RépubliqueAlgérienneDémocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mouloud MAMMERI, Tizi-Ouzou

Faculté de Génie Electrique et d'Informatique
Département d'Automatique

MEMOIRE DE FIN D'ETUDES

En vue de l'obtention du diplôme

De MASTER professionnel EN AUTOMATIQUE
OPTION : Automatique et informatique industrielles

Thème

Simulation d'un Automatisme Industriel sous Matlab-Simulink.

Dirigé par : Mr MAIDI

Présenté par : Ait Yacoub Terkia Yasmine

Soutenu le : 12 /09 /2013

Promotion 2013

Remerciements

Je remercie tout d'abord DIEU Tout Puissant pour m'avoir accordé la grâce de réaliser ce travail.

Je remercie tout particulièrement mon encadreur monsieur MAIDI d'avoir accepté de m'encadrer ainsi, pour son soutien moral, qui ma guidé avec bonne volonté.

Je remercie les membres du jury qui, en dépit de leurs multiples occupations, ont accepté de juger ce travail.

Je remercie mes très chers parents et grands parents. Aucun acte ou expression ne pourra exprimer mes sentiments envers vous.

Je remercie ma cousine Lynda pour tout ce qu'elle a fait pour m'aider durant ce projet.

Mes remerciements vont également à mes frères, sœurs, cousins, cousines ;Puissiez-vous trouvez en ce travail une source de motivation et de persévérance aussi bien sur le plan intellectuel que dans tout autre domaines de la vie.

Je remercie mes oncles et tantes. Aucune expression ne pourra exprimer mes sentiments envers vous.

Enfin je remercie tous ceux qui ont participé de près ou de loin à la réalisation de ce travail.

Dédicace

Je dédie ce travail à :

- A l'être le plus cher dans mon cœur. A ma mère **Malika**, qui sans ses soutiens et conseils je n'aurais pas pu arriver là.

- A la mémoire de mon cher père **Mustapha.**

- A mes très chers frères de cœur anis, Sofiane et mouloud Midou.

- A ma sœur Sarah.
- A ma chère tante Nadia

- A mes oncles et mes tantes et toute ma famille.

- A ma meilleur amie yamina

- A ma promotion.

- A tous ceux que j'aime bien et je respecte.

sommaire

Table des matières

Introduction générale

INTRODUCTION :

Les systèmes automatisés ont été conçus à l'origine pour remplacer l'homme dans l'accomplissement de tâches fastidieuses, dangereuses ou dépassant ses capacités physiques. De nos jours, le domaine d'application de ces systèmes a été largement étendu puisque leur utilisation s'est démocratisée au point d'envahir notre vie quotidienne. On les trouve plus seulement dans les systèmes industriels complexes ; ils ont fait leur apparition dans les produits de consommation les plus courants (électroménager, domotique, jouets . . .).

Dans ce nouveau contexte leur but est souvent d'améliorer le confort et la sécurité de leurs utilisateurs.

Le domaine de l'automobile est un bon exemple de cette évolution. Hier, les systèmes automatisés étaient surtout utilisés dans les ateliers de montage des véhicules. Aujourd'hui le véhicule lui-même intègre de nombreux systèmes automatisés tels que, par exemple, le contrôle moteur, le freinage ABS, la boîte de vitesses automatique ou bien encore la direction assistée. Prenons par exemple cette dernière, il y a quelques années réalisée mécaniquement, elle peut aujourd'hui être électrique et pilotée par microprocesseur. Elle permettra, dans les prochaines générations de véhicule, un contrôle automatique de la conduite (fonction de pilote automatique). La tendance actuelle est au remplacement des commandes mécaniques par des systèmes d'asservissement électronique.

Cette évolution a été rendue possible grâce aux progrès réalisés dans le domaine de la conception des circuits électroniques permettant de construire des microprocesseurs de plus en plus puissants.

Ces systèmes de contrôle numériques dits systèmes temps réel confèrent aux systèmes automatisés auxquels ils appartiennent une plus grande précision, un comportement plus sûr et plus intelligent.

Le cycle de conception et de développement de ces systèmes nécessite une collaboration forte et permanente entre automaticiens et informaticiens. Dans une première phase, l'automaticien est chargé de définir un modèle mathématique du système. Il doit prendre en compte les problèmes de discrétisation liés à une réalisation numérique du système de contrôle. Ce passage d'un modèle continu à un modèle discret se traduit par des contraintes de temps (liées à l'échantillonnage des signaux) sur la fréquence et le temps d'exécution des calculs que devra effectuer le système temps réel pour suivre le modèle défini. Dans une deuxième phase, l'informaticien réalise les programmes qui doivent implanter ces calculs sur le système temps réel. Lors de cette phase d'implantation, l'informaticien doit être capable de garantir que ses

programmes satisfont les contraintes de temps d'exécution définies par l'automaticien. La dernière phase du cycle de conception consiste à relier physiquement le système temps réel au système à contrôler. Il faut alors vérifier que le système automatisé se comporte comme prévu. Dans la majorité des cas, il est nécessaire de tester les programmes afin que le système de contrôle se comporte comme le modèle défini. Il est aussi nécessaire d'observer le comportement du système automatisé.

Ces observations permettront d'établir les écarts qu'il y a entre le fonctionnement réel et le fonctionnement voulu du système automatisé. Si cet écart est acceptable le système est validé. Si ce n'est pas le cas, on retourne à la première phase de développement afin de modifier ou d'affiner les modèles. Ce cycle de développement est effectué autant de fois que nécessaire jusqu'à la validation du système automatisé.

La réalisation d'un système temps réel nécessite une bonne maîtrise des outils fournis par la théorie de l'automatique lors de la phase de modélisation et de simulation, ainsi qu'une bonne maîtrise de l'informatique temps réel lors de la phase d'implantation. La recherche sur la programmation des systèmes temps réel tente de fournir des méthodes de développement permettant de gérer au mieux cette grande complexité et d'aboutir à la réalisation de systèmes sûrs et efficaces. C'est dans ce but qu'ont été développées les méthodes dites "formelles". Ces méthodes cherchent à définir et à regrouper des outils de spécification (description haut niveau la plus proche du modèle du système), des outils de vérification permettant de simuler et de vérifier les propriétés de la spécification ainsi que des outils de génération automatique des programmes temps réel implantant la spécification.

Réunir sous un même environnement de développement, un langage de spécifications, des outils de vérification et de génération automatique de programmes temps réel capables de prendre en compte différents types d'architectures (notamment multiprocesseurs) et capables de garantir que le code généré est conforme à la spécification, constituerait l'outil de conception idéal capable d'aboutir rapidement à la réalisation d'un système sûr et optimisé. Réaliser un tel outil est l'un des challenges actuels de la recherche sur la programmation des systèmes temps réel.

Cependant il est maintenant possible de trouver des logiciels dans la plupart des parties d'une voiture, d'un avion, d'un téléphone portable, ou les systèmes de contrôle d'imagerie médicale. Deux qualificatifs sont généralement associés à ces systèmes : l'adjectif embarqué qui caractérise la relation étroite entre le matériel et le logiciel et l'adjectif critique qui révèle l'aspect sécuritaire des systèmes industriel.

Dans la présente étude, nos investigations visent à explorer le principe d'une nouvelle méthodologie de traduction du programme de gestion d'un API dans l'environnement de Matlab/Simulink pour pouvoir simuler le comportement que peut avoir un automatisme industriel et ainsi évité les dommages qui peuvent survenir lors de la réalisation.

Ainsi ce présent mémoire va être présenté en 3 chapitres, dont le premier chapitre va être consacré pour la présentation des API et toutes leurs architectures ainsi que leurs langages de programmation.

Le deuxième chapitre va s'intitulé sur la modélisation industriel et simulation des processus. Et pour finir, nous allons présenter dans le troisième chapitre le bloc de fonction de Matlab/Simulink des processus industriel commandé par des API.

Chapitre I

Les Automates Programmables Industriels

I. Introduction :

Bien que Les automates programmables industriels aient beaucoup de définitions, on peut affirmer qu'ils sont les membres à semi-conducteur de la famille d'ordinateur, qui utilise des circuits intégrés au lieu des dispositifs électromécaniques pour mettre en application des fonctions de commande. Ils peuvent être considérés en des termes simples en tant qu'ordinateurs industriels avec une architecture particulièrement conçue dans leurs unités centrales (le cerveau d'API) et leurs circuits d'interface d'entrée-sortie (I/O) avec le monde réel.

Les API sont capables de stocker des instructions, telles que l'ordonnancement, la synchronisation, le compte, la logique, l'arithmétique, la manipulation de données, et la communication, pour commander les machines industrielles et les processus.

I. 2) Historique :

C'est Modicon qui créa en 1968, aux USA, le premier automate programmable. Son succès donna naissance a une industrie mondiale qui s'est considérablement développé depuis. L'automate programmable représente aujourd'hui l'intelligence des machines et des procèdes automatisé de l'industrie, des infrastructures et du bâtiment.

Dans les années 80 les automates deviennent de plus en plus gros:

- De plus en plus d'entrées/sorties.
- Un automate commande plusieurs machines (**Architecture centralisée).**

Dans les années 90, on utilise des automates plus petits relies entre eux par des réseaux **(Architecture décentralisée).**

De nos jour, les entrées / sorties sont aussi décentralisées.

L'automate possède de plus en plus de ports de communication et de moins en moins d'entrées sorties en local.

I. 3) Définition :

Un Automate Programmable Industriel (API : Automate programmable industriel ou, en anglais, PLC : Programmable Logic Controller) est un appareil électronique de traitement de l'information (remplacement de logique a relais câblée) qui effectue des fonctions d'automatisme programmées telles que :

- La logique combinatoire

- Le sequencement
- La temporisation
- Le comptage
- Le calcul numérique
- L'asservissement, la régulation

-Et qui permet de commander, mesurer et contrôler au moyen de signaux d'entrées/sorties (numériques ou analogiques) toutes machines et processus, en environnement industriel.

I. 4) Architecture matérielle d'un API :

I. 4.1) 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 micros automates. Il intègre le processeur, l'alimentation, les entrées / 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 avec plus connecteurs).

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

I. 4.1.1) Configurations mécaniques de l'aspect extérieur :

Il y a quatre types communs de conception mécanique des API :

-API Single-board ou API a armature ouverte.

- API compact ou simple-boîte (parfois désigné sous le nom d'une brique d'API ou API shoebox)

-API Semi-modularisé.

-API modularisé (API modulaire ou types de support).

- **API Single-board ou API a armature ouverte:**

D'une part, les API single-board sont des API de base disponible sur une carte électronique simple. Ils sont totalement d'un seul bloc (excepté une alimentation d'énergie) et, une fois installés dans un système, ils sont simplement montés à l'intérieur d'un coffret de commande sur les impasses filetées .Les API single board sont très peu coûteux, facile a programmer, petit, et consomment peu de puissance, mais, d'une façon générale, ils n'ont pas un grand nombre d'entrées et de sorties, et ont un ensemble d'instruction légèrement limité. Ils sont plus adaptés à de petites et simples applications de commande.

- **API compact ou simple-boîte d'API (parfois désigné sous le nom d'une brique d'API ou d'API shoebox) :**

Ces API sont également logé dans une seul case avec tous les points de raccordement d'entrée /sortie, de puissance et de commande situés sur une seul unité. Dans ce cas-ci, ils sont connus en tant que API compact. Ce genre de contrôleurs programmables est généralement choisi selon la mémoire disponible de programme et le nombre exigés des entrées / sorties et leurs tension pour adapter à l'application. Ce type d'API compact est utilisé généralement pour de petits contrôleurs programmables et fourni comme paquet compact intégral complet avec l'alimentation d'énergie, le processeur, la mémoire, et les unités d'entrée-sortie. Typiquement un tel API pourrait avoir 6, 8, 12, ou 24 entrées et 4, 8, ou 16 sorties et une mémoire qui peut stocker environ 300 à 1000 instructions.

- **API Semi-modularisé :**

Quelques systèmes compacts ont la capacité d'être prolongé pour faire face à plus d'entrées / sorties en liant des boîtes d'entrée-sortie entre elles. Ce genre d'API est connu en tant qu'unités semi-modularisé. Ces systèmes ont généralement un port d'expansion (une interconnexion douille) ce qui permettra l'addition des unités spécialisées comme les compteurs à grande vitesse et les entrée analogique et les unités de sortie ou les entrées/ sorties discrètes additionnelles. Ces unités d'expansion sont branchées directement à la case principale ou reliées à lui par un câble plat ou à tout autre câble approprié.

- **API modularisé (API modulaire ou types de support) :**

Ce sont des systèmes avec de plus grands nombres d'entrées /sorties et des unités plus sophistiquées, avec une plus grande sélection d'options, et ils sont susceptibles d'être modulaires et conçus pour s'adapter dans des supports. Le type modulaire se compose des modules séparés pour l'alimentation d'énergie, le processeur...etc, qui sont souvent montés sur des rails dans un coffret en métal. Le type de support peut être employé pour toutes les tailles des contrôleurs programmables et a diverses unités fonctionnelles emballées dans de différents modules qui peuvent être branchés en douilles dans un support bas. Le mélange des modules exigés pour un but particulier est décidé par l'utilisateur et les unités approprier branchés au support. Ainsi il est facile d'augmenter le nombre de raccordements d'entrée-sortie en ajoutant simplement plus de modules d'entrée/sortie ou d'augmenter la mémoire en ajoutant plus d'unités de mémoire.

Les interfaces de puissance et de données pour des modules dans un support sont fournies par les conducteurs de cuivre dans la carte mère du support et quand des modules sont glissés dans un support, ils s'engagent dans des connecteurs dans la carte mère.

I. 4.1.2) Fonctions réalisées *:*

Les automates compacts permettent de commander des sorties en T.O.R et gèrent parfois des fonctions de comptage et de traitement analogique.

Les automates modulaires permettent de réaliser de nombreuses autres fonctions grâce à des modules intelligents que l'on dispose sur un ou plusieurs racks. Ces modules ont l'avantage de ne pas surcharger le travail de la CPU car ils disposent bien souvent de leur propre processeur.

§ Principales fonctions :

- Cartes d'entrées / sorties : Au nombre de 4, 8, 16 ou 32, elles peuvent aussi bien réaliser des fonctions d'entrées, de sorties ou les deux.

-Ce sont les plus utilisées et les tensions disponibles sont normalisées (24, 48, 110 ou 230V continu ou alternatif ...).

-Les voies peuvent être indépendantes ou commune.

-Les cartes d'entrées permettent de recueillir l'information des capteurs, boutons ...etc, qui lui sont raccordés et de la matérialiser par un bit image de l'état du capteur.

-Les cartes de sorties offrent deux types de technologies : les sorties à relais électromagnétiques (bobine, contact) et les sorties statiques (à base de transistors).

- Cartes de comptage rapide : elles permettent d'acquérir des informations de fréquences élevées incompatibles avec le temps de traitement de l'automate.

Exemple : signal issu d'un codeur de position.

- Cartes de commande d'axe : Elles permettent d'assurer le positionnement avec précision d'élément mécanique selon un ou plusieurs axes. La carte permet par exemple de piloter un servomoteur et de recevoir les informations de positionnement par un codeur. L'asservissement de position pouvant être réalisé en boucle fermée.
- Cartes d'entrées / sorties analogiques : Elles permettent de réaliser l'acquisition d'un signal analogique et sa conversion numérique (CAN) indispensable pour assurer un traitement par le microprocesseur.
- La fonction inverse (sortie analogique) est également réalisée tel que Les grandeurs analogique sont normalisées : 0-10V ou 4-20mA.

§ **Autres cartes** :

- Cartes de régulation PID.
- Cartes de pesage.
- Cartes de communication (Ethernet ...).
- Cartes d'entrées / sorties déportées.

I. 4.2) Aspect interne :

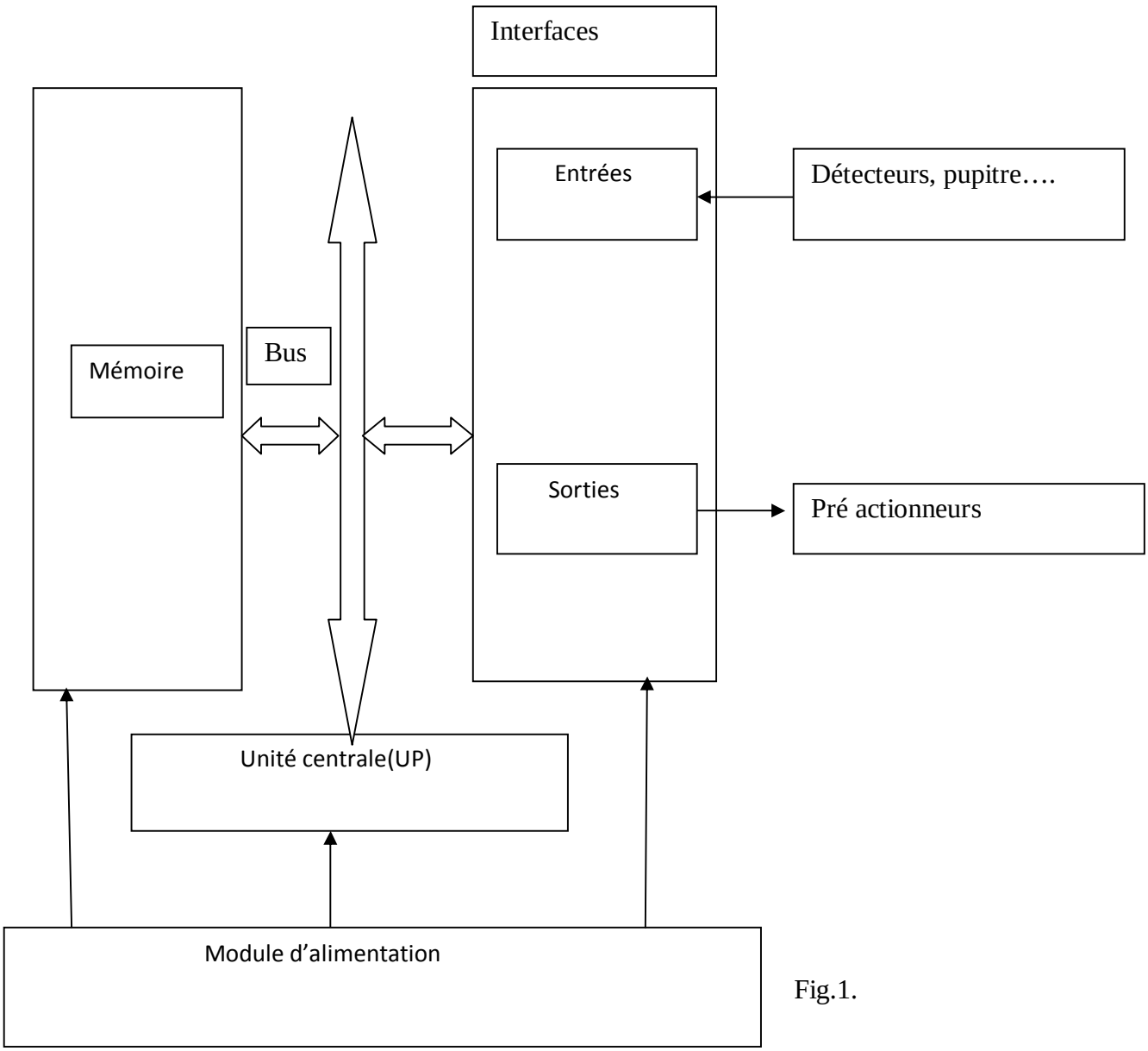


Fig.1.

Les blocs typiques pour un contrôleur programmable généralement sont: Le processeur, le support de support, les modules d'entrée /sortie, module d'alimentation d'énergie et l'unité de programmation.

- **Le processeur :**

  (Également connu sous le nom d'unité centrale de traitement), comme dans les unités d'un seul bloc, est généralement spécifié selon la mémoire requise pour que le

programme soit mis en application. Le processeur comprend une mémoire, ports de communication périodique pour l'imprimeur, lien de LAN d'API et le dispositif de programmation externe et, dans certains cas, l'alimentation d'énergie de système pour actionner le processeur et des modules d'entrée-sortie. Noter que, dans des versions modularisées inclut des dispositifs tels que des boucles d'avertissement plus élevées de fonctions de maths, de PID et des commandes de programmation facultatives.

- **Le support de support :**

Est habituellement un cadre en métal avec une carte mère de carte électronique qui fournit des moyens pour monter les modules d'entrée-sortie de l'API et le processeur.

Les supports de support sont spécifiés selon le nombre de modules exigés pour mettre en application le système. Ils fournissent des raccordements de données et de puissance au processeur et des modules par l'intermédiaire de la carte mère. Pour les unités centrales de traitement qui ne contiennent pas une alimentation d'énergie, le support tient également l'alimentation d'énergie modulaire. Il y a des systèmes dans lesquels le processeur est monté séparément et relié par le câble au support. Le support de support peut être disponible pour monter directement à un panneau ou peut être installé dans un coffret standard d'équipement. Les supports de support sont « cascadable » ainsi que plusieurs peuvent être reliés ensemble pour permettre à un système de s'adapter à un grand nombre de modules d'entrée-sortie.

- **Les modules d'entrée /sortie :**

Sont spécifiés selon les signaux d'entré/sortie liés à l'application considérer. Ces modules se rangent dans les catégories des compteurs à grande vitesse discrets ou analogiques. Les modules discrets d'entrée-sortie sont généralement capables de manipuler 8 ou 16 et, dans certains cas 32 type de marche-arrêt d'entrée/sortie par module. Des modules sont spécifiés comme entrée ou sortie mais généralement pas les deux en même temps bien que quelques fabricants offrent maintenant les modules qui peuvent être configurés avec des points d'entrée et de sortie dans la même unité. Le module peut être spécifié comme AC seulement ou DC seulement ou AC/DC avec les valeurs de tension pour lesquelles il est conçu. Les modules d'entrée/sortie analogique sont disponibles et spécifiés selon la résolution et la tension désirée ou la gamme du courant. Comme avec les modules discrets, ceux-là sont généralement de type entrés ou sortie ; cependant quelques fabricants fournissent les entrées/sorties analogique dans le même module et ils peuvent directement accepter des entrées de thermocouple pour la mesure et la surveillance de la température par l'API.des entrées pulsées

a l'API peuvent être acceptées on utilisant un module-compteur à grande vitesse. Ce module peut être capable de mesurer la fréquence d'un signal d'entrée d'un tachymètre ou toute autre fréquence produisant du dispositif et il peut également compter les impulsions entrantes si désirés. Généralement, la fréquence et le compte sont fournis par le même module en même temps si tous les deux sont exigés dans l'application. le registre du module d'entrée / sortie transfère les mots de 8ou16 bit d'information à et de l'API et ils sont généralement des nombres (BCD ou binaire) produits des commutateurs de roue codeuse ou des systèmes d'encodeur pour que l'entrée ou les données soient produites à un dispositif d'affichage par l'API. D'autres types de modules peuvent être disponibles selon le fabricant de l'API et de ses possibilités. Ceux-ci incluent les modules de communication spécialisés pour tenir compte du transfert de l'information à partir d'un contrôleur à l'autre.

- **Module d'alimentation d'énergie :**

Spécifique, dépend de l'API du fabricant étant utilisé dans l'application. Dans certains cas une alimentation d'énergie capable de fournir toute la puissance exigée par le système fournis par le processeur. Si l'alimentation d'énergie est un module séparé, elle doit être capable de livrer un plus grand courant que la somme de tous les courants requise par les autres modules. Pour des systèmes avec l'alimentation d'énergie à l'intérieur du module d'unité centrale de traitement, il peut y avoir quelques modules dans le système qui exigent la puissance excessive non fournie par le processeur en raison de la tension ou les conditions du courant qui peuvent seulement être réalisées par l'addition d'une deuxième source d'énergie. C'est généralement vrai si les modules de communication analogique ou externe sont présents puisque ceux-là exigent une alimentation de ± DC qui, dans le cas des modules analogiques, doivent être bien réglés.

- **L'unité de programmation** :

Permet à l'ingénieur ou au technicien de suivre et éditer le programme à exécuter. Sous sa forme la plus simple ce peut être un dispositif tenu dans la main avec un clavier numérique pour la saisie de programme et un dispositif d'affichage (LED ou LCD) pour le visionnement des étapes ou des fonctions de programme. Des systèmes plus avancés utilisent un PC séparé qui permet au programmeur d'écrire, regarder, éditer et télécharger le programme a l'API. Ceci est accompli avec le logiciel de propriété industrielle fourni par le fabricant de l'API. Ce logiciel permet également au programmeur ou à l'ingénieur de surveiller l'API pendant qu'il lance le programme. Avec ce système de surveillance, des choses telles que les enroulements

internes, les registres, les temporisateurs et d'autres articles non évidents extérieurement peuvent être surveillés pour déterminer l'opération appropriée. En outre, les données de registre interne peuvent être changées s'il y a lieu à amélioré l'opération du programme. Ceci peut être avantageux en mettant le programme au point.

La communication avec le contrôleur programmable se fait par l'intermédiaire d'un câble relié à un port de programmation spécial sur le contrôleur. Le raccordement au PC peut être par un port série ou d'une carte consacrée installée dans l'ordinateur.

I. 5) Structure d'un système automatisé :

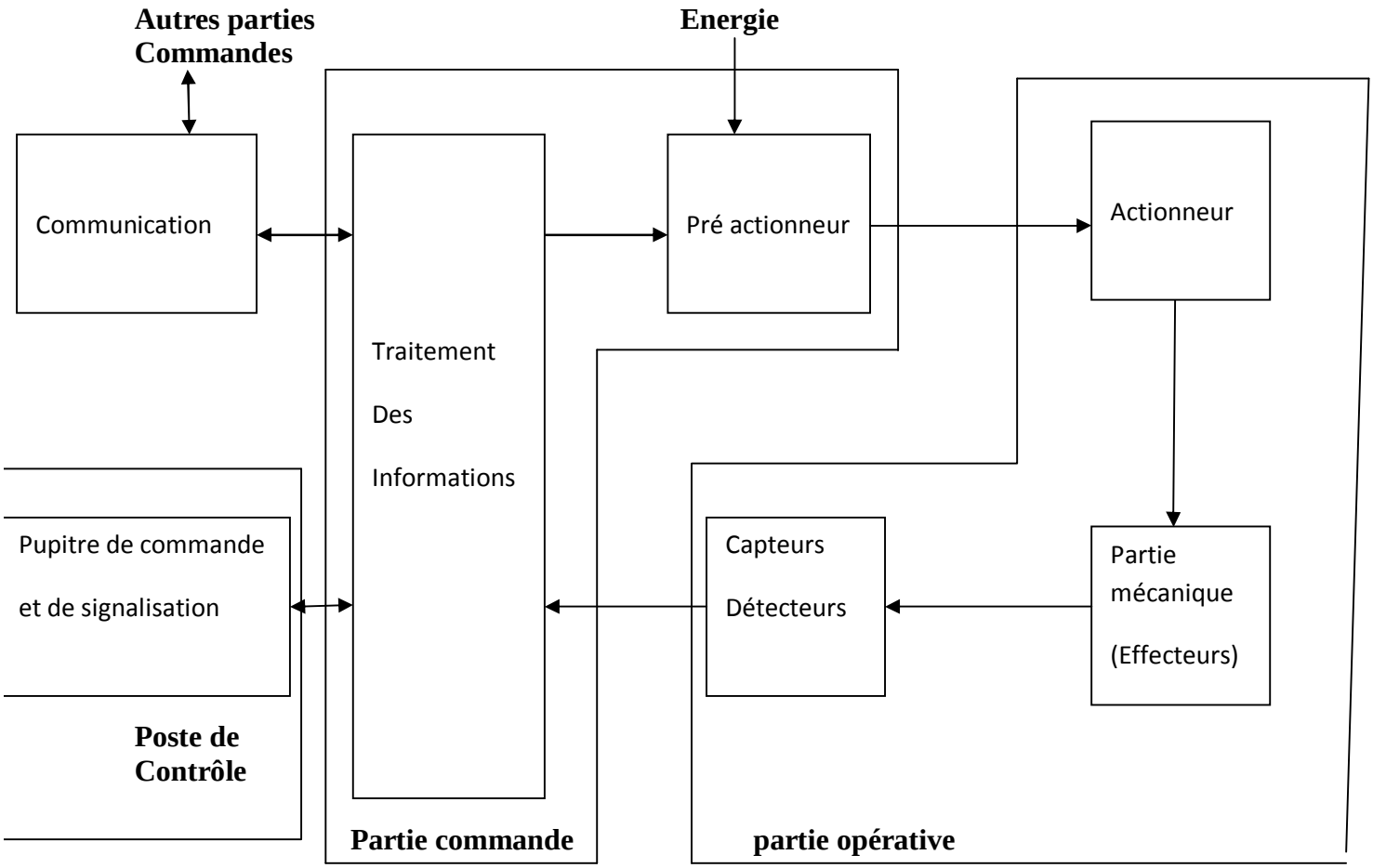


Fig.2.

- **Partie opérative :** Elle agit sur la matière d'œuvre afin de lui donner sa valeur ajoutée en consommons de l'énergie :

-Les **actionneurs** (moteurs, vérins) agissent sur la partie mécanique du système qui agit à son tour sur la matière d'œuvre pour donner le signal d'enclenchement du cycle.

-Les **capteurs / détecteurs** permettent d'acquérir les divers états du système.

- **Partie commande :** Elle donne les ordres de fonctionnement à la partie opérative :

- Les **prés actionneurs** permettent de commander les actionneurs et ils assurent le transfert d'énergie entre la source de puissance (réseau électrique, pneumatique …) et les actionneurs. Exemple : contacteur, distributeur …

Ces pré actionneurs sont commandés à leur tour par le bloc de **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. Et 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)) il va commander les pré actionneurs 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 du cycle …).

-Il permet également de visualiser les différents états du système à l'aide des voyants, de terminal de dialogue ou d'interface homme-machine (IHM).

I. 6) Domaines d'emploi des automates :

On utilise les API dans tous les secteurs industriels pour la commande des machines (convoyeur, emballage ...) ou des chaînes de production (automobile, agroalimentaire ...) comme il peut également assurer les fonctions de régulation des processus (métallurgie, Chimie ...).Et il est de plus en plus utilisé dans le domaine du bâtiment (tertiaire et industriel) pour le contrôle du chauffage, de l'éclairage, de la sécurité et les alarmes.

I. 7) Nature des informations traitées par l'automate :

Les informations peuvent être de type :

- Tout ou rien (T.O.R.) : l'information ne peut prendre que deux états (vrai/faux, 0 ou 1). C'est le type d'information délivrée par un détecteur, un bouton poussoir …etc.
- Analogique : l'information est continue et peut prendre une valeur comprise dans une plage bien déterminée. C'est le type d'information délivrée par un capteur (pression, température …)
- Numérique : l'information est contenue dans des mots codés sous forme binaire ou bien hexadécimale. C'est le type d'information délivrée par un ordinateur ou un module intelligent.

I. 8) Traitement du programme automate :

Tous les automates fonctionnent selon le même mode opératoire :

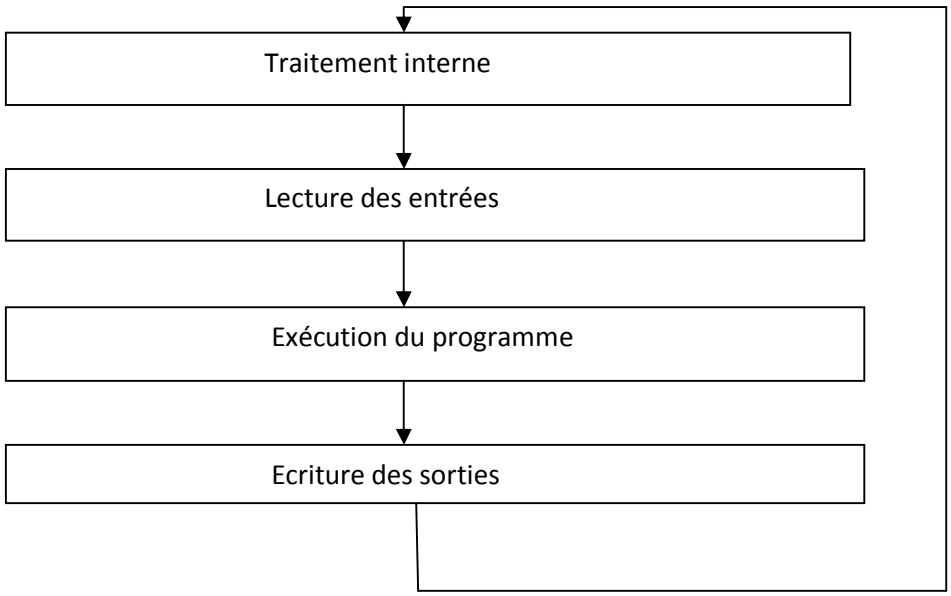


Fig.3.

- Traitement interne : L'automate effectue des opérations de contrôle et met à jour certains paramètres des systèmes (détection des passages en RUN / STOP, mises à jour des valeurs de l'horodateur, ...).
- Lecture des entrées : L'automate lit les entrées (de façon synchrone) et les recopie dans la mémoire image des entrées.
- Exécution du programme : L'automate exécute le programme instruction par instruction et écrit les sorties dans la mémoire image des sorties.
- Ecriture des sorties : L'automate bascule les différentes sorties (de façon synchrone) aux positions définies dans la mémoire image des sorties.

Ces quatre opérations sont effectuées continuellement par l'automate (fonctionnement cyclique).

I. 8.1) Temps de réponse :

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.

Le temps de réponse total est au plus égal à deux fois le temps de scrutation (sans traitement particulier).

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.

Dans certains cas, on ne peut admettre un temps de réponse aussi long pour certaines entrées.ces entrées pourront alors être traitées par l'automate comme des événements (traitement événementiel) et prises en compte en priorité (exemples : problème de sécurité, coupure d'alimentation ...).

Certains automates sont également dotés d'entrées rapides qui sont prises en compte avant le traitement séquentiel mais le traitement évènementiel reste prioritaire.

Exemple :

Les automates TSX micro (Télémécanique) offrent deux types de structure logicielle :

- Une structure mono tâche :

Le programme n'est lié qu'à une seule tâche : la tâche maîtresse.

- Une structure multitâche :

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

La tâche rapide est alors périodique pour laisser le temps à la tâche maîtresse de s'exécuter (la moins prioritaire). La tâche événementielle est prioritaire sur les autres tâches.

I. 9) programmation :

I. 9.1) Définition d'un programme:

Suite logique et séquentiel d'instructions qui commande le traitement des diverses données ainsi que la gestion de l'environnement (clavier, écran, imprimante, …). La façon d'écrire les instructions dépend du langage de programmation utilisé. Chaque langage à ses spécificités d'écriture et notamment de syntaxe, mais beaucoup se ressemblent. A vrai dire, l'ordinateur ne comprend qu'un seul langage qui est le " langage machine ". Par conséquent le langage de programmation comme le C, C++, JAVA, (Visual) Basic, Perl, PHP va jouer le rôle de traducteur entre vous et la machine de manière à rendre l'approche beaucoup plus facile et pratique.

I. 9.2) Différences principales entre les langages:

– La syntaxe

– Langages séquentiels, langages orienté-objet

– Langages interprétés, langages exécutés

– Langages spécialement adaptés :

- A Internet (pages web): PHP, Perl, JAVA
- Aux simulations: Matlab

La terminologie autour de Visual Basic qui va être utilisé dans le chapitre 3 peut mener à une certaine ambiguïté.

Visual Basic est critiqué pour sa gestion mémoire peu performante et pour les codes construit on utilisant des instructions peu académiques, pouvant donner de mauvaises habitudes de programmation et permettant d'écrire un code peu performant.

VBA = Visual Basic pour Application Disponible avec MS Office (Excel, Access, Word).

VB = Visual Basic Langage de programmation contenu dans MS Office mais également intégré dans Visual Studio pour faire des applications indépendantes.

I. 9.3) Généralités sur le Visual Basic:

- **Définition :**

Visual Basic est un des langages les plus utilisés pour l'écriture d'applications commerciales. Il a également été très utilisé dans le monde de l'ingénierie et de la recherche appliquée en raison de sa capacité à permettre des développements très rapides permettant ainsi aux scientifiques de se consacrer davantage à l'algorithmique et moins aux aspects formels du codage.

- **Avantages :**

  – Simplicité

  – Disponible dans MS Office

  – Accès à toutes les fonctions de Windows

    - Systèmes de fenêtres, boîtes de dialogue, API

- **Inconvénients :**

  – Dépend beaucoup des versions utilisées de Windows et MS Office

  – Cher si on veut être en règle

- **types de variables utilisés :**

  Les variables sont essentielles à tous les programmes et tous les langages et elles permettent le stockage à court terme des paramètres du programme. Souvent des valeurs numériques ou des caractères qui correspondent à des zones mémoire de l'ordinateur.

  En Visual Basic, on considère environs 12 types de variables tel que les plus utilisées sont:

  – String pour stocker des chaînes de caractères

  – Integer pour stocker des valeurs entières

  – Double pour stocker des valeurs décimales

  – Long pour stocker des grandes valeurs entières

  – Boolean pour stocker soit un 0 soit un 1(un bit)

  En VB, il n'est pas obligatoire de déclarer les variables, mais vivement conseillé

I. 9.4) Langages de programmation des automates :

Il existe 5 langages de programmation des automates qui sont normalisés au plan mondial par la norme CEI 61131-3.

Chaque automate se programmant via une console de programmation propriétaire ou par un ordinateur équipé du logiciel constructeur spécifique.

- **Liste d'instructions (IL : Instruction list) :**

Langage textuel de même nature que l'assembleur (programmation des microcontrôleurs).

Ex:

LD A

ANDN B

ST C

- **Langage littéral structuré (ST : Structured Text)** :

Langage informatique de même nature que le Pascal, il utilise les fonctions comme if ... then ... else ... (si ... alors ... sinon ...).

Ex:

C = A AND NOTB

- **Langage à contacts (LD : Ladder diagram)** :

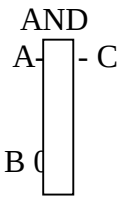
Langage graphique développé pour les électriciens. Il utilise les symboles tels que : contacts, relais et blocs fonctionnels et s'organise en réseaux (labels). C'est le plus utilisé.

Ex :

- **Blocs Fonctionnels (FBD : Function Bloc Diagram)** :

Langage graphique où des fonctions sont représentées par des rectangles avec les entrées à gauche et les sorties à droites. Les blocs sont programmés (bibliothèque) ou programmables.

Ex :



- **Programmation à l'aide du GRAFCET (SFC : Séquential Function Chart) *:***

Le GRAFCET est un langage de spécification utilisé par certains constructeurs d'automate (Schneider, Siemens) pour la programmation. Parfois associé à un langage de programmation. Il permet une programmation aisée des systèmes séquentiels tout en facilitant la mise au point des programmes ainsi que le dépannage des systèmes.

On peut également traduire un GRAFCET en langage contacts et l'implanter sur tout type d'automate.

Certains logiciels permettent une programmation totale en langage GRAFCET et permettent de s'adapter à la plupart des automates existants (logiciels CADEPA ou AUTOMGEN ou Matlab/Simulink).

I. 10) Conclusion :

Dans cette partie nous avons présenté d'une manière générale L'Automate Programmable Industriel (API) adapté à l'environnement industriel, qui réalise des fonctions d'automatisme pour assurer la commande de pré actionneurs et d'actionneurs à partir d'informations logique, analogique ou numérique.

Chapitre II

Modélisation Industriel et Simulation de Processus

II. 1) introduction :

Pourquoi modelant ?

Afin d'étudier, analyser et contrôler des systèmes, il est nécessaire de les connaitre très bien pour ainsi avoir un modèle mathématique qui les décrit. D'une part, Ce modèle peut être employé dans un outil de simulateur d'ordinateur (comme MATLAB/Simulink), généralement un tel modèle sera représenté d'une manière complexe et complète afin de décrire d'une manière précise et réaliste en tant que possible le vrai comportement de système. De l'autre, au cas où une commande serait nécessaire pour les buts d'analyse ou de conception d'un système de contrôle, une représentation plus simple de ce modèle sera exigée, mais en tenant compte toujours de l'essence du modèle et des caractéristique de son comportement .Par conséquent, le but en modelant des techniques est de réaliser des modèles d'une façon simple ou complexe selon l'application et l'objectif désiré.

Dans cette section, nous nous concentrerons brièvement sur le développement des modèles de commande qui décrivent les processus industriels qui seront utiles aux contrôleurs.

II. **2) Approche théorique :** Cette approche consiste à établir un modèle à partir des lois physiques. Ici, l'ingénieur peut trouver la difficulté de contrôler toutes les lois physiques qui participent à la réalisation du modèle qui pourrait être très complexe et ainsi difficile à contrôler. D'ailleurs, un autre inconvénient de cette approche est que de vrais phénomènes ne sont pas tenus compte comme les composants portant des tolérances, bruit et des effets de perturbation…

II. **3) Approche ou identification expérimentale :** Quand un système n'est pas approprié à l'approche théorique due à beaucoup de raisons (telles que sa complexité, une connaissance inachevée de la structure de système ou en raison d'une variation imprévisible de ses dispositifs), il est nécessaire de recourir à une autre approche qui permet l'accomplissement des modèles valides et appropriés. Cette deuxième approche consiste a analyser le système on se basons sur l'étude de ses signaux de sortie devant un ensemble bien connu de signaux d'entrée. Dans cette approche, on n'adopte pas n'importe quelle hypothèse au sujet des caractéristiques du système parce que habituellement son étude est difficile et limite la qualité du modèle.

Beaucoup d'expérience ont démontré que la meilleure solution est la combinaison des deux approches, toutes les fois qu'il est possible. Dans ce cas-ci, deux étapes sont habituellement effectuées : L'étape d'analyse et puis l'étape expérimentale.

Dans l'étape d'analyse, les lois physiques et les conditions de travail (modes d'opération) seront tenues en compte afin d'établir l'hypothèse. À l'étape expérimentale, à partir de l'hypothèse qu'on fait l'analyse puis les mesures expérimentales obtenues seront considérées pour déterminer les coefficients du modèle mathématique.

Afin d'obtenir la réaction du système, il est nécessaire de le stimuler grâce aux variables d'entrée qui sont produites de l'environnement du système à l'étude. Il existe deux genres de variables d'entrée de système : Ceux qui peuvent être commandés, et ceux qui ne peuvent pas être commandés et qui sont automatiquement produits par l'environnement (connu sous le nom de perturbations, Fig.4). Les variables produites par le système ce sont les variables de sortie et elles influencent sur l'environnement. Ces variables sont mesurables et, parfois, observables.

Supposer un système comme présenté dans la fig.4. Principalement, deux problèmes peuvent surgir avec ce système :

- **Problème ou analyse direct :**

Sachant (entrée, système), trouvaille (sorties) ;

Ce problème a une solution unique et ça s'appelle un problème d'analyse.

- **Problèmes inverses** :

1) sachant (entrée, sortie), trouvaille (système) ;

. Ce problème n'a pas une solution unique mais un nombre infini de solutions. C'est un problème d'identification de structure et d'évaluation d'état, et ça s'appelle un problème de la synthèse.

2) sachant (système, sortie), trouvaille (entrée) ;

. C'est un problème de commande (instrumentation).

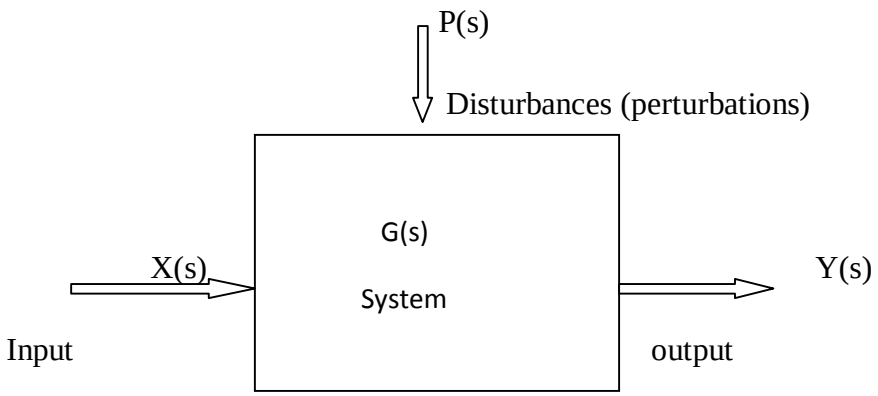


Fig.4.

II. 4) Spectre de modélisation et de simulation

Un système peut appartenir à différentes disciplines, et chacune présente les différents aspects qui devraient être tenus en compte en traitant ce système. Le modeleur peut faire face à une sorte de systèmes qui présentent seulement des données historiques, par exemple, des systèmes d'opinion publique ou des systèmes de secteurs sociaux (voir fig.5).

Des modèles peuvent seulement être établis de l'expérimentation et de la modélisation technique, c'est une identification qui est principalement employée quand la structure de système est inconnue. Dans ce cas-ci, les modèles sont appelés des modèles de boîte noire, et ils peuvent être représentés avec des équations différentielles.

Dans le monde réel, il existe beaucoup de systèmes complexes dont l'évolution dépend de diverses variables (le temps, l'espace…).et la manière la plus appropriée de les décrire est par des équations a dérivé partielles (PDE) parce que ce sont des systèmes avec des paramètres de perturbations. Dans le domaine des sciences environnementales (écologie, pollution, biodiversité…) les modèles développés pour ces systèmes se font par la prévision et l'expérimentation des stratégies de gestion.

En conclusion, il y a un autre genre de systèmes dont leurs lois physiques sont parfaitement connu se qui permet la connaissance de la structure de système, et elles peuvent être établies en utilisant des modèles de boîte blanche .a partir de ces modèles les équations différentielles sont produites et, dans la plupart des cas, elles sont assez franches pour être représentées avec des équations différentielles ordinaires (ODEs) avec des paramètres qui sont concentrés.

Par exemple, les circuits électriques et électronique, les procédés de commande chimique, la commande industrielle et les systèmes aérospatiaux peuvent être représentés avec des modèles de boîte blanche. Dans ces cas, les modèles obtenus sont employés pour concevoir un contrôleur pour contrôler le processus. On peut voir dans la fig.5, allons des modèles de boîte noire aux modèles de boîte blanche qu'il y a tous les modèles qu'un modeleur peut trouver dans le monde réel.

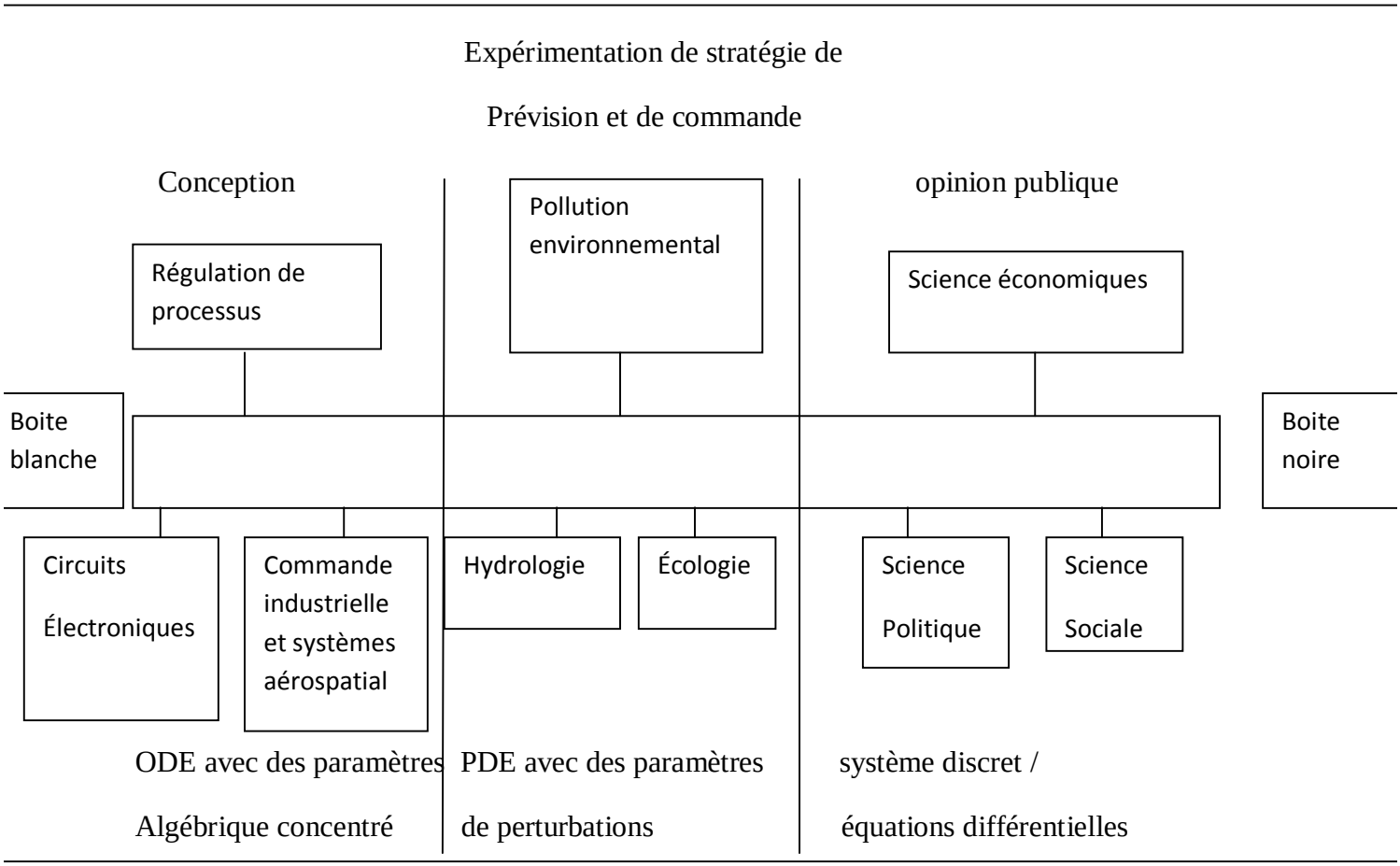


Fig.5. modélisation : de la boîte blanche et de la boîte noire

II. 5) Représentation de modèle mathématique :

Des systèmes linéaires Temps-Invariables (LTI) sont représentés par des ODE, mais ces expressions sont trop compliquées pour manœuvrer. Ainsi, les modeleurs emploient des expressions plus faciles pour représenter ces équations, par exemple, la transformer de Laplace qui permet d'obtenir la fonction de transfert décrivant le système, principalement utilisé dans des systèmes SISO (à sortie unique et à entrée unique). Une alternative pour représenter un système est par la représentation de l'espace d'état, principalement utilisée dans des systèmes de MIMO (à sorties multiples et à entrées multiples). La caractéristique principale de la représentation de l'espace d'état est de représenter les variables internes du

système (tout le système est indiqué) tandis que la fonction de transfert représente seulement le rapport entre la sortie et l'entrée du système.

Dans l'automatique, une représentation de l'espace d'état est un modèle mathématique d'un système physique dont les entrées/ sorties sont exprimés par des vecteurs et les variables d'état par des équations différentiel et algébriques qui sont écrits sous forme de matrice (cette dernière peut être faite quand le système dynamique est linéaire et à temps invariable).

La représentation de l'espace d'état dans le domaine temporelle (également connue sous le nom de « approche de domaine temporelle ») fournit une manière commode et compacte de modeler et analyser des systèmes avec des entrées / sorties multiples tel que avec p entrées et q sorties, nous pourrions écrire q x p de transformé de Laplace pour coder toutes les informations du système.

À la différence de l'approche de domaine fréquentielle, l'utilisation de la représentation de l'espace d'état n'est pas limitée aux systèmes dont leurs composants est linéaires et les conditions initiales nulles. Et on peut toujours représenter l'état du système comme vecteur dans cet espace.

II. 6) Les étapes d'un projet de simulation :

Il est important de savoir les objectifs pour lesquels le modèle sera employé pour pouvoir choisir le meilleurs modèle a appliqué.

Après avoir choisi le meilleur modèle adapté aux objectifs, le modèle de système sera développé en utilisant les lois de physiques et/ou d'identification, selon la situation.

Puis, le modèle doit être mis en application et validé avec des vraies données obtenues (base de données).

Pour valider le modèle n'importe quel critère d'erreur est employé, et si le modèle a une bonne qualité alors le processus sera terminé, dans le cas opposé le modèle est reconsidéré et le processus de sélection commence à rechercher encore le meilleur modèle à équipe au système et tout le processus sera répété jusqu'à ce que nous soyons satisfaits du modèle choisi.

Le processus entier peut être vu dans fig.6.

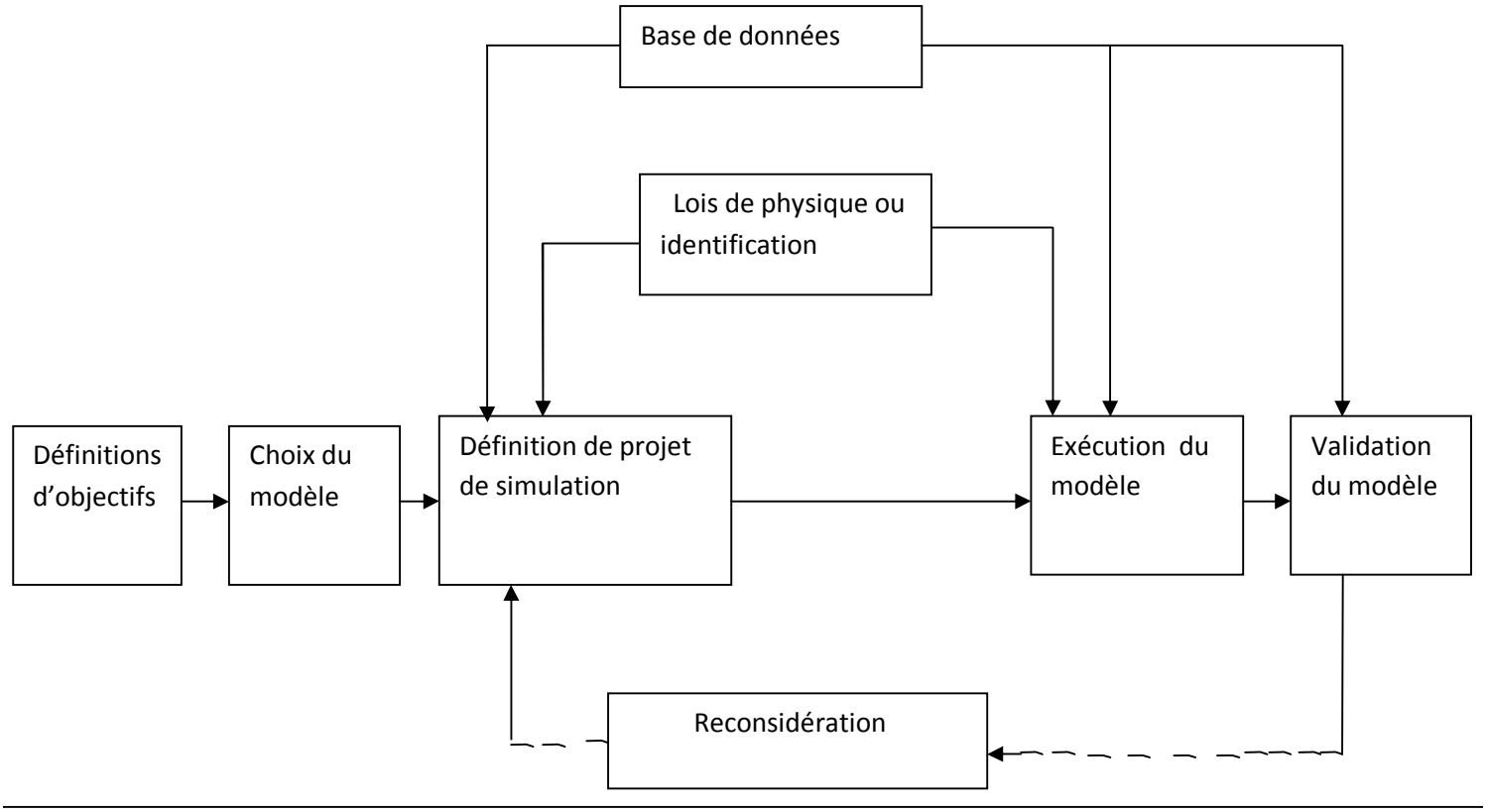


Fig.6. les étapes de simulation

II. 7) Logiciel de simulation :

La modélisation et la simulation mathématiques émergent en tant que technologies principales dans la technologie.

Les outils automatisés appropriés pour l'intégration avec des méthodes de conception traditionnelles sont essentiels pour répondre aux futurs besoins de la technologie efficace.

La simulation interactive fournit une méthode flexible et facile à utiliser pour définir les performances du modèle. Pendant la course de simulation interactive, l'utilisateur peut changer les valeurs des entrées du modèles (signaux de perturbations), les paramètres de système et les conditions initiales des variables d'état tout on percevant immédiatement comment ces changements affecte le modèle dynamique. Par conséquent, la simulation interactive facilite le développement et l'amélioration des performances du modèle et aide à

comprendre le comportement du système. Ces possibilités sont particulièrement utiles quand le modèle est employé pour les buts éducatifs.

II. 7.1) Simulation :

La modélisation des systèmes discrets est beaucoup moins facile à réaliser que la modélisation des systèmes continus pour lesquels la théorie de l'automatique fournit un ensemble de méthodes et d'outils mathématiques beaucoup plus important. C'est pourquoi, afin de simplifier les calculs, l'automaticien cherche à utiliser un modèle pseudo-continu de son système de commande plutôt qu'un modèle discret. La simulation hybride qui consiste à simuler l'ensemble du système automatisé (simulation d'un processus continu relié au système temps réel discret) est indispensable pour valider le modèle à partir duquel le système temps réel doit être conçu. Elle permet de paramétrer les lois de commande afin de compenser les approximations réalisées par la modélisation pseudo-continue. Ce sont les erreurs de modélisation qui ont le plus d'impact sur le temps de développement car elles sont situées à la première étape du cycle de conception. Il faut alors attendre la dernière étape de validation pour les détecter, ce qui entrainera alors la réexécution complète du cycle de conception.

Cette simulation, si elle est suffisamment fiable, permet de valider une fois pour toute le modèle du système temps réel et d'évaluer les propriétés dynamiques du système. Ainsi la simulation hybride devrait être systématiquement associée à l'étape de modélisation car c'est seulement par ce biais qu'il est possible de choisir la loi de commande adéquate et de la paramétrer. Cette simulation hybride, pour être la plus proche de la réalité, doit nécessairement prendre en compte les retards induits par les temps de calcul car le temps de réponse d'une loi de commande est lié au retard introduit par la latence des calculs qui la composent. Elle doit essentiellement prendre en compte les algorithmes de commande, mais il peut aussi être nécessaire de simuler l'ensemble des algorithmes afin de connaître l'influence sur le comportement dynamique du système et les discontinuités dues aux changements de mode.

II. 7.2) Vérification :

La spécification doit pouvoir être vérifiée formellement du point de vue logique et du point de vue comportemental. La vérification logique a pour but de s'assurer que la spécification réalisée est cohérente du point de vue de la sémantique du langage employé. Elle consiste par exemple dans un graphe flot de données à s'assurer que les horloges des signaux d'entrées d'un sommet du graphe sont identiques et que le graphe ne comporte pas de boucles

génératrices d'inter-blocages à l'exécution. La vérification comportementale cherche à garantir que la spécification est conforme au cahier des charges. Elle permet de prouver, entre autre, que certains comportements du système ne seront jamais possibles. Par exemple, si on spécifie un système de commande d'aiguillage d'un système ferroviaire, la vérification comportementale doit permettre de vérifier qu'aucun train ne pourra accéder à une portion de voie déjà occupée par un autre train. Il pourra aussi être montré, sur un système de boîte de vitesses automatique d'un véhicule, que l'enclenchement de la vitesse la plus faible ne sera jamais déclenché alors que le véhicule roule à une vitesse élevée. Ce type de vérification est aussi très important car c'est grâce à elle que l'on peut garantir la fiabilité du système (à condition que l'implantation soit conforme à la spécification).

Dans le cadre d'un outil de spécification multi-formalisme, la vérification logique doit être réalisée pour chacun des formalismes et à tous les niveaux de la spécification. Dans le cadre d'une interface graphique, cette vérification pourra être réalisée en ligne au fur et à mesure de la construction des graphes. Elle devra interdire certaines erreurs, par exemple en autorisant seulement des connexions cohérentes entre les éléments du graphe. La vérification comportementale ne s'intéresse qu'au contrôle du système, elle ne sera donc appliquée qu'aux formalismes décrivant cet aspect.

Seule une approche formelle autorise la vérification. Certains outils basés sur ce type d'approche (Statemate Magnum, Sildex et Orccad) permettent d'exporter la spécification vers des outils autorisant la simulation hybride tels que Simulink ou SystemBuild prenant en compte l'architecture matérielle sur laquelle est implantée la spécification afin de pouvoir intégrer les temps de calcul dans la simulation.

II. 7.3) Implantation :

Un outil de spécification disposant de toutes les fonctionnalités citées précédemment ne sert pas à grand chose si la spécification doit être implantée "à la main" par les ingénieurs, car dans ce cas, il n'est pas possible de garantir que l'implantation est conforme à la spécification et donc que le système possède les propriétés montrées par la vérification logique et comportementale. De plus, l'implantation "à la main" des algorithmes induit un coût financier élevé lié à des temps de développement et de débogage importants. Ce coût sera d'autant plus élevé que l'on aura cherché à optimiser le code. Notons aussi que la maintenance et les modifications ultérieures du système seront très coûteuses.

L'outil de spécification doit donc impérativement pouvoir être connecté à un outil d'implantation ou en intégrer un. Celui-ci doit non seulement prendre en compte les architectures distribuées hétérogènes, mais il doit aussi garantir que l'implantation réalisée est

conforme à la spécification, aussi bien du point de vue logique que temporel, et que cette implantation utilise au mieux les ressources matérielles disponibles. Ainsi, le surcoût engendré par l'exécutif doit être minimal et le parallélisme potentiel des algorithmes doit être exploité.

II. 7.4) Validation :

La validation du système consiste à connecter le système temps réel au processus qu'on cherche à commander, et à observer que les propriétés et le comportement du système automatisé sont conformes au cahier des charges. Cette étape nécessite donc de mesurer et d'observer l'évolution des différents paramètres du processus que le système temps réel cherche à commander.

Dans le cadre de la conception de systèmes embarqués, il est souvent difficile d'intégrer au système les instruments de mesure adéquats. De plus, la structure matérielle intrinsèque du système fournit déjà ces instruments de mesure. Ainsi, idéalement, il doit pouvoir être possible, pour faciliter le débogage et valider le système, et pouvoir visualiser l'évolution de certains signaux pendant l'évolution du système automatisé ou de pouvoir la mémoriser afin de l'analyser par la suite. Ces visualisations ou mémorisations doivent pouvoir être intégrées au code de l'application au gré du concepteur du système.

II. 8) Outils d'aujourd'hui de simulation :

Il y a un grand nombre de logiciel de simulation sur le marché. Toutes les langues et représentations de modèle et certains outils sont de propriété industrielle. Il y a des outils d'usage universel tels qu'ACSL, MATLAB-Simulink qui sont basés sur la même méthodologie de modélisation, blocs d'entrée-sortie, comme dans l'effort d'étalonnage précédent, CSSL, de 1967.

Il est important d'accentuer que, à quelques exceptions, tous les paquets de simulation sont seulement forts dans un domaine et ils ne sont pas capables de modeler des composants dans d'autres domaines raisonnablement. C'est un inconvénient important puisque les systèmes techniques deviennent de plus en plus hétérogènes avec des composants de beaucoup de domaines de technologie. JMAG fournit la technologie du dernier cri pour entourer des phénomènes physiques étendus exactement dans le modèle de simulation. JMAG précisent des appuis d'analyse supérieurs dans la conception électromécanique. En outre, l'épice, dans ses différentes versions (PSpice, HSpice, etc.) est le logiciel de simulation principal dans le domaine de l'électronique et de l'électrotechnique.

PSIM est un autre paquet de simulation spécifiquement conçu pour l'électronique de puissance et la commande de moteur.

En conclusion, indépendamment du logiciel des processus industriel, et des logiciel de simulation il existe une autre ligne de simulateurs : Les simulateurs d'API. Dans ce domaine, les lotisseurs peuvent trouver des progiciels comme PC-SIM qui a une bonne option entre d'autres dispositifs pour l'étude de programmation d'API, parce qu'il a un environnement graphique très bon. Un autre progiciel SIMTSX qui permet de debuggé certains marque commerciale d'API sans la présence de la machine ou du processus et laisse valider les programmes de API et les fonctions de commande associées, et entraine la commande et l'entretien des operateurs avant la prise en charge de l'équipement sur l'emplacement. Quelques outils basés sur PC de simulation de processus ont été développés, on utilisant les technologies des microcontrôleurs qui sont conçus pour travailler avec n'importe quel type d'API. La modélisation des API peut être réduite à l'émulation du programme de gestion d'API mais beaucoup d'approches peuvent être encore adoptées concernant le programme des API.

II. **9) Conclusion** :

Les modèles servent à représenter et déterminer le comportement de systèmes et à accomplir ainsi au moins trois buts : Prévision, apprenant la nouvelle compression de règles et/ou de données.

Plusieurs auteurs ont développé des paquets spécifiques pour la vérification du programme des API. Souvent ces programmes vérifient seulement la structure du programme sans vérifier si elle atteint les objectifs désirés de commande. L'autre approche est la génération du programme d'API avec d'autres formalismes, tels que des réseaux de Pétri, des diagrammes d'état. Si le formalisme original est sans erreur ceci pourrait être un outil valable pour développer les programmes d'API. D'autres auteurs ont développé des progiciels pour traduire les programmes d'API au code de DSP, de sorte qu'il puisse être employé dans le matériel API mais aucune de ces approches n'est prévue pour être employée dans l'environnement de Matlab/ Simulink.

Chapitre III

Méthodologie de traduction de programme de gestion d'un API dans l'environnement de Matlab/Simulink.

III. 1) Introduction :

La littérature appropriée identifie que l'essai pratique d'un contrôle du processus de cycle d'automation et de commande par un Automate programmable(API) est un problème bien connu. Il y a plusieurs solutions qui peuvent être mises en application, comme les batteries à commutateurs et les interfaces de machine humaines (HMI), commande de surveillance et acquisition de données (SCADA), ou des outils de simulation. L'utilisation des LED et des commutateurs est extrêmement embrouillant et inintéressante, Cette approche, seulement valide quand de petits processus sont considérés. Quelques systèmes de HMI et de SCADA permettent ces caractéristiques mais il sont très cher et non prévu à cette fin et considère habituellement des protocoles de propriété.

L'utilisation de Matlab/Simulink n'a pas été une approche ordinaire pour enseigner l'automation industrielle et les Processus commandés par des API.

Dans ce chapitre nous allons présenter un outil qui peut être utilisé pour mettre en application le programme de gestion d'un API dans l'environnement de Matlab/Simulink tout on Supposant que les modèles des processus industriels sont mis en application dans Matlab/Simulink.

L'idée est de considérer le programme de gestion d'un API comme bloc de fonction dans l'environnement de Matlab/Simulink qui commandera le modèle du processus industriel tant que la simulation fonctionne. L'objectif principal du travail décrit en ce chapitre est de traduire automatiquement le programme de gestion d'un API, écrit comme liste d'instruction dans la langue de logiciel de Matlab/Simulink.

III. 2) Présentation de Matlab/Similink :

III. **2.1) MATLAB** : est un logiciel commercial de calcul numérique/scientifique, et de visualisation et programmation très performant et convivial développé par la société « The MathWorks Inc ». Notez que ce n'est cependant pas un logiciel de calcul algébrique ou symbolique (pour cela, voir les logiciels commerciaux Mathematica ou Maple, ou le logiciel libre Maxima).

Le nom de MATLAB vient de MATrix LABoratory. Les éléments de données de base manipulés par MATLAB étant des matrices (pouvant bien évidemment se réduire à des vecteurs et des scalaires) qui ne nécessitent ni déclaration de type ni dimensionnement. Contrairement aux langages de programmation classiques (scalaires), les opérateurs et

fonctions MATLAB permettent de manipuler directement et interactivement ces données matricielles, rendant ainsi MATLAB particulièrement efficace en calcul numérique, analyse et visualisation de données en particulier.

Mais MATLAB est aussi un environnement de développement ("progiciel") à part entière ; son langage d'assez haut niveau, doté notamment de structures de contrôles, fonctions d'entrée-sortie et de visualisation 2D et 3D et des outils de construction d'interface a usage graphique (GUI)... permet à l'utilisateur d'élaborer ses propres fonctions ainsi que de véritables programmes ("M-files") appelés scripts vu le caractère interprété de ce langage.

MATLAB est disponible sur tous les systèmes d'exploitation standards (Windows, GNU/Linux, MacOS X...). Le champ d'application de MATLAB peut être étendu aux systèmes non linéaires et aux problèmes associés de simulation avec le produit complémentaire **SIMULINK**. Les capacités de MATLAB peuvent en outre être enrichies par des fonctions spécialisées regroupées au sein de dizaines de "**toolboxes**" (boîtes à outils qui sont des collections de "M-files") couvrant des domaines très variés:

- analyse de données
- statistiques
- mathématiques symboliques (accès au noyau Maple V)
- analyse numérique (accès aux routines NAG)
- traitement d'image, cartographie
- traitement de signaux (et du son en particulier)
- acquisition de données et contrôle de processus (gestion ports série/parallèle, cartes d'acquisition)
- instrumentation
- Logique floue
- Finance

etc...

Une interface de programmation applicative(API) rend finalement possible l'interaction entre MATLAB et les environnements de développement classiques (exécution de routines C ou Fortran depuis MATLAB, ou accès aux fonctions MATLAB depuis des programmes C ou Fortran).

Ces caractéristiques et d'autres encore font aujourd'hui de MATLAB un standard incontournable en milieu académique dans les différents domaines de l'ingénierie et la recherche scientifique.

III. 2.2) Simulink :

Simulink est une interface graphique permettant de décrire des graphes flots de données dont les blocs sont des fonctions décrites avec Matlab. Ces graphes sont proches de la représentation schéma-bloc utilisé généralement par l'automaticien pour représenter les lois de commande d'un système automatisé. Il permet de spécifier et paramétrer rapidement un prototype grâce à des simulations prenant à la fois en compte le comportement continu du processus et le comportement discret du système temps réel évitant ainsi de longues phases de tests et il intègre une importante bibliothèque de blocs prédéfinis (gain, intégrateur, filtre...etc.).

Il est possible de construire hiérarchiquement de nouveaux blocs, soit en utilisant des blocs existants, soit en créant de nouveaux blocs à partir de fonctions Matlab tel que les nouveaux blocs créés par les utilisateurs peuvent être regroupés dans des librairies pouvant être réutilisées dans d'autres applications.

III. 2.3) Opérateurs logiques :

Les opérateurs logiques (fonctions) ont pour arguments des expressions logiques et retournent les valeurs logiques vraies (**1**) ou fausses (**0**). Les opérateurs logiques principaux sont les suivants :

Opérateur ou fonction	**Description**
~ expression **not** (expression) **!** expression	-**Négation** logique (rappel: NON 0 => 1 ; NON 1 => 0)
expression1 **&** expression2 **and** (expression1, expression2)	-**ET** logique. Si les expressions sont des matrices, retourne une matrice (rappel: 0 ET 0 => 0 ; 0 ET 1 => 0 ; 1 ET 1 => 1)
expression1 **&&** expression2	- **ET** logique "short circuit". A la différence de **&** ou **and,** cet opérateur est plus efficace, car il ne prend le temps d'évaluer expression2 que si expression1 est vraie. En outre: . MATLAB: n'accepte que les expressions sous forme de matrices.

expression1 | expression2 **or** (expression1, expression2)	**-OU** logique. Si les expressions sont des matrices, retourne une matrice (rappel: 0 OU 0 => 0 ; 0 OU 1 => 1 ; 1 OU 1 => 1)
expression1 || expression2	**-OU** logique "short circuit". A la différence de | ou **or,** cet opérateur est plus efficace, car il ne prend le temps d'évaluer expression2 que si expression1 est fausse. En outre: . MATLAB: n'accepte que les expressions sous forme de matrices
xor (expression1, expression2)	- **OU EXCLUSIF** logique (rappel: 0 OU EXCL 0 => 0 ; 0 OU EXCL 1 => 1 ; 1 OU EXCL 1 => 0)

- les principales techniques d'**affectation** de vecteurs (usage des crochets **[ ])** et d'**adressage** de ses éléments (usage des parenthèses **( )**)

III. 2.4) Principes de base relatifs aux fonctions :

On pourrait déclarer plusieurs fonctions dans un M-file, mais seule la première est accessible de l'extérieur (les suivantes ne pouvant être appelées que par la première).

- le nom du M-file doit être rigoureusement identique au nom de la première fonction, et le fichier doit commencer, en 1ère ligne, par la déclaration de la première fonction.

-La déclaration d'une fonction définit le nom de la fonction et ses arguments arg_entree et arg_sortie (séparés par des virgules) selon la syntaxe suivante :

function [arg_sortie**, ...]=**nom_fonction **(**arg_entree**, ...)** (les crochets ne sont pas obligatoires s'il n'y a qu'un arg_sortie) Se succèdent donc, dans cet ordre :

1. déclaration de la fonction (ligne ci-dessus)

2 .lignes de commentaires (commençant par le caractère **%)** qui décrivent la fonction pour le système d'aide en-ligne, à savoir :

• la "H1-line" (qui sera retournée par la commande lookfor mot-clé)

• les lignes du texte d'aide (qui seront affichées par la commande help nom_fonction)

3. éventuelle déclaration de variables globales ou statiques

4. instructions proprement dites de la fonction (qui va affecter les variables arg_sortie)

5. et instruction(s) return signalant le(s) point(s) de sortie de la fonction.

III. 3) Méthodologie de traduction de l'API/Matlab :

Afin de comprendre entièrement les avantages de la méthodologie proposée de traduction, on suppose que le processus industriel est déjà modelé dans l'environnement de Matlab/Simulink, comme présenté dans fig.7.

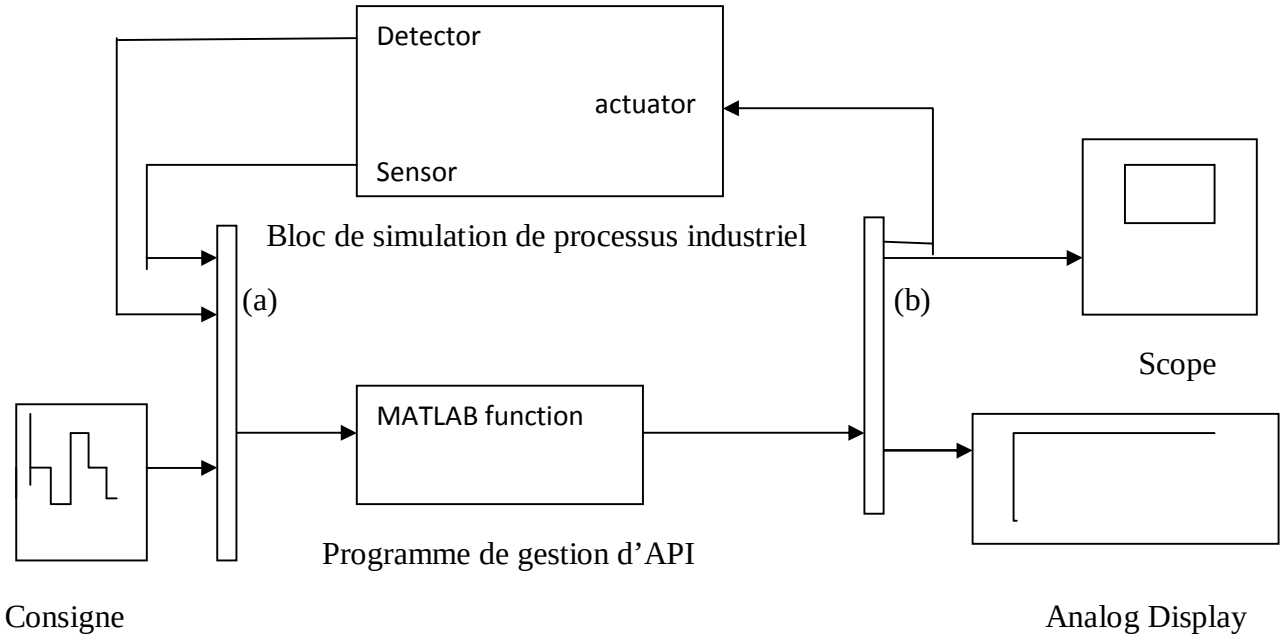


Fig.7. Interaction d'opération d'API et le processus industriel

Les processus industriel commandés sont simulé dans un bloc de Matlab/Simulink appelé « bloc de simulation de processus industriel ». Les sorties de ce bloc sont les sondes et les signaux des détecteurs du processus qui seront employés comme entrées au bloc de Matlab/Simulink appelé « programme de gestion d'API ». Ce bloc émulera l'opération de l'API et ses sorties correspondront aux sorties de l'API qui seront relié à l'entrée des déclencheurs dans le bloc de simulation de processus industriel.

Le bloc de programme de gestion de l'API est la clef de voûte de la méthodologie proposée, Ce bloc de fonction est un m-dossier de Matlab et il émulera l'opération cyclique de l'API.

Afin d'établir automatiquement ce bloc, les procédures suivantes doivent être accomplies :

1. prendre un processus commandé par un API déjà modelé dans un bloc existant dans le bloc de simulation de processus industriel développé dans l'environnement de Matlab/Simulink .

2. Considérer les caractéristiques fonctionnelles de ce processus commandé par un API.

3. Considérer un API spécifique pour commander le processus.

4. Élaborons le programme de gestion respectif de l'API selon les caractéristiques fonctionnelles considérées, on employant par exemple la méthodologie de GRAFCET.

5. Noter le programme de gestion de l'API en utilisant l'un des langages de programmation du fournisseur.

6. Sauvegarder le programme de gestion de l'API comme langage de programmation orienté vers le texte dans un dossier des textes.

7. exécuter le paquet développé de traduction de « API-Matlab/Simulink » afin de convertir le programme de gestion de l'API en langue de Matlab/Simulink (le bloc de fonction de m-dossier de Matlab/Simulink « programme de gestion de l'API » devrait être automatiquement produit).

8. Examiner le programme de gestion développé de l'API avec le modèle considéré du processus commandé par l'API (bloc de simulation de processus industriel).

9. Élaborent les adaptations dont on a besoin afin de mettre la commande de programme pour travailler correctement.

Avant de traduire automatiquement le programme de gestion de l'API dans la langue de Matlab/Simulink Le paquet proposé de traduction de « API-Matlab/Simulink », exigera les informations suivantes :

1. Type d'API ;

2. Le nombre d'entrées /sorties de l'API ;

3. Dossier de programme de gestion de l'API pour la traduction.

III. 3.1) Type d'API :

Le choix du type de l'API est essentiel pour établir les règles de traduction en conséquence à la syntaxe du programme du fabricant. Bien qu'elles soient toutes basées sur la logique

booléenne, chaque fabricant d'API développe sa propre syntaxe de programmation. De cette façon le paquet de traduction devrait connaître le fabricant de l'API afin d'appliquer les règles adéquate à la traduction.

III. 3.2) Le nombre d'entrées/sorties de l'API :

Le nombre d'entrées/sorties de l'API définit clairement les arguments de fonction de Matlab/Simulink « programme de gestion de l'API ». Cette fonction sera responsable d'exécuter le programme de gestion de l'API dans l'environnement de Matlab/Simulink, qui sera créée comme m-dossier des textes.

-la structure du programme qui illustre ces entrées/sorties est représenté de la sorte qui suit (fonction1) :

```
Function [output]=PLC control program (di1,…,di'n',…,ai1,…,ai'n')

……

(Programme de gestion de l'API dans le langage de Matlab/Simulink)        (1)

…...

Output=[do1,…,do'p',…,ao1,…,ao'p']
```

[**FONCTION 1**]

- Les di'1' à di 'n' dénotent les entrées numériques de l'API, et ai'1' à ai'm' dénotent les entrées analogiques de l'API, do'1' à do'p' dénotent les sorties numériques de l'API et ao'1' à ao'q' dénotent Les sorties analogiques de l'API. n, m, p et q dénotent, respectivement, le nombre d'entrées numériques et analogiques et le nombre de sorties numériques et analogiques. Il est important de noter que n+m définissent la dimension du bloc de Mux (a) dans fig. 7. De même p+q définissent la dimension du bloc de Demux (b) dans fig. 7.

III. 3.3) Dossier de programme de gestion de l'API pour la traduction :

Le programme de gestion d'un API peut être écrit dans un ensemble large de langages de programmation. Les modèles des logiciels des API et l'ensemble référé de langues sont établis et définis dans la norme 61131-3 du CEI. Chaque fabricant offre différents genres de langages

de programmation appropriés, ayant pour résultat un ensemble typique de cinq langages de programmation :

- Liste d'instruction : ressemble Très étroitement à l'assembleur et il peut être considéré comme langage de programmation de bas niveau des textes.
- Langage Ladder : Historiquement dérivé du câblage des circuits électriques et il ne permet pas la complexité et la modularité.
- Programmation par GRAFCET : c'est un langage de programmation graphique comme les réseaux de pétri qui structure les éléments internes de l'API dans des étapes (liées aux actions) et des transitions (entre les étapes).
- Schéma de bloc fonctionnel: Un autre langage graphique où les blocs de fonction du processus traites les multiples signaux de l'API.
- Texte structuré : Dérivé du langage de programmation de Pascal, c'est un langage de programmation à niveau élevé qui permet la complexité et la modularité.

-La représentation typique d'un programme de gestion d'API est caractérisée par l'utilisation d'un langage graphique connu sous le nom de diagramme d'échelle (ladder diagram). Cependant, presque chaque paquet des logiciels de programmation d'API permet l'utilisation des langages de programmation orientés vers le texte. D'ailleurs, ils permettent la conversion automatique entre les diagrammes d'échelle et les langages de programmation orientés vers le texte, et vice-versa.

-La méthodologie proposée de traduction considérera que le programme de gestion d'API est écrit comme langage de programmation orienté vers le texte, dans un dossier standard des textes. Ceci ne représente pas un problème parce que, comme référé, presque chaque paquet de logiciel de programmation d'API permet la sauvegarde du programme de gestion des API dans ce format.

La figue 8 ci-dessous montre le dossier texte d'un simple programme de gestion d'un API de Siemens.

```
//
// COMMENTAIRES DE TITRE DE PROGRAMME
//
RÉSEAU 1
LD      I0.0
A       I0.1
LD      I0.2
A       I0.3
OLD
=       Q0.0
//
RÉSEAU 2
LD      I0.4
LD      I0.5
CTU     C5, +6
//
END
```

Fig.8. Dossier standard des textes de programme de gestion d'API

-Le paquet de traduction de programme de gestion d'un API est un outil logiciel développé dans un Visual Basic, qui convertit automatiquement le dossier des textes de programme de gestion de l'API en m-dossier correspondant dans l'environnement de Matlab/Simulink. Ce m-dossier, contenant le programme de gestion de l'API décrit dans la langue de Matlab/Simulink organisé comme la fonction de Matlab définie en (fonction 1).

Ainsi Sachant le nombre d'entrées/ sorties de l'API, l'outil de conversion établit le nombre correct d'arguments d'entrée/sortie pour la fonction (1). La traduction du programme de gestion de l'API lui-même repose sur un ensemble de règles de traduction qui s'appliquée à l'ensemble de la liste d'instruction de l'API (la liste qui se présente dans le dossier de programme de gestion de l'API pour la traduction).

III. 3.4) Traduction de différents type d'instruction d'un API dans Matlab/Simulink :

Une liste pleine d'instruction d'API peut être en gros divisée en :

- Booléen
- Comparaison
- Sortie
- Temporisateur
- Compteur
- Maths
- Incrément/décrément
- Déplacement/décalage
- Commande de programme
- Autre

-considérant une liste d'instruction d'API de Siemens. Les instructions booléennes seront traduites dans la langue de Matlab/Simulink en utilisant des fonctions booléennes standard de Matlab, comme présenté dans le tableau 1, où Ix.y dénote une entrée numérique et Q x.y dénote une sortie numérique, x et y sont, respectivement, l'octet et le bit d'entrée-sortie numérique considérée. En outre, le do_g est une variable de Matlab dénotant la sortie numérique g et le di_h est une variable de Matlab dénotant l'entrée numérique h. L'état booléen VRAI sera représenté dans l'environnement de Matalb par le `1' et l'état booléen FAUX par le `0'. Des combinaisons de diverses instructions booléennes seront converties en utilisant les règles ci-dessous (Un exemple est montré sur la dernière rangée du tableau 1).

Instruction booléen	Instruction d'API	Traduction Matlab/Simulink
AND	LD Ia.b A Ic.d =Qe.f	do_g=di_h&di_i

OR	LD Ia.b O Ic.d =Qe.f	do_g=di_h|di_i
NOT	LDN Ia.b =Qc.d	do_g=~di_i
combinaison de diverse instruction booléenne	LD I0.0 A I0.1 LD I0.2 OLD =Q0.0	do_1= (di_1&di_2) |di_3

Tableau 1: traduction des instructions booléenne

- Les instructions de maths d'un API sont habituellement booléennes permises. Ceci implique l'utilisation de la fonction 'if' de Matlab afin de représenter leur comportement. On considère comme exemple, les nombres entiers d'un API ajoutant les instructions présentées dans le tableau 2 avec AIW0 et AQW0 qui représentent, respectivement, l'entrée analogique de l'API et la sortie analogique de l'API. La variable ao_1 est une variable de Matlab dénotant la première sortie analogique et ai_1 est une variable de Matlab dénotant la première entrée analogique. De l'autre part, les programmes de gestion des API emploient souvent les drapeaux internes (bit et les variables internes de la mémoire) pour représenter des états ou pour stocker les valeurs analogiques. Toutes les fois que le paquet de traduction trouve une mémoire interne d'un API il lui assigne automatiquement une variable de Matlab/Simulink.ces variables sont habituellement dénotés comme m ou v, car ils sont numériques ou analogiques. L'instruction de multiplication (MUL) d'un API comporte souvent l'utilisation d'une mémoire interne auxiliaire, comme présenté dans le tableau 2, où l'instruction de MOVW est également employée (entrant la valeur d'un mot variable de 16 bit dans des autres).notons aussi que la variable interne VD d'un API se rapporte à un mot à 32 bits.

Instructions	Instructions d'un API	Traduction Matlab/Simulink
Incrémentation	LD I0.0 +I AIW0,AQW0	If di_1 do_1=ai_1+ao_1 End

Multiplication	LD I0.0
	MOVW +6,VD4
	MUL +9,VD4
		If di_1
		V_4=+6
		V_4=+9*v_4
		End

Tableau 2: traduction des instructions non-booléenne

- Les instructions des compteurs des API (CTUD-compteur incrément/décrément) peuvent exiger plusieurs entrées booléennes : une pour le compteur d'incrémentation, l'autre pour le compteur de décrémentation (si c'est le cas) et tout autre pour la réinitialisation du compteur. Puisque le compte est seulement effectué sur l'incrémentation de l'entrée booléenne, le Matlab/Simulink devrait tenir compte de l'état précédent de cette entrée booléenne.

- Le tableau 3 présente un exemple d'un compteur, où di-1-prev est une variable de Matlab dénotant l'état précédent de la variable di-1. L'état précédent signifie l'état dans le cycle précédent de programme de gestion de l'API. Dans cet exemple, pour l'atteinte du compte 4 le compteur change son état booléen à chaque vérification. Dans l'environnement de Matlab c-10 dénote l'état booléen du compteur numéro 10, et c-10-value dénote la valeur de compte du même compteur.

Instruction	Instruction de l'API	Traduction dans Matlab/Simulink
Compteur	LD I0.0 // increment LD I0.1 // decrement LD I0.2 // reset du compteur CTUD C10, +4	If di_1 &~di_1_prev C_10_value=c_10_value+1 End If di_2&~di_2 C_10_value=c_10_value-1 End If di_3 C_10_value=0 C_10=0 End If c_10_value>=+4 C_10=1 End

Tableau 3: traduction des instructions d'un compteur

- Les instructions des temporisateurs d'un API, c'est comme sur les temporisateurs à retard qui exigent habituellement seulement une entrée numérique pour le compte comme présenté sur la figure 9. Le temporisateur (T33 dans l'exemple) travail toutes les fois que l'entrée numérique respective (I2.0 dans l'exemple) est valide et elle devient une entrée booléenne vrai quand elle atteint son temps de préréglage (3 secondes dans l'exemple).

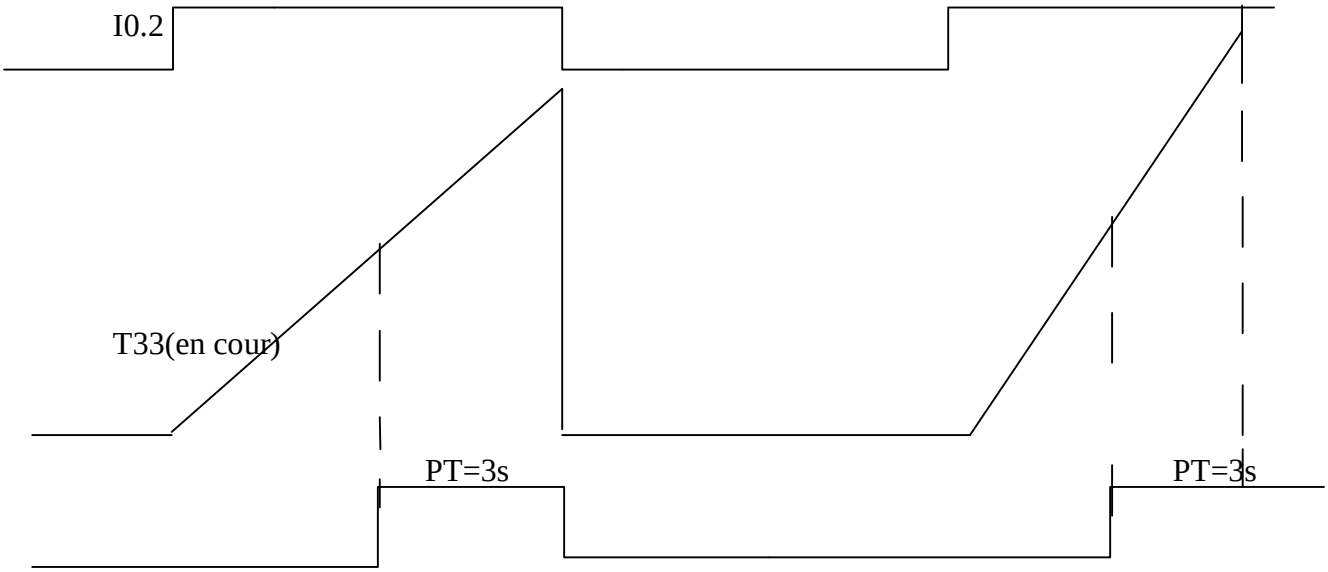


Fig.9 .les opérations d'un temporisateur à retard.

- La traduction du temporisateur précédent est présentée dans le tableau 4. Toutes les fois que le temporisateur commence (son entrée numérique correspondante di_20 étant VRAI et le temporisateur n'a pas encore commencé) le temps de préréglage (t33_start) est ajouté à la valeur réelle d'horloge (clock_in) établissant ainsi le temps d'arrêt de temporisateur (t33_end_time). Après atteinte de ce temps d'arrêt la valeur logique du temporisateur (t1_bin) devient vraie et le temporisateur est remis en marche quand son entrée booléenne correspondante devient fausse.

Instruction	Instruction de l'API	Traduction dans Matlab/Simulink
Temporisateur	LD I2.0 TON T33, 3	% Start timer If di_20 &t33_start==0 T33_start=3; T33_end_time=clock_in+3; End % Timer ON If t33_start&clock_in>=t33_end_time

		T33_bin=1; End % Timer OFF If ~di_20 T33_bin=0 ; T33_start=0 ; End

Tableau 4: traduction des instructions d'un temporisateur

III. 4) Exemples d'application :

III. 4.1) machine de scierie :

Comme un premier exemple d'application nous allons considérer des booléens purs. On suppos automatiser la scierie présentée dans la fig. 10. Après pressurage du BOUTON DÉMARRER la machine de découpage se déplace vers la droite. La lame doit être reliée avant qu'elle atteigne les notations et elle s'arrêt après qu'elle les ait découpées. Actuellement la lame devrait augmentée jusqu'à ce qu'elle atteigne la position supérieure ainsi le mouvement ascendant devrait s'arrêter et la machine doit se déplacer vers la gauche jusqu'à sa position originale, où la lame devrait commencer a s'abaissée. Dans fig. 10 on représente également les commutateurs de limite (dénotés Si) qui sont employés pour commander les actions de la machine.

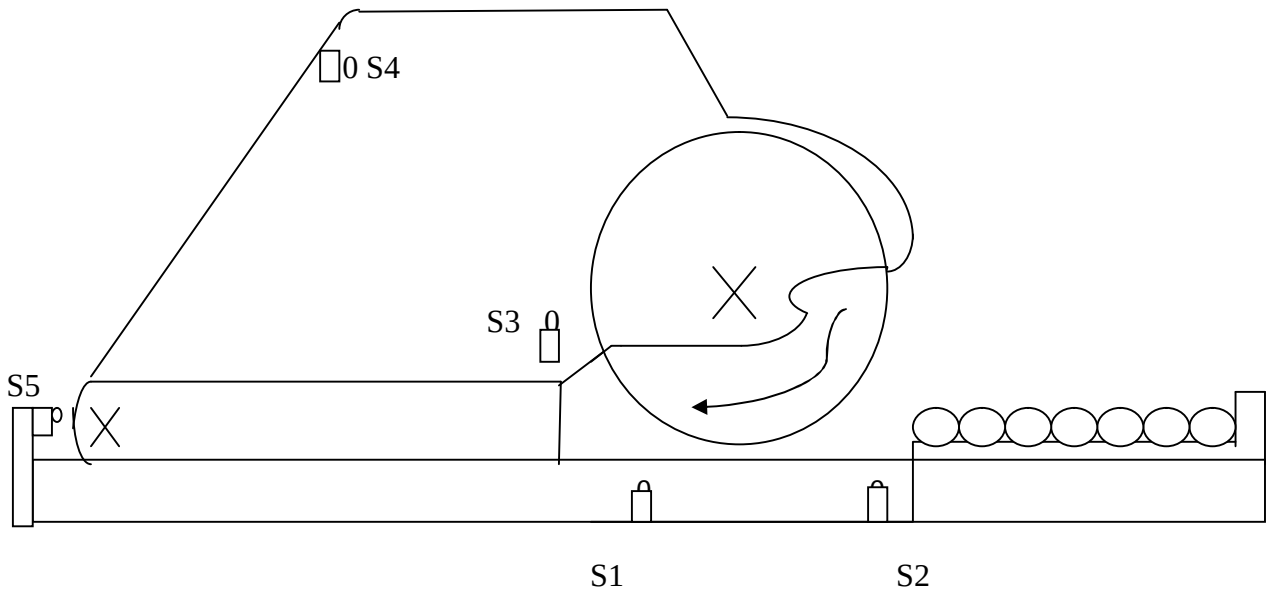


Fig.10. machine de scierie

Tableau 5 : présente la liste des entrées / sorties du processus dans l'API. Il montre chaque description de signal, son adresse dans l'API et l'assignement correspondant de Matlab/Simulink. Pour ce processus automatisé avec 6 entrées numériques et 5 sorties numériques. Un API de SIEMENS standard S7-200 avec 8 entrées numériques et 6 sorties numériques est considéré pour exécuter la commande du processus désiré.

Description du signal	Adresse d'I/O de l'API	Notes	Assignement correspondant de Matlab/Simulink
Start (démarrer)	I0.0	bouton poussoir	di1
Commutateur de scie en position ON	I0.1	commutateur de limite-S1	di2
Commutateur de scie en position OFF	I0.2	Commutateur de limite-S2	di3
Machine abaissée	I0.3	commutateur de limite-S3	di4
Machine augmenté	I0.4	commutateur de limite-S4	di5
machine à la position du début	I0.5	commutateur de limite-S5	di6
Déplacement à gauche	Q0.0	Moteur M1-gauche	do1
Déplacement à droite	Q0.1	Moteur M1-droite	do2
Rotation de la scie	Q0.2	Moteur M2	do3
Montée de la machine	Q0.3	Moteur M3-vers le haut	do4
Descente de la machine	Q0.4	Moteur M3-vers le bas	do5

Tableau 5 : traduction des instructions booléenne

. L'algorithme de commande du processus a été élaboré en concordance avec le GRAFCET, (Graphe Fonctionnel de Commande Etape Transition) ou à SFC (diagramme séquentiel de fonction) présenté dans la figue 11.

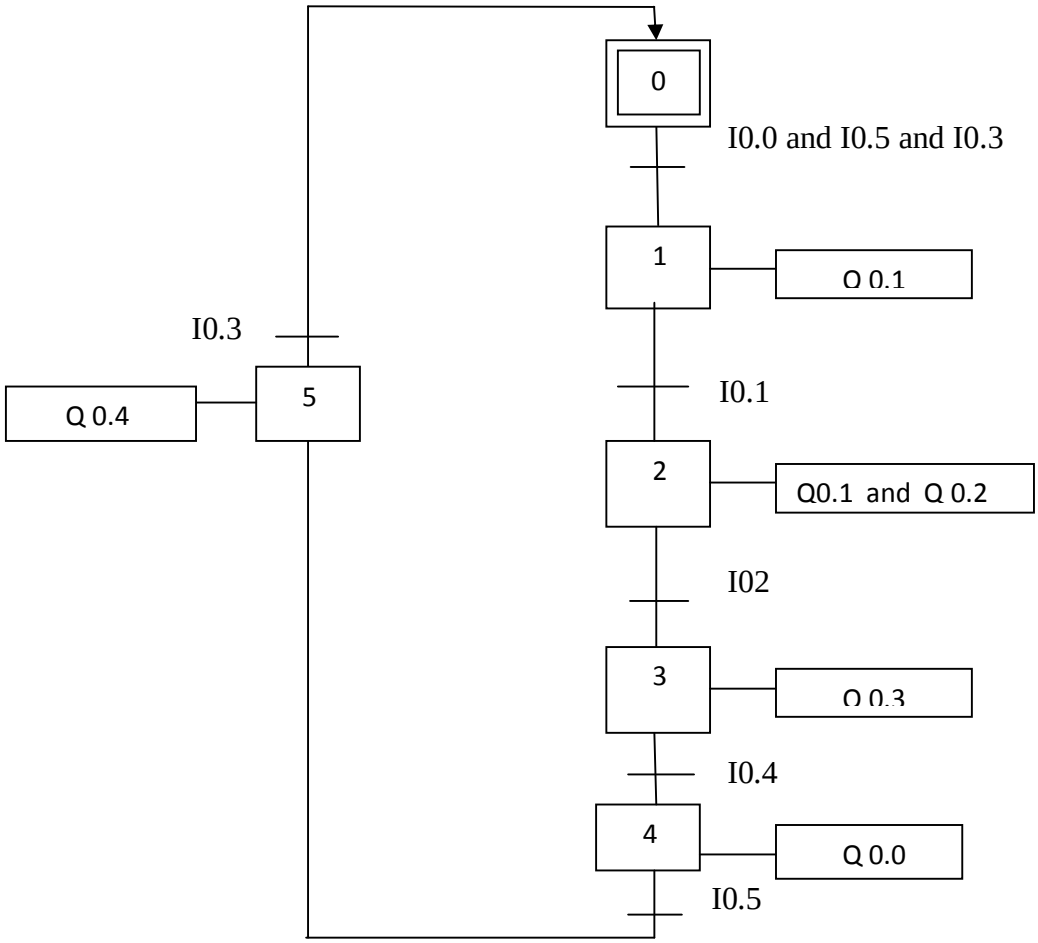


Fig.11. GRAFCET de scierie

De ce GRAFCET le programme de gestion de l'API est écrit comme langage de programmation texte dans un dossier standard des textes .en appliquant le paquet de translation API-Matlab/Simulink développé à ce dossier des textes, il produit automatiquement le bloc de fonction de m-dossier de Matlab/Simulink «Programme de gestion d'API ». Le type d'API et le nombre d'entrée-sortie ont été également considérés en tant qu'arguments de paquet de traduction du procédé. Le m-dossier obtenu, contient la version de Matlab/Simulink du programme de gestion développé et il est présenté dans la fonction 2, où le mi dénote chaque étape du GRAFCET et mi_a sa valeur booléenne précédente.

```
function[output]=cpu222(di1,di2,di3,di4,di5,di6,di7,di8,ai1,ai2,ai3,ai4,clock_in)

global m0 m0_a m1 m1_a m2 m2_a m3 m3_a m4 m4_a m5 m5_a

% Initialisation des sorties

aux=0;do1=aux; do2=aux; do3=aux; do4=aux; do5=aux; do6=aux ; ao1=aux ; ao2=aux ;
ao3=aux ; ao4=aux ;

% Évolution des étapes du GRAFCET

m0= (m5&di3)|(m0&~m5);

m1= (m0&di1&di6&di4)|(m1&~m0);

m2= (m1&di2)|(m2&~m1);

m3= (m2&di3)|(m3& ~ m2);

m4= (m3&di5)|(m4&~m3);

m5= (m5&di6)|(m5&~m4);

% Génération des sorties                                    (2)

do1=m1;

do2=m2;

do3=m3;

do0=m4;

do4=m5;

% Actualisation des étapes précédente du GRAFCET

m0_a=m0;

m1_a=m1;

m2_a=m2;

m3_a=m3;
```

```
m4_a=m4;

m5_a=m5;

output= [do1 do2 do3 do4 do5 do6 ao1 ao2 ao3 ao4];
```

[FONCTION 2]

III. 4.2) Reservoir d'eau :

Dans ce deuxième exemple nous considérons un réservoir à eau avec écoulement aléatoire de l'eau à l'entrée (y compris les opérations d'un temporisateur et celle d'un compteur) , dont de niveau devrait être gardé entre 4 et 5 mètres de taille (fig.12.) afin d'accomplir ce but une valve électrique de sortie (avec un débit de 8 l/sec) est commandée par un API. L'API contrôle cette valve selon l'information qui provient des détecteurs à deux niveaux (installés à la taille de 4 et 5 mètres respectivement). Dans cet exemple on désire également compter le nombre de fois que la valve de sortie est enclenchée. Toutes les fois que ce nombre atteint 8, deux types d'alarmes devraient être produise : un signal de lumière et un vibreur. Le vibreur, cependant, devrait seulement être activé une seconde après que le compteur a atteint 8 fois. En plus, pour l'essai, un signal "Reset" externe est considéré à 3 et 12 secondes.

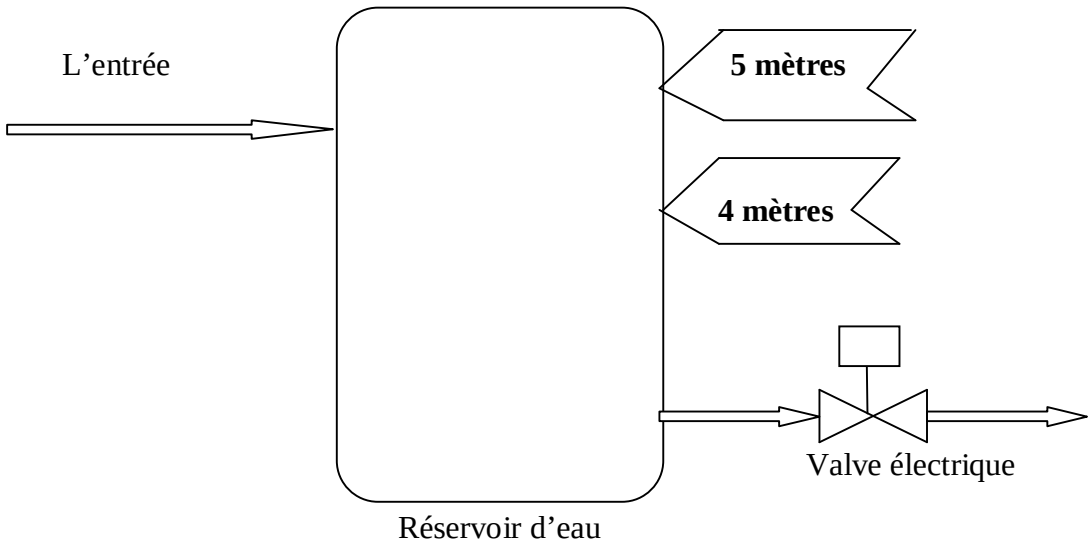


Fig.12. commande de niveau du réservoir d'eau.

La figue 13 ci-dessous : présente l'environnement de Matlab/Simulink pour cette application. Le sous-système « système de réservoir d'eau » contient le modèle simulé du réservoir d'eau, avec ses détecteurs et déclencheurs. Le sous-système « API » contient la version de Matlab/Simulink de la commande de programme développée de l'API. Réellement ce sous-système simule l'action de l'API sur le processus. Les entrées de l'API sont : « Commutateur

maximum de niveau d'eau 'KMAX' », « commutateur minimum de niveau d'eau 'KMIN' », et « reset externe du compteur ». Les deux premières entrées sont les signaux fournis par les détecteurs à deux niveaux installés dans le réservoir d'eau. Les sorties de l'API sont : « valve de sortie », « compteur=8 », «Compteur d'alarme » et « compteur de valve ». Cette dernière sortie est une sortie analogique avec le nombre de fois que la valve de sortie a été actionnée.

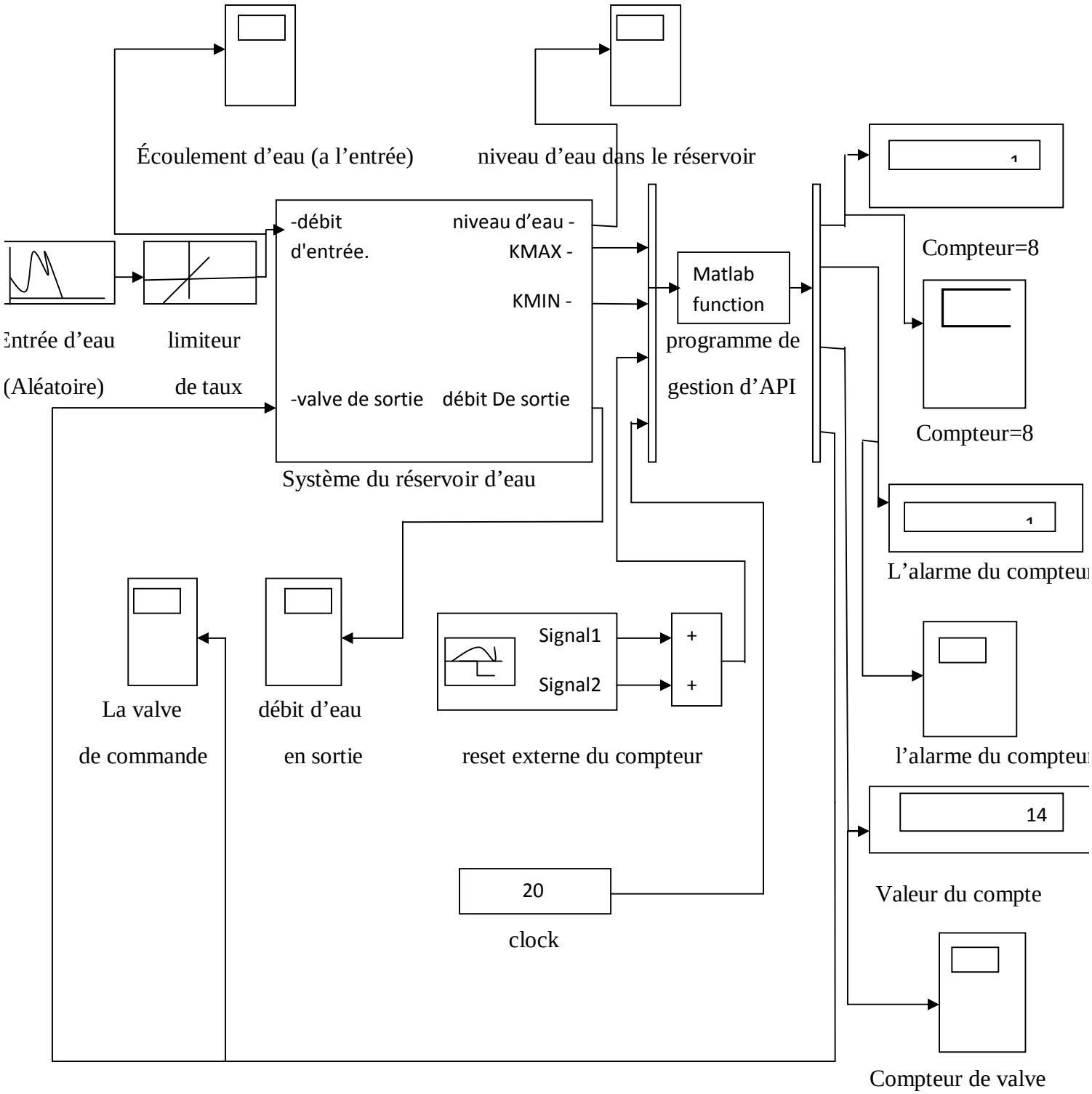
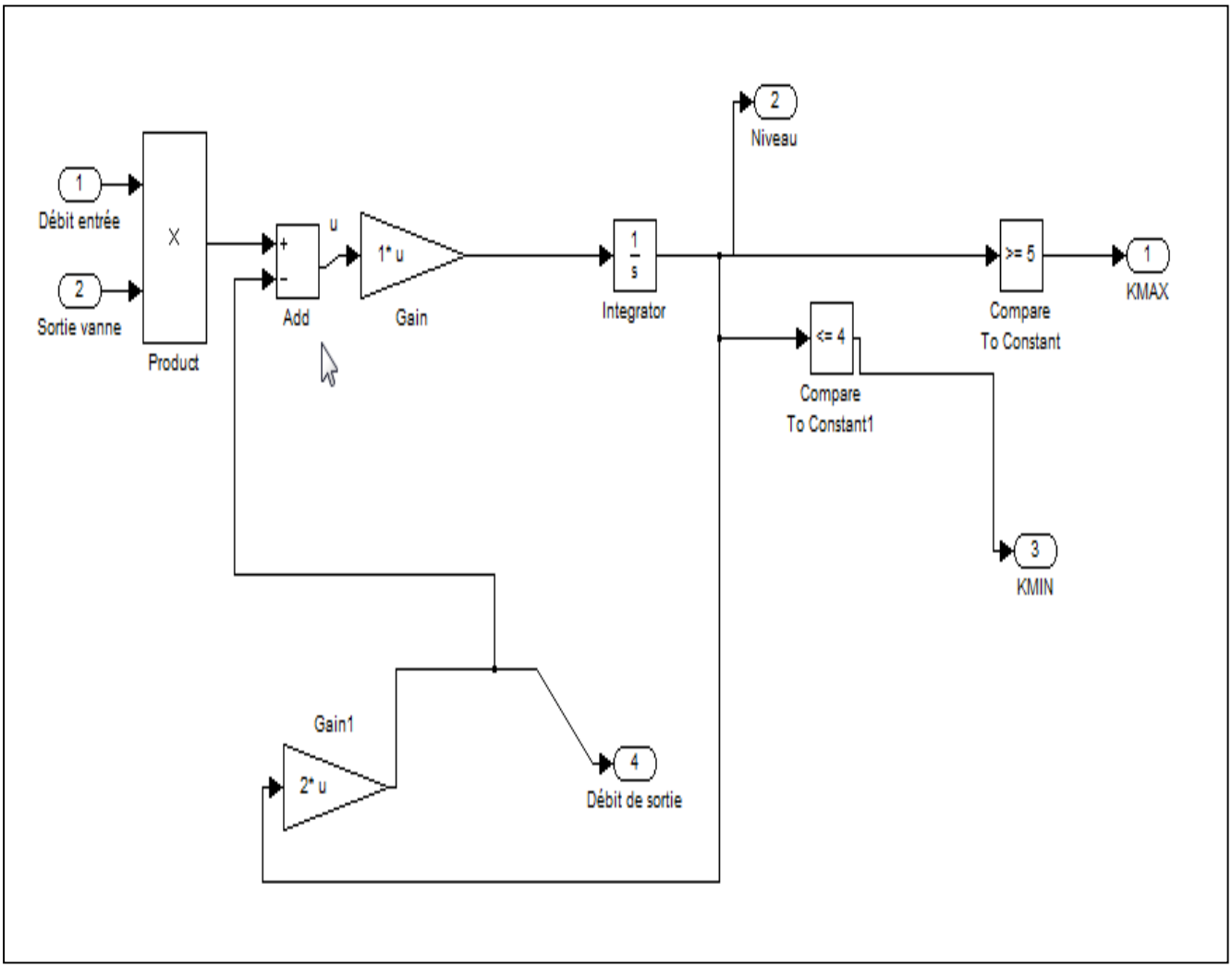


Fig.13. Modèle de Matlab/Simulink du système commandé de réservoir d'eau.

Tel que Le sousystème du réservoir d'eau se résume à cette équation : dh/dt=1/s*[Qe-Qs]

Avec : Qe :le débit d'entrée

Qs :le débit de sortie



[Le sousysteme du réservoir d'eau].

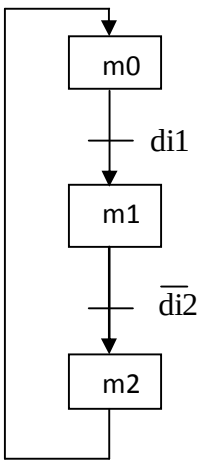
L'API choisi pour cette application était encore un SIEMENS S7-200 avec 8 entrées numériques, 4 entrées analogiques, 6 sorties numériques et 4 sorties analogiques. Ceci définit automatiquement le nombre et le type d'arguments d'entrée/ sortie du sous-système de Matlab/Simulink « programme de gestion de l'API ». Les arguments d'entrée sont les entrées de SIEMENS S7-200 et l'horloge, et les arguments de sortie de ce sous-système sont les sorties de SIEMENS S7-200.

Le tableau 6: définit les données de l'API et leurs représentations respectives dans l'environnement de Matlab/Simulink, où la description de chaque variable de données est également présentée.

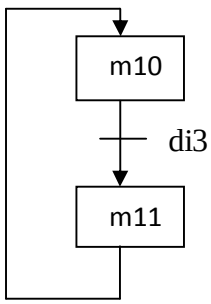
Description de signal	Signal d'API	Notes	Assignement dans Matlab/Simulink
Niveau maximum	I0.0	Commutateur	di1
Niveau minimum	I0.1	Commutateur	di2
Reset du compteur externe	I0.2	Bouton poussoir	di3
Valve	Q0.0	Valve électrique	do1
Alarme du signal lumineux	Q0.1	8 signaux opérations de valve	do2
Alarme du signal du vibreur	Q0.2	retard d'une seconde après le signal lumineux	do3
Valeur du compteur	QW10	Valeur du compteur	ao1
Compteur d'opérations de valve	C1	Compteur interne (CTU)	c1
Temporisateur du compteur d'alarme	T1	Temporisateur interne (on-delay)	t1
Etape 0 du GRAFCET 1	M10.0	Marque interne	m0
Etape 1 du GRAFCET 1	M10.1	Marque interne	m1
Etape 2 du GRAFCET 1	M10.2	Marque interne	m2
Etape 0 du GRAFCET 2	M20.0	Marque interne	m10
Etape 1 du GRAFCET 2	M20.1	Marque interne	m11

Tableau 6 : Représentation de Matlab/Simulink des données d'API du système de réservoir d'eau.-

Afin d'accomplir les caractéristiques désirées, deux GRAFCET sont considérer, un pour la commande de valve et un autre pour la remise du compteur.



GRAFCET1 : contrôle de valve



GRAFCET2 : remise du compteur

En plus, le compteur et le temporisateur ont été également considérés dans la liste d'instruction pour la signalisation. En utilisant les modalités de conversion précédemment décrites, la liste d'instruction du programme de gestion de l'API (modèle SIEMENS) est convertie en une fonction de Matlab/Simulink présentée en (fonction 3).

-La fonction (3) ci-dessous s'appelle dans le sous-système « programme de gestion de l'API » (voir la fig.7).

```
function[output]=cpu222(di1,di2,di3,di4,di5,di6,di7,di8,ai1,ai2,ai3,ai4,clock_in)

global m0 m0_a m1 m1_a m2 m2_a m3 m3_a m10 m10_a m11 m11_a c1 c1_bin t1_start t1_end_time t1_bin

% Initialisation des sorties

aux=0;do1=aux; do2=aux; do3=aux; do4=aux; do5=aux; do6=aux ;

ao1=aux ; ao2=aux ; ao3=aux ; ao4=aux ;

% grafcet 1(commande de valve)

m0= (m2)|(m0&~m1);

m1= (m0&di1)|(m1&~m2);

m2= (m1&~di2)|(m2&~m0);

% grafcet 2(reset du compteur)

m10= (m11)|(m10&~m11);

m11= (m10&di3)|(m11&~m10);

% compteur siemens (CTU)                                                    (3)

% actualisation du compteur

if m1==1 & ~m1_a

c1=c1+1;
```

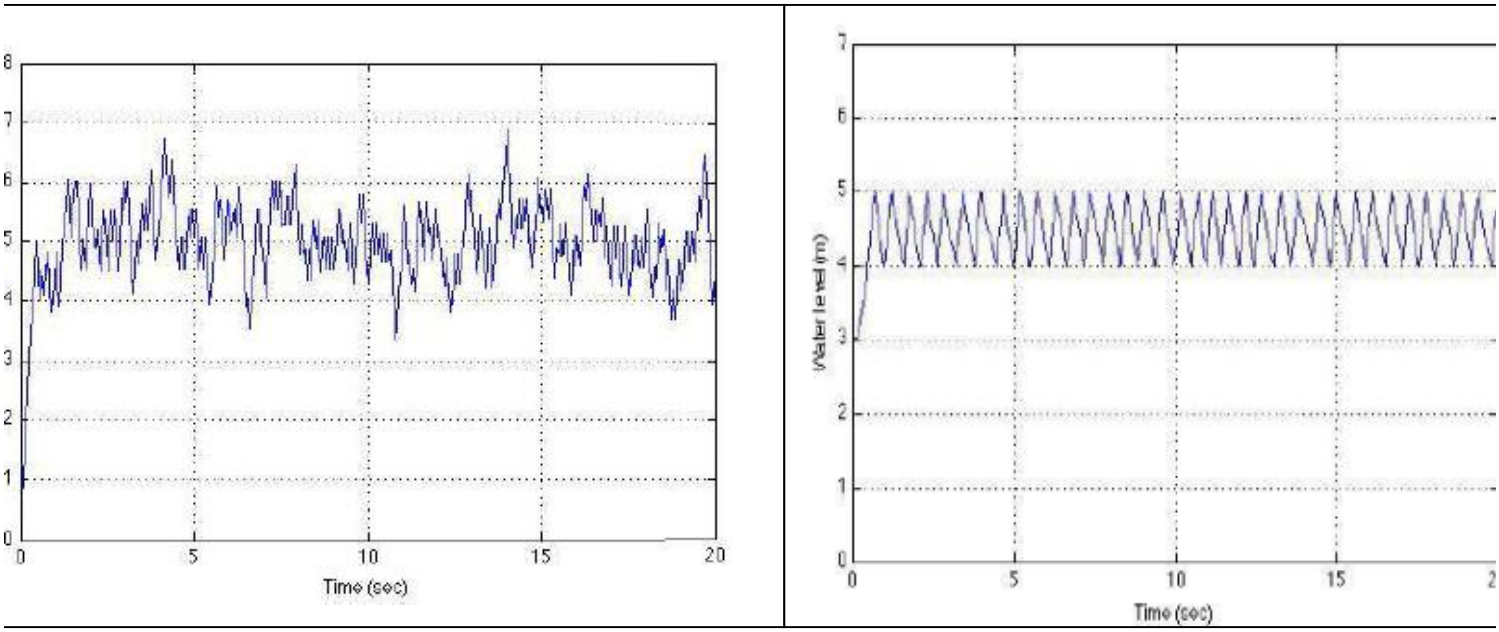
```
 end
% compteur en mode SET
 if c1==8
   c1_bin=1 ;
 end
% compteur en mode RESET
  if m11
    c1=0;
    c1_bin=0;
   end
% temporisateur siemens (on_delay)
% temporisateur en mode START
   if c1_bin & t1_start==0
 t1_start=1;
 t1_end_time=clock_in+1;
 end
% temporisateur en mode ON
  if t1_start & clock_in>=t1_end_time
   t1_bin=1;
   end
% temporisateur en mode OFF
  if ~c1_bin
  t1_bin=0;
```

```
  t1_start=0;
  end
% génération des sorties
 do1=m1 ;
 do2=c1_bin ;
 do3=t1_bin;
 ao1=c1;
 ao2=ai1;
% actualisation des étapes précédente du GRAFCET
  m0_a=m0;
  m1_a=m1;
  m2_a=m2;
  m3_a=m3;
  m10_a=m10;
  m11_a=m11;
  output=[do1 do2 do3 do4 do5 do6 ao1 ao2 ao3 ao4];
```

[FONCTION 3]

Toutes les fois que la simulation fonctionne cette fonction agit en tant qu'un API commandant le processus de circuit d'eau.

Fig. 14 : ci dessous présente l'écoulement d'eau aléatoire d'entrée et le niveau de réservoir d'eau. On peut voir que le programme de gestion développé (mis en application dans le sous-système de simulation de l'API) peut se conformer aux caractéristiques désirées, gardant le niveau d'eau entre 4 et 5 mètres de taille.



(a) écoulement d'eau d'entrée (b) niveau du réservoir d'eau

Fig. 14(a,b) : résultat de simulation sur Matlab/Simulink du système du réservoir d'eau Controller par l'API.

Fig.15 : ci dessous présente le compte cumulatif de l'opération de valve et des deux alarmes considérées. On voit clairement l'effet du compteur externe de remise (à 3 et 12 secondes), et le retard d'une seconde (dû à l'opération de temporisateur) entre l'alarme sous forme de lumière (ligne épaisse grise) et l'alarme de vibreur (ligne mince noire).

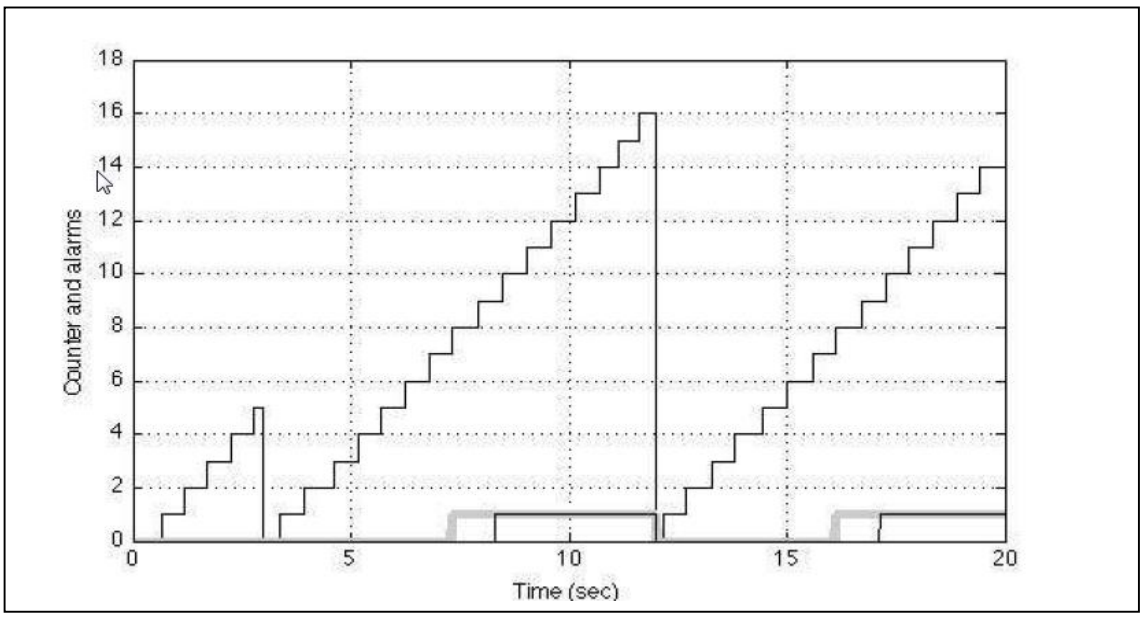


Fig.15 :l'évolution du conteur ainsi que l'alarme généré du réservoir d'eau contrôlé par l'API.

III. 5) Conclusion :

Une nouvelle approche pour l'essai des programmes de control de l'API pour l'enseignement de l'automation et le control des processus a été présentée. Cette approche est basée sur la langue de logiciel de Matlab/Simulink. Le programme de gestion de l'API est traduit en un bloc de fonction de Matlab, dans l'environnement de Matlab/Simulink, qui agira sur le modèle du processus industriel tant que la simulation fonctionne. Le paquet développé de traduction traduit automatiquement le programme de gestion de l'API, écrit comme liste d'instruction, dans la langue de logiciel de Matlab/Simulink. Le paquet de traduction produit un m-dossier obtenu en appliquant un ensemble de règles de traduction qui convertissent la liste d'instruction de l'API en langue de Matlab. Ce m-dossier est intégré dans la simulation de Matlab/Simulink du processus comme bloc de fonction appelé « programme de gestion de PLC ».

Conclusion générale

CONCLUSION :

Afin de faciliter les interactions entre les automaticiens et les informaticiens, nous avons établit un lien entre les terminologies couramment employés dans ces deux disciplines qui se traduisent à l'implantation des logiciel qui permettent la commande des processus industriels.

Nous avons élaboré dans notre travail Une méthode de développement qui fourni un outil de spécification permettant de prendre en compte les besoins de l'automaticien pour concevoir ces modèles ainsi que pour la commande des systèmes automatisé,

Pou cela nous avons établit le lien entre l'environnement de Matlab/Simulink qui permet la traduction automatique des programme de gestion des API.

L'automaticien utilise généralement une représentation graphique proche des graphes flot de données (schéma-bloc) et pour décrire les discontinuités dans le comportement du système (changement de mode de fonctionnement) il utilise généralement une représentation graphique de type flot de contrôle (GRAFCET, réseaux de Pétri), pour cela nous avons présenté les instructions nécessaire qui permettent de traduire ces représentation en un ensemble de m-dossier qui se traduit automatiquement en un bloc de fonction qui commandera le processus automatisé tant que la simulation fonctionne.

Référence bibliographie

References bibliographiques:

[1] Mikell P. Groover, Automation, Production Systems and Computer Integrated Manufacturing, Prentice Hall, 1987.

[2] Andrew Kusiak; Computational Intelligence in Design and Manufacturing, John Wiley & Sons, 2000.

[3] S. B. Morriss, Automated Manufacturing Systems, McGraw Hill, 1994.

[4] Matlab/Simulink, http://www.mathworks.com/

[5] L.A. Bryan and E.A. Bryan. '*Programmable Controllers. Theory and Implementation*'. Atlanta, Georgia, USA: Industrial Text Company Publication. 2nd Edition. 1997.

[6] John R. Hackworth and Frederick D. Hackworth, Jr. '*Programmable Logic Controllers: Programming Methods and Applications*'. Ed. Prentice Hall. 2004.

[7] Enrique Mandado, J. Marcos Acevedo, Celso Fernández, José I. Armesto and Serafín Pérez. '*Autómatas Programables. Entorno y Aplicaciones*' (in Spanish). Madrid: Ed. Thomson-Paraninfo. 2005.

[8] José Luís Romeral and Josep Balcells Sendra. '*Autómatas Programables*' (in Spanish). Barcelona: Ed. Marcombo. Boixareu Editores. 1996.

[9] C. Martin, F. Esquembre and J.L. Guzman, "Interactive Simulation of Object-oriented Hybrid Models by Combined Use of EJS, MATLAB/SIMULINK and MODELICA/DYMOLA", Proceedings 18th European Simulation Multiconference Graham Horton (c) SCS Europe, 2004.

[10] H. Elmqvist and S. E. Mattsson, "An Introduction to the Physical Modeling Language Modelica", Proc. of the 9th European Simulation Symposium, ESS'97, Oct 19-23,1997.

[11] Ejs, Easy Java Simulations, http://fem.um.es/Ejs/

[12] V Pinto, S. Rafael, J: F: Martins; "PLC controlled industrial processes on-line simulator";IEEE International Symposium on Industrial Electronics, ISIE 2007, June 2007, Vigo,Spain.

[13] G. L. Kim, P. Paul, Y. Wang, "UPPAAL in a nutshell", International Journal on Software Tools for Technology Transfer, 1, pp. 134-152, 1997.

[14] M. Chmiel, E. Hrynkiewicz, M. Muszynski, "The way of ladder diagram analysis for small compact programmable controller", Proceedings of the 6th Russian-Korean International Symposium on Science and Technology KORUS-2002, pp. 169-173,2002.

[15] Gi Bum Lee, Han Zandong, Jin S. Lee, "Automatic generation of ladder diagram with

control Petri Net", Journal of Intelligent Manufacturing, 15, 245±252, 2004.

[16] HyungSeok Kim, Wook Hyun Kwon, Naehyuck Chang; "A translation method for ladder diagram with application to a manufacturing process", Proceedings of the IEEE International Conference on Robotics and Automation, pp. 793-798, Detroit, USA, 1999.

[17] IEC, International Electrotechnic Commission, *Preparation of Function charts fos Control Systems*, publication 848, 1988.

[18] Alain GONZAGA cours d'API.

[19] Kamil Š vancara, Petr Pivoňka The real-time communication between Matlab and the real process controlled by PLC." 7th International Research/Expert Conference" Trends in the Development of Machinery and Associated Technology"TMT 2003, Lloret de Mar, Barcelona, Spain, 15-16 September, 2003

[20]Gregoire Hamon and John Rushby. An Operational Semantics for Stateow. Journal on Software Tools for Technology Transfer (STTT), 9(5{6) :447{456, 2007.

[21]Gerard Berry and Georges Gonthier. The ESTEREL Synchronous Programming Language : Design, Semantics, Implentation. Science of Computer Programming, 19(2) :87{152, 1992.

[22]Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Je_rey D. Ullman. Compilers :Principles, Techniques, and Tools. Addison Wesley, second edition, 2006.

RESUME

MATLAB est un logiciel commercial de calcul numérique/scientifique, et de visualisation et programmation très performant et convivial. Le champ d'application de MATLAB peut être étendu aux systèmes non linéaires et aux problèmes associés de simulation avec le produit complémentaire SIMULINK.

Mais MATLAB est aussi un environnement de développement ("progiciel") à part entière ; son langage d'assez haut niveau, doté notamment de structures de contrôles, fonctions d'entrée-sortie et de visualisation 2D et 3D et des outils de construction d'interface a usage graphique (GUI)... permet à l'utilisateur d'élaborer ses propres fonctions ainsi que de véritables programmes ("M-files") appelés scripts vu le caractère interprété de ce langage.

Une interface de programmation applicative(API) rend finalement possible l'interaction entre MATLAB et les environnements de développement.

La clef de voûte de la méthodologie proposée est le programme de gestion de l'API présenté dans un bloc de fonction dans l'environnement de MATLAB/SIMULINK qui émulera l'opération cyclique de l'API tant que la simulation fonction.

Mots-clés : Automates programmables industriel (API), Architecture matériel d'un API, Langage de programmation, Les étapes d'un projet de simulation, Outils de simulation, Translation API/MATLAB.