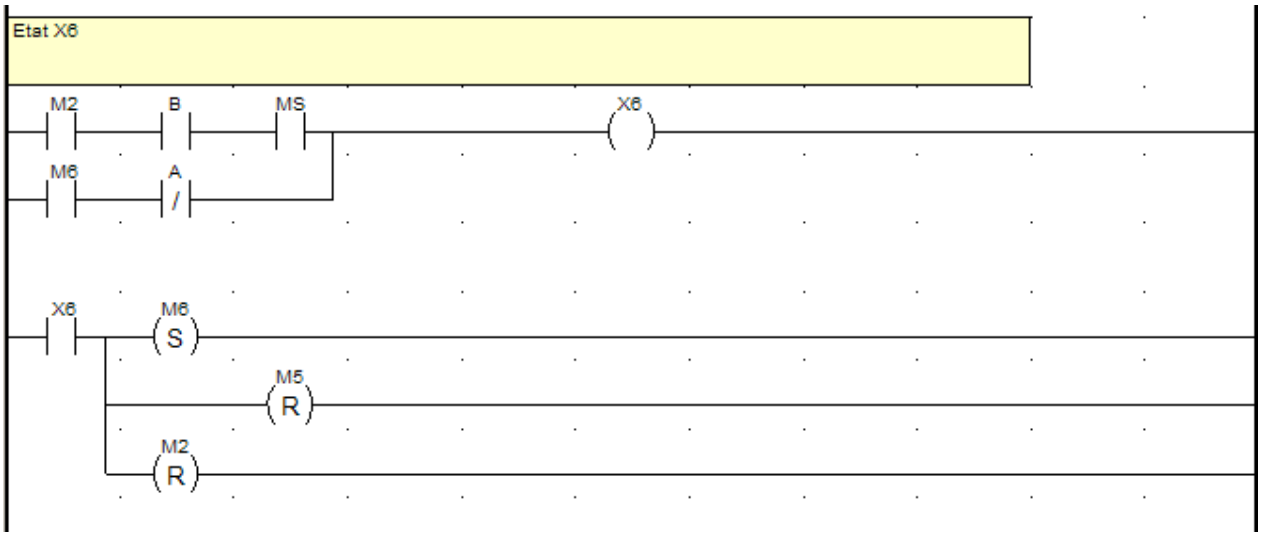
C.4.1. Programme du chariot

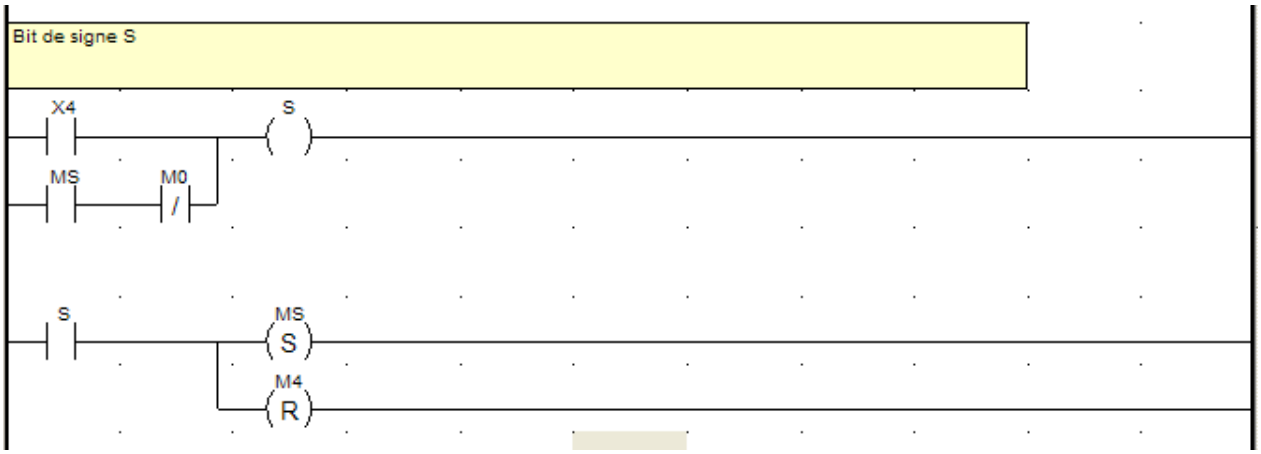


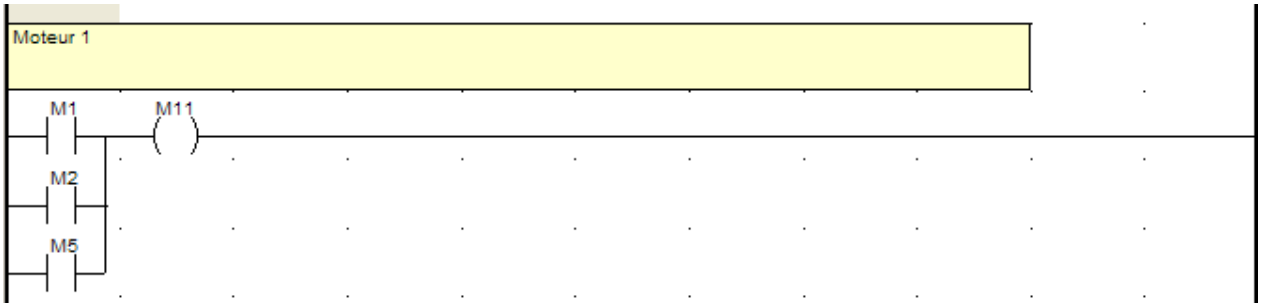

Etat X3
M2
C
MS
X3
M3
D
X3
M3
S
M2
R

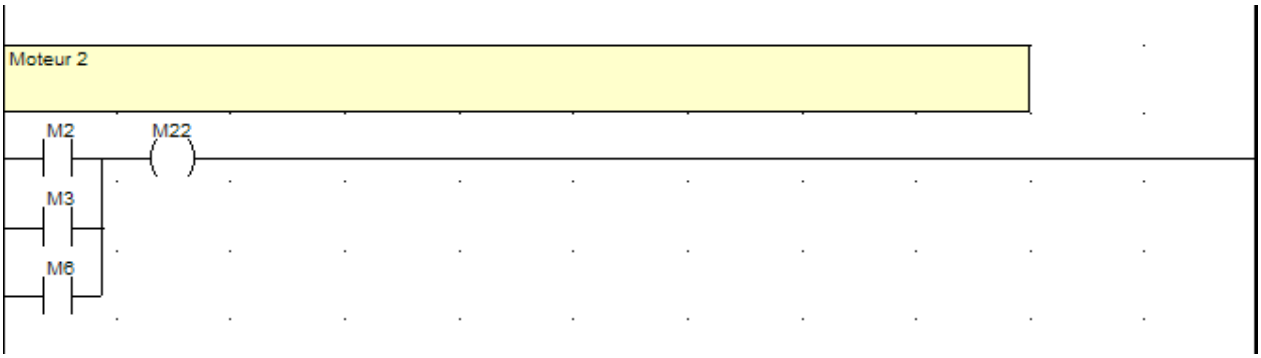

Etat X4
M3
D
X4
S
M4
FBI_3.Q
X4
M4
S
M3
R
FBI_3
TON
EN
ENO
M4
IN
Q
Q
T2
PT
ET
T


Etat X5
M5
C
X5
M4
FBI_3.Q
X5
M5
S
M4
R
X4
R


Etat X6
M2
B
MS
X6
M6
A
/
X6
M6
S
M5
R
M2
R


Bit de signe S
X4
S
MS
M0
/
S
MS
S
M4
R


Moteur 1
M1
M11
M2
M5


Moteur 2
M2
M22
M3
M6

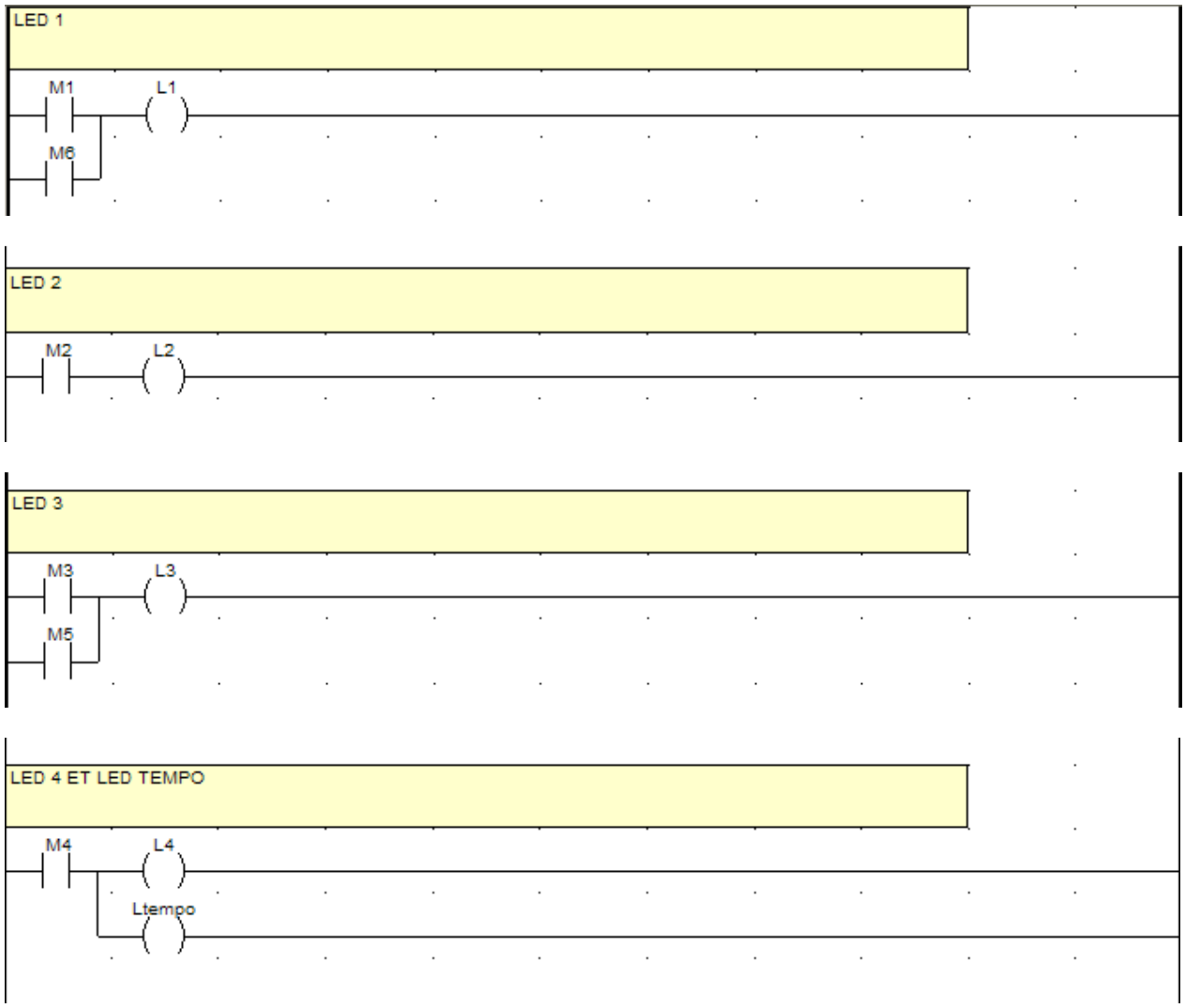


Figure C.1. Programme du chariot

C.4.1. Programme de simulation du circuit RC

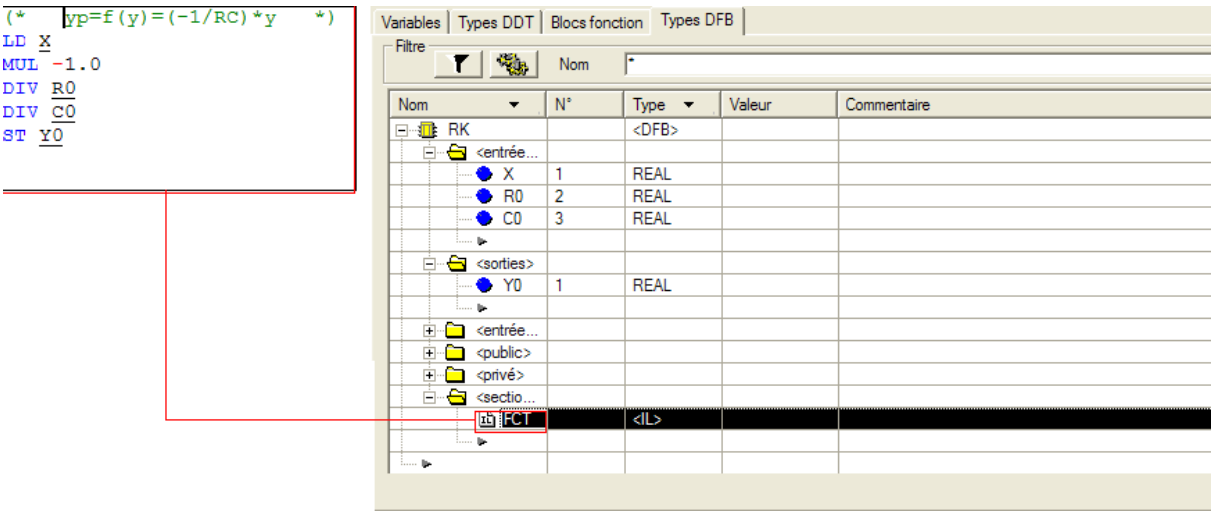


Figure.C.2. Création de la DFB de l'exemple « circuit RC »

```
(* Calcul du nombre d'itération N=(TF-T0)/H *)
LD TF
SUB T0
DIV H
REAL_TO_INT
ST N

(* Initialisation de y *)
LD YINIT
ST Y

(* Début de la boucle *)
START:

LD I
LT N
JMPCN FIN

(* Calcul de K1=fct(y)*)
CAL RK_1 (X:=Y,R0:=R,C0:=C,Y0=>K1 )

(* Calcul de K2=fct(y+K1*H/2)*)
LD K1
MUL H
DIV 2.0
ADD Y
ST Y1
CAL RK_1 (X:=Y1,R0:=R,C0:=C,Y0=>K2 )

(* Calcul de K3=fct(y+K2*H/2)*)
LD K2
MUL H
DIV 2.0
ADD Y
ST Y2
CAL RK_1 (X:=Y2,R0:=R,C0:=C,Y0=>K3 )
```

```
(* Calcul de K4=fct(y+K3*h)*)

LD K3
MUL H
ADD Y
ST Y3
CAL RK_1 (X:=Y3,R0:=R,C0:=C,Y0=>K4 )

(* Calcul du nouveau y:   y(i+1)=y(i)+(h/6)*(K1+2K2+2K3+K4)   *)

LD H
DIV 6.0
MUL (
LD 2.0
MUL K2
ADD (2.0
MUL K3
)
ADD K1
ADD K4
)
ADD Y
ST Y

(* Incrémentation de I *)

LD 1
ADD I
ST I

JMP START

(* Arret du programme *)
 FIN: HALT ()
```

Figure.C.3. Programme principal de l'exemple « circuit RC »

C.4.3. Programme de l' exemple « fabrication du mélange »

```
If rec1 then
recette:=1;
R3:=false;
R4:=false;
rec2:=false;
end_if;

if rec2 then
recette:=2;
R1:=false;
R2:=false;
rec1:=false;
end_if;

if (rec1=false and rec2=false) then
R5:=false;
R6:=false;
end_if;

CASE recette OF
     1: R1:=R5; R2:=R6; (* 1ère recette *)
     2: R3:=R5; R4:=R6;
     END CASE;


    If m=true then

    If  (N<=H1 and x=false) then
    R5:=true;
    end_if;

    if (H1<N and N<H2 and x=false) then
    R5:=false;
    R6:=true;
    end_if;

    if N>=H2 then
    x:=true;
    y:=true;
    R6:=false;
    Ag:=true;
    tempo:=true;
    end_if;

    FBI_1 (IN := tempo,
           PT := T0,
           Q => S);
```

```
if (tempo=true and S=false) then
   tempo:=false;
   Ag:=false;
   if N>L then
         R:=true;
end_if;
end_if;


if N<L then
        R:=false;
        x:=false;
   END_if;

If (R5 or R6) then
N:=N+1.0;
end_if;

if (y and N<L) then
    m:=false;
    y:=false;
    end_if;

end_if;

                If R then
                N:=N-1.0;
                end_if;
```

Figure C.4. Programme de l'exemple « Fabrication de mélanges »

C.4.4. Simulation de l' exemple « fabrication du mélange »

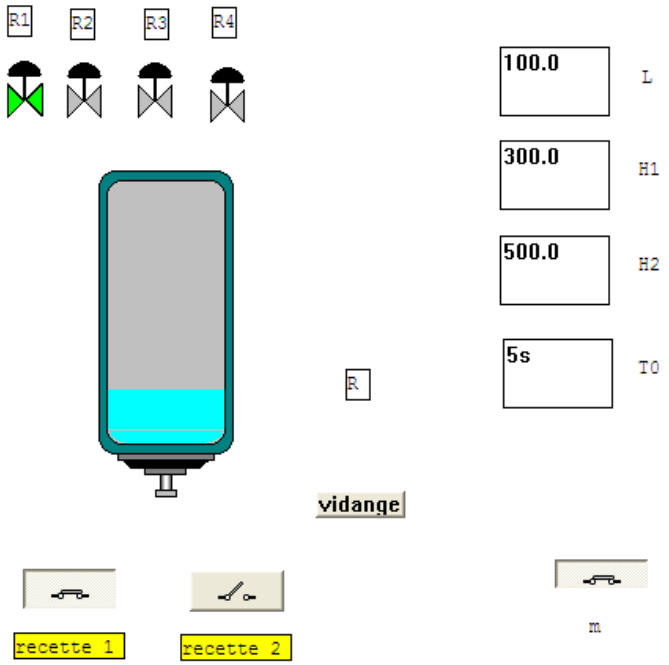


Fig. C.5. 1ère étape de la simulation

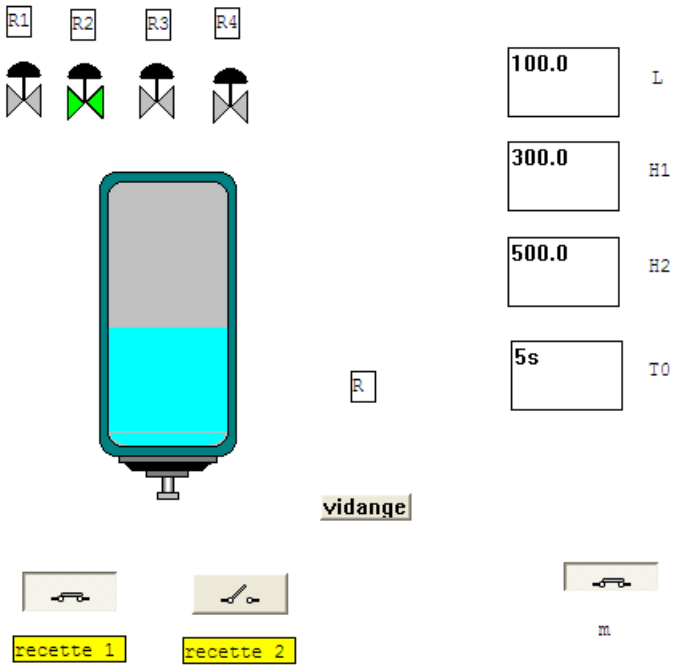


Fig. C.6. 2ème étape de l'application

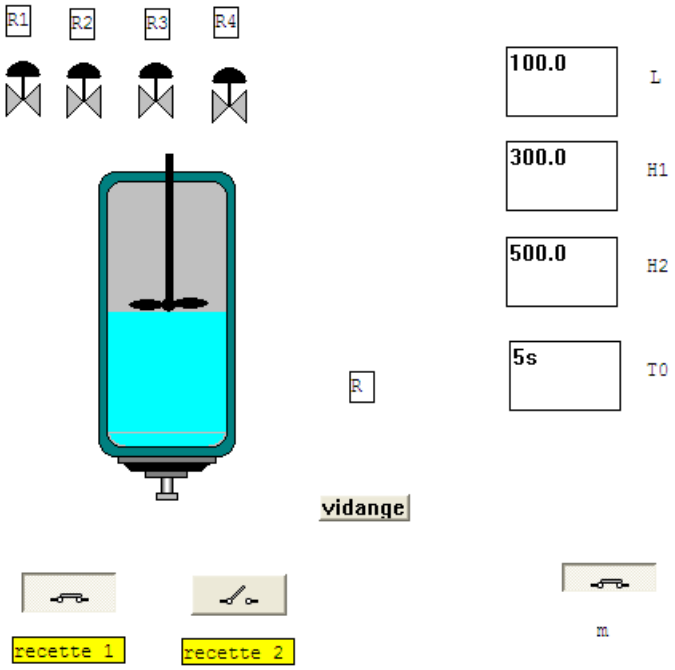


Fig. C.7. 3ème étape de l'application

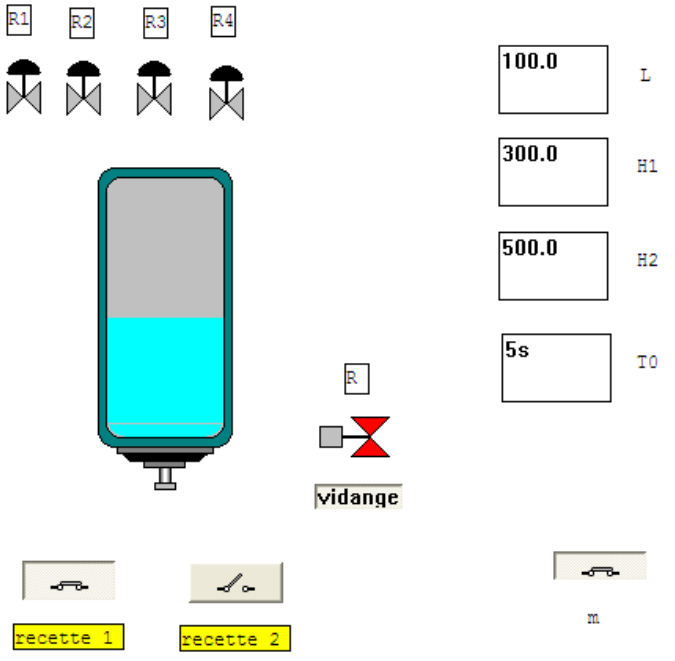


Fig. C.8. 4ème étape de l'application

C.4.4. Programme de l'exemple du Grafcet

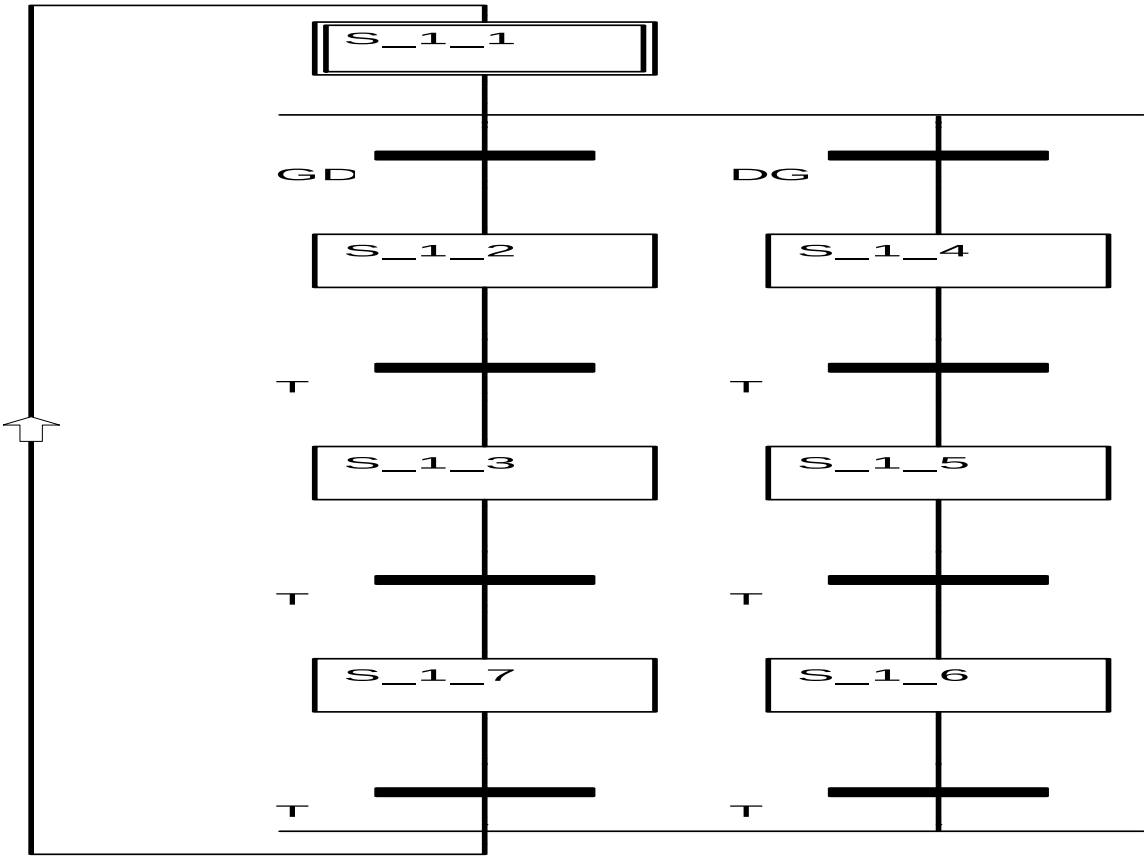


Figure. C.9. Programme exemple grafcet

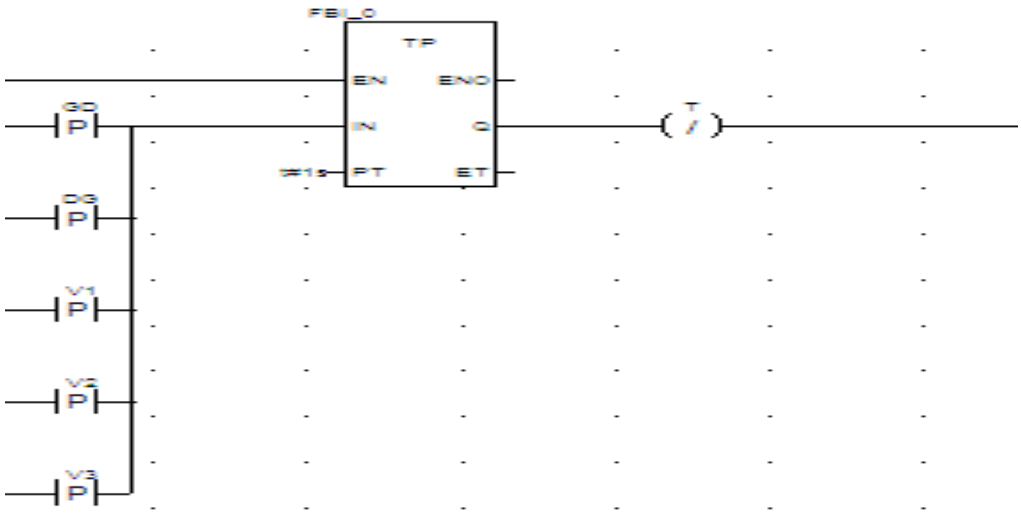


Figure. C.9. Programme de la temporisation

ANNEXE D

VIJEO DESIGNER

D.1. Catalogue des interfaces homme/machine

Machine cible	Abréviation	Série	Modèle
Magelis iPC Series	iPC	Compact iPC*1	Compact MPC KT52 NAX (1024x768)
			Compact MPC KT22 NAX (1024x768)
		Smart iPC*1	Smart MPC ST52 NDJ (1024x768)
			Smart MPC ST21 NAJ (800x600)
Magelis XBTGT Series	XBTGT, XBTGT2000 Series ou plus	XBTGT7000 Series	XBTGT7340 (1024x768)
		XBTGT6000 Series	XBTGT6340 (800x600)
			XBTGT6330 (800x600)
		XBTGT5000 Series	XBTGT5340 (640x480)
			XBTGT5330 (640x480)
			XBTGT5230 (640x480)
		XBTGT4000 Series	XBTGT4340 (640x480)
			XBTGT4330 (640x480)
			XBTGT4230 (640x480)
		XBTGT2000 Series	XBTGT2330 (320x240)
			XBTGT2220 (320x240)
			XBTGT2130 (320x240)
			XBTGT2120 (320x240)
			XBTGT2110 (320x240)
	XBTGT, XBTGT1000 Series	XBTGT1000 Series	XBTGT1100 (320x240)
			XBTGT1130 (320x240)
Magelis XBTG Series	XBTG	XBTG Series	XBTG6330 (800x600)
			XBTG5330 (640x480)
			XBTG5230 (640x480)

			XBTG4330 (640x480)
			XBTG4320 (640x480)
			XBTG2330 (320x240)
			XBTG2220 (320x240)
			XBTG2130 (320x240)
			XBTG2120 (320x240)
			XBTG2110 (320x240)

Tableau D.1. Machines cibles prises en charge :

D.2. Description des objets configurables

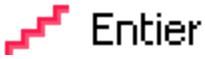
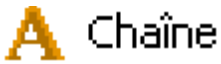
Objet	Opération	Description
Commutateur	Bit	Active et désactive les variables TOR.
	Mot	Ecrit des valeurs de type constante ou des valeurs de variables dans une adresse de mots spécifiée.
	Chaîne	Ecrit une chaîne dans la variable de chaîne définie ou ajoute une chaîne à la variable de chaîne définie.
	Ecran	Précise l'opération de modification d'écran, vers l'écran précédent ou vers un écran existant de la cible.
	Popup	Indique si la fenêtre popup doit se fermer ou s'afficher.
	Alarme	Indique les opérations d'alarmes : activation/désactivation du curseur, défilement vers le haut/bas, page précédente/suivante, accès à une ligne donnée, atteindre l'alarme la

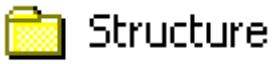
		plus récente et atteindre l'alarme la plus ancienne.
	Son	Définit les opérations de son (lecture ou arrêt du son).
	Langue	Définit les opérations relatives à la langue (modification de la langue du système ou de l'application utilisateur).
	Script	Exécute le script lorsque le commutateur est touché.
	Système	Configure les opérations système telles que le redémarrage ou la fermeture de Runtime, et l'ouverture du menu Configuration.
	Délai	Définit le délai d'exécution d'une opération après une pression. Définit aussi ce délai pour le contrôle des bits.
	Vidéo	Fournit plusieurs opérations sur appui vidéo.
Commutateur radio	Lorsqu'on utilise plusieurs commutateurs radio, seul un commutateur appartenant au même numéro de groupe peut être actif à un moment donné.	
Voyants	Modifie les couleurs ou le texte en fonction de la valeur des variables.	
	Fonctions spéciales	Voyants graphiques, tels qu'une tour ou un voyant rotatif.
	N-Etat	Les voyants à états multiples modifient les couleurs ou le texte selon les valeurs des variables.
Affichage des données	Affichage/saisie numérique	Affiche les valeurs de variables sous forme de valeurs numériques. Vous pouvez aussi saisir des nombres à partir du clavier.
	Affichage/saisie de texte	Affiche le texte stocké dans les variables de type chaîne. Vous pouvez aussi saisir du texte à partir du clavier.

	Affichage de la date	Affiche la date de l'horloge interne de la machine cible.
	Affichage de l'heure	Affiche l'heure de l'horloge interne de la machine cible.
Graphique à Barres	Vertical Horizontal	Représente les valeurs de variables sous forme de graphiques à barres. Le sens des barres peut être vertical ou horizontal.
Cadran	Représente les valeurs de variables sous forme de cadran.	
Affichage de message	Affiche la chaîne de texte associée avec la valeur actuelle d'une variable spécifique.	
Affichage d'image	Affiche l'image associée avec la valeur actuelle d'une variable spécifique.	
Sélecteur	Il s'agit d'un type de commutateur qui affiche l'état actuel d'une variable spécifiée et exécute des opérations lorsqu'il est touché.	
Résumé d'alarme	Affiche la liste des messages d'alarmes et fournit un enregistrement de tous ces messages. Vous pouvez aussi consigner ces messages sur un disque.	
Courbe de tendance	Affiche les valeurs de variables courantes dans la courbe de tendance.	
Affichage vidéo	Avec la fonction Vidéo, vous pouvez : • afficher la vidéo en direct depuis la caméra de la cible. • enregistrer la vidéo en direct depuis l'alimentation vidéo de la caméra cible. • lire la vidéo enregistrée sur la cible ou lire une vidéo ajoutée à la cible. • prendre des instantanés vidéo et les enregistrer dans un fichier ou les imprimer.	

Tableau D.2. Types d'objets configurables:

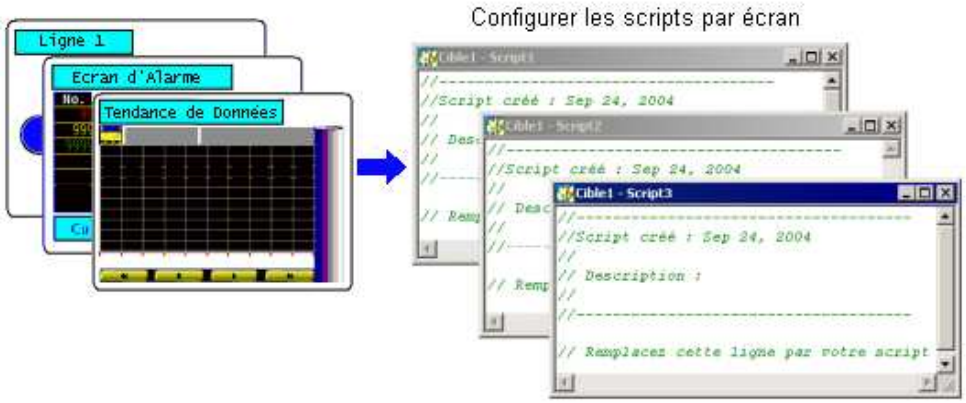
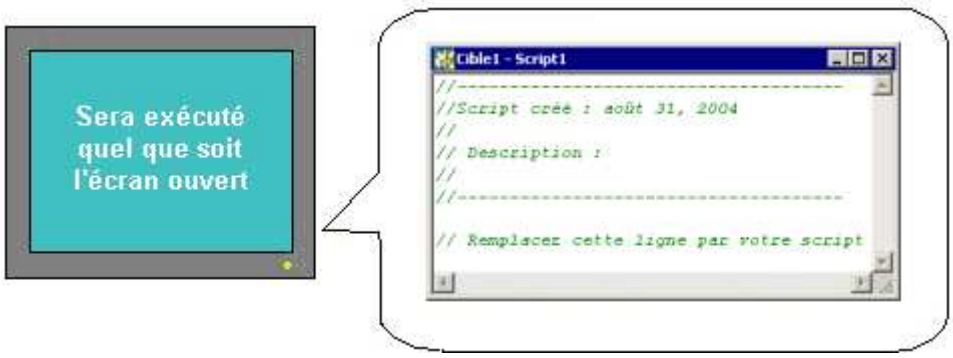
D.3. Types de variables dans Vijeo Designer

TOR 	Stocke une valeur de 0 ou 1.
Entier 	Stocke une valeur comprise entre -2147483648 et +2147483647. La plage de données d'une variable externe de type entier dépend des propriétés suivantes : Format de données, Signé et Longueur des données. Les variables de type entier : • ne stockent pas les décimales ; • s'exécutent plus rapidement que les variables de type flottant.
Flottant 	Stocke une valeur numérique, soit entre -3,4028e38 et +3,4028e38, soit entre -1,1754e-38 et +1,1754e-38. Les variables de type flottant pouvant stocker des valeurs plus importantes que celles du type entier. Utilisez-les pour stocker des calculs, des constantes, etc., lorsque la précision des données n'est pas une priorité. Si la précision des données est nécessaire, utilisez des entiers. Les valeurs de type flottant stockent des valeurs importantes, mais leur précision est limitée à sept chiffres. Par exemple, si l'on prend la valeur 1,2345678, seuls les sept premiers chiffres (1,234567) sont garantis. Après les sept premiers chiffres, les calculs sont effectués de manière approximative. Lors du Runtime, l'exécution d'une variable de type flottant est plus longue que celle d'une variable de type entier.
Chaîne 	Stocke entre 2 et 100 caractères. On peut définir le nombre de caractères stockés dans une chaîne avec la

	propriété Nombre d'octets.
Structure 	Les structures, comme les tableaux, peuvent être considérées comme des dossiers qui stockent plusieurs variables, qui peuvent être de différents types, ce qui les différentie des tableaux. L'organisation des variables en structures vous permet d'en améliorer la gestion. On peut préciser des variables Structure dans la boîte de dialogue Propriétés de l'animation pour les expressions mathématiques ou les scripts. 

Taleau D.3. Types de variables

D.4. Types de scripts

Exécution du script :	Utilisez :
Lorsqu'un écran spécifique est ouvert	Script d'écran Utilisé lorsque on veut exécuter le script uniquement quand l'écran associé est ouvert 
Lorsque l'application est en cours d'exécution	Script d'application Utilisé lorsque on veut exécuter le script tant que l'application est en cours d'exécution. 
Lorsqu'un objet est touché	Script d'appui Si on veut exécuter le script lorsqu'un objet est touché, ou lorsqu'une touche de fonction est touchée. Il existe trois manières de configurer des scripts d'appui : en ajoutant une animation sur appui à un objet et une opération Script, en dessinant un commutateur et en ajoutant une opération Script, ou en ajoutant une opération Script à une touche de fonction.

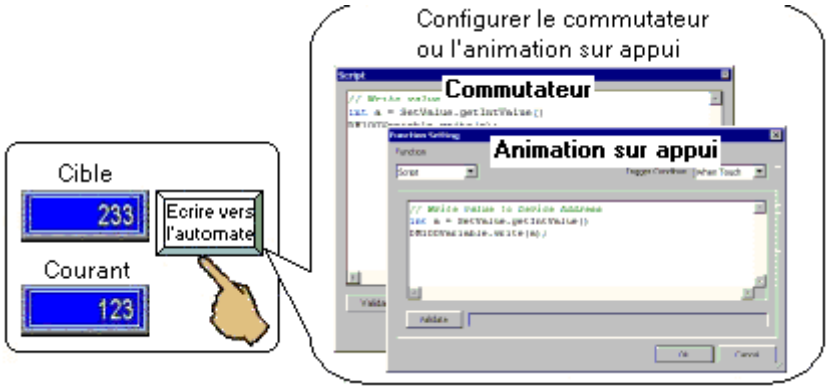
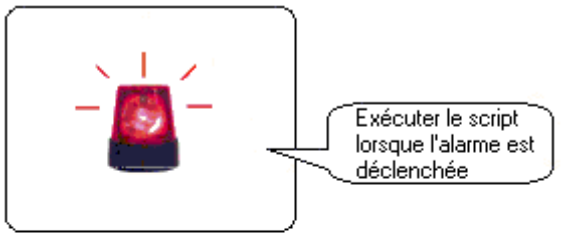
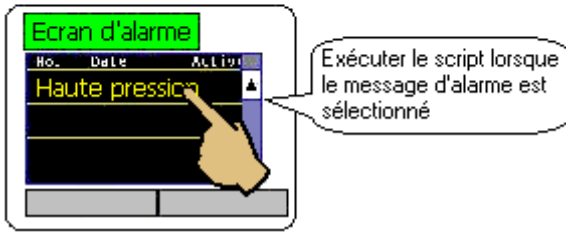
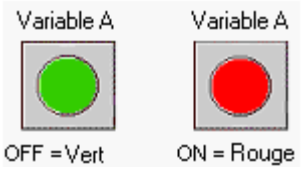
	
Lors du déclenchement d'une alarme	Script d'action Lorsqu'on veut exécuter le script quand une alarme se déclenche, il faut configurer un script d'action dans la variable associée.  

Tableau D.4. Types de scripts

D.5. Types d'animation

Type d'animation	Description
Animation de la couleur Modifier la couleur d'un objet	On peut facilement modifier la couleur d'un objet en changeant la valeur des variables. Cette animation permet, par exemple, de créer un voyant qui change de couleur, comme illustré ci-dessous. 
Animation de remplissage Afficher graphiquement les variations de niveau	On peut remplir progressivement une partie de l'objet avec une couleur en modifiant la valeur des variables. Le remplissage peut être effectué verticalement ou horizontalement. Cette animation permet de créer des graphiques à barres. Elle utilise des variables de type entier et des adresses de périphérique. 
Animation de la taille Afficher les variations de volume	On peut modifier la taille verticale ou horizontale d'un objet en changeant la valeur des variables. Cette animation permet, par exemple, d'afficher graphiquement les variations d'épaisseur de rouleaux, comme illustré ci-dessous. 
Animation Position Déplacer un objet verticalement et horizontalement	On peut déplacer un objet verticalement et horizontalement en modifiant la valeur des variables. Cette animation permet, par exemple, d'afficher graphiquement la position d'un produit sur un tapis roulant, comme illustré ci-dessous.

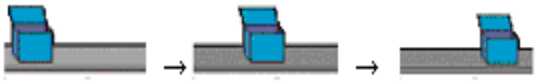
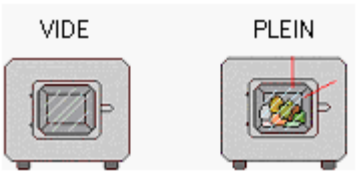
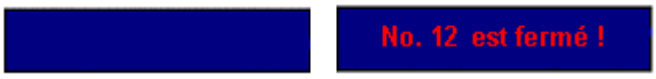
	
Animation Rotation Faire pivoter un objet	O peut faire pivoter un objet selon des angles spécifiques, en modifiant la valeur des variables. Cette animation permet, par exemple, d'afficher les variations de mesure avec l'aiguille d'un cadran (comme illustré ci-dessous) ou de montrer la position de pales d'un ventilateur en mouvement. 
Animation sur appui Utiliser un objet comme un commutateur	On peut créer un objet qui s'active ou se désactive (commutateur) ou qui écrit des données dans une variable. On peut utiliser jusqu'à 32 fonctions d'animation sur appui sur un objet.
Animation de la valeur Afficher ou saisir des données	On peut afficher des valeurs numériques dans l'écran ou autoriser la saisie de données à partir du clavier. Avec une variable TOR, vous pouvez afficher plusieurs messages en fonction des états disponibles. Cette animation permet d'afficher des données numériques, les états des voyants, du texte, mais également d'activer la saisie au clavier.
Animation Visibilité Afficher/masquer des objets	On peut rendre les objets visibles ou invisibles en modifiant la valeur des variables.  On peut également configurer le démarrage ou l'arrêt du clignotement d'un commutateur avec la valeur des variables. 

Tableau D.5 Types d'animation

D.5 Scripts de l'exemple d'application

Script1 : pour le remplissage du réservoir entre le niveau 1 et le niveau 2

```
int tmp=0;
int tmp1;
tmp = tank_level.getIntValue ();
tmp1 = level1.getIntValue ();
if (tmp<tmp1)
{
      tmp++;
tank_level.write ( tmp );

}
```

Script2 : pour le remplissage du réservoir entre les niveaux 2 et maximum

```
int tmp;
int tmp1;
int tmp2;
int tmpm;
int v1;
int v;
int v3;
int pmpp;
tmp = tank_level.getIntValue ();
tmp1 = level1.getIntValue ();
tmp2 = level2.getIntValue ();
tmpm = maximum.getIntValue ();
v1=var1.getIntValue ();
v3=var3.getIntValue ();
v=var.getIntValue ();
pmpp=pmp.getIntValue ();

if (tmp>=tmp1)
{
      pmpp=1;
      if (v==0)
            {
                  tmp++;
      tank_level.write ( tmp );
            }
}
if (tmp==tmp2)
{
      v1=1;
      v3=0;
}



      if (tmp==tmp1)
```

```
{
      v=0;
}
var.write ( v );
var1.write ( v1 );
var3.write ( v3 );
pmp.write ( pmpp );
```

Script 3 :

```
int tmp;
int tmp2;
int tmpm;
int v2;
int v3;
int v;
int v1;
tmp = tank_level.getIntValue ();
tmp2 = level2.getIntValue ();
tmpm = maximum.getIntValue ();
v2=var2.getIntValue ();
v3=var3.getIntValue ();
v=var.getIntValue ();
      if (tmp<=tmpm)
      {
            if (v2==0)
            {
              if (tmp>tmp2)
              {
                tmp--;
                tank_level.write ( tmp );
                v=1;
              }
            }
      }

      if (tmp==tmpm)
      {
            v2=0;
            v3=1;
      }



      var2.write ( v2 );
      var3.write ( v3 );

            var.write ( v );
```

Script 4 :

```
int tmp;
int tmp2;
int tmpm;
int v;
int v1;
int v2;
int v3;
tmp = tank_level.getIntValue ();
tmp2 = level2.getIntValue ();
tmpm = maximum.getIntValue ();
v1=var1.getIntValue ();
v3=var3.getIntValue ();
v=var.getIntValue ();
v2=var2.getIntValue ();

                    if (v3==0)
              {
                    if (v1==1)
                    {

                       if (tmp>=tmp2)
                         {
                            tmp++;
                            tank_level.write ( tmp );
                            v=1;
                            v2=1;
                         }
                    }

              }


       var.write ( v );
       var1.write ( v1 );
       var3.write ( v3 );
       var2.write ( v2 );
```

Script 5:

```
int v2;
int tmp;
int tmp2;
int v4;
v4 = var4.getIntValue ();
v2 = var2.getIntValue ();
tmp = tank_level.getIntValue ();
tmp2 = level2.getIntValue ();

if (tmp<tmp2)
{
      v2=1;
            }
var2.write ( v2 );
```

Script 6 :

```
int tmp;
int v4;
int tmpm;
int v5;
tmpm = maximum.getIntValue ();
v4 = var4.getIntValue ();
tmp = tank_level.getIntValue ();
v5 = var5.getIntValue ();

if (tmp>tmpm)
{
      v4=1;
       v5=2;

            }
      else
      {

         v4=0;
         v5=0;

      }
var4.write ( v4 );
var5.write ( v5 );
```

Figure D.6. Scripts de l'exemple d'application

ANNEXE E

APPLICATIONS

E.1. Programmes Runge-Kutta

E.1.1. Systèmes linéaires d'ordre 1, 2 et 3

```
(***initialisation***)

if %s21 then

y1:=y0;
y2:=dy0;
y3:=d2y0;

end_if;


(*équation différentielle d'ordre 1*)

if n=1 then


(***calcul des Ki***)

K1:=-a1*y-a2+a3*u;
K2:=-a1*(y+K1*h/2.0)-a2+a3*u;
K3:=-a1*(y+K2*h/2.0)-a2+a3*u;
K4:=-a1*(y+K3*h)-a2+a3*u;

(***affectation de la sortie***)


y:=y+(h/6.0)*(K1+2.0*K2+2.0*K3+K4);



(*********************************)

(*équation différentielle d'ordre 2*)

else if n=2 then

(*Calcul des Ki et Qi*)

K1:=y2;
Q1:=-a1*y2-a2*y1-a3+a4*u;

K2:=y2+Q1*h/2.0;
Q2:=-a1*(y2+Q1*h/2.0)-a2*(y1+K1*h/2.0)-a3+a4*u;

K3:=y2+Q2*h/2.0;
Q3:=-a1*(y2+Q2*h/2.0)-a2*(y1+K2*h/2.0)-a3+a4*u;

K4:=y2+Q3*h;
Q4:=-a1*(y2+Q3*h)-a2*(y1+K3*h)-a3+a4*u;
```

```
(* Calcul des yi *)
y1:=y1+h/6.0*(K1+2.0*K2+2.0*K3+K4);
y2:=y2+h/6.0*(Q1+2.0*Q2+2.0*Q3+Q4);

(* Affectation de la sortie *)
y:=y1;

(****************************************)



(*équation différentielle d'ordre *)

else if n=3 then

(* Calcul des Ki, Qi et Vi *)

K1:=y2;
Q1:=y3;
V1:=-a1*y3-a2*y2-a3*y1-a4+a5*u;

K2:=y2+Q1*h/2.0;
Q2:=y3+V1*h/2.0;
V2:=-a1*(y3+V1*h/2.0)-a2*(y2+Q1*h/2.0)-a3*(y1+K1*h/2.0)-a4+a5*u;

K3:=y2+Q2*h/2.0;
Q3:=y3+V2*h/2.0;
V3:=-a1*(y3+V2*h/2.0)-a2*(y2+Q2*h/2.0)-a3*(y1+K2*h/2.0)-a4+a5*u;

K4:=y2+Q3*h;
Q4:=y3+V3*h;
V4:=-a1*(y3+V3*h)-a2*(y2+Q3*h)-a3*(y1+K3*h)-a4+a5*u;

(* Calcul des Xi *)
y1:=y1+h/6.0*(K1+2.0*K2+2.0*K3+K4);
y2:=y2+h/6.0*(Q1+2.0*Q2+2.0*Q3+Q4);
y3:=y3+h/6.0*(V1+2.0*V2+2.0*V3+V4);

(* Affectation de la sortie *)
y:=y1;


end_if;
end_if;
end_if;
```

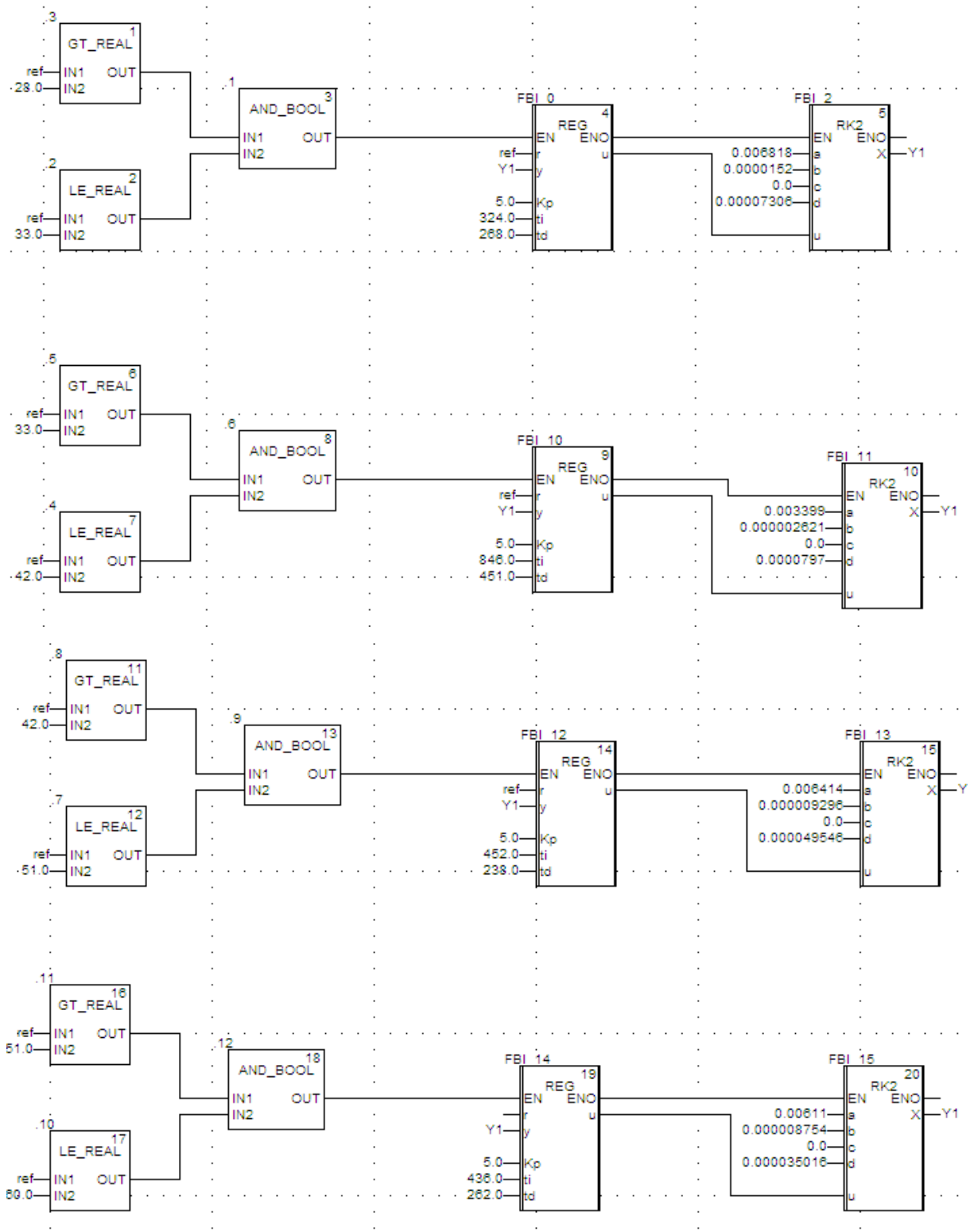
E.1.2. Système non linéaire

```
h:=0.02;

(*calcul des Ki et Vi*)

(*on pose pour simplifier les calculs:*)
t1:=t+h/2.0;
t2:=t+h;
X101:=X10+K1*h/2.0;
X102:=X10+K2*h/2.0;
X103:=X10+K3*h;
X201:=X20+V1*h/2.0;
X202:=X20+V2*h/2.0;
X203:=X20+V3*h;

K1:=X20;
V1:=-3.0*t*X20-3.0*t*t*X10+u*u;

K2:=X201;
V2:=-3.0*t1*X201-3.0*t1*t1*X101+u*u;

K3:=X202;
V3:=-3.0*t1*X202-3.0*t1*t1*X102+u*u;

K4:=X203;
V4:=-3.0*t2*X203-3.0*t2*t2*X103+u*u;


(* Calcul des Xi *)
X1:=X10+h/6.0*(K1+2.0*K2+2.0*K3+K4);
X2:=X20+h/6.0*(V1+2.0*V2+2.0*V3+V4);

X10:=X1;
X20:=X2;
t:=t+h;

(* Affectation de la sortie *)
X:=X1;
```

E.2. Programme de la DFB « PID » :

```
ttt:=ti*td;
tt:=SQRT_REAL (ttt);

h:=0.2;
e:=r-y;
s:=s+e;

P:=Kp*e;
In:=(Kp/ti)*s*h-(Kp*ti/tt)*a;
D:=Kp*td*(e-e0)/h;

u:=P+In+D;

if u>10.0 then
u1:=10.0;
a:=u-u1;
u:=u1;
else if u<-10.0 then
u:=-10.0;
a:=u1-u;
u:=u1;
else
a:=0.0;
end_if;
end_if;
```

E.3. Programme de la régulation du four :



E.4. Programme de la DFB « MCR »

```
(*Initialisations*)

IF %S21 then
 F[1][1]:=1000.0;
 F[2][2]:=1000.0;
 F[3][3]:=1000.0;
 F[4][4]:=1000.0;
 i:=0;
 FOR j:=1 TO 4 DO
  TETA[j]:=2.0;
 END_FOR;

end_if;

(*Stockage des entrées/serties*)

i:=i+1;
Y_vect[i]:=y;
u_vect[i]:=u;


if (i=3) then

                  (*formation du vecteur fi**)


fi[1]:=-y_vect[1];
fi[2]:=-y_vect[2];
fi[3]:=u_vect[1];
fi[4]:=u_vect[2];

             (*********************************************)

              (*Calcul du vecteur epsilone, eps=Y-teta'*fi*)

                  (*     calcul de teta'* fi         *)
y_estim:=teta[1]*fi[1]+teta[2]*fi[2]+teta[3]*fi[3]+teta[4]*fi[4];
                       (*  calcul de epsilon *)

 eps:=y_vect[3]-y_estim;

          (***mise à jour de F, F=F-(F*fi*fi'*F)/(1+fi'*F*fi)***)


                        (****calcul de G1=F*fi***)

for j:=1 to 4 do
G1[j]:=fi[1]*F[j][1]+fi[2]*F[j][2]+fi[3]*F[j][3]+fi[4]*F[j][4];
end_for;
```

```
                              (*    G2=G1*fi'    *)

for j:=1 to 4 do
for j1:=1 to 4 do
G2[j][j1]:=G1[j]*fi[j1];
end_for;
end_for;
                               (*     G3=G2*F    *)
for j:=1 to 4 do
for j1:=1 to 4 do
G3[j][j1]:=F[1][j1]*G2[j][1]+F[2][j1]*G2[j][2]+F[3][j1]*G2[j][3]+F[4][j1]*G2[j
end_for;
end_for;


                  (*calcul du scalaire a=1/(1+fi'*F*fi)    *)

                              (*   G4=fi'*G1   *)
G4:=fi[1]*G1[1]+fi[2]*G1[2]+fi[3]*G1[3]+fi[4]*G1[4];

                              (* calcul de a    *)
 if G4=-1.0 then
  G4:=-1.01;
 end if;

 a:=1.0/(1.0+G4);

                               (* calcul de F1=a*G3   *)

 FOR i2:=1 TO (4) DO
  FOR j2:=1 TO (4) DO

   F1[i2][j2]:=a*G3[i2][j2];

  END_FOR;
 END_FOR;

                               (* calcul de F=F-F1   *)

 FOR i2:=1 TO (4) DO
  FOR j2:=1 TO (4) DO

   F[i2][j2]:=F[i2][j2]-F1[i2][j2];

  END_FOR;
 END_FOR;

           (********mise à jour de teta=teta+F*fi*eps**********)
```

```
                    (****calcul de G11=F*fi***)
for j:=1 to 4 do
G11[j]:=fi[1]*F[j][1]+fi[2]*F[j][2]+fi[3]*F[j][3]+fi[4]*F[j][4];
end_for;


                          (*calcul de teta*)
 FOR i1:=1 TO 4 DO

  TETA[i1]:=TETA[i1]+eps*G11[i1];

 END_FOR;


         (******mise à jour du stockage des entrées/sorties*****)

 FOR j:=1 TO 3 DO
  y_vect[j]:=y_vect[j+1];
  u_vect[j]:=u_vect[j+1];
 END_FOR;

 i:=i-1;
end_if;

TETA1:=TETA[1];
TETA2:=TETA[2];
TETA3:=TETA[3];
TETA4:=TETA[4];
```

E.5. Programme de simulation de la SBPA

```
if %S21 then
X10:=true;
X20:=true;
X30:=true;
X40:=true;
X50:=true;

end_if;


X1:=XOR(X40,X50);
X2:=X10;
X3:=X20;
X4:=X30;
X5:=X40;
```

```
if X5=true then

u:=10.0;

else

u:=0.0;

end_if;

X10:=X1;
X20:=X2;
X30:=X3;
X40:=X4;
X50:=X5;
```

E.6. Programme de simulation d'un système de type ARX d'ordre 2

```
Y_vect[i+1][1]:=y;
U_vect[i+1][1]:=u;

i:=i+1;

if i>1 then

y:=-teta1*y_vect[1][1]-teta2*y_vect[2][1]+teta3*u_vect[1][1]+teta4*u_vec

y_vect[1][1]:=y_vect[2][1];

i:=i-1;
end_if;
```