AUTOMATISME

Auto2 R2.9

I Les systèmes automatisés

II Les outils de description

III Le logiciel Control Expert

INSTITUT UNIVERSITAIRE
DE TECHNOLOGIE
UNIVERSITÉ D'ANGERS

Compétences ciblées

- Concevoir la partie GEII d'un système
- Vérifier la partie GEII d'un système

Apprentissages critiques

- AC11.01 : Produire une analyse fonctionnelle d'un système simple
- AC11.02 : Réaliser un prototype pour des solutions techniques logiciel
- AC11.03 : Rédiger un dossier de fabrication à partir d'un dossier de conception
- AC12.01 : Appliquer une procédure d'essais
- AC12.02 : Identifier un dysfonctionnement
- AC12.03 : Décrire un dysfonctionnement

Descriptif

Acquis d'apprentissage visés

• Enumérer les principes de base d'un automate (C1-N1-AC1) ;
• Décrire le fonctionnement des pré-actionneurs et actionneurs (électriques, pneumatiques, hydrauliques…) (C1-N1-AC1, C1-N1-AC2) ;
• Identifier les parties commande, opérative et supervision d'un système automatise (C1-N1-AC1, C1-N1-AC2) ;
• Programmer un automate avec un langage simple (C1-N1-AC2) ;
• Lire un schéma de câblage et expliquer l'intégration de l'automate dans celui-ci (C1-N1-AC1, C1-N1-AC2).
• Analyser une architecture d'un système automatise (C1-N1-AC1) ;
• Développer la partie commande d'un système automatise à partir d'une unité de traitement en utilisant un langage approprie (C1-N1-AC2, C1-N1-AC3) ;
• Réaliser l'interfaçage et le branchement des entrées-sorties de la partie opérative d'un système automatise (C1-N1-AC2, C1-N1-AC3) ;
• Vérifier le fonctionnement d'un système automatise simple (C2-N1) ;
• Proposer des modifications simples de programme pour respecter un cahier des charges (C2-N1-AC2, C2- N1-AC3).

Pour rappel

SEMESTRE 2

			type de B.U.T.			tertiaire	SAÉ			Ressources										
UE	Compétence	Niveau de la	Composantes essentielles			Apprentissages critiques	SAÉ 2.1	SAÉ 2.2	Portfolio	R2.01 Anglais	R2.02 Culture et Com.	R2.03 Vie de l'entreprise	R2.04 Outils Mathématiques et Logiciels	R2.05 PPP	R2.06 Automatisme	R2.07 Informatique	R2.08 Electronique	R2.09 Energie	R2.10 Phys app	
UE 2.1	Concevoir la partie GEII d'un système	niveau 1 de la compétence	En adoptant une approche holistique intégrant les innovations technologiques en lien avec la	En produisant l'ensemble des documents nécessaires pour le client et les différents prestataires	En communiquant de façon adaptée avec les différents acteurs avant et pendant la phase de	Produire une analyse fonctionnelle d'un système simple	X		X	X	X	X	X	X	X	X	X	X	X	
						Réaliser un prototype pour des solutions techniques matériel et/ou logiciel	X		X	X	X	X	X	X	X	X	X	X	X	
						Rédiger un dossier de fabrication à partir d'un dossier de conception	X		X	X	X		X	X	X		X		X	
						Coefficients	5		1	0,5	0,5	0,5	1,3	0,5	1,3	1,3	1,3	1,3	0,5	15,00
UE 2.2	Vérifier la partie GEII d'un système	Niveau 1 de la compétence	En tenant compte des spécificités matérielles, réglementaires et	En mettant en oeuvre un plan d'essais et d'évaluations, dans une	En tenant compte des enjeux économiques, environnementaux et	Appliquer une procédure d'essais		X	X	X	X		X	X	X	X	X		X	
						Identifier un dysfonctionnement		X	X	X	X		X	X	X	X	X	X	X	
						Décrire les effets d'un dysfonctionnement		X	X	X	X		X	X	X	X	X	X	X	
						Coefficients		5	1	0,5	0,5	0,5	1,3	0,5	1,3	1,3	1,3	1,3	0,5	15,00
																		Somme coefficients UE		30

I Les systèmes automatisés

L'automatisme consiste à réaliser l'étude de la commande d'un système industriel plus ou moins complexe.

Les systèmes pilotés par des automatismes sont implantés un peu partout, les principaux domaines :

- L'agroalimentaire
- L'automobile
- L'aéronautique
- Le traitement de l'eau
- La surveillance des autoroutes
- La surveillance et le contrôle des bâtiments
- …

L'intérêt d'un automate est qu'il est capable de récupérer un nombre important de données en entrée **capteurs**, pour piloter un nombre important de sorties **actionneurs.**

Les technologies ayant beaucoup évoluées ces dernières années, les temps de traitement, temps de réponse, capacité de stockage de données, sécurité des commandes font que l' automate - en tant que composant d'automatisme ou sous forme de logiciel dans un ordinateur industriel - peut aujourd'hui être employé pour des traitements plus complexes, plus rapides, si besoin temps réel.

L'automatisme répond parfaitement aux besoins de l'industrie 4.0. Les automates communiquent avec des systèmes industriels complexes : caméra, capteur intelligent, robot industriel ou collaboratif et permettent l'échange d'informations avec des logiciels de supervision, maintenance, production, simulation,…

1.1 Historique

L'origine des automatismes est très ancienne, déjà les égyptiens fabriquaient des statuettes articulées.

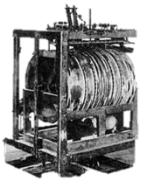


Figure 1: le fluteur

En 1737 le « fluteur » pouvait jouer 11 airs de flute différents.

Figure 2: canard de Vaucanson

En 1738, le canard de Vaucanson pouvait manger, boire et cancaner comme un vrai canard.

Une des premières machines «industrielles » automatisées fut le métier à tisser de Jacquard (1801).

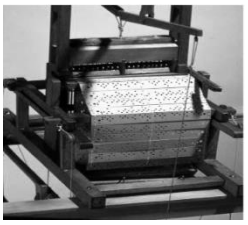


Figure 3 : métier à tisser de Jacquard

En 1950, le premier robot manipulateur industriel fut créé par Georges Devol. Il sera ensuite utilisé sur les chaines de production General Motors, puis adapté pour des besoins dans le nucléaire.

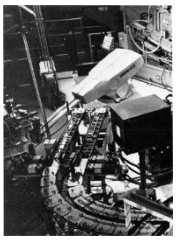


Figure 4: Robot de Devol

Dans les années 1980-1990, l'architecture de l'automate peut commander plusieurs machines à la fois et ses capacités de traitement augmentent avec la multiplication des entrées/sorties.

Dans les années 2000, on diminue le nombre des entrées/sorties intégrées, que l'on déporte au plus près des capteurs et actionneurs. L'évolution du réseau d'échange de données (Ethernet), prend la forme d'une architecture décentralisée avec des automates plus petits.

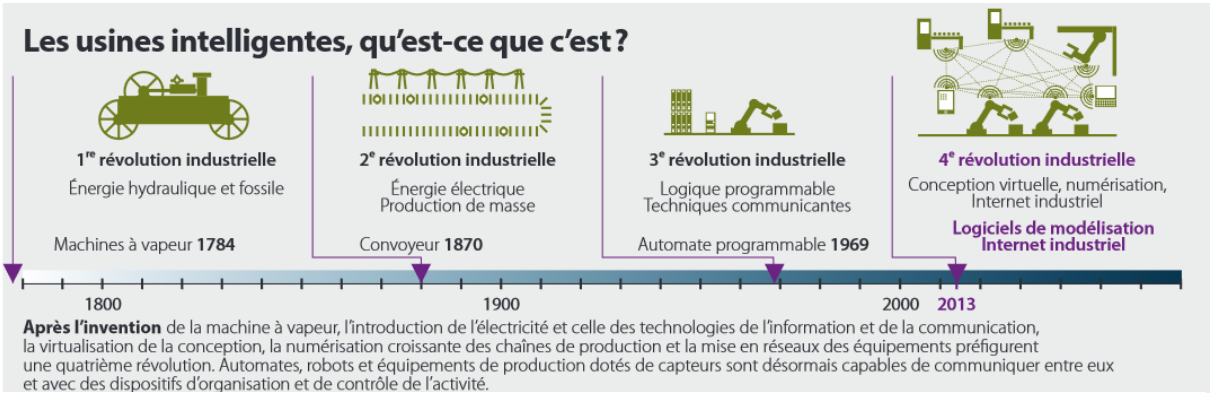


Figure 5: la 4ème révolution (*technologie 193*)

Depuis 2013 de nombreux fabricants de solutions d'automatisme travaillent sur l'usine du futur en développant des outils logiciels grâce auxquels l'industriel peut, dans le cas idéal, programmer tout à la fois son automate, définir ses interfaces homme-machine, configurer ses variateurs de fréquences, piloter ses commandes d'axes…

1.2 Les automates programmables

Un Automate Programmable Industriel (API) est un dispositif électronique programmable. Il envoie des ordres vers des pré actionneurs à partir de données d'entrées (capteurs), de consignes et d'un programme informatique pour commander les systèmes industriels de façon séquentielle.

Figure 6: machine pack'r remplissage de bidon 1l

Tout système automatisé COMPORTE :

- Une **PARTIE OPERATIVE** (P.O.) : Elle agit sur la matière d'œuvre afin de lui donner sa valeur ajoutée.
- Une **PARTIE COMMANDE** (P.C.) : Elle donne les ordres de fonctionnement à la partie opérative.

1.2.1 Constitution

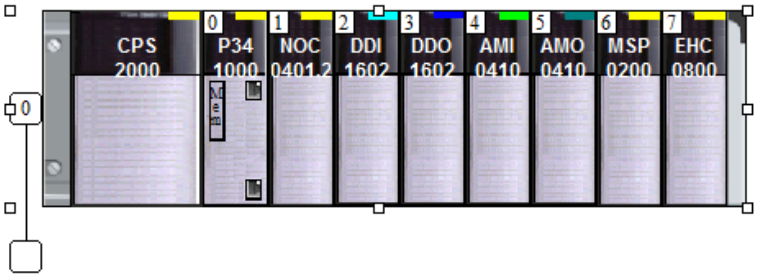
L'API est structuré autour d'un processeur ① (en anglais Central Processing Unit, CPU), d'une alimentation ②(depuis des tensions AC ou DC) et, de modules suivant les besoins de l'application, tel que:

- Des cartes d'entrées - sorties (en anglais Input - Output, I/O) numériques (Tout ou rien) ou analogiques
  - Cartes d'entrées pour brancher des capteurs, boutons poussoirs, ... ③
  - Cartes de sorties pour brancher des actionneurs, voyants, vannes, ...④

- Des modules de communication ⑤ Ethernet , Modbus, Modbus Plus, Profibus, RS-485, AS-i, CANopen, pour dialoguer avec d'autres automates, des entrées/sorties déportées, des supervisions, des robots, des caméras, ou autres interfaces homme-machine (IHM, en anglais Human Machine Interface, HMI)), ...

- Des modules dédiés métiers⑥, tels que le comptage rapide, le pesage...

- Des modules d'interface pour la commande de mouvement⑦, dits modules Motion, tels que démarreurs progressifs, variateurs de vitesse, commande d'axes.



Schneider rack 8 positions

1.2.2 Cycle ETS

Tous les automates fonctionnent selon le même mode opératoire :

• **Traitement interne** : L'automate réalise des opérations de paramétrage et de contrôle et met à jour certains paramètres systèmes, il réinitialise son chien de garde.

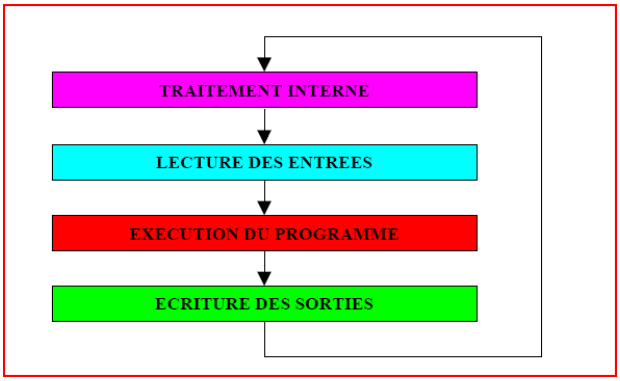


Figure 8: Cycle de l'automate

• **Lecture des entrées** : L'automate lit les entrées (de façon synchrone) et les recopie dans la mémoire image des entrées **MI**.

• **Exécution du programme** : L'automate exécute le programme instruction par instruction dans l'ordre et écrit les sorties dans la mémoire image des sorties **MQ**.

• **Ecriture des sorties** : L'automate recopie la table image des sorties **MQ** sur les sorties physiques. Toutes les sorties basculent en même temps !

Ces quatre opérations sont effectuées continuellement par l'automate (fonctionnement cyclique).
On appelle scrutation l'ensemble des quatre opérations réalisées par l'automate et le temps de scrutation est le temps mis par l'automate pour traiter la même partie de programme. Ce temps est de l'ordre de la dizaine de millisecondes pour les applications standards.

Le temps de réponse total (TRT) est le temps qui s'écoule entre le changement d'état d'une entrée et le changement d'état de la sortie correspondante :

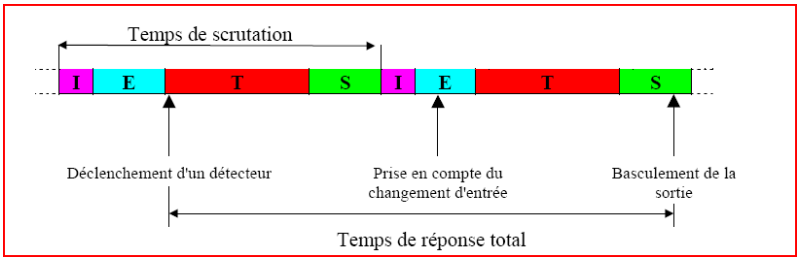


Figure 9: Temps de scrutation

Le temps de réponse total est au plus égal à deux fois le temps de scrutation.
Le temps de scrutation est directement lié au programme implanté. Ce temps peut être fixé à une valeur précise (fonctionnement périodique), le système indiquera alors tout dépassement de période.

La plupart des automates offrent deux types de structure logicielle :

- ✓ Une structure monotâche :

Le traitement se fait de la façon décrite page précédente. Le programme n'est alors lié qu'à une seule tâche : la tâche maître.

- ✓ Une structure multitâche :

A la tâche précédente peut être rajouté deux autres tâches : la tâche rapide et la tâche événementielle.

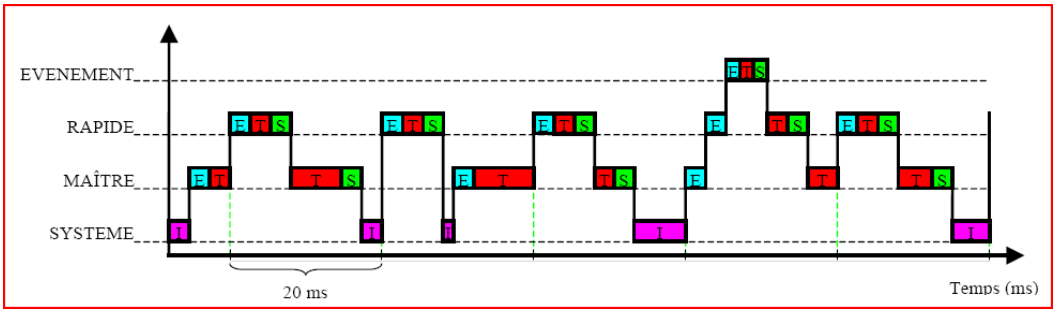


Figure 10: Structure multitâche

1.3 Les langages de programmation de l'automatisme

Les API doivent pouvoir être programmés facilement. Ceci a conduit les constructeurs à concevoir des langages d'application spécialement adaptés à la réalisation d'automatisme. Dans la norme CEI-61131-3 on distingue 5 langages:

- *Les langages graphiques :*
  - **LD** : .....
  - **FBD**: .....
  - **SFC**: .....

- *Les langages textuels :*
  - **IL** : .....
  - **ST**: .....

La plupart des automates industriels peuvent être programmés avec ces 5 langages. Plusieurs langages peuvent être utilisés au sein du même projet.

II Les outils de description

Il existe différents outils d'analyse permettant la mise en place d'un système automatisé.

2.1 Schéma Bloc Fonctionnel*:*

L'outil «maison» pour le bilan des entrées/sorties et l'analyse de leurs commandes.

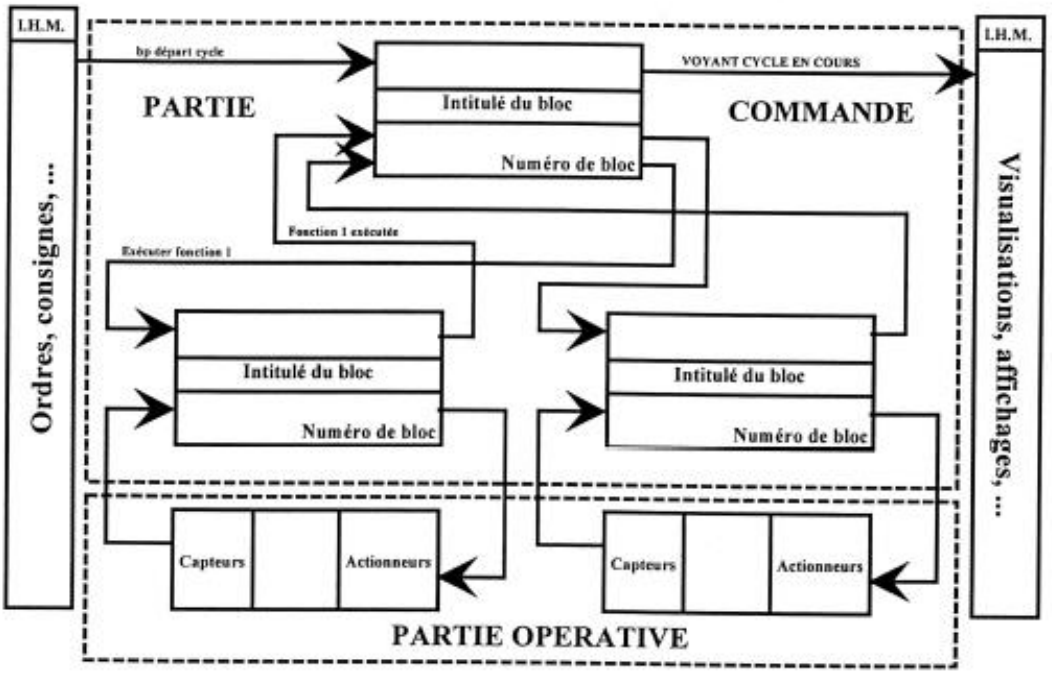
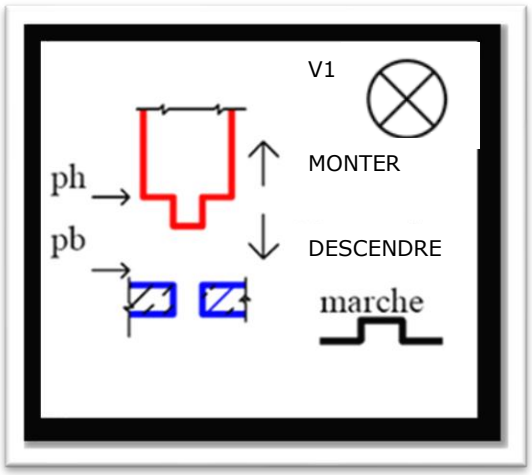
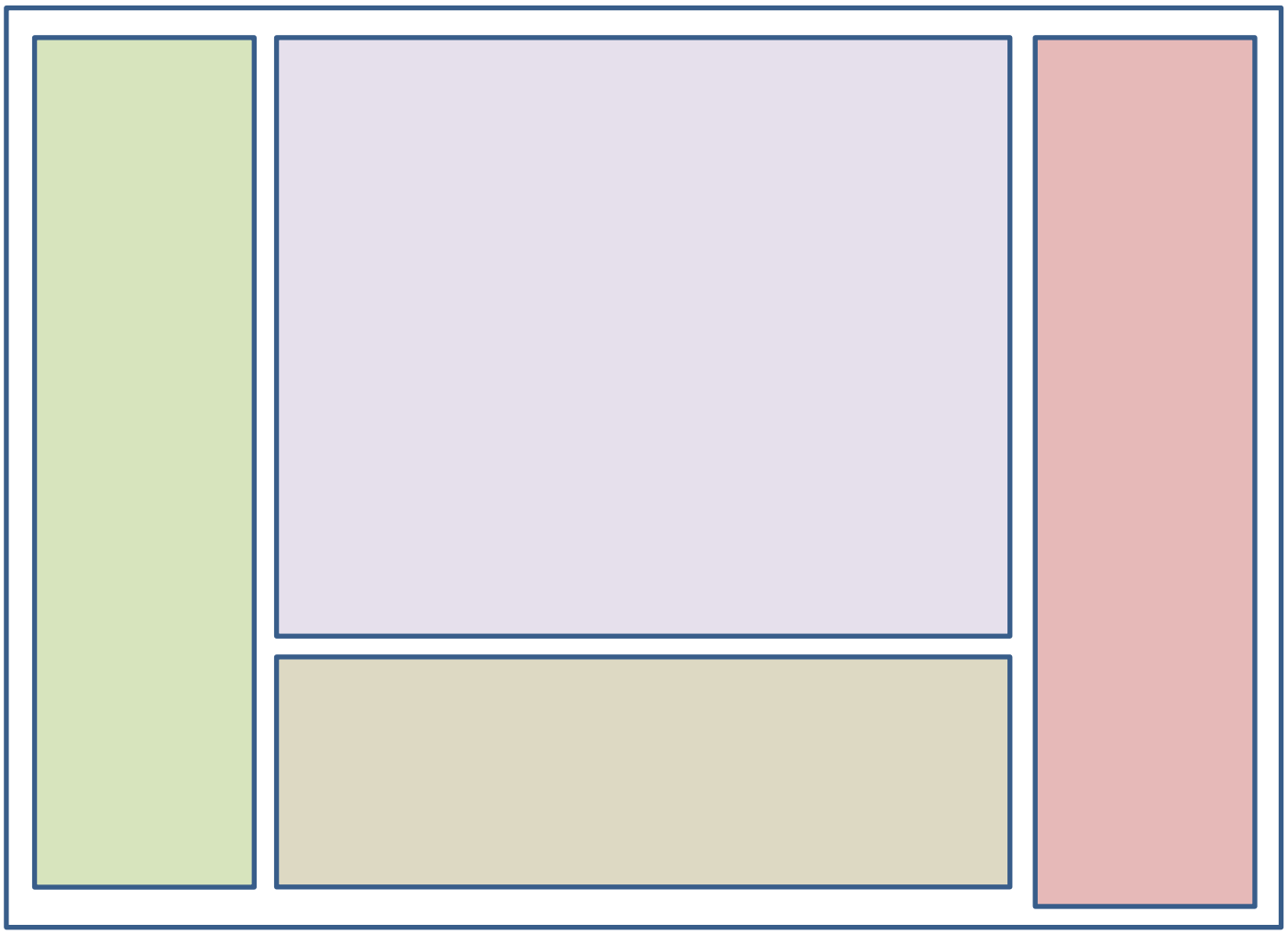


Figure 11: Schéma Bloc Fonctionnel « Maison »

Exemple de schéma bloc fonctionnel :

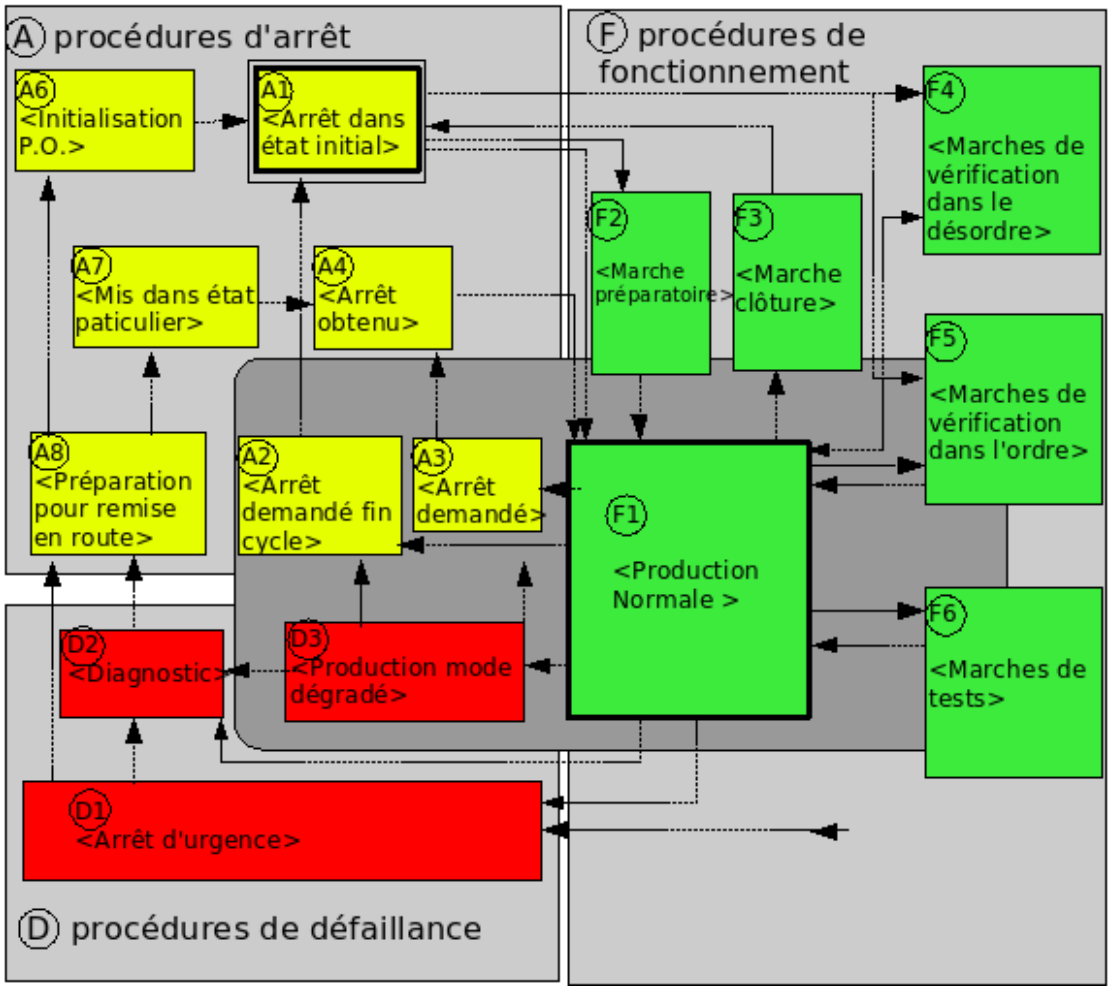


Entrées IHM :marche
Capteurs : ph, pb
Sorties IHM : V1
Actionneurs : MONTER, DESCENDRE



2.2 Le GEMMA Guide d'étude des modes de marche et d'arrêt

états de fonctionnement: F1 F2 F3 F4 F5 F6
états d'arrêt: A1 A2 A3 A4 A5 A6 A7
états de défaillance: D1 D2 D3



2.3 Le Grafcet

Ce diagramme fonctionnel, le GRAFCET (**Graphe Fonctionnel de Commande Etapes Transitions**) permet de décrire les comportements attendus de l'automatisme en imposant une démarche rigoureuse, évitant ainsi les incohérences dans le fonctionnement.

Le GRAFCET se compose d'un ensemble :

- d'**étapes** auxquelles sont associées des actions,
- de **transitions** auxquelles sont associées des réceptivités,
- de **liaisons orientées** reliant les transitions aux étapes et inversement

2.3.1 Les étapes

L'étape est représentée par un carré.

Symbole complet :

.

L'étape initiale :

Lors du déroulement du processus, les étapes sont activées les unes après les autres. Parmi les étapes, certaines sont initialement activées au début du fonctionnement, on les appelle les **étapes initiales**

2.3.2 Les actions associées aux étapes :

A chaque étape peuvent être associées une ou plusieurs actions. Elles traduisent ce qui doit être fait ou commandé à chaque fois

Une seule action
Plusieurs actions

Remarque :
Une étape peut ne comporter aucune action. C'est souvent le cas des étapes initiales. Il peut aussi s'agir d'étape d'attente

2.3.3 Les transitions

La transition indique la possibilité d'évolution du cycle. Elle est soit validée, soit non validée. Une transition entre deux étapes se représente par une barre perpendiculaire aux liaisons orientées.

2.3.4 Réceptivité associée aux étapes

2.3.5 Liaison orientée

Une liaison orientée relie toujours une étape à une transition et inversement. Les liaisons du haut vers le bas ne comportent pas de flèches. Dans le cas contraire, il faut en utiliser.

2.3.6 Règles de syntaxe

L'alternance étape-transition et transition-étape doit toujours être respectée quelle que soit la séquence parcourue.

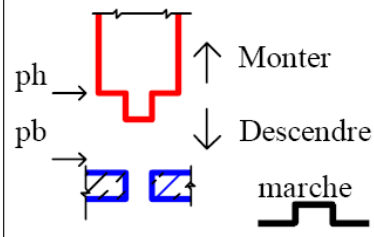


Figure 1 Poinçonneuse

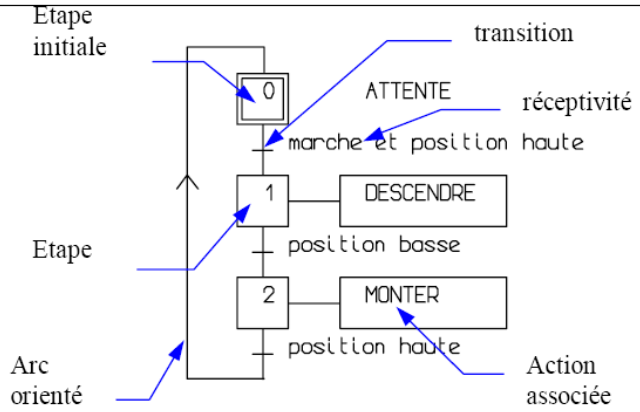


Figure 2 Grafcet de la poinçonneuse

2.3.7 Les actions

Les actions continues

- Type N Une action continue ou «normale» ou «non mémorisée» n'est exécutée que si l'étape n associée est active

- Type D (Delay)

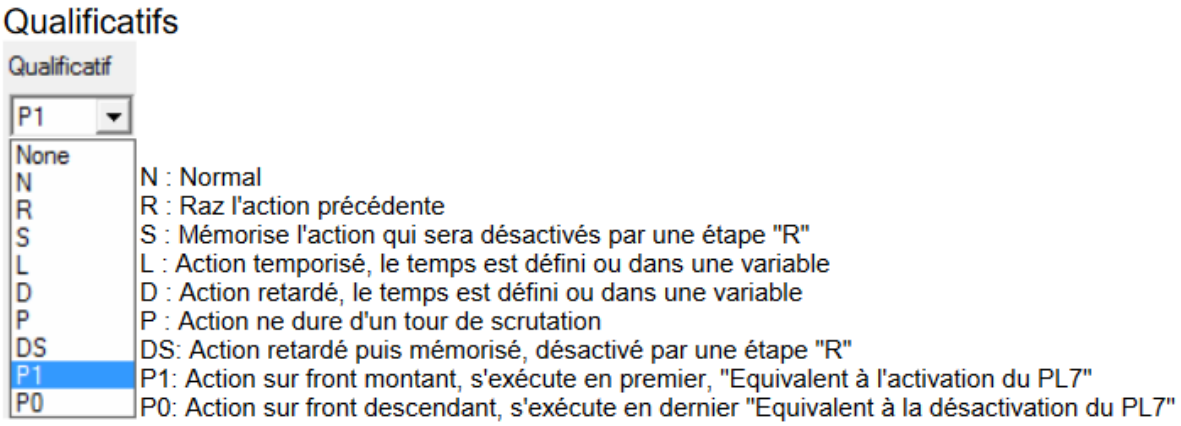
- Type L (Limited) :

- Les actions mémorisées

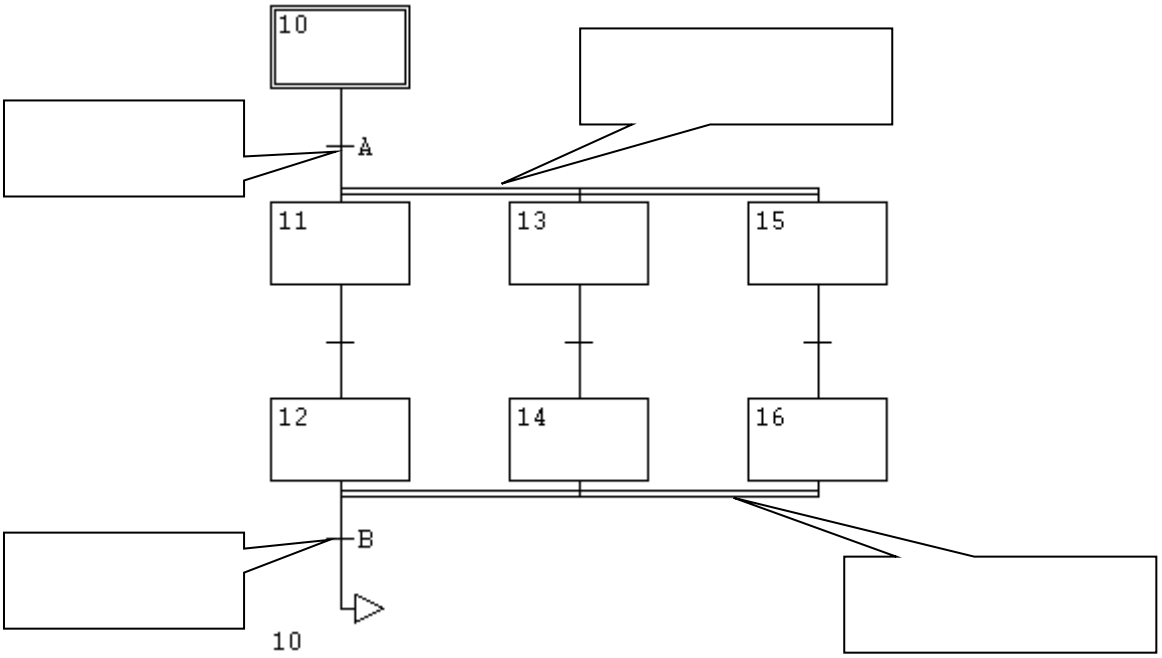
Une étape à action mémorisée permet de mettre la sortie correspondante dans un état spécifié lors de son activation. Sa désactivation ne remet pas la sortie associée à son état d'origine : le passage dans un autre état de cette sortie devra être décrit explicitement par une autre étape.

- Qualificatifs d'action

Selon le logiciel utilisé il peut exister d'autres types d'action :



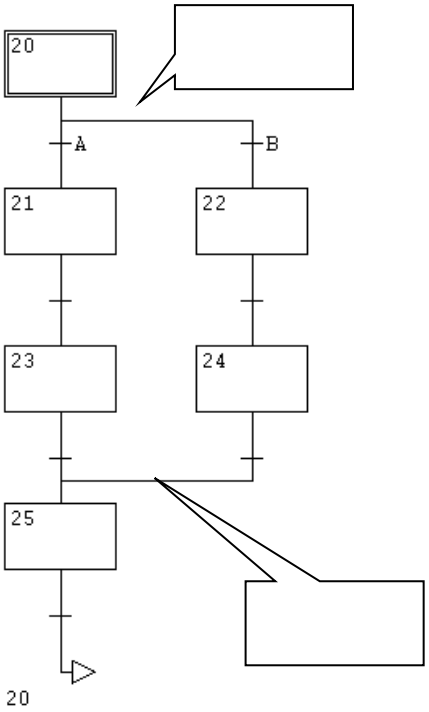
2.3.8 Divergence et convergence en ET (séquences simultanées)



Divergence en ET : lorsque la transition A est franchie, les étapes 11, 13 et 15 sont actives.
Convergence en ET : la transition B sera validée lorsque les étapes 12, 14 et 16 seront actives. Si la réceptivité associée à cette transition est vraie, alors celle-ci est franchie.

REMARQUES :
Après une divergence en ET, on trouve une convergence en ET.

Le nombre de branches parallèles peut-être supérieur à 2.
La réceptivité associée à la convergence peut-être de la forme = 1. Dans ce cas la transition est franchie dès qu'elle est active.



2.3.9 Divergence et convergence en OU (aiguillage)

Divergence en OU : l'évolution du système vers une branche dépend des réceptivités A et B associées aux transitions.
Convergence en OU : après l'évolution dans une branche, il y a convergence vers une étape commune.

REMARQUES:
A et B ne peuvent être vrais simultanément (conflit).
Après une divergence en OU, on trouve une convergence en OU.
Le nombre de branches peut-être supérieur à 2.
La convergence de toutes les branches ne se fait pas obligatoirement au même endroit.

2.3.10 Les règles d'évolution :

Le Grafcet fonctionne en suivant 5 règles d'évolution :

Règle N°1 :

L'initialisation précise l'étape ou les étapes actives au début du fonctionnement. On la repère en doublant les côtés des symboles correspondants. Il peut y avoir plusieurs étapes initiales dans un grafcet.
Remarque : Les étapes initiales sont activées inconditionnellement en début de cycle.

Règle N°2 :

Une transition est soit validée, soit non validée. Elle est validée lorsque toutes les étapes immédiatement précédentes sont actives.
Elle ne peut être franchie que :
Lorsqu'elle est validée, et que la réceptivité associée à la transition est vraie.

Règle N°3 :

Le franchissement d'une transition entraîne l'activation **simultanée** de toutes les étapes immédiatement suivantes et la désactivation de toutes les étapes immédiatement précédentes.

Règle N°4 :

Plusieurs transitions simultanément franchissables sont simultanément franchies.

Règle N°5 :

Si au cours du fonctionnement, une même étape doit être activée et désactivée simultanément, elle reste active.
Remarque : la durée de franchissement d'une transition ne peut jamais être rigoureusement nulle, même si d'après les règles 3 et 4, elle peut être rendue aussi petite que possible. Il en est de même de la durée d'activation d'une étape

2.3.11 Illustration des règles du grafcet

Règle n°1 : **Situation initiale**

Règle n°2 : **Franchissement d'une transition**

Règle n°3 : **Evolution des étapes actives**

Règle n°4 : **Evolution simultanée**

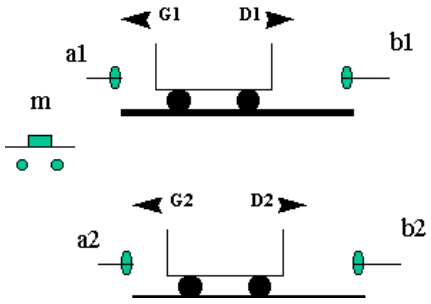
Règle n° 5 : **Activation et désactivation simultanée d'une étape**

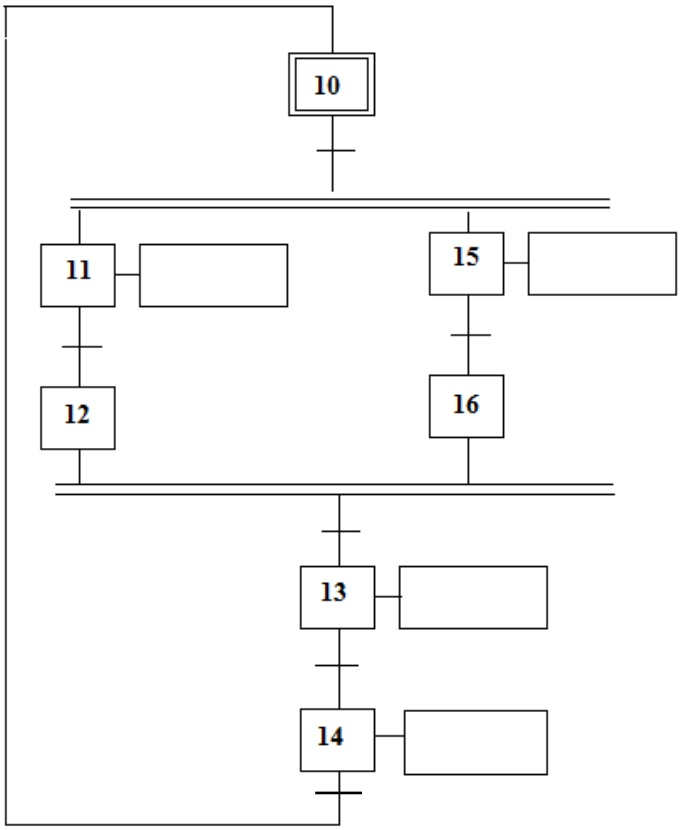
2.3.12 Exemple

Chariot

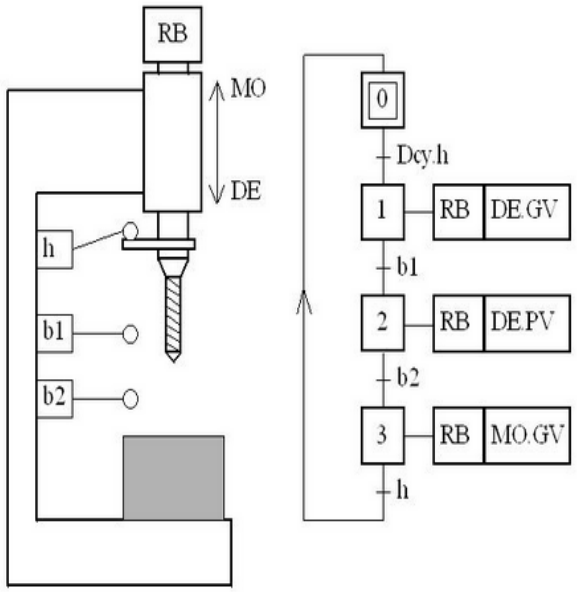
Cahier des charges :

- L'appuie sur « m » commande le déplacement des chariots vers la droite (D1 et D2) jusque b1 et b2.
- Quand les deux chariots sont arrivés alors le chariot 1 repart vers la gauche jusque a1
- Puis le chariot 2 repart vers la gauche jusque a2.
- Lorsque l'on appuie sur « m » le cycle recommence





2.3.13 Chronogrammes d'un système séquentiel



Dcy
h
b1
b2
%X0
%X1
%X2
%X3
MO
DE
GV
PV
RB

III Le logiciel CONTROL EXPERT

Control Expert est intégré dans la suite Ecostruxure, ce logiciel est la suite du logiciel UNITY. C'est un logiciel professionnel de Schneider permettant le développement de programmes et l'exploitation des automatismes Modicon : M340, M580.

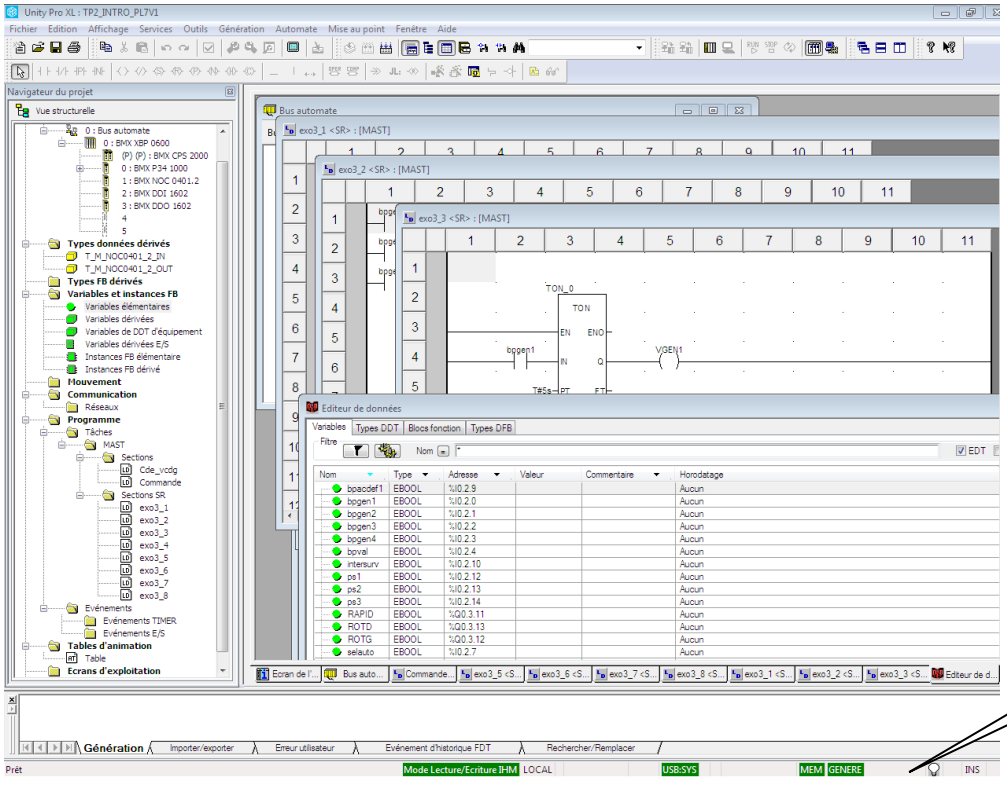
Control Expert possède :

- 5 langages de programmation IEC61131-3 :
  - Langage à Blocs Fonctions (FBD :Function Bloc Diagram)
  - Langage Ladder (LD)
  - Langage List (IL)
  - Langage Littéral Structuré (ST)
  - Diagramme Fonctionnel en Séquence (SFC : Sequential Fonctionnal Chart, grafcet)
- Librairies de Blocs fonctions (DFB) personnalisables.
- Simulateur sur PC avant la mise en service.
- Test et diagnostic intégrés.

Ergonomie du logiciel

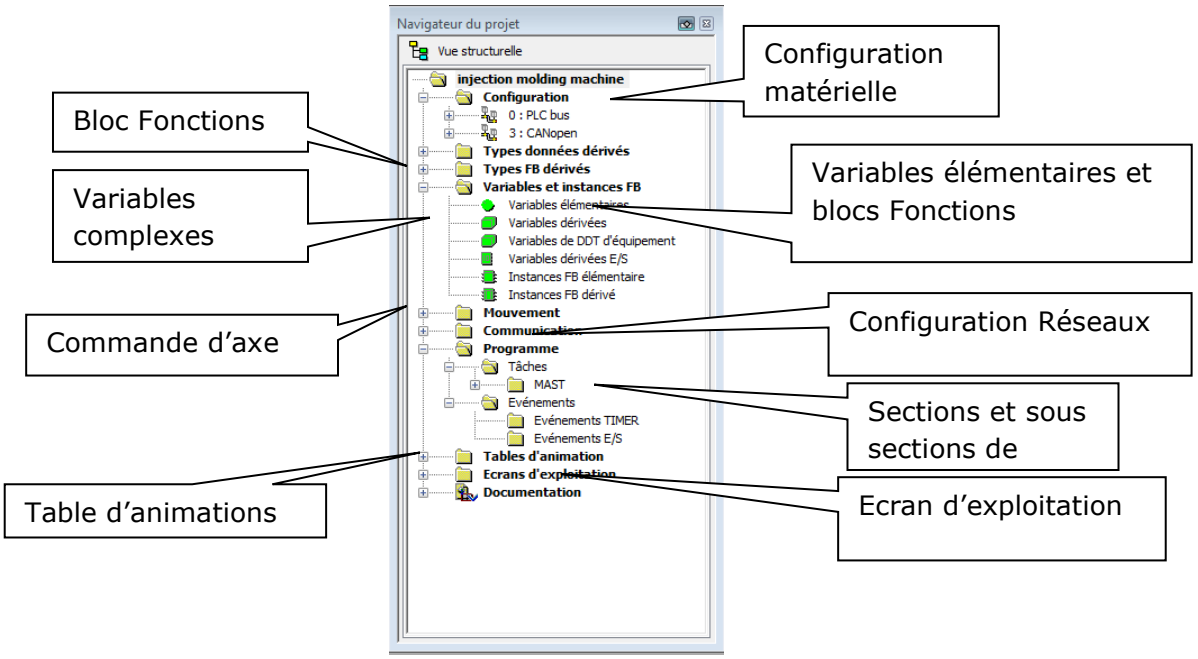
A l'ouverture du logiciel, une fenêtre principale s'ouvre contenant :

- La barre d'outils
- La zone projet
- La fenêtre de visualisation
- La barre d'état



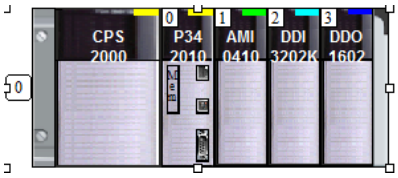
3.1 Navigateur de projet

C'est l'éditeur qui permet de naviguer à l'intérieur du projet :

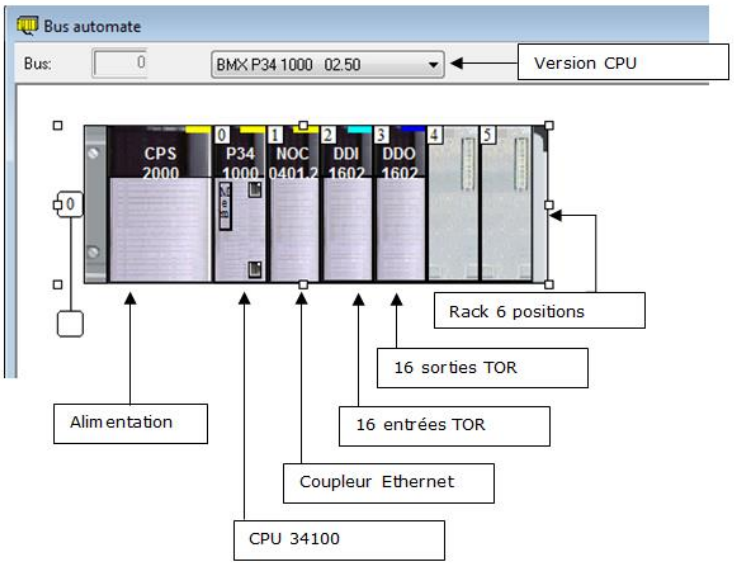


3.2 Configuration matérielle :

La configuration du matériel utilisée dans l'application se fait en double cliquant sur les emplacements du rack



Le premier module, l'alimentation, est toujours placé par défaut et au 1er emplacement.
Il faut ensuite renseigner les autres cartes à chaque emplacement.

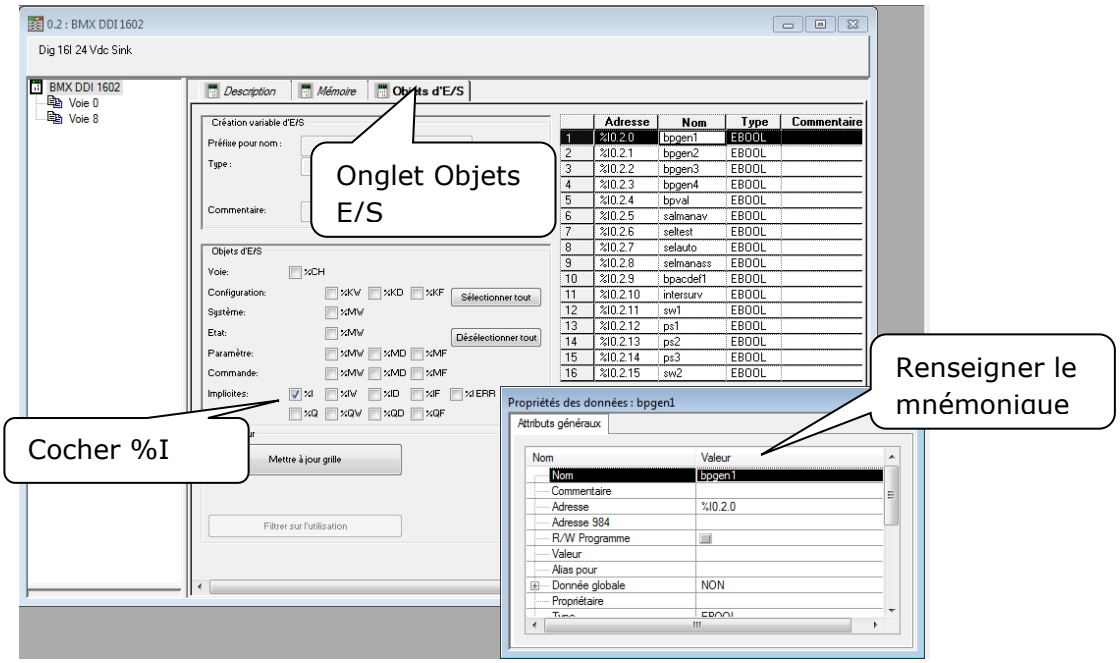


3.3 Les variables

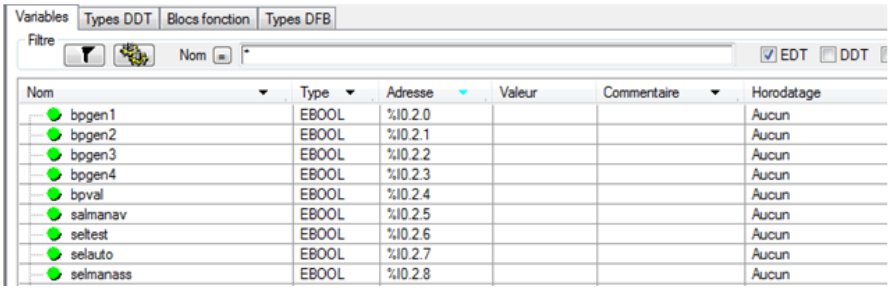
3.3.1 Les variables d'entrées sorties

La déclaration des mnémoniques peut se faire à partir du menu **configuration matérielle** ou dans le **menu variable** :

Menu configuration matérielle



3.3.2 Menu Variable



Les entrées sorties TOR utilisées sur nos maquettes sont des variables élémentaires de type EBOOL.

3.3.3 Les adresses

Une entrée va s'écrire par exemple %I 0.2.0

- %I : Variable Modicon
- I : Entrée
- 0 : Numéro du rack
- 2 : Numéro de l'emplacement
- 0 : Numéro de la voie

3.3.4 Les variables élémentaires

Les types de **variables élémentaires** (**EDT**, Elementary Data Type) définis par la norme IEC 61131-3

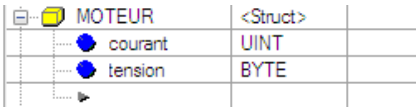
Nom du type	Description	Format (bits)	Valeur par défaut
BOOL	Booléen	1	0 (=False)
EBOOL	Booléen avec détection de fronts et forçage	3	0 (=False)
INT	Entier	16	0
DINT	Entier double	32	0
UINT	Entier non-signé	16	0
UDINT	Entier double non signé	32	0
TIME	Entier double non signé représentant une durée	32	T=#0s
DATE	Date	32	D#1990-01-01
TOD	Heure du jour	32	TOD#00:00:00
DT	Date et heure	64	DT#1990-01-01-00:00:00
REAL	Nombre réel	32	0.0
STRING	Chaîne de caractère	16	
BYTE	Octet	8	0
WORD	Mot	16	0
DWORD	Mot double	32	0

3.3.5 Les variables dérivées

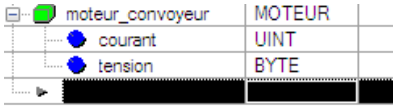
Les variables dérivées sont des variables créées à partir de types de données dérivées.

- Les types de **données dérivées** (**DDT**, Derived Data Type) : ces types sont créés par l'utilisateur. Il existe deux sortes de DDT :
  - les tableaux qui sont des regroupements de variables de même type,
  - les structures qui sont des regroupements de variables de types différents.

Ex : on souhaite utiliser une variable contenant deux champs distincts :



Création du type Moteur et des champs courant et tension



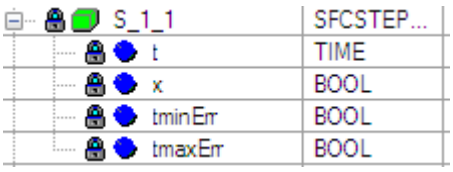
Création de la variable moteur_convoyeur de type MOTEUR

La variable pourra être modifiée en renseignant les champs :
moteur_convoyeur.courant :=52
moteur_convoyeur.tension :=24

- Les types de **données dérivées d'entrées/sorties** (**IODDT**, Input/Output Derived Data Type) : ceux-ci sont prédéfinis par le constructeur et rattachés aux cartes métier, cartes spécifiques que l'on peut trouver sur le rack (carte de pesage, carte de commande d'axe,…).

- Les types **de données dérivées d'équipement** (**DDT d'équipement**, Derived Data Type) : ces types sont créés par le fabricant pour l'utilisation de variables dérivées (tableaux, structure) de cartes spécifiques.

Lors de la création d'un programme en SFC, le logiciel crée des variables Dérivées DDT de type SFCSTEP_STATE pour chaque étape de grafcet.



S_1_1.t correspond au temps d'activation de l'étape C'est une variable de type TIME (T#)

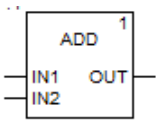
S_1_1.X est un booléen qui vaut true lorsque l'étape est active

S_1_1.tminErr et tmaxErr sont des booléens qui passent à true lorsque le temps dépasse le temps de contrôle min ou max.

3.4 Les fonctions

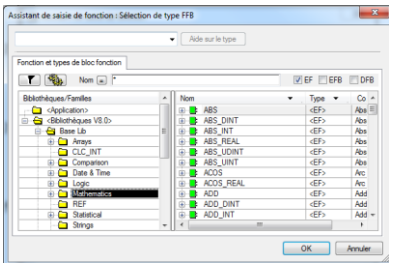
Trois types de foncions existent :

3.4.1 Les **fonctions élémentaires** (**EF**, Elementary Function) qui possèdent 32 entrées maximum, une seule sortie et pas d'état interne.

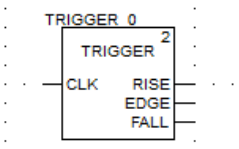


Par exemple le Bloc ADD

que l'on insère dans le programme en sélectionnant l'assistant de saisie de fonction

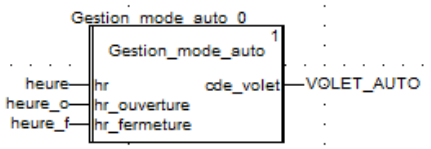


3.4.2 Les **blocs fonctions élémentaires** (**EFB**, Elementary Function Block) qui possèdent jusqu'à 32 entrées, 32 sorties et des états internes.

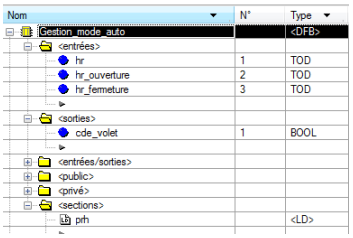


Par exemple le Bloc TRIGGER que l'on insère dans le programme en sélectionnant l'assistant de saisie de fonction comme le cas précédent et que l'on « instancie » (trigger_0 par défaut)

3.4.3 Les **blocs fonctions dérivés** (**DFB**, Derived Function Block) qui possèdent jusqu'à 32 entrées, 32 sorties et des états internes. Ceux-ci sont créés par l'utilisateur dans un projet, mais pourront être placés en bibliothèques et exportés pour pouvoir être utilisés dans d'autres projets.



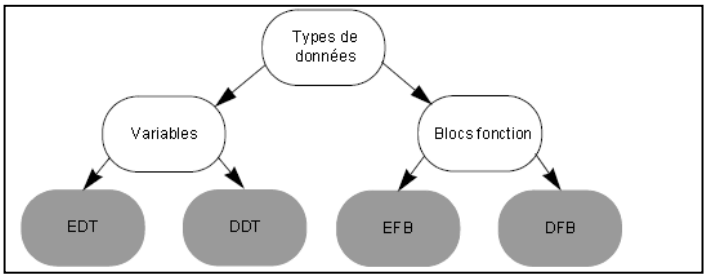
Par exemple le Bloc Gestion_mode_auto développé par un utilisateur que l'on insère dans le programme en sélectionnant l'assistant de saisie de fonction comme le cas précédent et que l'on « instancie » (Gestion_mode_auto_0)
Avant d'appeler ce bloc, il aura fallu au préalable créer le type dérivé :



Les sections de programme traduisant le comportement du DFB pourront être écrites dans l'un de ces 4 langages : Ladder Diagram (LD), Structured Text (ST), Instruction List (IL), Function Block Diagram (FBD).

Les EFB et les DFB devront être instanciés pour pouvoir être utilisés dans une application.

Pour résumé :



Famille	Définition
EDT	Types de données élémentaires (Elementary data types) tels que : • Bool, • Int, • Byte, • Word, • Dword, • etc...

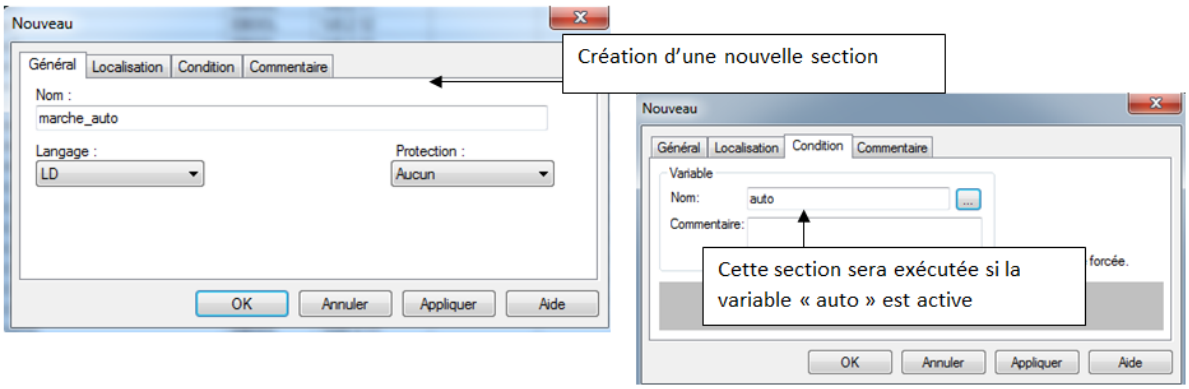
DDT	Types de données dérivés (Derived data types) tels que :
	• tableaux, qui contiennent des éléments de même type:
	  • tableau de Bool (tableau d'EDT),
	  • tableau de tableaux (tableau de DDT),
	  • tableau de structures (tableau de DDT)
	• structures, qui contiennent des éléments de type différents :
	  • structure de Bool, Word, etc... (structure d'EDT),
	  • structure de tableaux, structure de structures, structure de tableaux\structures (structure de DDT),
	  • structure de Bool, tableaux, etc... (structure d'EDT et DDT),
	  • structure concernant les données d'entrées/sorties (structure d'I/ODDT),
	  • structures contenant des variables restituant les propriétés et l'état d'une action ou transition d'un diagramme fonctionnel en séquence (Sequential Function Chart).

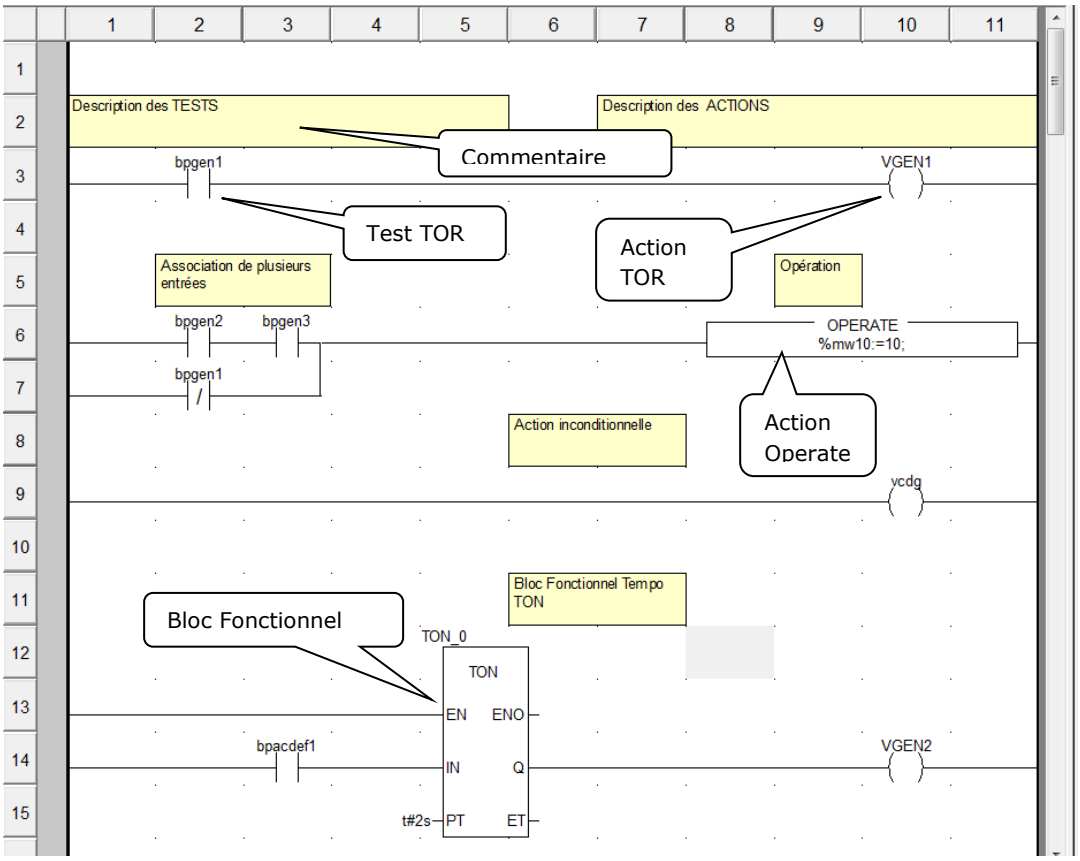
EFB	Blocs fonctions élémentaires (Elementary function blocs) écrits en langage C, il sont constitués : • de variables d'entrées, • de variables internes, • de variables de sorties, • d'un algorithme de traitement.
DFB	Blocs fonctions utilisateurs (Derived function blocs) écrits en langage d'automatisme (Litteral Structuré, Liste d'instructions, etc...), ils sont constitués : • de variables d'entrées, • de variables internes, • de variables de sorties, • d'un algorithme de traitement.

3.5 Le langage de programmation Ladder

3.5.1 Description

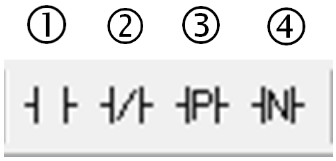
Ouverture d'une section
La création d'une nouvelle section se fait en sélectionnant nouvelle section de l'onglet section du navigateur de projet





3.5.2 La barre d'outils :

3.5.3 Les outils de tests ***TOR***:



① : Détection si entrée NL1

② : Détection si entrée NL0

③ : Détection du front montant

④ : Détection du front descendant

3.5.4 Les outils d'action ***TOR*** *:*

① : recopie du test (sortie NL1 si test vrai)

② : recopie de l'inverse du test (sortie NL0 si test vrai)

③ : action de mémorisation Set → NL1

④ : action de mémorisation Reset → NL0

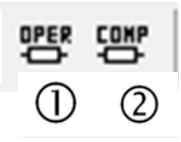
⑤ : bobine de détection de front positif

⑥ : bobine de détection de front négatif

⑦ : arrêt du programme

⑧ : appel d'un sous-programme

3.5.5 Les blocs ***compare et operate***



① : Il s'agit d'un bloc opération qui permet des affectations de variables non booléennes ou instructions écrites en Structuré

② : Il s'agit d'un bloc de comparaison pour tester des expressions de

① : Permet de sélectionner une donnée de la table des variables

② : Assistant de saisie de bloc Fonctionnel

③ : Ouvre l'éditeur du navigateur de bibliothèque de type

*3.5.6 Les outils de **sélection***

3.5.7 Les actions de **contrôle**

① : Si l'état de la liaison à gauche est 1, un saut est exécuté jusqu'à l'étiquette (dans la section courante)

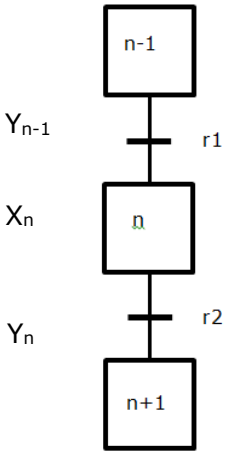
② : Il s'agit d'un repère (destination du saut)

③ : Force le retour au programme appelant

3.5.8 La traduction d'un grafcet en ladder

Le ladder peut être utilisé pour coder une application séquentielle décrite à l'aide d'un grafcet.

La solution suivante permet de respecter les 5 règles du grafcet

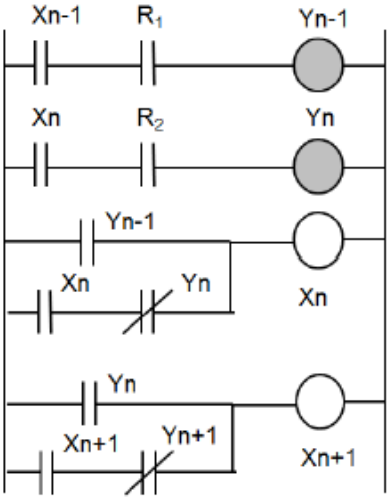


L'équation logique du franchissement de la t

$Y_{n-1} = X_{n-1}.\ r1$

L'équation logique de l'étape Xn :

$X_n = Y_{n-1} + X_n.\overline{Y_n}$



La solution consiste alors à coder :

1- en premier dans la section toutes les équations logiques de franchissement

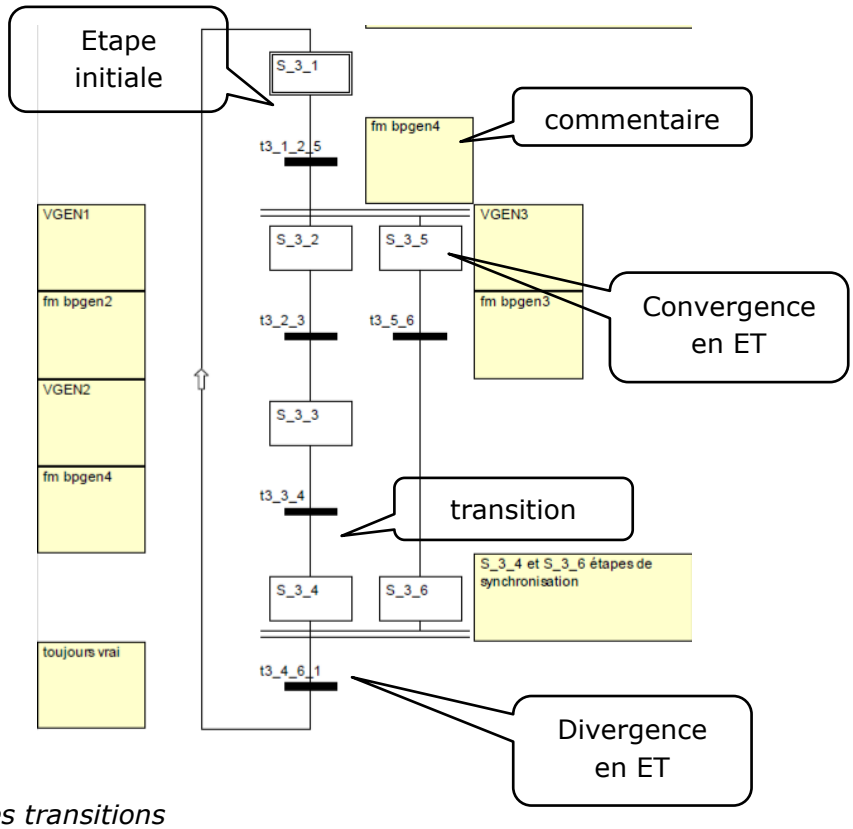
2- toutes les étapes en fonction des équations de franchissement

3- toutes les actions à la fin en fonction des étapes

3.6 le SFC (Diagramme Fonctionnel en Séquence)

Une section SFC contient les objets de création de programme suivant :

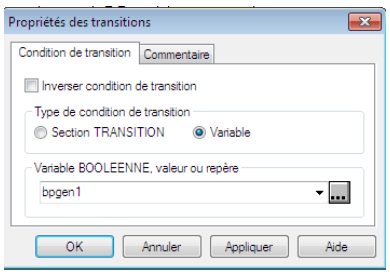
- Etape
- Macroétape
- Transition
- Saut
- Liaison
- Divergence en OU
- Convergence en OU
- Divergence en ET
- Convergence en ET



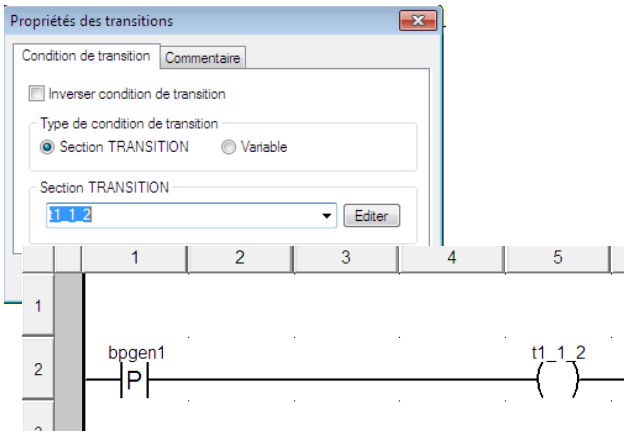
3.6.1 Le codage des transitions

Il existe deux types de transition :

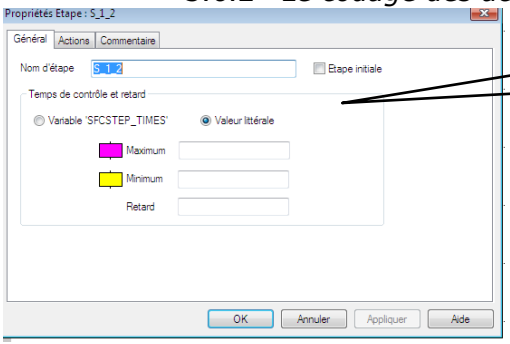
→Variable



→ Section transition



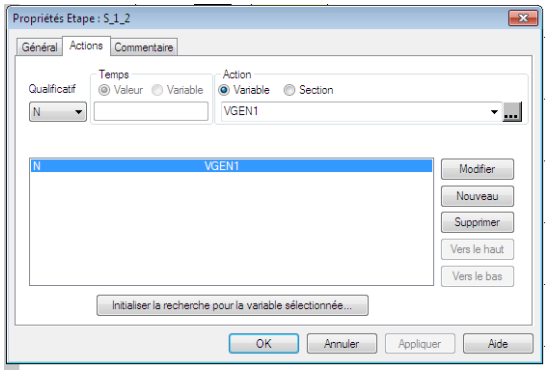
3.6.2 Le codage des actions

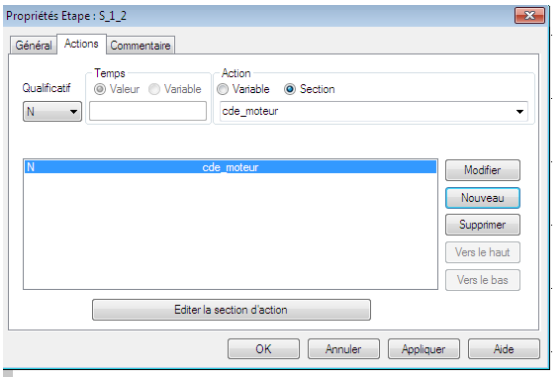


A cocher si étape initiale

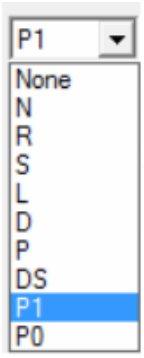
Les actions peuvent être codées en commandant une variable ou en créant une section

Dans les deux cas il faut utiliser un qualificatif





Les qualificatifs



N : Normal
R : Raz l'action précédente
S : Mémorise l'action qui sera désactivés par une étape "R"
L : Action temporisé, le temps est défini ou dans une variable
D : Action retardé, le temps est défini ou dans une variable
P : Action ne dure d'un tour de scrutation
DS: Action retardé puis mémorisé, désactivé par une étape "R"
P1: Action sur front montant, s'exécute en premier, "Equivalent à l'activation du PL7"
P0: Action sur front descendant, s'exécute en dernier "Equivalent à la désactivation du PL7"

3.7 Le langage littéral structuré

Le ST (langage littéral structuré) est un langage évolué de type algorithmique, il est qualifié de « textuel » par opposition aux langages « graphiques » comme le ladder ou le SFC.
Il se programme sous forme de phrases et contient un certain nombre d'instruction.
Ce langage est particulièrement bien adapté à la programmation de fonctions arithmétiques, et pour la gestion des communications.
Il ressemble beaucoup au langage « C ».

3.7.1 Les instructions de base

INSTRUCTION SUR LES BITS

:= pour une affectation
NOT, OR, AND, XOR, SET, RESET
RE : Front montant
FE : front descendant

```
VGEN1 :=%m1 ;

VGEN2 := not bpgen2 ;

VGEN3 := bpgen2 or bpgen3 and bpgen4 ;

 Equivalent à :

VGEN3 := bpgen2 or (bpgen3 and bpgen4) ;
```

INSTRUCTION DE COMPARAISON

= < > <= >= <>

```
VDEF1 := %MW10>20 ;

VDEF2 :=%MW15=%MW6 ;
```

3.7.2 Les structures de base

STRUCTURE INCONDITIONNELLE

Une suite d'actions séparées par des ";"
<Action>;<Action>;<Action>;
<Action>;
<Action>;
Une action finit toujours par un ";"

STRUCTURES CONDITIONNELLE

```
IF <condition> THEN
        <programme>
END_IF;
```

```
IF A>B THEN
        SET(VDEF3) ;
END_IF ;
```

```
IF <condition> THEN
        <programme>

ELSEIF <condition> THEN
        <programme>
        <programme>
ELSE
        <Programme>
END_IF;
```

```
IF A>B THEN
        C :=SIN(A) * COS(B) ;
        B :=SUB (C,A) ;
ELSEIF A=B THEN
        C :=ADD(A,B) ;
ELSE   C :=DIV(A,B) ;
END_IF ;
```

STRUCTURES ITÉRATIVES

WHILE <condition> DO
<programme>
END_WHILE;

```
X :=1 ;
WHILE  X<= 100  DO
X :=X+4 ;
END_WHILE ;
```

REPEAT
<programme>
UNTIL <condition> END_REPEAT;

```
X :=1 ;
REPEAT X :=X+2 ;
UNTIL X>=101
END_REPEAT ;
```

STRUCTURE REPETITIVE

FOR <indice>:=<valeur départ> TO <Valeur arrivé> DO
<programme>
END_FOR ;

```
FOR i :=1 TO 50 DO
C :=C*COS(B) ;
END_FOR ;
```

SELECTION DE L'INSTRUCTION

CASE <select> OF
1: <programme> ;
2: <programme> ;
3,4: <programme> ;
ELSE : <programme> ;
END_CASE ;

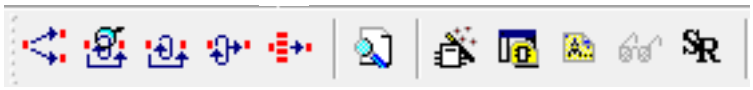
```
CASE %MW52 OF
            12: C :=COS(B) ;
            2: C :=COS(A) ;
            3,4: C :=COS(A)*10 ;
            ELSE C :=0 ;
END_CASE ;
```

AUTRES INSTRUCTIONS

Il existe de très nombreuses instructions permettant de manipuler les bits, mots, tableaux, modules métiers,…

3.7.3 Le menu

① ② ③ ④ ⑤



① : Instruction IF

② : Instruction FOR

③ : Instruction WHILE

④ : Instruction REPEAT

⑤ : Instruction CASE

L'assistant de saisie permet d'appeler les blocs fonctionnels.

- Compteur CTU

```
CTU_0 (CU := (*BOOL*),
       R := (*BOOL*),
       PV := (*INT*),
       Q => (*BOOL*),
       CV => (*INT*));
```

Les champs BOOL ou INT sont à compléter avec vos variables

- Tempo TON :

```
TON_0 (IN := (*BOOL*),
       PT := (*TIME*),
       Q => (*BOOL*),
       ET => (*TIME*));
```

- Conversion REAL en DINT :

```
(*DINT*) := REAL_TO_DINT (IN := (*REAL*));
```

- Ecriture sur un périphérique ou objet distant

```
WRITE_VAR (ADR := (*ANY_ARRAY_INT*),
           OBJ := (*STRING*),
           NUM := (*DINT*),
           NB := (*INT*),
           EMIS := (*ANY_ARRAY_INT*),
           GEST := (*ANY_ARRAY_INT*));
```

NOTES :

NOTES :

NOTES :