NOTES DU COURS D' INTRODUCTION AUX AUTOMATES

Djamba Tunda-Olembe

REPUBLIQUE DEMOCRATIQUE DU CONGO
ENSEIGNEMEMENT SUPERIEUR ET UNIVERSITAIRE
Université Catholique Maria Malkia
FACULTE DES SCIENCES INFORMATIQUES

COURS D'INTRODUCTION AUX AUTOMATES

(L1 TECHNOLOGIE ET RESEAUX)

ANNEE 2023

PRESENTATION DU COURS

I. PRE-REQUIS

RESEAUX INFORMATIQUES, ARCHITECTURE DES MICROPROCESSEURS ET ELECTRONIQUE

II. LIMINAIRE

La science des automates ou Automatique est une branche scientifique qui traite des systèmes ou l'être humain est remplacé dans ses taches de contrôle et de commande par un dispositif plus performant appelé Automate. Les automates sont parfois appelés suivant l'auteur du texte qui les décrit : Automates finis pour les théoriciens de l'Automatique et des réseaux ; Machines à nombre d'état finis pour les électroniciens concepteurs de circuits numériques ; ou Séquenceurs pour les concepteurs d'unité centrales d'ordinateurs. On distingue des automates finis et des automates industriels. Les automates finis sont utilisés dans les différents contextes : pour décrire les lexèmes d'un langage de programmation dans un compilateur ou pour décrire les fonctions de traductions des transducteurs. Les automates industriels sont les organes de traitement de l'information des systèmes automatisés. Ils jouent un rôle très important dans les entreprises industrielles. Actuellement, la majorité des installations industrielles de production sont contrôlées et commandés par les ordinateurs industriels.

III. OBJECTIFS DU COURS

a) Objectifs généraux :

1. L'étudiant doit comprendre et maitriser les concepts et la terminologie des automates finis;
2. L'étudiant doit comprendre la structure et la modélisation ses systèmes automatisés linéaires et logiques ;
3. L'étudiant doit comprendre la structure et le fonctionnement des automates programmable industriel ;
4. L'étudiant doit comprendre et maitriser les méthodes de représentation symbolique en automation et régulation ;
5. L'étudiant doit comprendre et la structure et le fonctionnement des réseaux des automates et l'Internet industriel.

b) Objectifs spécifiques :

1. L'étudiant doit être capable de manipuler les formalismes de description et Représentation graphique des automates finis ;
2. L'étudiant doit être capable d'analyser et de modéliser les systèmes automatisés linéaires et logiques ;
3. L'étudiant doit être capable de sélectionner et de mettre œuvre un automate programmable industriel ;

4. L'étudiant doit être capable de lire et d'interpréter un schéma d'automation et instrumentation ;
5. L'étudiant doit être capable d'analyser un réseau des automates industriels.

IV.METHODE DE COMMUNICATION

1. Cours théorique orienté vers la pratique ;
2. Travaux et discussions en groupe ;
3. Organisation TD et TP ;
4. Autoformation.
5. Lecture du support du cours

V. METHODE D'EVALUATION

1. Travaux pratiques (5 points) ;
2. Interrogations (5 points) ;
3. Examen (10 points)

VI. PLAN DU COURS

- Chapitre 1 : Généralités sur les automates ;
- Chapitre 2 : Les automates finis ;
- Chapitre 3 : Les systèmes automatisés linéaires et logiques ;
- Chapitre 4 : Les automates programmables industriels ;
- Chapitre 5 : Automation et instrumentation ;
- Chapitre 6 : Introduction aux réseaux des automates et à l'Internet industriels.

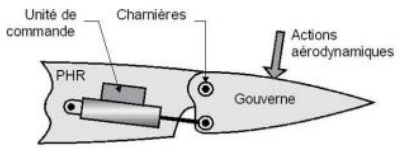
VII. BIBLIOGRAPHIE

1. BARTHELEMY F. : cours sur les automates, Département informatique, CNAM/ France, Année 2015.
2. NOUVEL DAMIEN : cours d'introduction aux automates, Département informatique, Université François Rabelais, TOURS / France, Année 2015.
3. BERGOUGNOUX L. : cours sur les automates programmables industriels, POLYTECH Marseille, année 2014.
4. LONGEOT ET AL. : Automatique industrielle, Edition DUNOD, Paris, 2006.

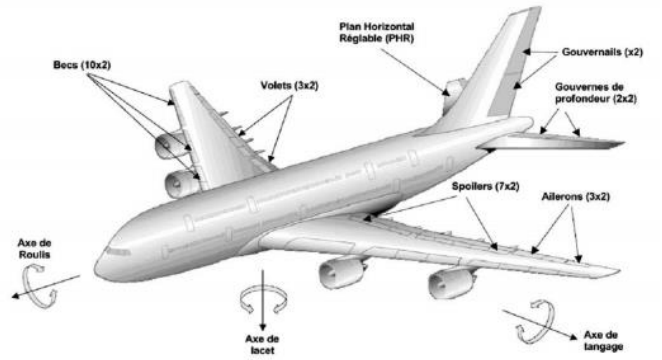
CHAP.1 GENERALITES SUR LES AUTOMATES

I.1 GENERALITES SUR LES AUTOMATES INDUSTRIELS

 # Commandes de vol primaires de l'Airbus A380



Précision, Fiabilité.



Définition **:** **Automatique**, c'est l'ensemble de théories et de techniques pour la prise de décision et la commande des systèmes. En anglais "*Automatic Control*" (faux ami : commande).

L'automatique est ainsi la discipline scientifique permettant de caractériser les systèmes automatisés et de choisir/concevoir/réaliser la commande des systèmes. Les systèmes de commande s'inspirent le plus souvent de l'homme.

Définition **:** Un système **automatisé** ou **automatique** est un système réalisant des opérations et pour lequel l'homme n'intervient que dans la programmation du système et dans son réglage.

Les buts d'un système automatisé sont de réaliser des tâches complexes ou dangereuses pour l'homme, effectuer des tâches pénibles ou répétitives ou encore gagner en efficacité et en précision.

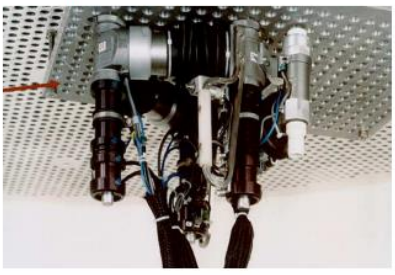
Quelques exemples **:**

 # Capsuleuse Ravoux

Tâche répétitive,

Cadence élevée.

 # Robot TRIBAR, inspection des conduites de centrale nucléaire : *Tâche dangereuse, nécessitant précision et fiabilité*



Le distributeur de billets

Les feux de carrefour

ROBOTIQUE INDUSTRIELLE

La robotique crée des machines capables d'accomplir des travaux déterminés. Dotées de capteurs et susceptibles de réagir à leur environnement grâce à leur intelligence informatique, elles peuvent disposer d'une certaine autonomie de calcul, de mouvement et d'action.

Le robot n'a pas forcément l'allure humanoïde, il peut prendre toutes les apparences voir n'être qu'une créature virtuelle de pur logiciel. Toutes les tâches répétitives intellectuelles ou manuelles seront à sa portée.

Le robot deviendra le troisième achat le plus important du foyer après la maison et l'automobile. La robotique est probablement l'équivalent industriel dans ce siècle de l'automobile au siècle dernier.

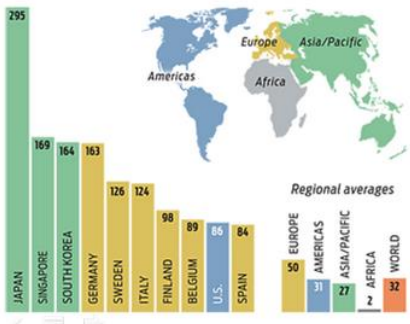
Pour les entreprises, le robot constitue une source d'économies certaines. La meilleure efficacité résultant de la combinaison entre robots et humains.

Les robots industriels

Quelques images

FANUC

ABB

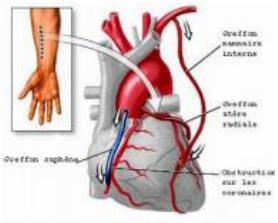


KUKA

Robot

Robotique médicale

Contexte et motivation



✓ Pontage coronaire est un geste très délicat

✓ **Solution classique 1 :** - grandes incisions au thorax

- arrêter le cœur

- utiliser une machine cœur-poumon

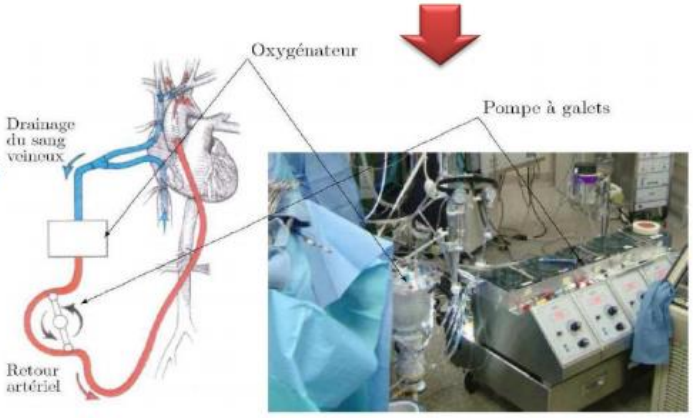
Inconvénients :

✘ Risques de complications opératoires

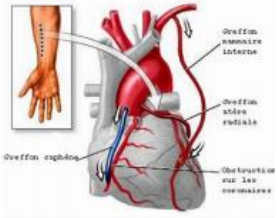
✘ Effets secondaires indésirables

✘ Douleurs importantes

✘ Temps de récupération long



Contexte et motivation



✓ Pontage coronaire est un geste très délicat

✓ **Solution classique 1 :** - grandes incisions au thorax

- arrêter le cœur

- utiliser une machine cœur-poumon

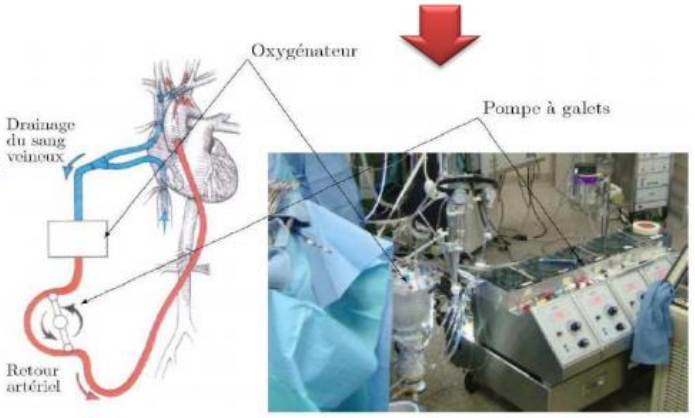
Inconvénients :

✘ Risques de complications opératoires

✘ Effets secondaires indésirables

✘ Douleurs importantes

✘ Temps de récupération long



‘Chirurgie cardiaque mini-invasive’

Contexte et motivation

Avantages :

- ✓ Traumatisme réduit
- ✓ Temps de récupération moins long
- ✓ Cicatrices plus petites

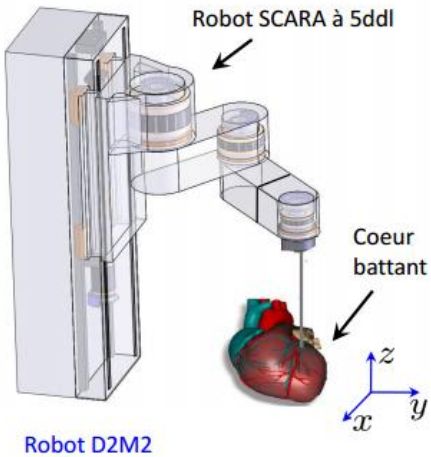
MAIS

- ✘ Le chirurgien doit travailler sur un cœur battant
- ✘ Compensation manuelle des mouvements physiologiques (très fatigante)
- ✘ Nécessite beaucoup de concentration / précision
- ✘ Mobilité réduite

Meilleur Solution

Chirurgie mini-invasive robotisée

- ✓ Compensation des mouvements physiologique
- ✓ En chirurgie cardiaque à cœur battant
- ✓ Une architecture de commande à retour d’effort
- ✓ Application à un robot SCARA à 5ddl
- ✓ Implémentation en simulation / en temps réel



Robot D2M2

Test de faisabilité

Solution proposée
Architecture de commande adaptative à retour d'effort

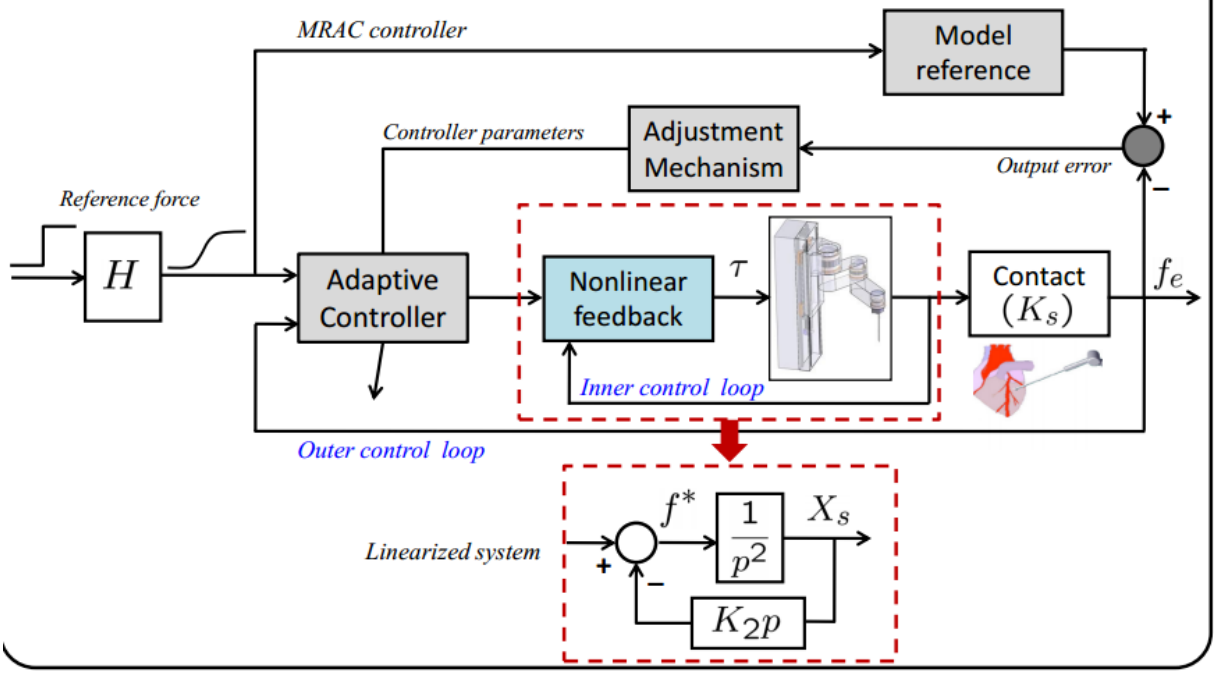
Comporte deux boucles de commande

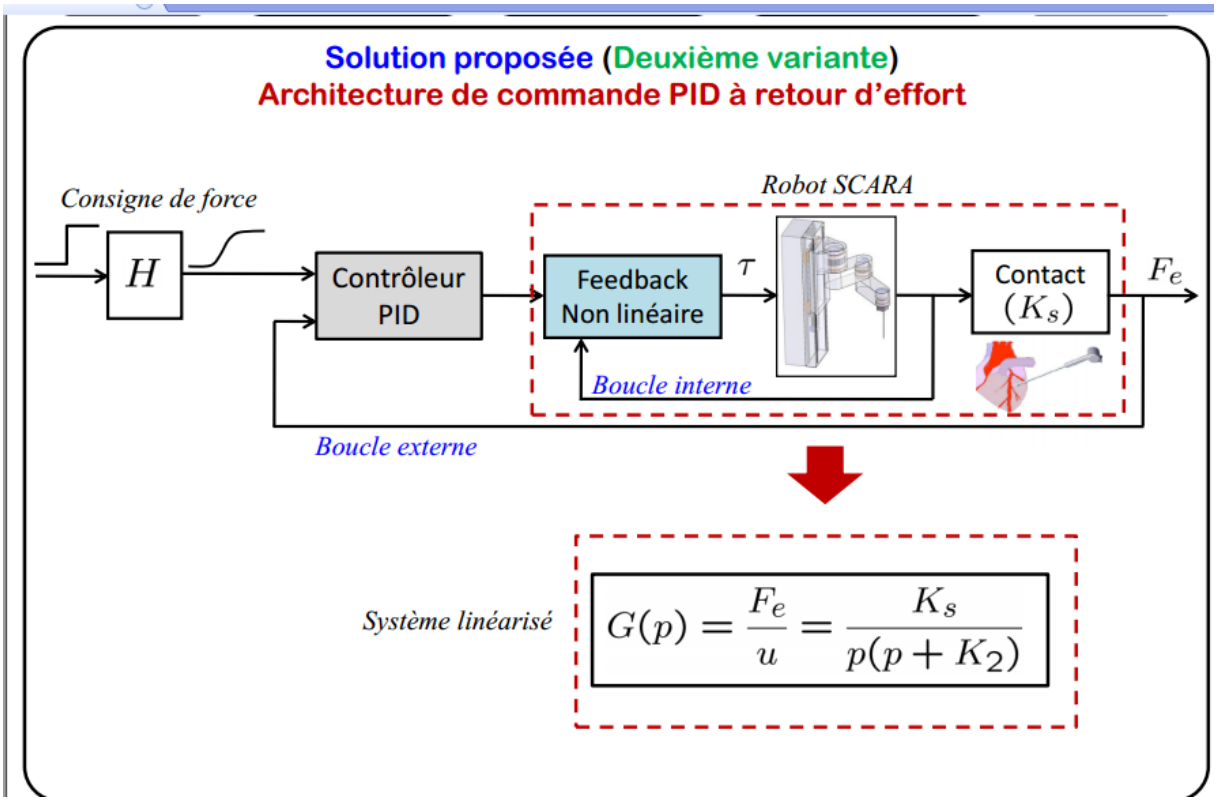
Une boucle de commande interne à base d'un retour d'état non linéaire
Une boucle de commande externe à base d'un contrôleur linéaire

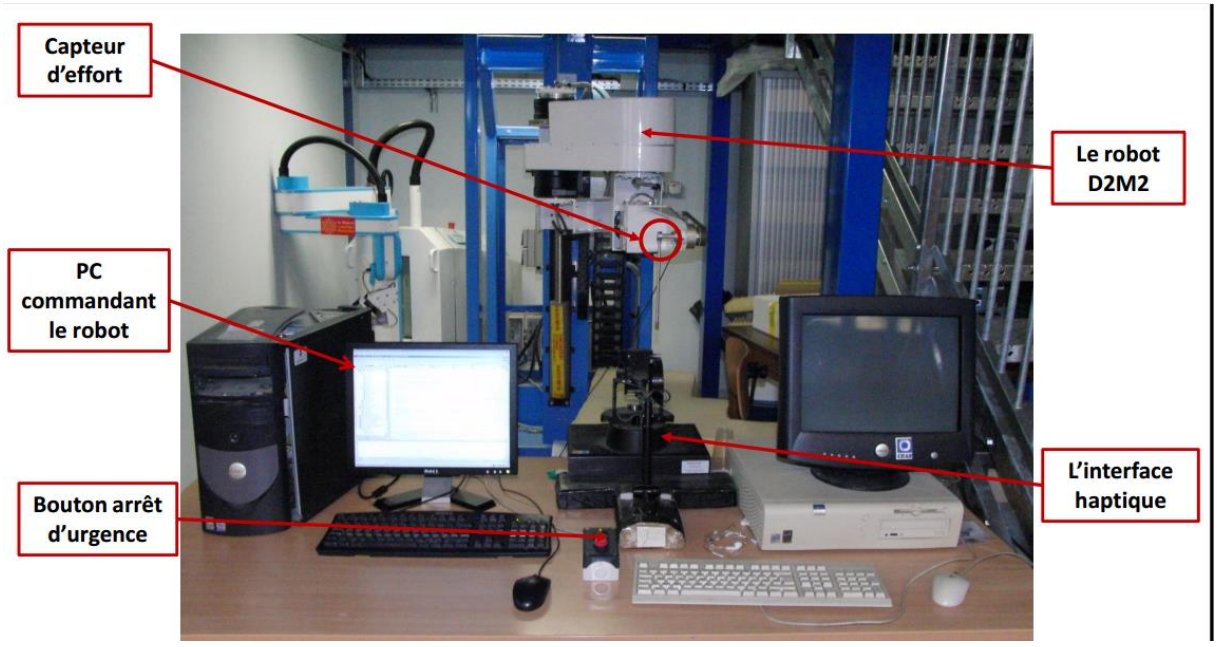
Deux variantes selon le contrôleur linéaire choisi

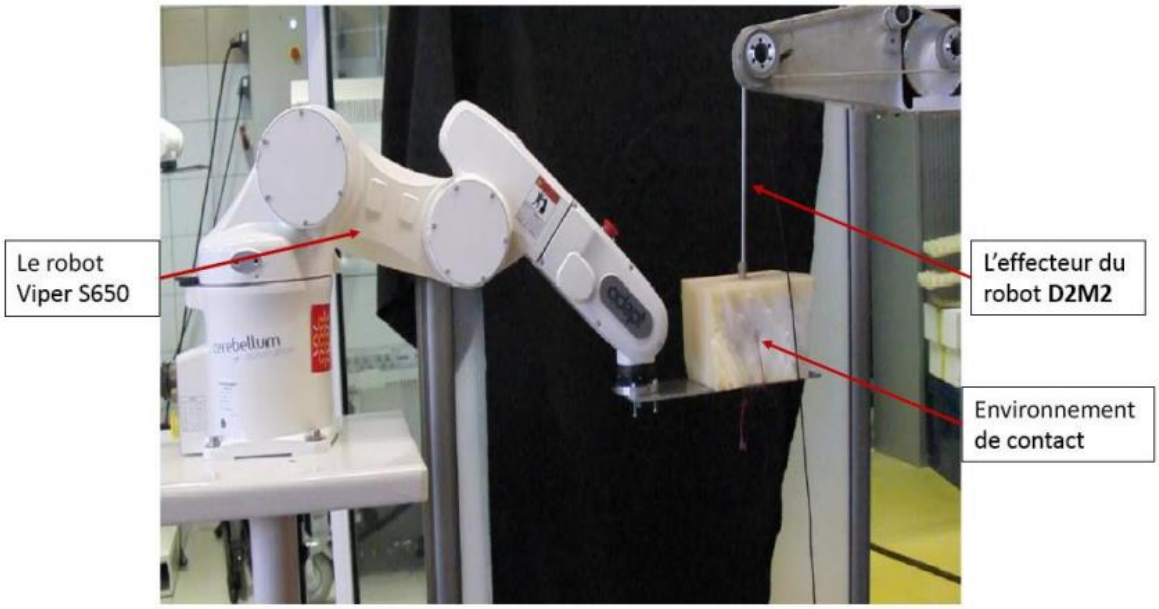
- Une architecture de commande à base d'un contrôleur adaptatif MRAC
- Une architecture de commande à base d'un contrôleur PID

Solution proposée (Première variante)
Architecture de commande adaptative à retour d'effort




Solution proposée (Deuxième variante)
Architecture de commande PID à retour d'effort
Consigne de force
H
Contrôleur PID
Robot SCARA
Feedback Non linéaire
τ
Contact (Ks)
Fe
Boucle interne
Boucle externe
Système linéarisé
G(p) = Fe/u = Ks/p(p + K2)


Capteur d'effort
Le robot D2M2
PC commandant le robot
L'interface haptique
Bouton arrêt d'urgence



Contrôle d'attitude d'un satellite

- Mode normal
- Mode de contrôle d'orbite
- Mode de survie
- Mode de transition

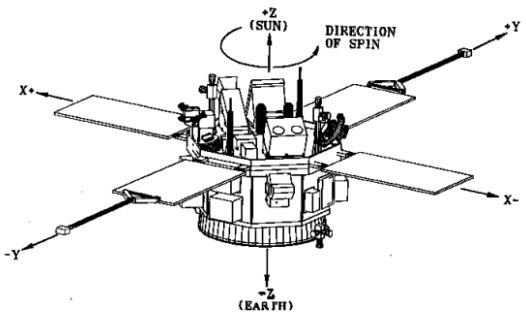
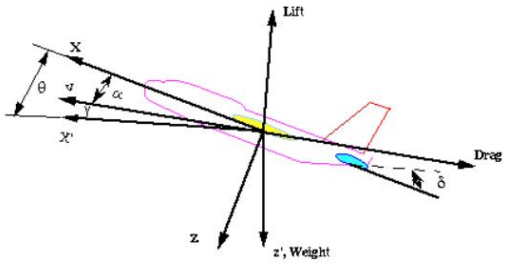


FIGURE 1 – Satellite FRD5

- Orientation (attitude) à contrôler (antennes, capteurs, panneaux solaires)
- Pointage géocentrique, inertiel...
- Perturbations gravitationnelles, bruits électriques, flexibilité, vents solaires

Commande longitudinale d'un avion civil



- Contrôle de la vitesse d'avancement, de la pente γ, de l'assiette longitudinale θ et de l'altitude h
- δ gouverne de profondeur et poussée des moteurs
- Rafales, charge variable, confort des passagers...

STRUCTURE D'UN SYSTEME AUTOMATISE

Ce chapitre permet de comprendre la structure d'un Système Automatisé de Production et de définir les différentes parties de ce système. Un système de production est dit automatisé lorsqu'il peut gérer de manière autonome un cycle de travail préétabli qui se décompose en séquences et/ou en étapes.

Les systèmes automatisés, utilisés dans le secteur industriel, possèdent une structure de base identique. Ils sont constitués de plusieurs parties plus ou moins complexes reliées entre elles :

- la partie opérative (PO) ;
- la partie commande (PC) ou système de contrôle/commande (SCC) ;
- la partie relation (PR) de plus en plus intégrée dans la partie commande.

1.1.1 La partie opérative

C'est la partie visible du système. Elle comporte les éléments du procédé, c'est à dire :

- des pré-actionneurs (distributeurs, contacteurs) qui reçoivent des ordres de la partie commande ;
- des actionneurs (vérins, moteurs, vannes) qui ont pour rôle d'exécuter ces ordres. Ils transforment l'énergie pneumatique (air comprimé), hydraulique (huile sous pression) ou électrique en énergie mécanique ;
- des capteurs qui informent la partie commande de l'exécution du travail. Par exemple, on va trouver des capteurs mécaniques, pneumatiques, électriques ou magnétiques montés sur les vérins. Le rôle des capteurs (ou détecteurs) est donc de contrôler, mesurer, surveiller et informer la PC sur l'évolution du système.

1.1.2 La partie commande

Ce secteur de l'automatisme gère selon une suite logique le déroulement ordonné des opérations à réaliser. Il reçoit des informations en provenance des capteurs de la Partie Opérative, et les restitue vers cette même Partie Opérative en direction des pré-actionneurs et actionneurs. L'outil de description de la partie commande s'appelle le GRAphe Fonctionnel de Commande Étape / Transition (GRAFCET).

1.1.3 La partie relation

Sa complexité dépend de l'importance du système. Elle regroupe les différentes commandes nécessaires au bon fonctionnement du procédé, c'est à dire marche/arrêt, arrêt d'urgence, marche automatique, etc... L'outil de description s'appelle le Guide d' Études des Modes de Marches et d'Arrêts (GEMMA).

Les outils graphiques, que sont le GRAFCET et le GEMMA, sont utilisés par les automaticiens et les techniciens de maintenance.

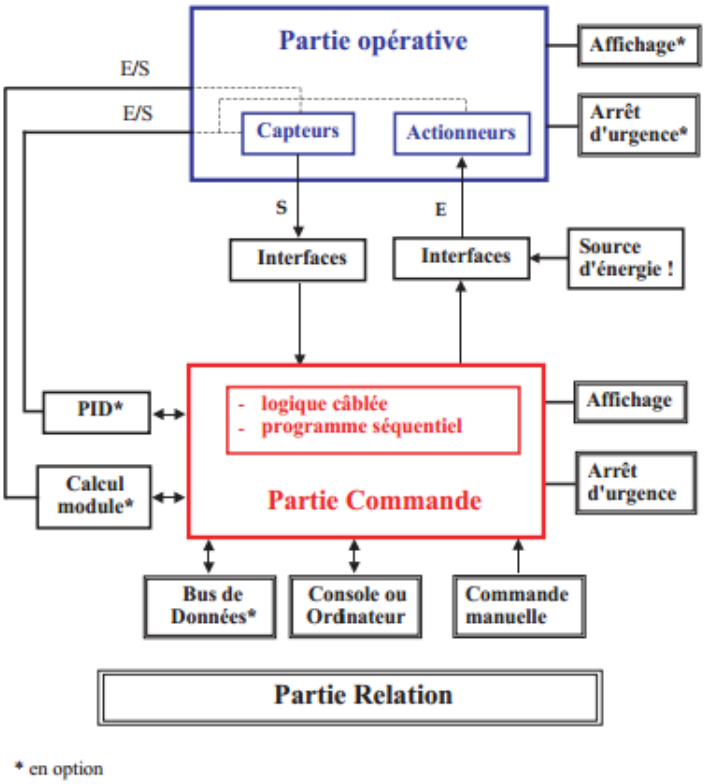
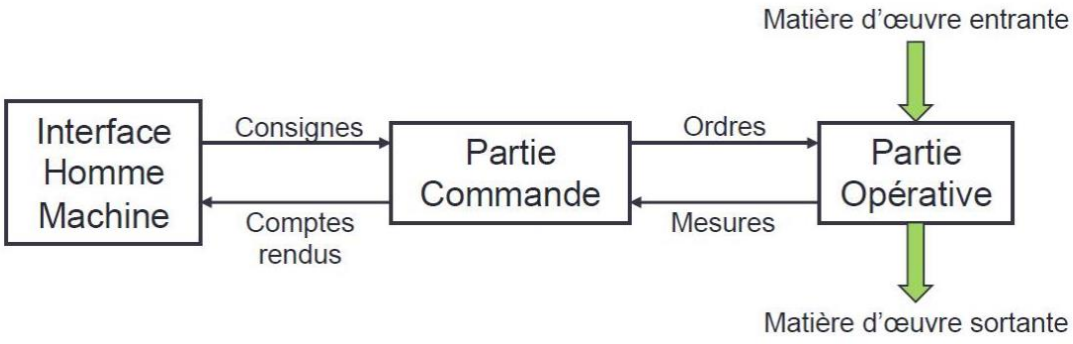


FIG. 1.1 – Procédé automatisé

Un système automatisé peut être décomposé en trois grandes parties :

- ✓ Une partie opérative (PO) assurant la conversion de puissance et l'action sur la matière d'œuvre.
- ✓ Une partie commande (PC) assurant la mesure en continu sur le processus, le traitement des données par comparaison aux consignes et le pilotage de la partie opérative.
- ✓ Une interface homme / machine (IHM) permettant de définir les consignes et de surveiller l'évolution.



II – Classification des systèmes

1. Les différentes natures d'information

On a vu (Analyse fonctionnelle, §VII – 1.) qu'à chaque chaîne fonctionnelle d'un système correspond une chaîne d'information et une chaîne d'énergie. L'automatique s'intéresse à la **chaîne d'information**. Les systèmes automatisés vont alors être classés en fonction de la nature des informations de commande ou de mesure, et également en fonction de la nature du traitement de ces informations.

On distingue deux types d'informations : **analogiques** et **discrètes (logiques)**.

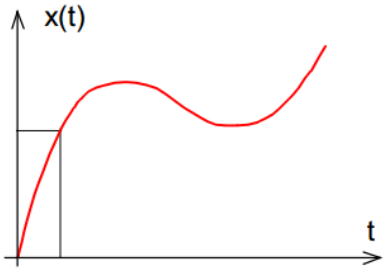
Définition **:** une **information (signal) analogique** est une information qui peut prendre toutes les valeurs possibles dans un intervalle donné. Les grandeurs physiques, comme la température, la vitesse, la pression, la tension (…) sont des informations analogiques. Une information analogique peut être représentée par une courbe (voir page suivante).

Définition **:** une **information (signal) discrète** est constituée d'un ensemble fini de valeurs. On distingue :

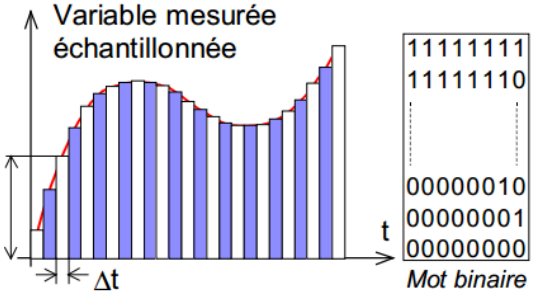
Information binaire (0 ou 1, vrai/faux, noir/blanc, Tout Ou Rien (TOR)).

Information binaire

Information numérique sous la forme d'un mot binaire, constitué de plusieurs variables binaires (bits[1]). Information généralement issue d'un traitement d'une information analogique (échantillonnage, codage[2]).



Information analogique



Information numérique

2. Classification des systèmes

On peut alors distinguer les systèmes automatisés suivants :

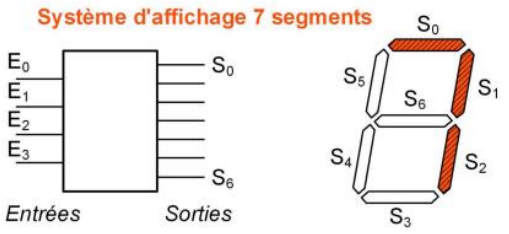
Variables logiques - Systèmes automatisés à logique combinatoire

Pour un tel système les sorties dépendent exclusivement d'une combinaison des entrées, sans prendre en compte "l'histoire" du système. A un état d'entrées, correspond un et un seul état en sortie. Aucune mémoire des états précédents des entrées et des sorties n'est conservée. L'information logique est traitée de manière instantanée.

Les grandeurs y sont manipulées sous formes d'états binaires, ce qui justifie l'utilisation de l'algèbre de BOOLE[3], et des notions liées au codage de l'information.

Exemple : afficheur sept segments

L'information, chiffre compris entre 0 et 9, est fournie par un nombre binaire sur 4 bits, soit pour notre afficheur quatre entrées (E_0, E_1, E_2 et E_3) et en sortie les segments seront allumés ou éteints. Pour chaque combinaison des quatre entrées, doit correspondre un et un seul état des sorties, correspondant à l'affichage correct de l'information.



Variables logiques - Systèmes automatisés à logique séquentielle

Qualifié de système à mémoire généralisée, les sorties du système sont élaborées à partir d'un ensemble de signaux logiques, mais dépendent aussi de la chronologie des événements logiques. "L'histoire" du système est prise en compte.

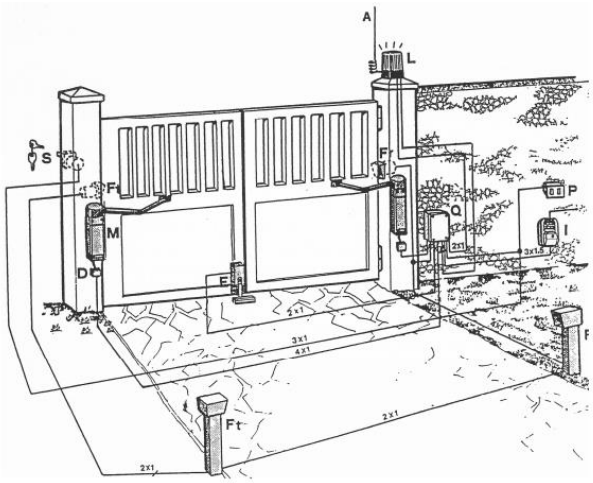
En effet les états précédents des entrées et des sorties sont mémorisés, et influent sur l'évolution du système. A une combinaison d'entrées, peuvent correspondre plusieurs combinaisons des sorties.

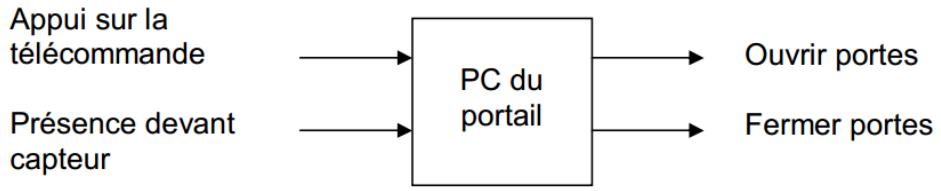
Exemple de système logique :
Le portail automatisé.

Pour simplifier, on s'intéresse aux éléments suivants :

- ✓ Les 2 portes
- ✓ Les 2 moteurs
- ✓ La télécommande
- ✓ La cellule photo électrique

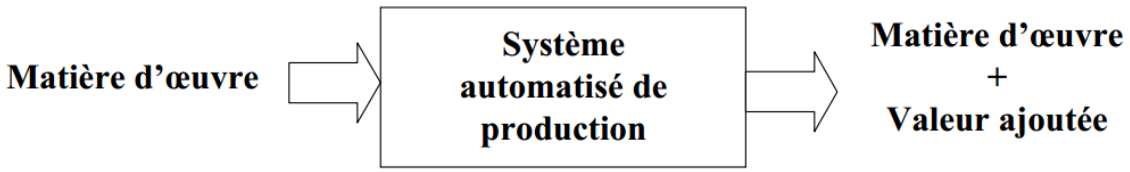
Entrées/sorties de la Partie commande (PC) du portail : (les entrées sont les informations, les sorties sont les ordres)



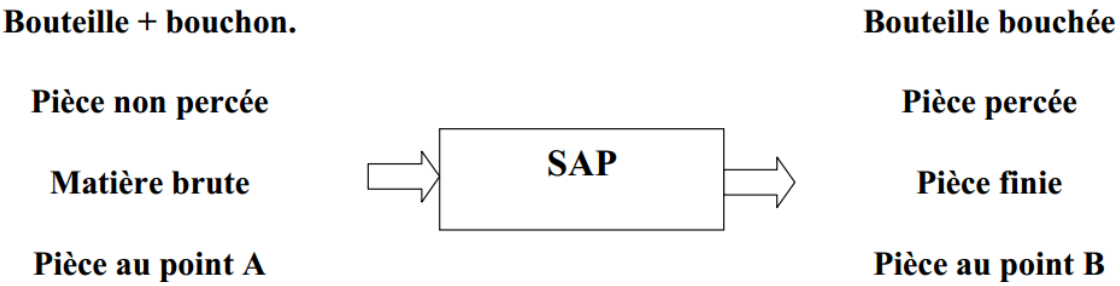


Les entrées et les sorties sont sous la forme tout ou rien (1 ou 0) (vrai ou faux), on les appelle des variables logiques.

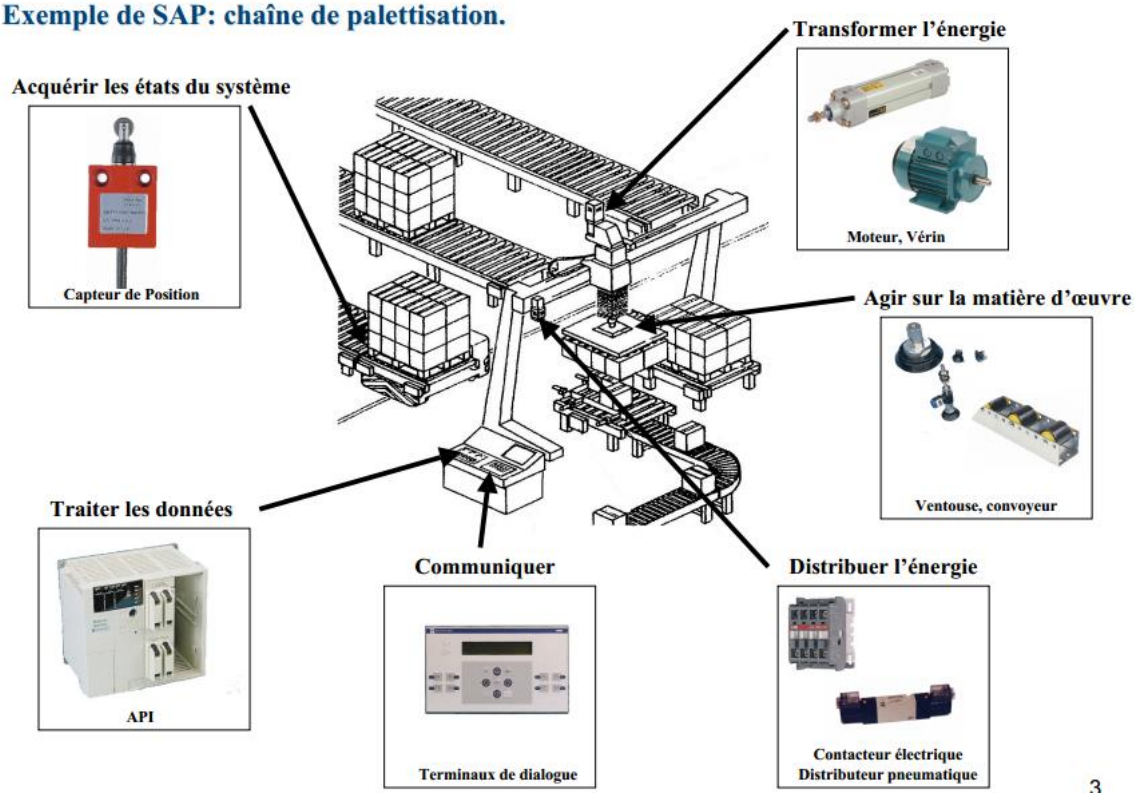
SYSTEME DE PRODUCTION AUTOMATISE SEQUENTIEL



- Système Automatisé de Production (SAP):
  Système autonome de création de valeur ajoutée.
  Sous des impératifs de sécurité, productivité, adaptabilité…



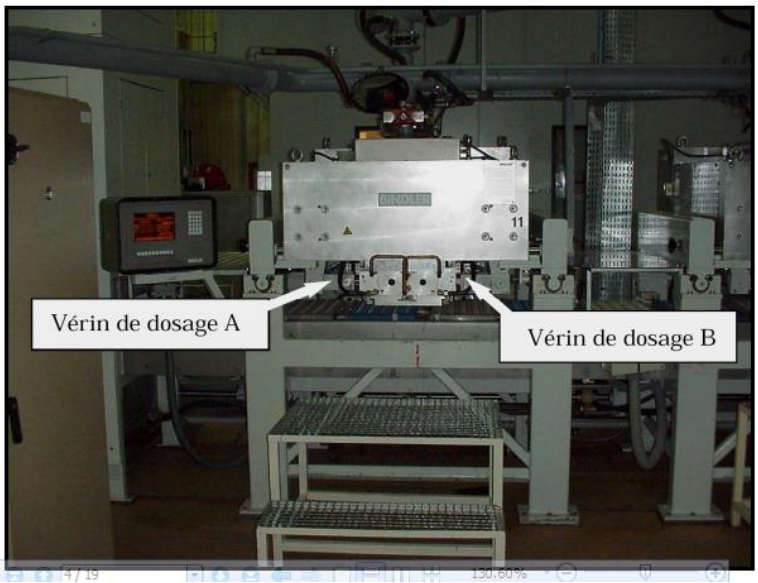
Exemple de SAP: chaîne de palettisation.



3

Exemple : ligne de production industrielle de chocolats

Le système est une ligne de production industrielle de sujets creux en chocolat (petits et grands sujets, par exemple Père Noël, lapin, œufs, …) Le processus d'injection (réalisé par la doseuse) fait l'objet d'une commande séquentielle.



Variables analogiques ou numériques - Systèmes automatisés continus

Les signaux traités sont analogiques ou numériques, et leurs valeurs ne peuvent être prédéterminées. Les sorties (asservies ou non) sont des grandeurs continues pour un processus donné.

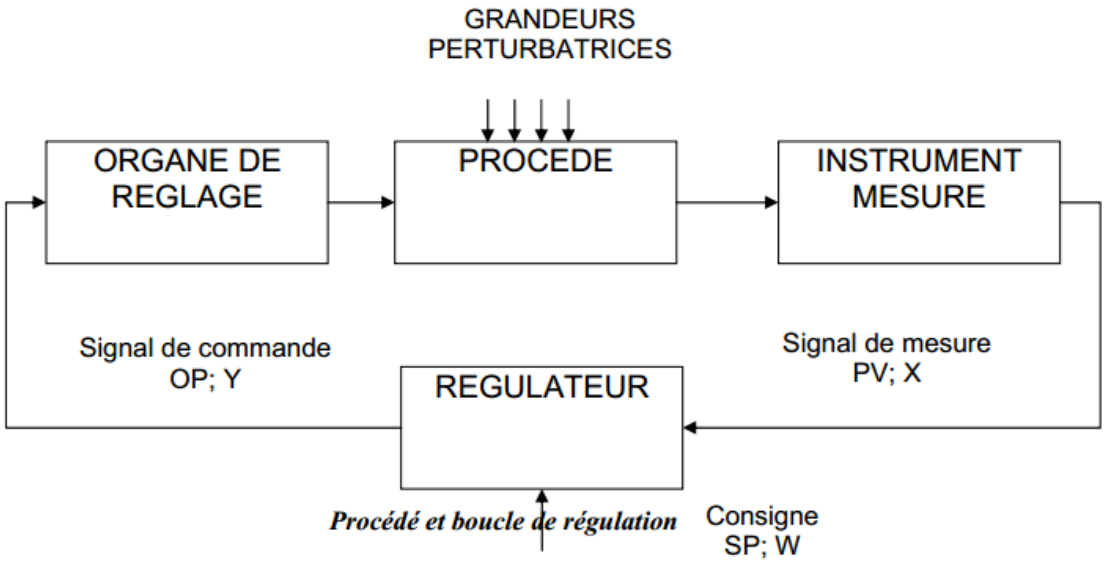
Systèmes automatisés asservis : ils sont l'objet du cours qui suit, pour de tels systèmes une mesure de la sortie est réalisée en permanence et sa valeur comparée à l'entrée (sortie souhaitée) puis corrigée. Ces systèmes permettent d'obtenir toutes les caractéristiques nécessaires aujourd'hui dans beaucoup de systèmes pluri-techniques [**Rapidité, Précision, Stabilité**]. Les asservissements sont classés en deux familles : les systèmes **régulateurs** et les systèmes asservis **suiveurs**.

Systèmes **régulateurs** : la consigne d'entrée est fixe, ils sont destinés à assurer une sortie constante.

Exemple : Régulation de température

Systèmes asservis **suiveurs** ou en poursuite d'une loi de référence : la consigne d'entrée varie constamment et l'objectif est d'ajuster en permanence la sortie au signal d'entrée.

La figure suivante illustre un procédé régulé par une boucle fermée. Dans ce cas l'association procédé et instruments constitue un système asservi, de ce fait la réponse à un échelon de consigne est généralement du type apériodique.



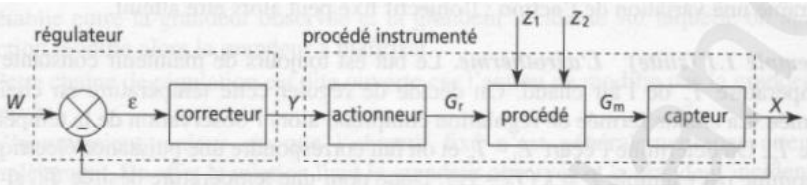


Figure 1.10 Constitution d'une chaîne fermée de régulation.

Les éléments constitutifs d'une chaîne fermée de régulation (fig. 1.10) peuvent être déclinés en deux catégories.

➤ Les éléments fonctionnels

Ils sont représentés par des rectangles.

Le régulateur comprend le soustracteur (ou comparateur) et le correcteur.

Le soustracteur reçoit la consigne et le signal de mesure dont il effectue la différence. Le résultat de cette différence est appelé écart (ou erreur).

Le correcteur est chargé d'élaborer un signal de commande à partir de l'écart constaté afin d'obtenir les performances fixées par le cahier des charges (stabilité, précision, rapidité…).

L'actionneur est commandé par le signal de commande provenant du régulateur. C'est l'organe de puissance de la chaîne de régulation. Il agit sur la grandeur réglante du procédé pour modifier la grandeur physique à maîtriser.

Le capteur (ou transmetteur) élabore la mesure de la grandeur à maîtriser et la transmet au régulateur.

I.2 GENERALITES SUR LES AUTOMATES FINIS

DEFINITION DE BASE

Définition 1 *Alphabet, chaîne*
On appelle alphabet un ensemble fini de symbole. Une chaîne sur un alphabet Σ est une séquence éventuellement vide de symboles de Σ. La séquence vide est notée ϵ (epsilon). Les autres séquences sont notées par la juxtaposition des symboles qui les composent.

Définition 2 *Langage*
Un langage est un ensemble de chaînes sur un alphabet Σ.

Définition 3 *Automate fini*
Un automate fini est un quintuplet $A = (\Sigma, Q, \delta, i, F)$ où :

- *Σ est un ensemble fini de symboles appelé alphabet.*
- *Q est un ensemble fini dont les éléments sont appelés* états.
- *δ est une relation de $Q \times \Sigma \times Q$ appelée* transition *ou* ensemble des transitions *de A.*
- *i est un état de Q appelé* état initial.
- *F est un sous-ensemble de Q appelé* ensemble des états finals *de A.*

Définition 4 *Représentation graphique*
Un automate fini peut être représenté graphiquement comme un graphe orienté dont les sommets sont les états et les arcs sont les transitions. Une transition (q_1, x, q_2) est représentée par un arc reliant les sommets q_1 et q_2, étiqueté par x.

Définition 5 Transducteur à états finis

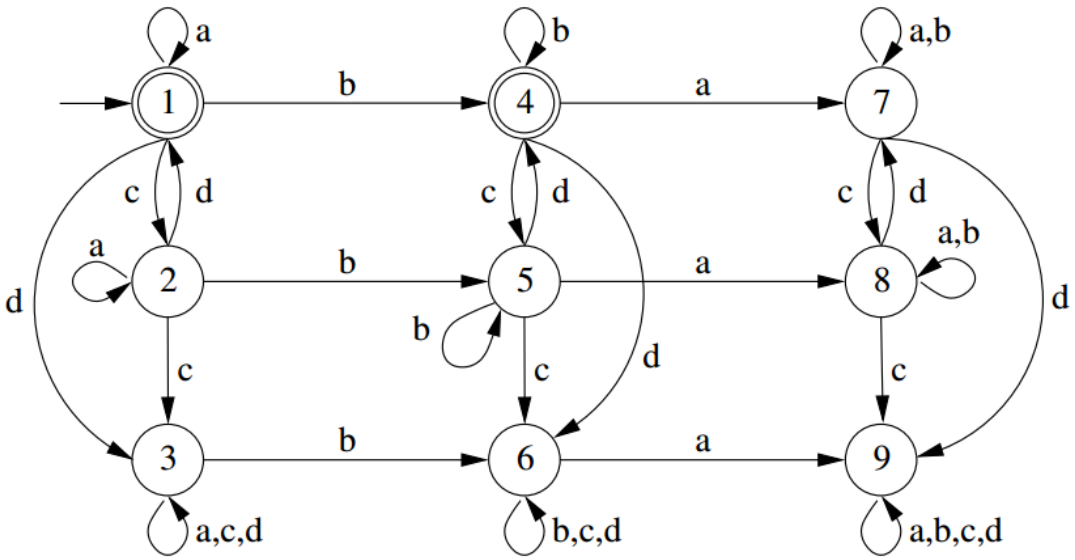
Selon le dictionnaire Trésor de la Langue Française informatisé, un transducteur est un dispositif ou élément d'une chaîne de communication (mécanique, électrique, etc.) recevant un message sous une certaine forme et le transformant en une autre. Selon le Larousse, transducteur est un synonyme de traducteur.

Un transducteur fini à état est un automate fini qui non seulement reconnaît un ensemble régulier de chaînes, mais encore qui traduit chaque chaîne de cet ensemble dans une autre chaîne appartenant à un autre langage régulier. Concrètement, comme un automate fini, c'est un graphe orienté étiqueté, mais chaque transition est étiquetée par un (ou zéro) symbole à reconnaître et un (ou zéro) symbole utilisé pour construire la traduction, séparés par un point-virgule.

Voyons un exemple qui concerne la traduction de quelques chiffres du français à l'anglais (un→one, deux → two et trois→three).

Un automate fini est un sextuplet $A = (\Sigma_i, \Sigma_o, Q, \delta, i, F)$ *où :*

- Σ_i *et* Σ_o *sont deux ensembles finis de symboles appelés respectivement alphabet d'entrée et alphabet de sortie.*
- Q *est un ensemble fini dont les éléments sont appelés* états.
- δ *est une relation de* $Q \times (\Sigma_i \cup \{\epsilon\}) \times (\Sigma_o \cup \{\epsilon\}) \times Q$ *appelée* transition *ou* ensemble des transitions *de* A.
- i *est un état de* Q *appelé* état initial.
- F *est un sous-ensemble de* Q *appelé* ensemble des états finals *de* A.



UTILISATION DES AUTOMATES FINIS

Les automates finis et les expressions régulières sont utilisées dans différents contextes.

- Pour décrire des traitements textuels (par exemple, recherche de sous-chaîne). Dans les traitements de textes, les langages de script (awk, perl), etc.
- Pour décrire les léxèmes d'un langage dans un compilateur ou un interpréteur (langage de programmation, de script, de description de document, d'interrogation de base de donnée). De même pour les fichiers de configuration d'applications.
- Pour décrire l'étage morpho-lexical des langues naturelles (dictionnaires, lexiques).
- Pour décrire les propriétés dynamiques d'un système. Par exemple dans des langages de description comme UML (diagrammes états-transitions) ou SA-RT.
- Pour décrire le flot de contrôle d'un programme séquentiel (par exemple, pour faire des tests d'accessibilité d'instructions).

Les propriétés des automates finis sont très intéressantes. Grâce à la déterminisation et à la minimisation, toute description sous forme d'automate fini peut être exécutée efficacement. Les opérations ensemblistes ainsi que la concaténation et l'étoile ouvrent la voie à la modularité et au traitement d'exceptions. Des boîtes à outil efficace existent qui permettent de décrire et d'exécuter de très gros automates (plusieurs millions d'états et de transitions).

MACHINE SEQUENTIELLE ALGORITHMIQUE

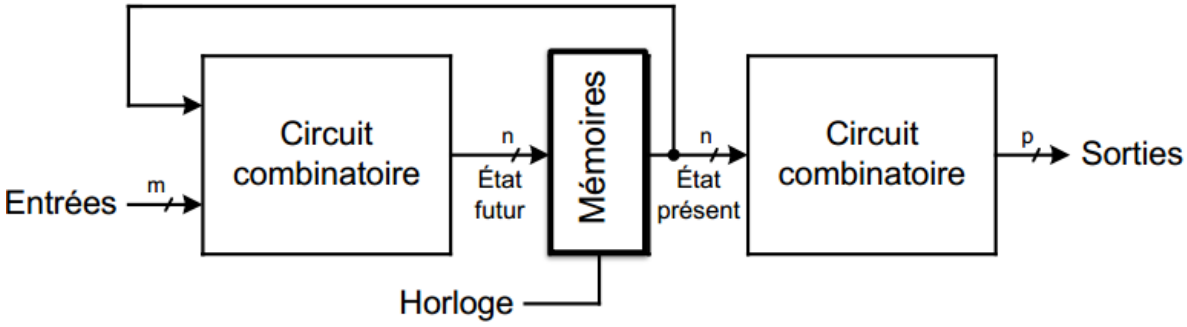
Qu'est-ce qu'une machine séquentielle algorithmique (MSA)?

- Circuit séquentiel qui peut se retrouver dans un nombre fini d'états. Une fois dans un état, le système doit demeurer dans cet état pour une durée déterminée.

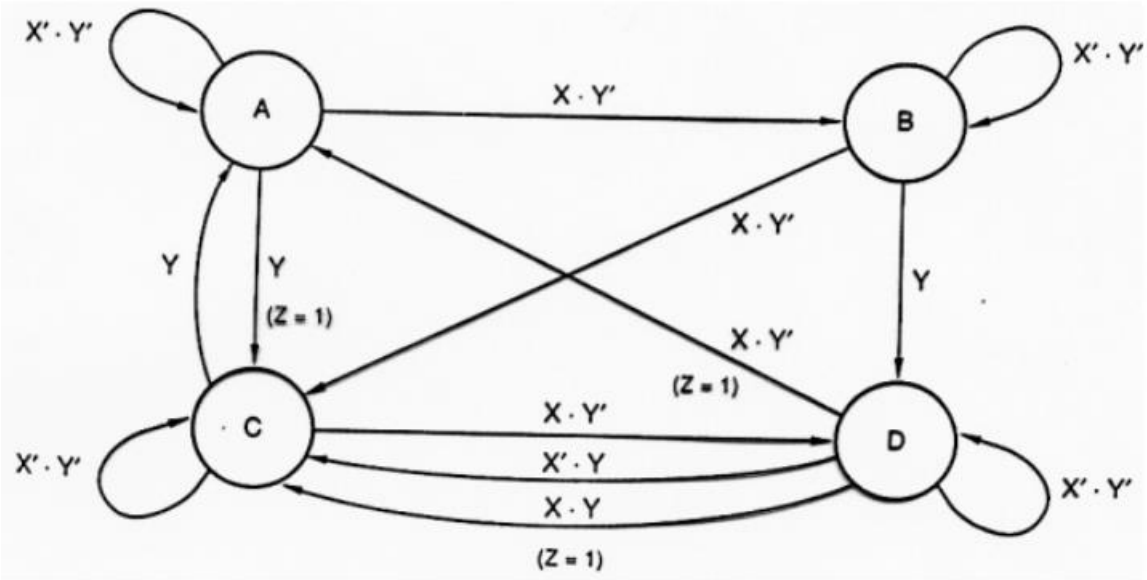
Autres appellations:

- Machine à états
- Machine à états finis
- Séquenceur
- Contrôleur
- Automate
- Machine séquentielle
- Machine séquentielle algorithmique.

Structure des machines à états finis



Exemples de diagramme d'états



I.3 GENERALITES SUR LES AUTOMATES PROGRAMMABLES INDUSTRIELS

1- Introduction

Les **Automates Programmables Industriels (API)** sont apparus aux Etats-Unis vers 1969 où ils répondaient aux désirs des industries de l'automobile de développer des chaînes de fabrication automatisées qui pourraient suivre l'évolution des techniques et des modèles fabriqués.

Un Automate Programmable Industriel (**API**) est une machine électronique programmable par un personnel non informaticien et destiné à piloter en ambiance industrielle et en temps réel des

procédés industriels. Un **automate programmable** est adaptable à un maximum d'application, d'un point de vue traitement, composants, language. C'est pour cela qu'il est de construction modulaire.

Il est en général manipulé par un personnel électromécanicien. Le développement de l'industrie à entraîner une augmentation constante des fonctions électroniques présentes dans un automatisme c'est pour ça que l'API s'est substitué aux armoires à relais en raison de sa souplesse dans la mise en œuvre, mais aussi parce que dans les coûts de câblage et de maintenance devenaient trop élevés.

2- Pourquoi l'automatisation ?

L'automatisation permet d'apporter des éléments supplémentaires à la valeur ajoutée par le système. Ces éléments sont exprimables en termes d'objectifs par :

- Accroître la productivité (rentabilité, compétitivité) du système
- Améliorer la flexibilité de production ;
- Améliorer la qualité du produit
- Adaptation à des contextes particuliers tel que les environnements hostiles pour l'homme (milieu toxique, dangeureux.. nucléaire...), adaptation à des tâches physiques ou intellectuelles pénibles pour l'homme (manipulation de lourdes charges, tâches répétitives parallélisées...),
- Augmenter la sécurité, etc...

Figure 4.1 : Automate SIEMENS S5-95U

3– Structure générale des API :

Les caractéristiques principales d'un automate programmable industriel (**API**) sont :
coffret, rack, baie ou cartes

- Compact ou modulaire
- Tension d'alimentation
- Taille mémoire
- Sauvegarde (EPROM, EEPROM, pile, …)
- Nombre d'entrées / sorties
- Modules complémentaires (analogique, communi-cation,..)
- Langage de programmation

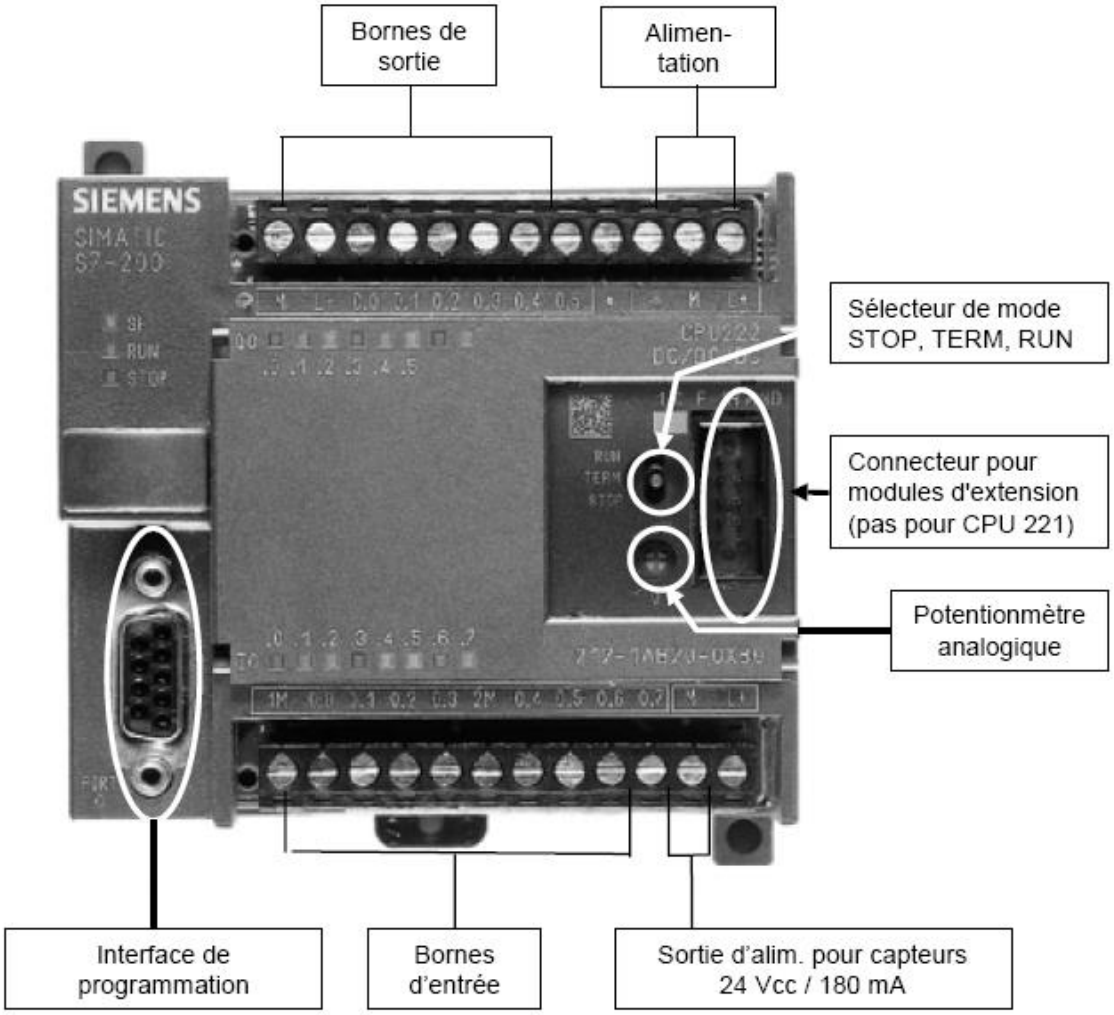


Figure 4.2 : Aspect extérieur d'un automate S7-200 CPU222

Des API en boîtier étanche sont utilisés pour les ambiances difficiles (température, poussière, risque de projection ...)

supportant ainsi une large gamme de température, humidité ...
L'environnement industriel se présentent sous trois formes :

- environnement physique et mécanique (poussières, température, humidité, vibrations);
- pollution chimique ;
- perturbation électrique. (parasites électromagnétiques)

Figure 4.3 : Automate Modulaire

4- Structure interne d'un automate programmable industriel (API) :

Les API comportent quatre principales parties (Figure 4.4) :

- Une unité de traitement (un processeur CPU);
- Une mémoire ;
- Des modules d'entrées-sorties ;
- Des interfaces d'entrées-sorties ;
- Une alimentation 230 V, 50/60 Hz (AC) - 24 V (DC).

La structure interne d'un **automate programmable industriel** (API) est assez voisine de celle d'un système informatique simple, L'unité centrale est le regroupement du processeur et de la mémoire centrale. Elle commande l'interprétation et l'exécution des instructions programme. Les instructions sont effectuées les unes après les autres, séquencées par une horloge.

Deux types de mémoire cohabitent :
- **La mémoire Programme** où est stocké le langage de programmation. Elle est en général figée, c'est à dire en lecture seulement. (ROM : mémoire morte)
- **La mémoire de données** utilisable en lecture-écriture pendant le fonctionnement c'est la RAM (mémoire vive). Elle fait partie du système entrées-sorties. Elle fige les valeurs (0 ou 1) présentes sur les lignes d'entrées, à chaque prise en compte cyclique de celle-ci, elle mémorise les valeurs calculées à placer sur les sorties.

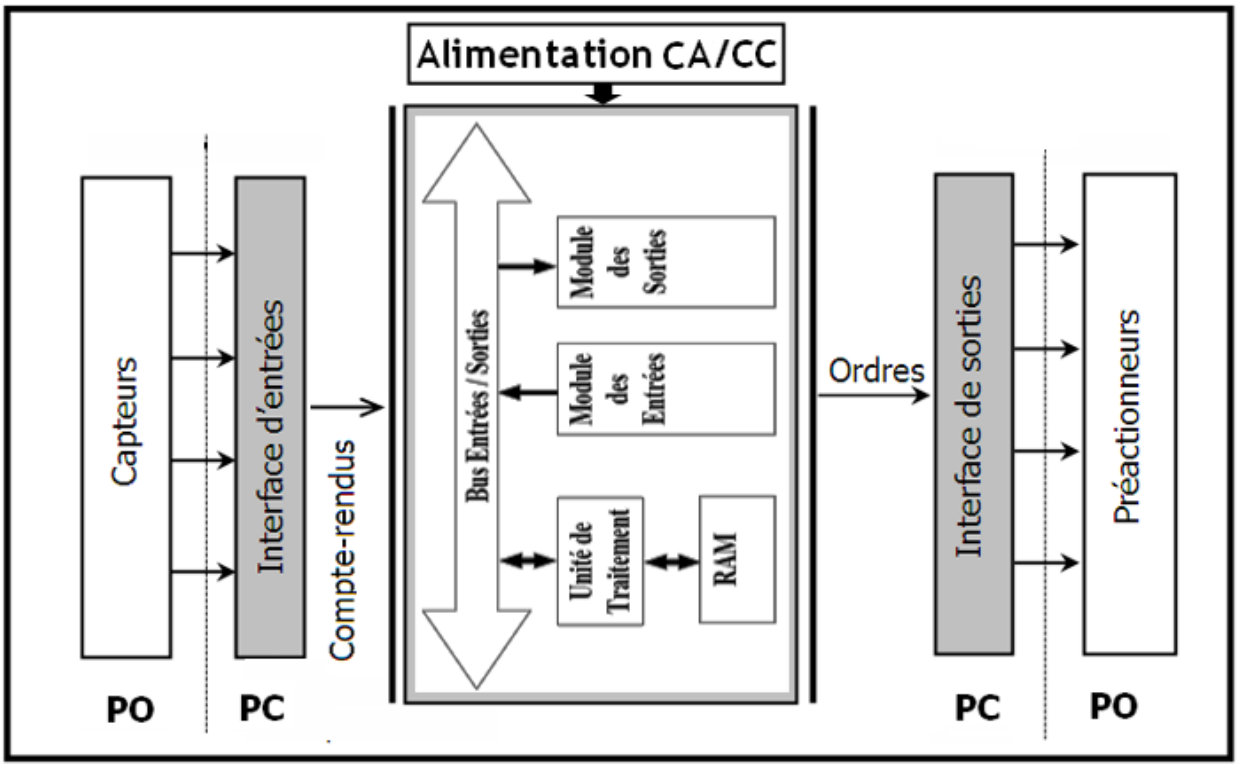


Figure 4.4 : Structure interne d'un automates programmables industriels (API)

5- Fonctionnement :

L'automate programmable **reçoit** les informations relatives à l'état du système et puis **commande** les pré-actionneurs suivant le programme inscrit dans sa mémoire.
Généralement les automates programmables industriels ont un fonctionnement cyclique (Figure 4.5). Le **microprocesseur** réalise toutes les fonctions logiques ET, OU, les fonctions de temporisation, de comptage, de calcul... Il est connecté aux autres éléments (mémoire et interface E/S) par des liaisons **parallèles** appelées ' **BUS** ' qui véhiculent les informations sous forme binaire.. Lorsque le fonctionnement est dit synchrone par rapport aux entrées et aux sorties, le cycle de traitement commence par la prise en compte des entrées qui sont figées en mémoire pour tout le cycle.

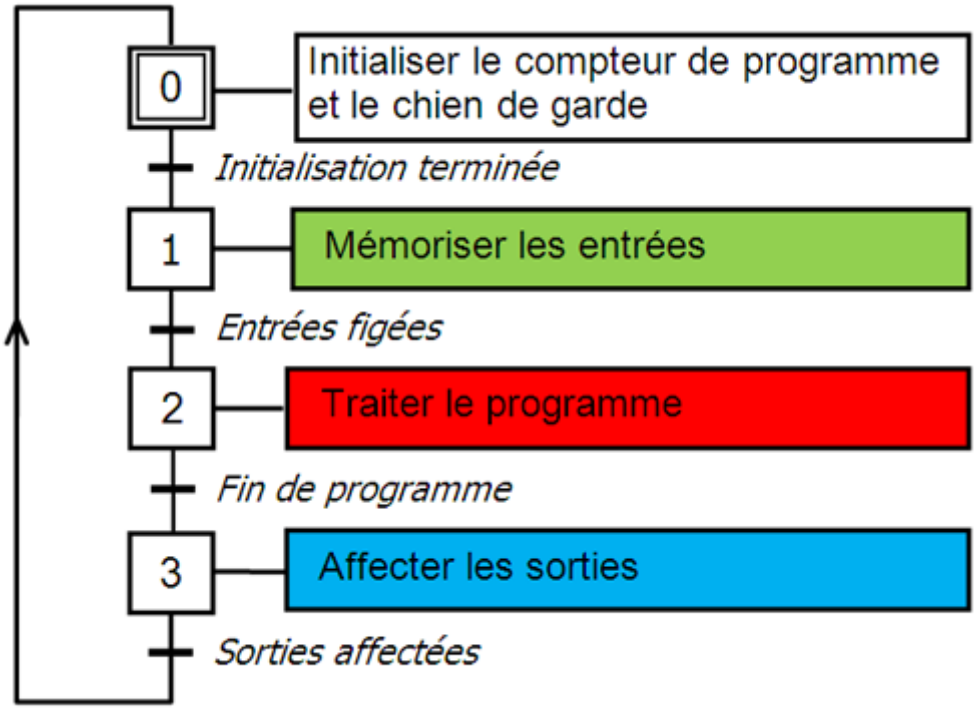


Figure 4.5 : Fonctionnement cyclique d'un API

I.4 GENERALITES SUR LES RESEAUX DES AUTOMATES

4.1- Principe

Avec le développement des systèmes automatisés et de l'électronique, la recherche de la baisse des coûts et la nécessité actuelle de pouvoir gérer au mieux la production et a partir du moment où tous les équipements sont de type informatique, il devient intéressant de les interconnecter à un mini-ordinateur ou à un automate de supervision (Figure 4.12).

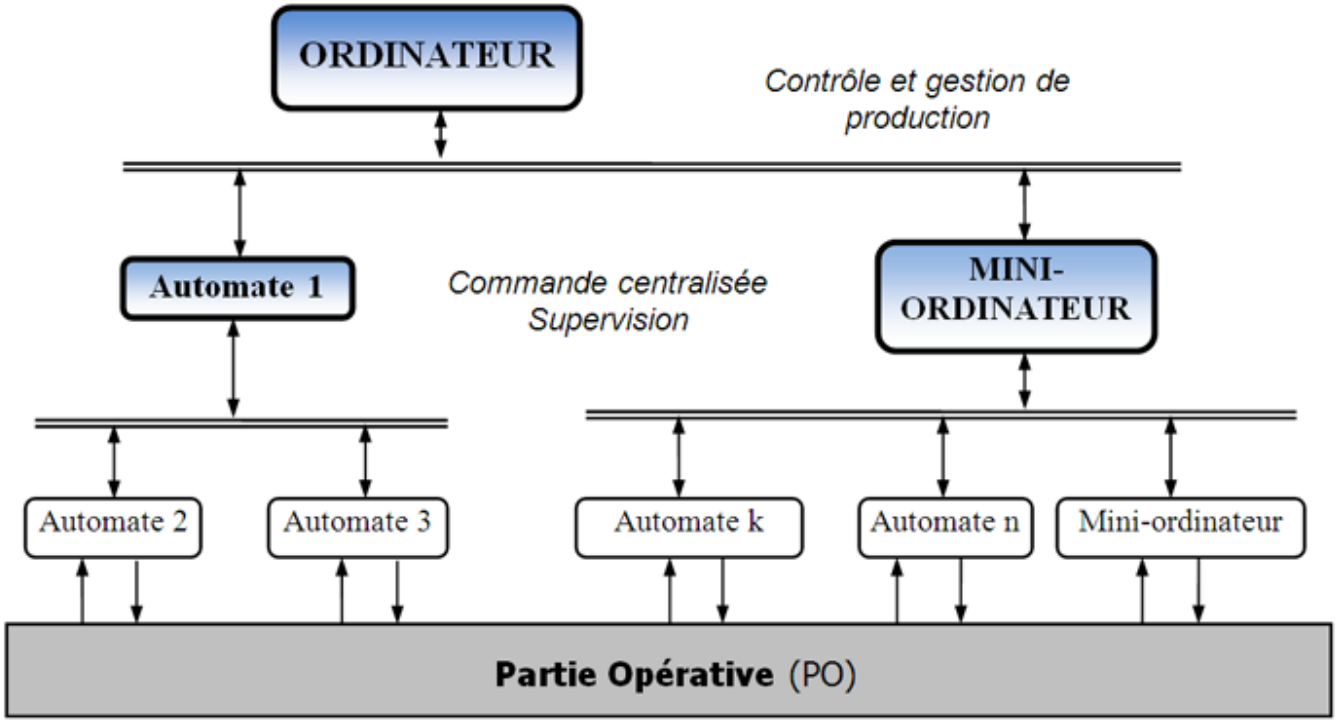


Figure 4.12: Exemple d'une structure de contrôle et gestion de production

L'interconnexion entre deux automates peut être réalisée très simplement en reliant une ou plusieurs sorties d'un automate à des entrées de l'autre et vice-versa (Figure 4.13).

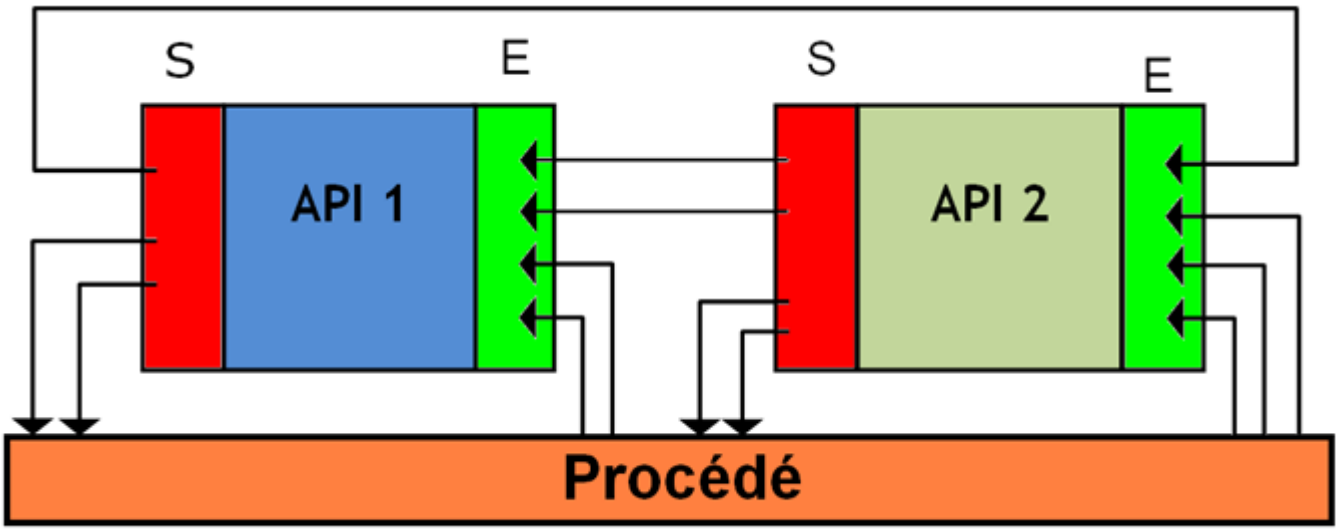


Figure 4.13: Interconnexion simple (Entrées/Sorties) entre deux automates (API)

Cette méthode ne permet pas de transférer directement des variables internes d'un automate sur l'autre, de sorte que celles-ci doivent être converties par programme en variables de sortie avant leur transfert. Elle devient coûteuse en nombre d’entrées/sorties mobilisé pour cet usage et lourde du point de vue du câblage, lorsque le nombre de variables qui doivent être échangées devient important.

4.2- Bus de terrain

Pour diminuer les coûts de câblage des entrées / sorties des automates, sont apparus les bus de terrains. L'utilisation de blocs d'entrées / sorties déportés à permis tout d'abord de répondre à cette exigence.

Les interfaces d'entrées/sorties sont déportées au plus près des capteurs. Avec le développement technologique, les capteurs, détecteurs ... sont devenus intelligents" et ont permis de se connecter directement à un bus.

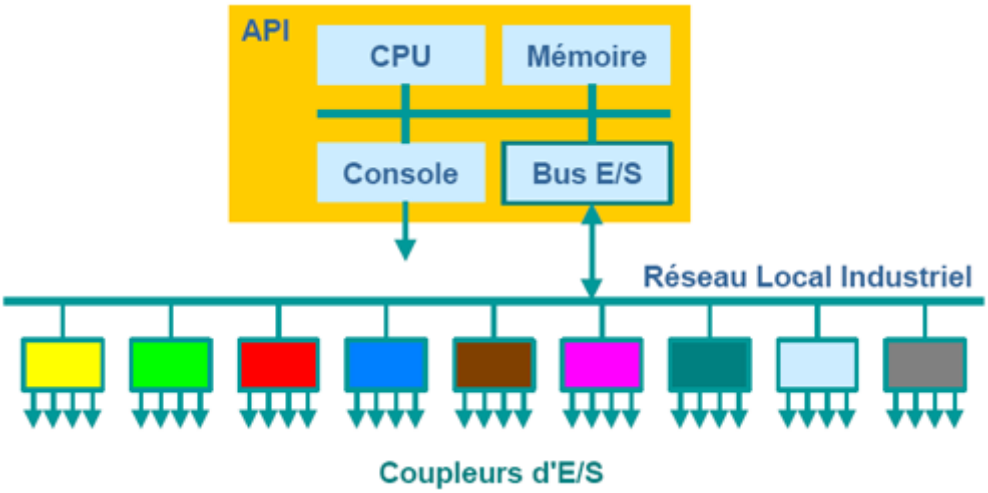


Figure 4.14: Interconnexion par entrées/sorties déportées

Plusieurs protocoles de communication et des standards sont apparus pour assurer le "multiplexage" de toutes les informations en provenance des capteurs / préactionneurs apr exemple le bus ASi (Actuators Sensors interface) est un bus de capteurs/actionneurs de type Maître / Esclave qui permet de raccorder 31 esclaves (capteurs ou préactionneurs) sur un câble spécifique (deux fils) transportant les données et la puissance.
Ce bus est totalement standardisé et permet d'utiliser des technologies de plusieurs constructeurs

Avantages des bus de terrain :

- Réduction des coûts de câblage et possibilité de réutiliser le matériel existant
- Réduction des coûts de maintenance

Inconvénients des bus de terrain :

- Taille du réseau limitée
- Latence dans les applications à temps critique
- Coût global

4.3- Différents types de réseaux d'automates :

4.3.1- Réseau en étoile :

Un centre de traitement commun échange avec chacune des autres stations. Deux stations ne peuvent pas échanger directement entre elles (Figure 4.15). Exemple le réseau de terrain BITBUS de la société INTEL

Avantages :

- Grande vitesse d'échange.
- Différent types de supports de transmission.
- Pas de gestion d'accès au support.

Inconvénients :

- Coût global élevé.
- Evolutions limitées.
- Tout repose sur la station centrale.

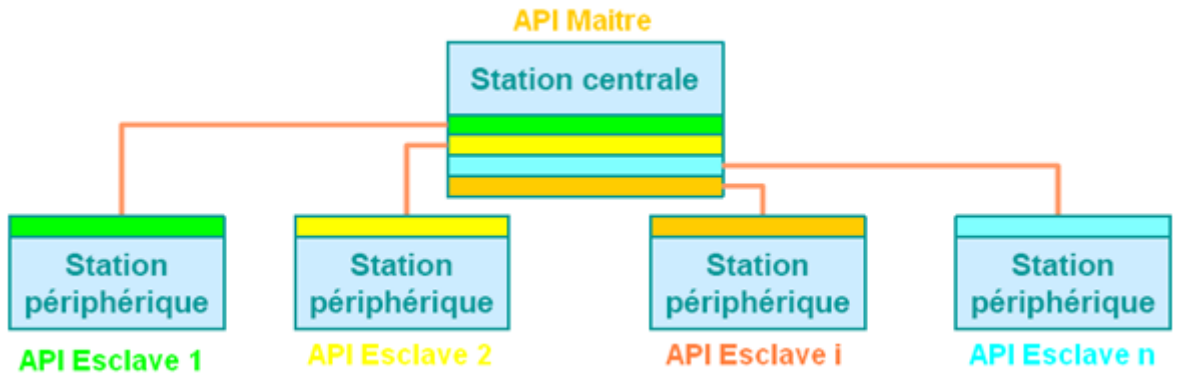


Figure 4.15: Interconnexion par entrées/sorties déportées

4.3.2- Réseau en anneau :

Chaque station peut communiquer avec sa voisine. Cette solution est intéressante lorsqu'une station doit recevoir des informations de la station précédente ou en transmettre vers la suivante (Figure 4.16).

Figure 4.16: Topologie Anneau

Avantages :

- Signal régénéré donc fiable.
- Contrôle facile des échanges (le message revient à l'émetteur).

Inconvénients :

- Chaque station est bloquante.
- Une extension interrompe momentanément le réseau.

4.3.3- Réseau hiérarchisé :

C'est la forme de réseaux la plus performante. Elle offre une grande souplesse d'utilisation, les informations pouvant circuler entre-stations d'un

même niveau ou circuler de la station la plus évoluée (en général un calculateur) vers la plus simple, et réciproquement (Figure 4.17).

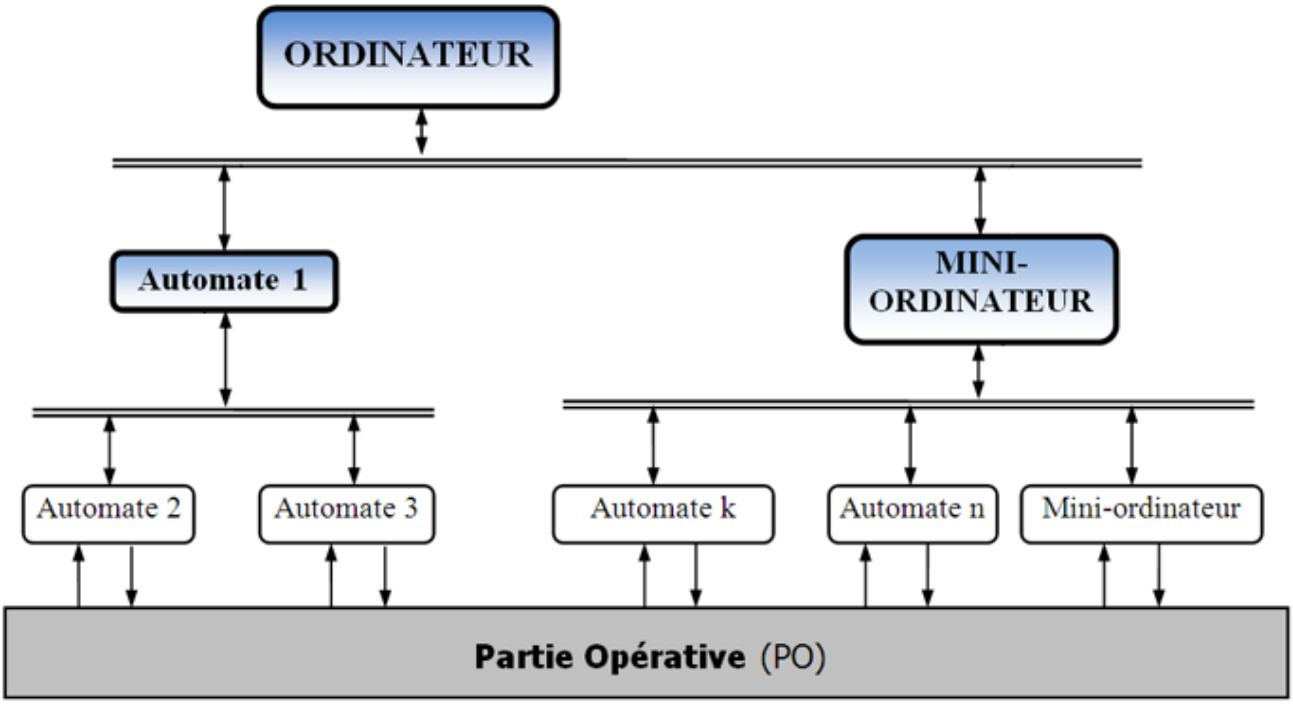
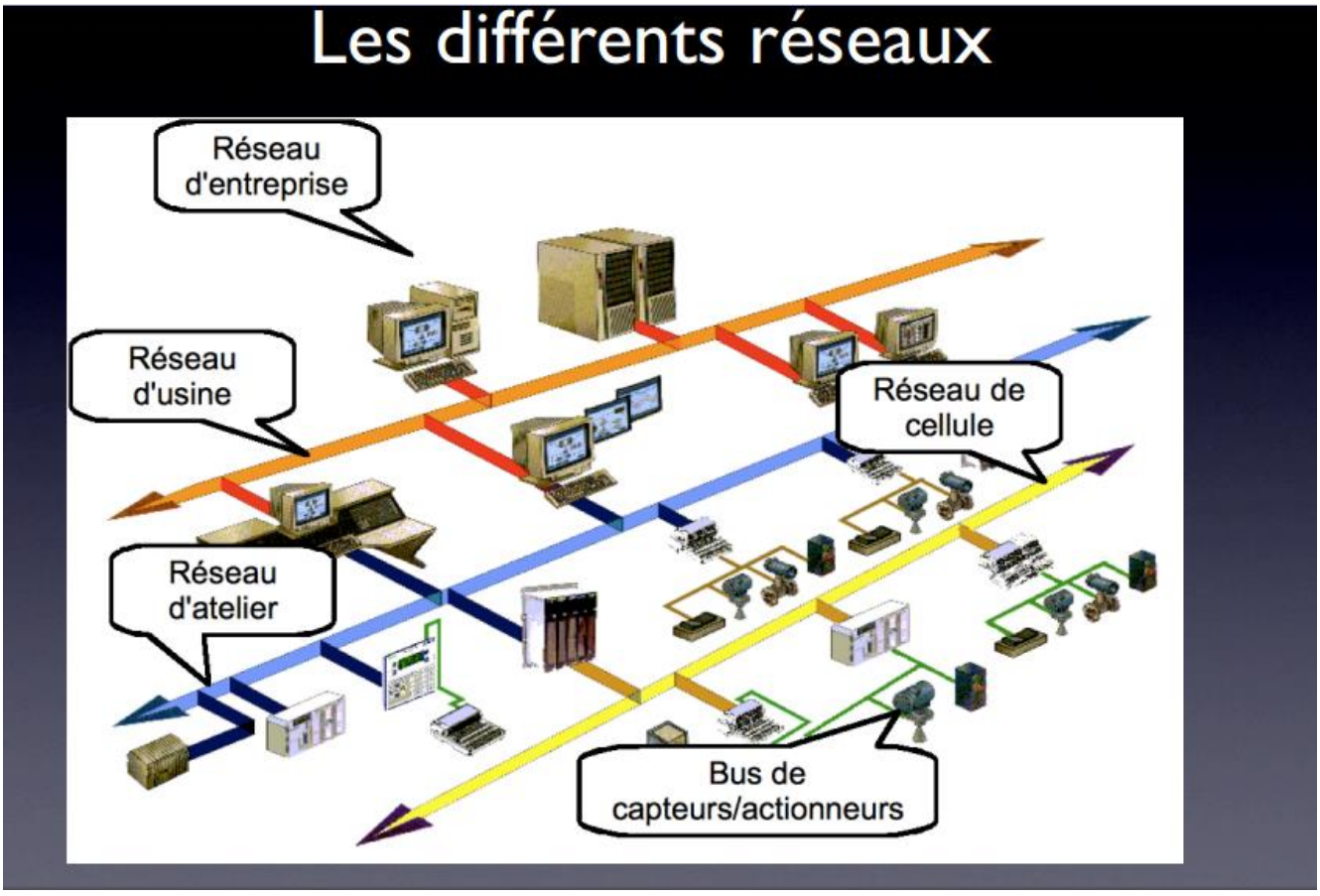
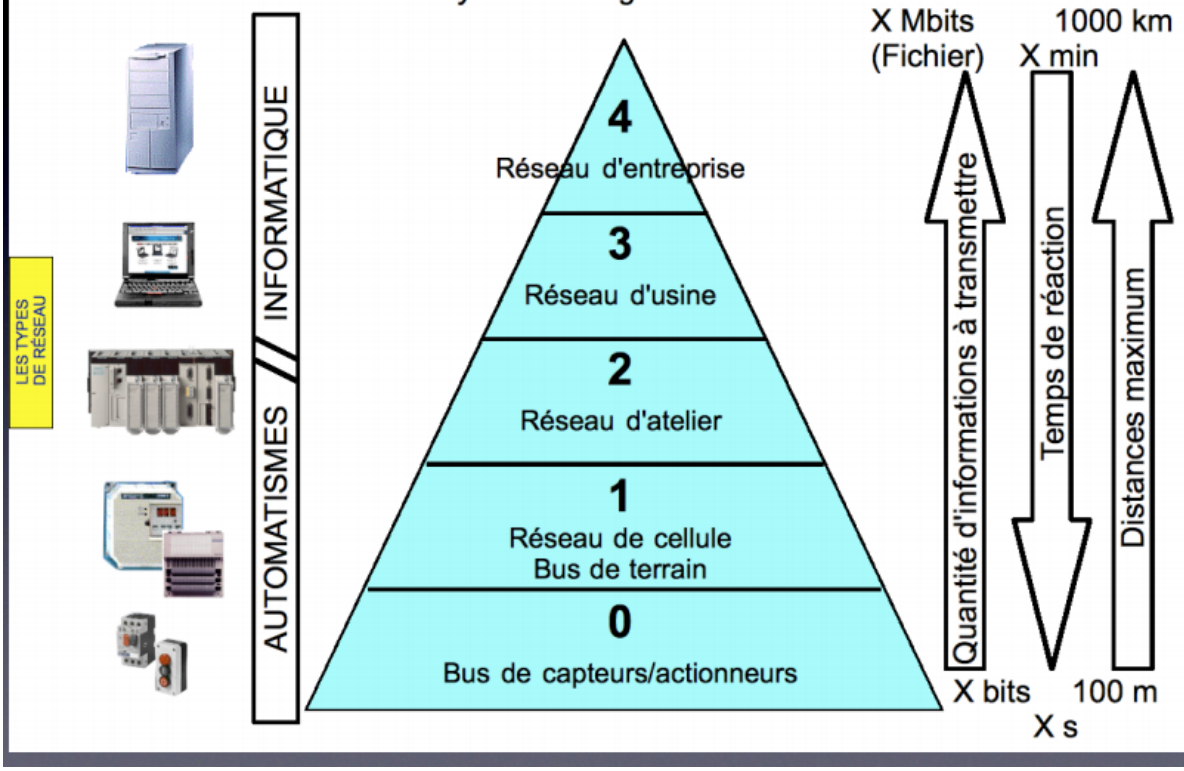


Figure 4.17: Réseau hiérarchisé



La pyramide CIM

CIM : Computer Integrated Manufacturing
Système Intégré de Production



Différents réseaux locaux industriels

- 4 grandes familles
  - ***SensorBus*** : Bus de capteurs/actionneurs
    - Au plus près des capteurs et des actionneurs
    - Déterministes
    - Temps de réponse très courts, actions réflexes
    - *Exemples :* As-i, Canopen
  - ***DeviceBus*** Bus de périphérie d'automatismes
    - Communication inter-automates
    - Partiellement déterministes
    - Orientés manufacturier haute vitesse
    - *Exemples :* Device WorldFip (DWF), FIPWAY, Profibus DP

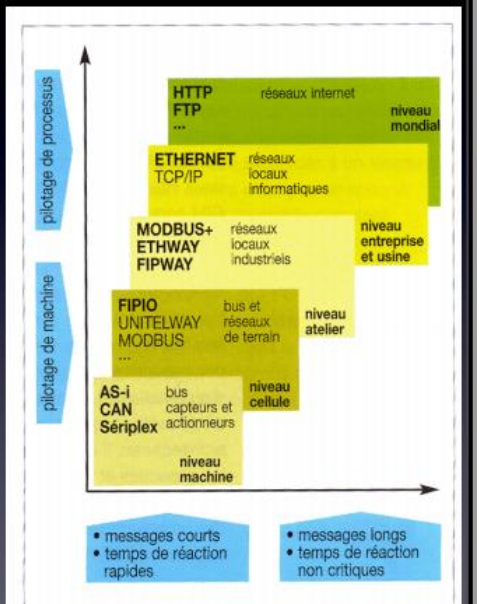
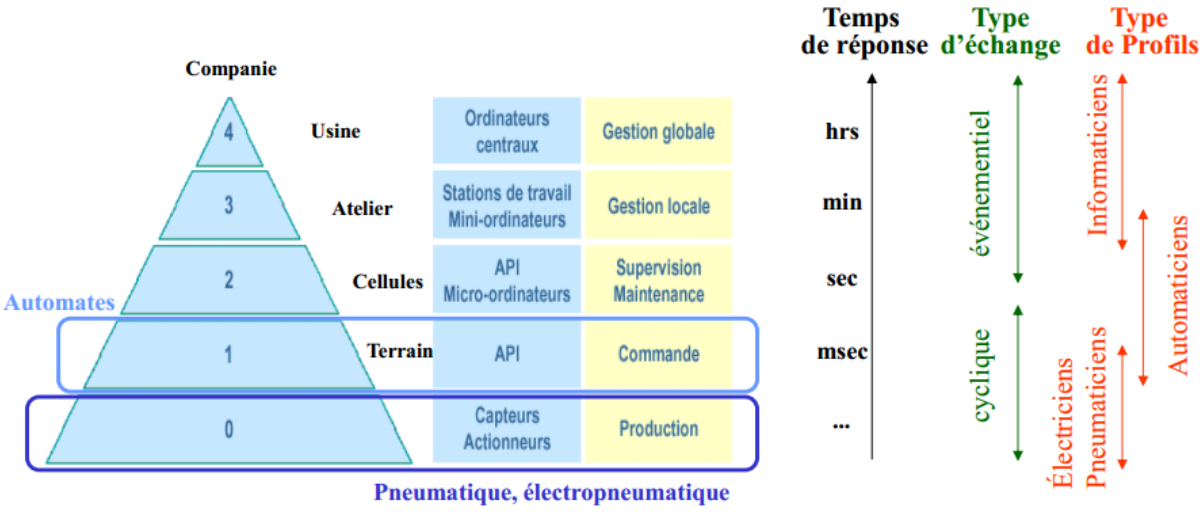


Figure 4. A chaque niveau correspond un bus ou un réseau.


Companie
4
Usine
Ordinateurs centraux
Gestion globale
3
Atelier
Stations de travail Mini-ordinateurs
Gestion locale
2
Cellules
API Micro-ordinateurs
Supervision Maintenance
Automates
1
Terrain
API
Commande
0
Capteurs Actionneurs
Production
Pneumatique, électropneumatique
Temps de réponse
Type d'échange
Type de Profils
hrs
min
sec
msec
...
événementiel
cyclique
Infomaticiens
Automaticiens
Électriciens
Pneumaticiens

CHAP.II LES AUTOMATES FINIS

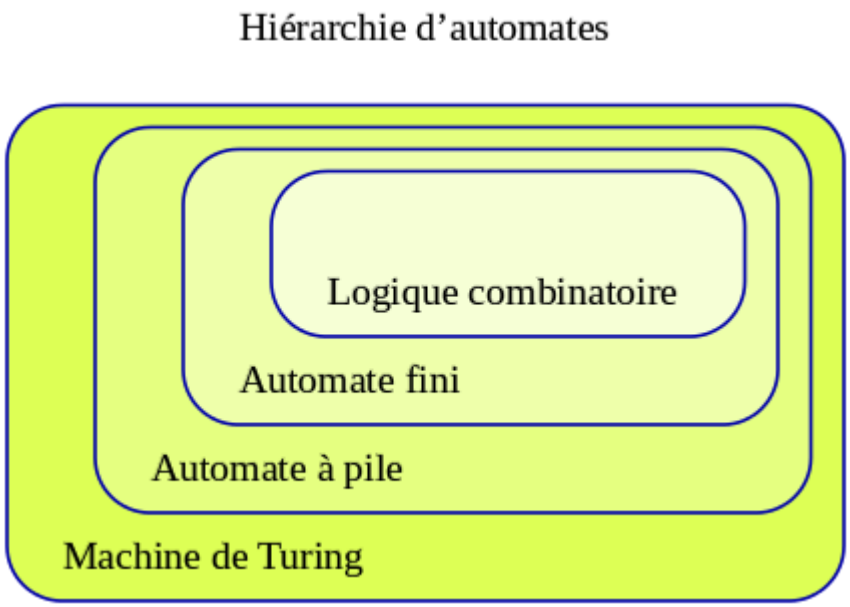
II.1 GENERALITES SUR LES AUTOMATES FINIS

Un automate fini ou automate avec un nombre fini d'états (en anglais *finite-state automaton* ou *finite state machine*) est un modèle mathématique de calcul, utilisé dans de nombreuses circonstances, allant de la conception de programmes informatiques et de circuits en logique séquentielle aux applications dans des protocoles de communication, le contrôle des processus, la linguistique et même la biologie. Un automate fini est une construction abstraite, susceptible d'être dans un nombre fini d'*états*, un seul état à la fois ; l'état où il se trouve est appelé l'« état courant ». Le passage d'un état à un autre est dirigé par un événement ou une condition ; ce passage est appelé une « transition ». Un automate particulier est défini par la liste de ses états et par les conditions des transitions.

On rencontre couramment des automates finis dans de nombreux appareils qui réalisent des actions déterminées en fonction des événements qui se présentent. Un exemple est un distributeur automatique de boissons qui délivre l'article souhaité quand le montant introduit est approprié, un autre les ascenseurs qui savent combiner les appels successifs pour s'arrêter aux étages intermédiaires, les feux de circulation capables de s'adapter aux voitures en attente, ou des digicodes qui analysent la bonne suite de chiffres.

Les automates finis peuvent modéliser un grand nombre de problèmes, parmi lesquels la conception assistée par ordinateur pour l'électronique, la conception de protocoles de communication, l'analyse syntaxique de langages et autres applications d'ingénierie. Dans la recherche en biologie et en intelligence artificielle, les automates finis ou des hiérarchies de telles machines ont été employés pour décrire des systèmes neurologiques. En linguistique, ils sont utilisés pour décrire les parties simples de grammaires de langues naturelles. En vérification de programmes (*model checking*), des automates finis, avec parfois un nombre très important d'états, sont employés.

Si l'on considère les automates finis comme un concept abstrait de calcul, leur puissance est pourtant faible ; ils ont bien moins de puissance de calcul qu'une machine de Turing. En d'autres termes, il y a des tâches qu'un automate fini ne peut pas accomplir alors qu'une machine de Turing peut le faire. Ceci est principalement dû au fait qu'un automate fini a une mémoire limitée par son nombre d'états.



Les automates finis sont étudiés dans le cadre plus général de la théorie des automates. La théorie des automates est l'étude des machines abstraites qui permettent de formaliser les méthodes de calcul. L'objet traité par un automate est un mot d'un langage. Pour arriver à la généralité souhaitée, on convertit un « problème » en un langage, et la résolution du problème, en l'analyse d'un élément de ce langage.

On représente chaque instance d'un « problème » par un mot. Savoir si l'instance du problème a une solution se ramène à tester si ce mot appartient au langage des mots représentant les instances ce problème et qui ont une solution. Un automate qui résout le problème prend en entrée un mot et décide s'il est accepté ou non.

Par exemple, le problème de savoir si un entier *N* est premier (test de primalité) peut se traduire comme suit : on représente tous les entiers naturels par des chaînes binaires (écriture en base 2). Dans ce langage, les mots représentant des nombres premiers forment un sous-ensemble. Le problème du test de primalité consiste alors à savoir si la chaîne binaire représentant un nombre *N* appartient à ce sous-ensemble ou non. Un automate approprié prend en entrée une chaîne binaire et l'accepte précisément lorsqu'elle représente un nombre premier.

La formulation des problèmes et de leur résolution (ou même de leur calculabilité) en termes de langage formels est à la base des hiérarchies de complexité, et des hiérarchies des langages formels.

Un autre domaine concerne la *transformation* de mots. Dans ce domaine, on utilise plutôt le terme de « machine » ou de transducteur. La linguistique, mais aussi la compilation, font usage de tels transducteurs pour l'analyse et la transformation de textes ou de programmes.

Les automates finis peuvent être classés principalement en deux catégories, les *accepteurs* et les *transducteurs*. Les accepteurs analysent la structure de la donnée fournie, et l'acceptent si elle est conforme à la spécification décrite par l'automate. Les transducteurs au contraire traduisent une chaîne de symboles en une autre, là encore selon l'algorithme codé dans l'automate. Dans certains cas, on peut rencontrer des variantes appelées *classificateurs* et *séquenceurs*.

Les accepteurs, également appelés reconnaisseurs produisent une sortie binaire, indiquant si l'entrée reçue est acceptée ou non. Chaque état d'un tel automate est soit un état d'acceptation, aussi appelé final ou terminal, ou un état de rejet. Si l'état courant, après la lecture de la totalité de l'entrée, est un état d'acceptation, l'entrée est acceptée, sinon elle est rejetée. L'entrée est généralement une suite de symboles (des lettres); il n'y a pas d'actions associées.

Les transducteurs finis génèrent en sortie des mots en fonction d'un mot d'entrée donné et d'actions associées aux états. Ils sont utilisés par exemple dans des applications de contrôle et dans le domaine de la linguistique informatique.

Un classificateur est similaire à un accepteur, mais possède au moins deux états terminaux. Un tel automate peut donc accepter plusieurs langages simultanément, selon la catégorie d'états à laquelle appartient l'état terminal atteint en fin de lecture.

Un séquenceur ou générateur est un cas particulier d'automate, opérant sur un alphabet d'entrée à une seule lettre. Un tel automate produit une seule séquence qui peut être interprétée comme le résultat d'un transducteur ou d'un classificateur.

Une distinction supplémentaire importante est celle entre automates finis déterministes et non déterministes. Dans **un automate déterministe**, chaque état possède au plus une transition pour chaque symbole d'entrée (et même exactement une dans le cas où l'automate est complet). Dans **un automate non déterministe**, un symbole d'entrée peut étiqueter une, plusieurs ou aucunes transitions pour un état donné. Cette distinction est importante en pratique, mais moins en théorie parce qu'il existe un algorithme (la construction par sous-ensembles) qui

permet de transformer un automate fini non déterministe en un automate fini déterministe avec la même fonctionnalité.

II.2 LES AUTOMATES FINIS DETERMINISTES(AFD)

II.2.1 INTRODUCTION

Les automates finis à états (automates finis en abrégé) offrent un formalisme de description peu puissant mais avec beaucoup d'algorithmes efficaces. Ils sont très utilisés notamment dans deux domaines : le traitement de chaînes de caractères et la description de comportement dynamique de systèmes.

Un automate fini est un graphe orienté dont les arcs sont étiquetés par des symboles. Voyons un exemple.

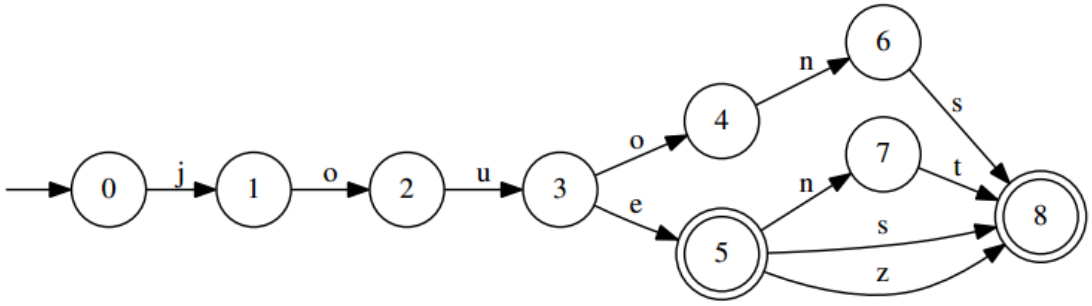


FIGURE 1 – exemple d'automate fini

La notation du graphe met en évidence deux types de noeuds particulier. D'une part l'état initial, qui est caractérisé par une flèche entrante venant de nulle part et sans étiquette (ici l'état 0). D'autre part les états finals, identifiés par un double cercle (ici les états 5 et 8). Il y a toujours exactement un état initial, alors qu'il peut y avoir 0, 1 ou plusieurs états finals. Les noeuds du graphe sont appelés états et les arcs sont appelés des transitions. Dans un tel graphe, on s'intéresse aux chemins qui vont de l'état initial à un état final. A chaque chemin est associée la chaîne des étiquettes des arcs parcourus.

Par exemple, il y a un chemin qui part de l'état 0, passe par les états 1, 2, 3 et arrive en 5, qui est un état final. La chaîne correspondante est *joue*. Il existe de même des chemins pour les chaînes *joues, jouent, jouez et jouons*. L'ensemble de ces chaînes forme ce que l'on appelle le langage de l'automate. Cet exemple est donc un automate qui représente le présent de l'indicatif du verbe *jouer*. Notez que des chemins différents partagent des parties communes. Par exemple, ici, le préfixe *jou* est représenté une fois alors qu'il appartient à toutes les chaînes.

II.2.2 QUELQUES DEFINITIONS

Définition 1 *Alphabet, chaîne*
On appelle alphabet un ensemble fini de symbole. Une chaîne sur un alphabet Σ est une séquence éventuellement vide de symboles de Σ. La séquence vide est notée ϵ (epsilon). Les autres séquences sont notées par la juxtaposition des symboles qui les composent.

Définition 2 *Langage*
Un langage est un ensemble de chaînes sur un alphabet Σ.

Définition 3 *Automate fini*
Un automate fini est un quintuplet $A = (\Sigma, Q, \delta, i, F)$ où :
- *Σ est un ensemble fini de symboles appelé alphabet.*
- *Q est un ensemble fini dont les éléments sont appelés* états.
- *δ est une relation de $Q \times \Sigma \times Q$ appelée* transition *ou* ensemble des transitions *de A.*
- *i est un état de Q appelé* état initial.
- *F est un sous-ensemble de Q appelé* ensemble des états finals *de A.*

L'ensemble des transitions δ est une relation, c'est-à-dire un ensemble de triplets. Cet ensemble est nécessairement fini puisque Q et Σ sont finis. Un automate fini est fait de composantes qui sont toutes finies (Σ, Q, δ, F), d'où le qualificatif de *fini*.

Définition 4 *Représentation graphique*
Un automate fini peut être représenté graphiquement comme un graphe orienté dont les sommets sont les états et les arcs sont les transitions. Une transition (q_1, x, q_2) est représentée par un arc reliant les sommets q_1 et q_2, étiqueté par x.

Nous avons déjà vu un exemple de représentation graphique (figure 1). Reprenons pour cet exemple les différentes définitions. L'alphabet de l'automate est l'ensemble $\{e, j, n, o, s, t, u, z\}$. Le quintuplet décrivant l'automate est le suivant : $(\Sigma, \{0, 1, 2, 3, 4, 5, 6, 7, 8\}, \delta, 0, \{4, 8\})$ avec
- $\Sigma = \{e, j, n, o, s, t, u, z\}$

– $\delta = \{(0, j, 1), (1, o, 2), (2, u, 3), (3, e, 4), (3, o, 5), (4, n, 6), (4, s, 8), (4, z, 8), (5, n, 7), (6, t, 8), (7, s, 8)\}$

Définition 5 *Chemin*
Soit $A = (\Sigma, Q, \delta, i, F)$ un automate. Un chemin de cet automate est une séquence $(q_0, x_0, q_1)(q_1, x_1, q_2) \dots (q_{n-1}, x_{n-1}, q_n)$ de transitions de δ telle que $n \geq 0$. La chaîne associée à ce chemin est $x_0 x_1 \dots x_{n-1}$. On notera un chemin de la façon suivante : $q_0 \xrightarrow{x_0} q_1 \xrightarrow{x_1} q_2 \dots q_{n-1} \xrightarrow{x_{n-1}} q_n$

Cette notion de chemin correspond à celle de chemin en théorie des graphes.

Définition 6 *Chemin succès*
Soit $A = (\Sigma, Q, \delta, i, F)$ un automate. Un chemin $q_0 \xrightarrow{x_0} q_1 \xrightarrow{x_1} q_2 \dots q_n$ de cet automate est appelé un **succès** *si $q_0 = i$ et $q_n \in F$.*

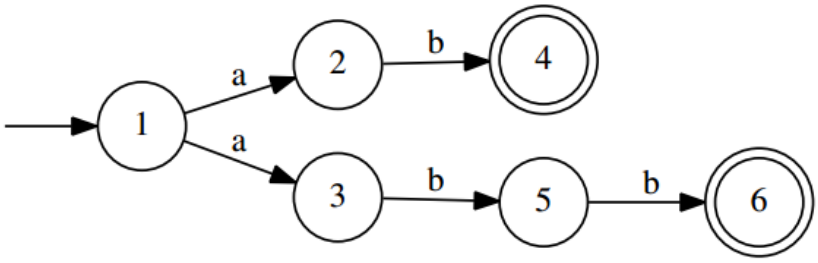
En d'autres termes, un chemin succès est un chemin qui part de l'état initial et qui arrive dans un état final.

Définition 7 *Chaîne associée à un chemin*
Soit $A = (\Sigma, Q, \delta, i, F)$ un automate et $q_0 \xrightarrow{x_0} q_1 \xrightarrow{x_1} q_2 \dots q_n$ un chemin de A. La chaîne associée à ce chemin est $x_0 \dots x_{n-1}$.

Définition 8 *Langage défini par un automate fini*
Soit A un automate. Le langage défini par cet automate est l'ensemble des chaînes associées aux chemins succès de cet automate.

Sur l'exemple de la figure 1, voici deux exemples de chemins.
– $0 \xrightarrow{j} 1 \xrightarrow{o} 2 \xrightarrow{u} 3 \xrightarrow{e} 4$
– $0 \xrightarrow{j} 1 \xrightarrow{o} 2 \xrightarrow{u} 3 \xrightarrow{o} 5 \xrightarrow{n} 7$
Les deux chemins commencent dans l'état initial 0. Le premier chemin se termine dans l'état 4 qui est un état final. Donc ce chemin est un succès. Le second chemin se termine dans l'état 7 qui n'est pas final. Ce n'est donc pas un succès. Les chaînes associées à ces deux chemins sont respectivement *joue* et *jouon*. Le language défini par l'automate est l'ensemble $\{joue, joues, jouons, jouez, jouent\}$.

Attention, une chaîne appartient à un langage s'il existe un chemin succès associé à cette chaîne. Cela ne signifie pas que tous les chemins partant de l'état initial et associés à cette chaîne sont des succès. Par exemple dans l'automate suivant, il y a deux chemins pour la chaîne ab.



Il y a le chemin $1 \xrightarrow{a} 2 \xrightarrow{b} 4$ qui est un succès parce que 4 est un état final et $1 \xrightarrow{a} 3 \xrightarrow{b} 5$ qui est un échec parce que 5 n'est pas un état final. La chaîne appartient au langage de l'automate puisqu'il existe un chemin succès. Donc l'existance d'un échec ne dit rien sur l'appartenance de la chaîne au langage.

II.2.3. EXECUTION D'UN AUTOMATE FINI

II.2.3.1 DEFINITIONS LIEES A L'EXECUTION D'UN AUTOMATE FINI

Nous avons vu dans la section précédente qu'un automate fini définit un langage, c'est-à-dire un ensemble de chaîne. Mais un automate est aussi une machine, que l'on peut exécuter pour tester l'appartenance d'une chaîne à ce langage ou pour générer les chaînes de ce langage.

Nous allons d'abord voir l'exécution d'un automate en reconnaissance.

Définition 9 *Parcours partiel, parcours complet, parcours succès*
Un parcours partiel d'automate est une paire comprenant un chemin partant de l'état initial et une chaîne. Un parcours complet est un parcours comprenant un chemin partant de l'état initial et la chaîne vide. Un parcours succès est un parcours complet dont le chemin est un succès, c'est à dire un chemin se terminant dans un état final.

Un parcours partiel reflète un état intermédiaire d'un parcours d'un chemin. Le dernier état du chemin donne l'état dans lequel on est arrivé et la chaîne est ce qu'il reste à reconnaître sur la suite du chemin. C'est un suffixe de la chaîne que l'on teste.

Définition 10 *Pas de calcul, calcul, calcul succès*
Un pas de calcul permet de passer d'un parcours partiel $(i \xrightarrow{x} \dots q, aw)$ *où* $a \in \Sigma$ *et* $w \in \Sigma^*$ *à un parcours partiel* $(i \xrightarrow{x} \dots q \xrightarrow{a} q', w)$ *s'il existe une transition* (q, a, q') *dans* δ.
Un calcul est une succession de parcours partiels $p_1, \dots p_n$ *tels que pour tout* i, *on peut passer de* p_i *à* p_{i+1} *par un pas de calcul.*
Un calcul succès est un calcul $p_1, \dots p_n$ *tel que* p_n *est un parcours succès.*

Un calcul est un moyen de parcourir un chemin du graphe à la recherche d'un chemin succès (c'est à dire un chemin partant de l'état initial et arrivant dans l'état final, étiqueté par w).

Propriété 1 *Soit* $A = (\Sigma, Q, \delta, i, F)$ *un automate. Une chaîne* w *appartient à* $L(a)$ *si et seulement si il existe un calcul succès commençant du parcours partiel* (i, w).

Attention, il faut qu'un calcul succès existe, cela ne signifie pas que tous les calculs partant de cette configuration doivent être des succès. S'il y a plusieurs chemins dans le graphe partant de l'état initial qui sont étiquetés par w, il suffit qu'un de ces chemins arrive dans un état final pour que w appartienne au langage. Pour prouver que w appartient à $L(A)$, il faut trouver un chemin succès. Pour prouver que w **n'appartient pas** à $L(A)$, il faut prouver qu'aucun chemin n'est un succès. Il faut donc avoir regardé **tous** les chemins pour tirer cette conclusion.

II.2.3.1 EXEMPLES D'EXECUTION D'UN AUTOMATE FINI

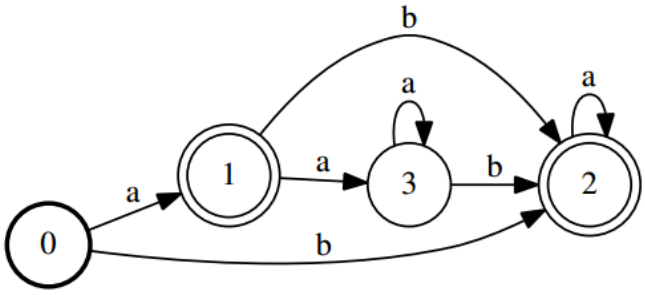


FIGURE 2 – automate à exécuter

Prenons comme exemple l'automate de la figure 2. Voyons pour cet automate un calcul qui permet de montrer que `aaba` appartient au langage. Le parcours partiel de départ est `(0,aaba)`. Le calul succès est le suivant :

$(0, aaba), (0 \xrightarrow{a} 1, aba), (0 \xrightarrow{a} 1 \xrightarrow{a} 3, ba), (0 \xrightarrow{a} 1 \xrightarrow{a} 3 \xrightarrow{b} 2, a), (0 \xrightarrow{a} 1 \xrightarrow{a} 3 \xrightarrow{b} 2 \xrightarrow{a}, \epsilon)$

Ce calcul est un succès parce que le dernier parcour a une chaîne vide et son chemin arrive dans l'état final 2.

On peut définir un notion de calcul et d'exécution d'automate en *génération*, pour énumérer les chaînes d'un automate. Nous ne détaillerons pas cela ici. Le principe général consiste à accumuler les symboles vus sur la partie du chemin déjà parcourue au lieu de stocker le reste de ce qui reste à trouver sur le chemin non encore parcouru.

REMARQUE

- Dans un automate fini déterministe, il n'y a qu'un seul et état initial i ;

- Dans un automate fini déterministe, pour tout état q_i étiqueté par la lettre x_i , il n' y a qu'une et une seule transition partant de q_i étiqueté par la lettre x_i ;

- Dans un automate fini déterministe, si on veut lire un mot à aucun moment on a un choix à faire. On doit commencer par l'unique état initial et lorsqu'on lit une lettre, il n'y a qu'une et une seule transition qu'on peut emprunter.

II.3 LES AUTOMATES FINIS NON DETERMINISTES (AFND)

Le modèle d'automate fini non déterministe généralise le cas des AFD. Comme nous le verrons bientôt, le non-déterminisme permet une plus grande souplesse bien utile dans certaines situations.

Définition II.2.1. Un *automate fini non déterministe* (AFND) est la donnée d'un quintuple

$$\mathcal{A} = (Q, I, F, \Sigma, \Delta)$$

où

- Q est un ensemble fini dont les éléments sont les *états* de $\mathcal{A}$,
- $I \subseteq Q$ est l'ensemble des *états initiaux*,
- $F \subseteq Q$ désigne l'ensemble des *états finals*,
- Σ est l'alphabet de l'automate,
- $\Delta \subset Q \times \Sigma^* \times Q$ est une *relation de transition* (qu'on supposera finie).

On peut dès à présent noter plusieurs différences entre les AFD et AFND. Dans le cas non déterministe, il est possible d'avoir plus d'un état intial; les labels des arcs ne sont plus nécessairement des lettres mais bien des mots de Σ^* et enfin, on n'a plus une fonction de transition mais une relation de transition. Pour représenter les AFND, nous utilisons les mêmes conventions que pour les AFD.

Exemple II.2.2. L'automate de la figure II.3 est un AFND ayant 1 et 3

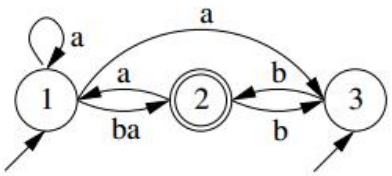


FIGURE II.3. Un AFND.

comme états initiaux, 2 comme état final et la relation de transition est

$$\Delta = \{(1, a, 1), (1, a, 3), (1, ba, 2), (2, a, 1), (2, b, 3), (3, b, 2)\}$$

Définition II.2.3. Un mot $w = w_1 \cdots w_k$ est *accepté* par un AFND $\mathcal{A} = (Q, I, F, \Sigma, \Delta)$ s'il existe $q_0 \in I$, $\ell \in \mathbb{N} \setminus \{0\}$, $v_1, \ldots, v_\ell \in \Sigma^*$, $q_1, \ldots, q_\ell \in Q$ tels que

$$(q_0, v_1, q_1), (q_1, v_2, q_2), \ldots, (q_{\ell-1}, v_\ell, q_\ell) \in \Delta,$$
$$w = v_1 \cdots v_\ell \quad \text{et} \quad q_\ell \in F.$$

En d'autres termes, cette condition signifie qu'il existe un chemin dans le graphe associé à $\mathcal{A}$ débutant dans un état initial, de label w et se terminant dans un état final. Naturellement, le langage *accepté* par un AFND $\mathcal{A}$ est l'ensemble des mots acceptés par $\mathcal{A}$ et se note encore $L(\mathcal{A})$. Enfin, deux AFND $\mathcal{A}$ et $\mathcal{B}$ sont dits *équivalents* si $L(\mathcal{A}) = L(\mathcal{B})$.

Exemple II.2.4. Si nous poursuivons l'exemple II.2.2, le mot ab est accepté car $1 \in I$, $(1, a, 3) \in \Delta$, $(3, b, 2) \in \Delta$ et $2 \in F$. Ceci se note schématiquement,

$$1 \xrightarrow{a} 3 \xrightarrow{b} 2.$$

A un mot, il peut correspondre plus d'un chemin. Par exemple, au mot baa, il correspond les chemins

$$3 \xrightarrow{b} 2 \xrightarrow{a} 1 \xrightarrow{a} 1,$$

$$3 \xrightarrow{b} 2 \xrightarrow{a} 1 \xrightarrow{a} 3$$

et

$$1 \xrightarrow{ba} 2 \xrightarrow{a} 1.$$

Ce sont les trois seules possibilités partant d'un état initial. Le mot baa n'est donc pas accepté par l'automate.

Remarque II.2.5. Dans la définition d'un AFND, rien n'empêche d'avoir des transitions "vides" du type

$$(q, \varepsilon, q') \in \Delta.$$

On parle parfois de ε-transitions. En particulier, on suppose implicitement que pour tout état q d'un AFND, on a

$$(q, \varepsilon, q) \in \Delta.$$

Exemple II.2.6. Considérons l'AFND suivant. Cet automate accepte le

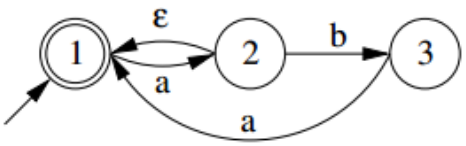


FIGURE II.4. Un AFND avec ε-transitions.

mot a car on a le chemin

$$1 \xrightarrow{a} 2 \xrightarrow{\varepsilon} 1 \in F.$$

Observons encore qu'il n'est pas possible depuis l'état initial de lire des mots débutant par b. Donc, contrairement à la situation déterministe où, à chaque mot correspondait exactement une exécution, ici, on peut avoir pour un mot donné plus d'un chemin dans le graphe, voire même aucun.

Définition II.2.7. Un AFND $\mathcal{A} = (Q, I, F, \Sigma, \Delta)$ est qualifié d'*élémentaire* si pour tout $(q, w, q') \in \Delta$,

$$|w| \leq 1.$$

Comme le montre le lemme suivant, on peut dans le cadre des AFND se restreindre au cas d'automates élémentaires. En effet, tout AFND est équivalent à un AFND élémentaire.

Lemme II.2.8. *Tout langage accepté par un AFND est accepté par un AFND élémentaire.*

Exemple II.2.9. Appliquons la méthode de construction donnée dans la preuve du lemme précédent à un exemple. Soit l'AFND non élémentaire $\mathcal{A}$ représenté à la figure II.5.

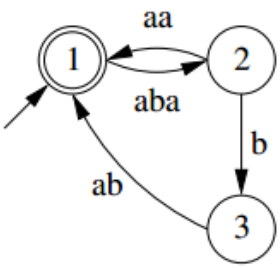


FIGURE II.5. Un AFND non élémentaire $\mathcal{A}$.

On obtient alors un automate élémentaire équivalent à $\mathcal{A}$ représenté à la figure II.6, les états supplémentaires ne portant pas de numéro.

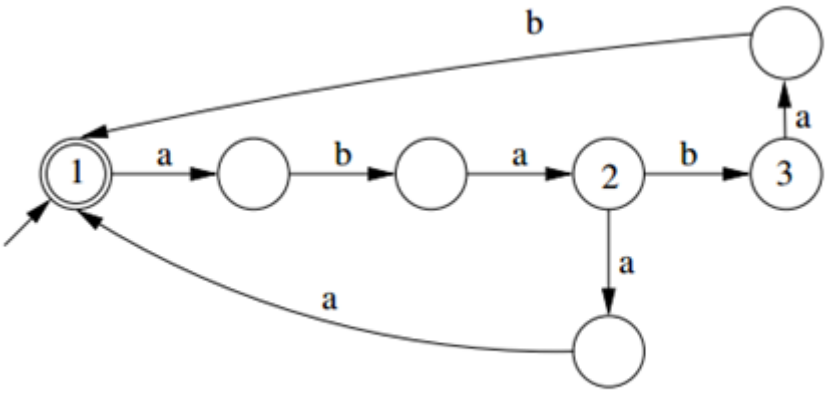


FIGURE II.6. Un AFND élémentaire équivalent à $\mathcal{A}$.

Remarque II.2.10. Soit $\mathcal{A} = (Q, I, F, \Sigma, \Delta)$ un AFND. Si $R \subseteq Q$ et $w \in \Sigma^*$, on note

$$R.w$$

l'ensemble des états atteints à partir des états de R en lisant w. Par exemple, avec l'automate de la figure II.4, $\{1\}.a = \{1, 2\}$ car

$$1 \xrightarrow{a} \mathbf{2} \text{ et } 1 \xrightarrow{a} 2 \xrightarrow{\varepsilon} \mathbf{1}.$$

On a aussi pour cet automate, $\{1\}.b = \emptyset$.

Dans le cas particulier de $w = \varepsilon$, on a toujours

$$R.\varepsilon \supseteq R.$$

En effet, on suppose implicitement que pour tout état q, $(q, \varepsilon, q) \in \Delta$ (cf. remarque II.2.5). Autrement dit, $R.\varepsilon$ est l'ensemble des états atteints depuis les états de R sans lire de lettres.

Si L est un langage sur Σ, on pose

$$R.L = \bigcup_{w \in L} R.w.$$

Le résultat suivant stipule que tout AFND est équivalent à un AFD. En d'autres termes, lorsqu'on s'intéresse à l'acceptation des mots d'un langage, un AFND n'est pas "*plus puissant*" qu'un AFD.

Proposition II.2.11 (Rabin et Scott[2]). *Tout langage accepté par un AFND est accepté par un AFD.*

II.4 LES NOTIONS DE LANGAGE FORMELLE ET DE LA COMBINATOIRE

Ce premier chapitre introduit quelques concepts fondamentaux de la théorie des langages formels et de la combinatoire sur les mots. La combinatoire des mots étudie les propriétés des suites de symboles. La théorie des langages formels englobe la théorie des automates et s'intéresse aux propriétés mathématiques des langages qui sont des ensembles de mots. Elle trouve notamment des applications en vérification et pour la compilation.

3.3 Langage régulier, non-déterminisme

Définition 11 *Langage régulier*
Un langage L est dit régulier s'il existe un automate fini A tel que $L = L(A)$.

Définition 12 *Automate fini non-déterministe*
Un automate fini $A = (\Sigma, Q, \delta, i, F)$ est dit non déterministe s'il existe dans δ deux transitions (q_1, x, q_2) et (q_1, x, q_3) telles que $q_2 \neq q_3$.

Un automate est non-déterministe si dans certains cas, il existe plusieurs chemins étiquetés par la même chaîne. Un automate fini est déterministe s'il n'est pas non-déterministe.

4 Propriétés de clôture

Nous allons nous intéresser à deux langages réguliers particuliers et à des opérations ensemblistes qui conservent la régularité.

Propriété 1 *Soit $A = (\Sigma, Q, \delta, i, F)$ un automate. Une chaîne w appartient à $L(a)$ si et seulement si il existe un calcul succès commençant du parcours partiel (i, w).*

Propriété 2 *Le langage vide est régulier.*

L'automate qui ne comprend que l'état initial qui n'est pas final ne possède aucun chemin succès. Il n'y a aucun état final, donc il n'y a pas de chemin succès. Donc le langage reconnu ne comporte aucune chaîne, c'est le langage vide (l'ensemble vide).

Propriété 3 *Le langage $\{\epsilon\}$ est régulier.*

L'automate qui ne comprend que l'état initial qui est également final reconnaît ce langage, s'il n'a aucune transition. Il convient de bien saisir la différence entre le langage vide qui ne contient aucune chaîne et ce langage qui contient une chaîne, la chaîne vide.

Propriété 4 *L'union de deux langages réguliers est un langage régulier.*

Définition 13 *Concaténation de langages*
Soient L_1 et L_2 deux langages. La concaténation de L_1 et L_2, notée $L_1.L_2$ est définie par : $L_1.L_2 = \{w_1 w_2 | w_1 \in L_1$ et $w_2 \in L_2\}$.

Propriété 5 *La concaténation de deux langages réguliers est un langage régulier.*

Définition 14 *Clôture sous concaténation*
Soit L un langage. On appelle clôture sous concaténation de L et on note L^* le langage définit par $L^* = \{w_1 \dots w_n | n \geq 0$ et $\forall i, 1 \leq i \leq n, w_i \in L\}$.

Propriété 6 *La clôture sous concaténation d'un langage régulier est un langage régulier.*

Propriété 7 *L'intersection de deux langages réguliers est un langage régulier.*

Soient $A_1 = (\Sigma, Q_1, \delta_1, i_1, F_1)$ et $A_2 = (\Sigma, Q_2, \delta_2, i_2, F_2)$. $A_1 \cap A_2 = (\Sigma, Q_1 \times Q_2, \delta, (i_1, i_2), F_1 \times F_2)$ avec $\delta = \{((q_1, q_2), x, (r_1, r_2)) | (q_1, x, r_1) \in \delta_1, (q_2, x, r_2) \in \delta_2\}$

Propriété 8 *Le complémentaire d'un langage régulier L, noté $\overline{L}$, est un langage régulier.*

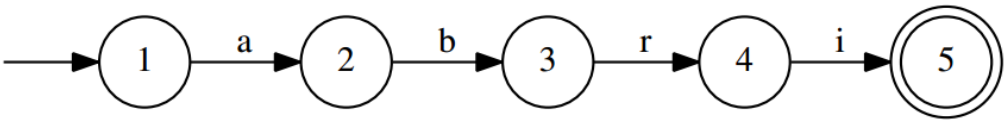
Propriété 9 *La différence ensembliste de deux langages réguliers est un langage régulier.*

Cette propriété est la conséquence des deux propriétés précédentes. En effet, $L_1 - L_2 = L_1 \cap \overline{L_2}$.

Les différentes propriétés étudiées dans cette section ont un intérêt direct pour la spécification. Par exemple, l'union est la base de la modularité. La différence ensembliste permet un traitement facile des exceptions. Illustrons cela sur un exemple.

Supposons que l'on veuille représenter avec un automate fini tous les mots du français pour une application de correction orthographique. On va faire une liste de tous les noms au singulier.

Chaque nom peut facilement être représenté avec un automate fini qui n'a qu'un chemin succès correspondant à ce mot. Par exemple `abri` est représenté par :



Le lexique de tous les noms au singulier peut être obtenu en faisant l'union de tous les automates ainsi construits. Appelons `noms_singulier` cet automate.

Pour avoir maintenant les noms au pluriel, il faut ajouter un s à la fin des mots. Cela peut être obtenu au moyen de l'opération de concaténation.

Mais il y a certaines exceptions qui ont un pluriel en `x`. On peut faire la liste de ces exceptions (pou, hibou, chou, etc). Appelons `exceptions` l'automate construit à partir de cette liste. Appelons `s_final` et `x_final` les deux automates qui contiennent respectivement les chaînes `s` et `x`.

La liste des mots au pluriel peut s'obtenir par :
`((noms_singulier-exceptions).s_final)U(exceptions.x_final)`

Dans une application réelle, il faut gérer aussi d'autres exceptions (pluriel des mots en *al*, en *ail*, en *eu* et pluriels irréguliers). Cela peut se faire avec le même genre d'utilisation des opérations ensemblistes.

5 Expression régulières

Les expressions régulières sont une façon de noter un langage régulier. Nous allons d'abord les définir, puis ensuite montrer qu'elles recouvrent exactement la notion de langage régulier.

La notion d'expression régulière est d'usage fréquent en informatique. En effet, on a souvent recours aux expressions régulières lorsqu'on désire rechercher certains motifs récurrents. Un exemple banal est celui d'un répertoire contenant divers fichiers :

```
> ls monrepertoire/
 memoire.aux    memoire.tex      picture001.jpg      rapsody.jpg
 memoire.dvi    picture001.jpg   presentation.exe    raw.jpg
 memoire.old    picture002.jpg   price-list.txt
 memoire.log    picture003.jpg   taches.txt
```

Si l'utilisateur désire afficher uniquement les images au format "JPEG" et comportant l'extension `.jpg`, il aura par exemple recours à une commande comme

```
ls *.jpg
```

De la même manière, s'il veut effacer tous les fichiers relatifs à `memoire`, il exécutera

```
rm m*
```

Les expressions régulières sont une importante notation pour spécifier formellement des modèles. Pour les définir correctement il nous faut faire l'effort d'apprendre un peu de vocabulaire nouveau :

Un langage sur un alphabet Σ est un ensemble de chaînes construites sur Σ. Exemples triviaux : ∅, le langage vide, {ε}, le langage réduit à l'unique chaîne vide. Des exemples plus intéressants (relatifs aux alphabets précédents) : l'ensemble des nombres en notation binaire, l'ensemble des chaînes ADN, l'ensemble des mots de la langue française, etc.

Si x et y sont deux chaînes, la concaténation de x et y, notée xy, est la chaîne obtenue en écrivant y immédiatement après x. Par exemple, la concaténation des chaînes anti et moine est la chaîne antimoine. Si x est une chaîne, on définit $x^0 = \varepsilon$ et, pour n > 0, $x^n = x^{n-1}x = xx^{n-1}$. On a donc $x^1 = x$, $x^2 = xx$, $x^3 = xxx$, etc.

Les opérations sur les langages suivantes nous serviront à définir les expressions régulières. Soient L et M deux langages, on définit :

Dénomination	Notation	Définition
l'union de L et M	L U M	{ x | x ϵ L ou x ϵ M}
la concaténation de L et M	LM	{xy | x ϵ L et y ϵ M}
la fermeture de Kleene de L	L*	{ $x_1x_2…x_n$ | xi ϵ L, n ϵ N et n ≥ 0}
la fermeture positive de L	L^+	{ $x_1x_2…x_n$ | xi ϵ L, n ϵ N et n > 0}

De la définition de LM on déduit celle de L^n = LL…L.

EXEMPLES. On se donne les alphabets L = {A,B…Z, a, b… z}, ensemble des lettres, et C = {0,1… 9}, ensemble des chiffres. En considérant qu'un caractère est la même chose qu'une chaîne de longueur un, on peut voir L et C comme des langages, formés de chaînes de longueur un. Dans ces conditions :

- L U C est l'ensemble des lettres et des chiffres,
- LC est l'ensemble des chaînes formées d'une lettre suivie d'un chiffre,
- L^4 est l'ensemble des chaînes de quatre lettres,
- L* est l'ensemble des chaînes faites d'un nombre quelconque de lettres ; ε en fait partie,
- C^+ est l'ensemble des chaînes de chiffres comportant au moins un chiffre,
- L(L U C)* est l'ensemble des chaînes de lettres et chiffres commençant par une lettre.

Définition d'une expression régulière.

Soit Σ un alphabet. Une expression régulière r sur Σ est une formule qui définit un langage L(r) sur Σ, de la manière suivante :

- ε est une expression régulière qui définit le langage {ε} ;
- Si a ϵ Σ, alors a est une expression régulière qui définit le langage(6) {a} ;
- Soient x et y deux expressions régulières, définissant les langages L(x) et L(y). Alors
    1. (x)|(y) est une expression régulière définissant le langage L(x) U L(y)
    2. (x)(y) est une expression régulière définissant le langage L(x)L(y)
    3. (x)* est une expression régulière définissant le langage (L(x))*
    4. (x) est une expression régulière définissant le langage L(x)

La dernière règle ci-dessus signifie qu'on peut encadrer une expression régulière par des parenthèses sans changer le langage défini. D'autre part, les parenthèses apparaissant dans les règles précédentes peuvent souvent être omises, en fonction des opérateurs en présence : il suffit de savoir que les opérateurs *, concaténation et |sont associatifs à gauche, et vérifient

priorité(*) > priorité(concaténation) > priorité(|)

Ainsi, on peut écrire l'expression régulière oui au lieu de (o)(u)(i) et oui|non au lieu de (oui)|(non), mais on ne doit pas écrire oui* au lieu de (oui)*.

DÉFINITIONS RÉGULIÈRES. Les expressions régulières se construisent à partir d'autres expressions régulières ; cela amène à des expressions passablement touffues. On les allège en introduisant des définitions régulières qui permettent de donner des noms à certaines expressions en vue de leur réutilisation. On écrit donc

- $d_1 \rightarrow r_1$
- $d_2 \rightarrow r_2$
- …
- $d_n \rightarrow r_n$

où chaque d_i est une chaîne sur un alphabet disjoint de Σ(7), distincte de d_1, d_2… d_{i-1}, et chaque r_i une expression régulière sur Σ U {d_1, d_2… d_{i-1}}.

EXEMPLE. Voici quelques définitions régulières, et notamment celles de identificateur et nombre, qui définissent les identificateurs et les nombres du langage Pascal :

Sélectionnez

```
lettre → A | B | ... | Z | a | b | ... | z
chiffre → 0 | 1 | ... | 9
identificateur → lettre ( lettre | chiffre )*

chiffres → chiffre chiffre*
fraction-opt → . chiffres | ε
```

```
exposant-opt → ( E (+ | - | ε) chiffres ) | ε
nombre → chiffres fraction-opt exposant-opt
```

NOTATIONS ABRÉGÉES. Pour alléger certaines écritures, on complète la définition des expressions régulières en ajoutant les notations suivantes :

- soit x une expression régulière, définissant le langage L(x) ; alors $(x)^+$ est une expression régulière, qui définit le langage $(L(x))^+$,
- soit x une expression régulière, définissant le langage L(x) ; alors (x)? est une expression régulière, qui définit le langage L(x) U { ε },
- si c_1, c_2… c_k sont des caractères, l'expression régulière $c_1|c_2|$ … $|c_k$ peut se noter $[c_1c_2 … c_k]$,
  - à l'intérieur d'une paire de crochets comme ci-dessus, l'expression c_1-c_2 désigne la séquence de tous les caractères c tels que $c_1 \leq c \leq c_2$.

Les définitions de lettre et chiffre données ci-dessus peuvent donc se réécrire :

Sélectionnez

```
lettre → [A-Za-z]
chiffre → [0-9]
```

Propriété 10 *Pour toute expression régulière p, il existe un automate fini qui reconnaît le langage défini par p.*

Cette propriété est une conséquence évidente des propriété de clôtures énoncées à la section précédente.

Propriété 11 *Pour tout automate fini A, il existe une expression régulière dénotant le langage $L(A)$.*

II.5 DETERMINATION MINIMISATION ET EPSILON TRANSITION

Propriété 12 *Pour tout langage régulier L, il existe un automate fini déterministe A tel que $L = L(A)$.*

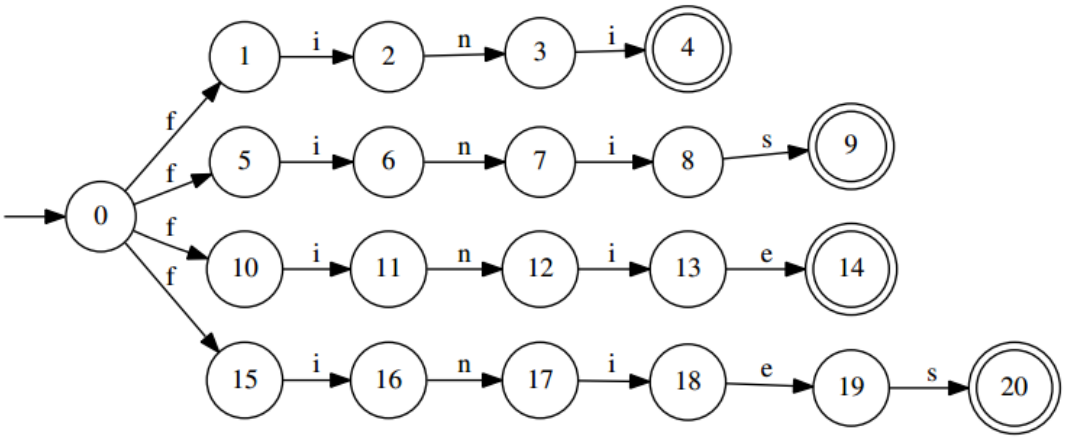
Par définition, un langage est régulier s'il existe un automate fini qui le reconnaît. La propriété énoncée spécifie de plus que s'il existe un automate qui reconnaît un langage, alors il existe nécessairement un automate fini déterministe qui le reconnaît. Cette propriété est intéressante opérationnellement, car, si l'on peut utiliser cet automate déterministe, on pourra savoir si une chaîne appartient ou non au langage en effectuant un seul calcul de l'automate.

Propriété 13 *Il existe un algorithme dit* algorithme de déterminisation *qui pour tout automate fini non déterministe A, calcule un automate fini déterministe A' tel que $L(A') = L(A)$.*

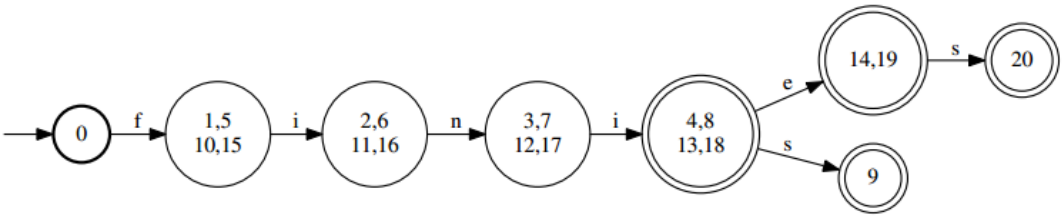
Cet algorithme utilise la notion d'ensemble de successeurs d'un état pour un symbole donné.

Définition 16 *Ensemble de successeurs*
Soit $A = (\Sigma, Q, \delta, i, F)$ un automate. On appelle ensemble des successeurs de l'état q pour un symbole x et on note $succ(q, x)$ l'ensemble des états r tels qu'il existe une transition (q, x, r) dans δ.

Prenons un exemple d'automate non déterministe qui décrit les différentes formes de l'adjectif *fini*, à savoir : *fini, finis, finie* et *finies*. Une construction naïve de l'automate consiste à écrire un chemin distinct pour chaque mot du langage.

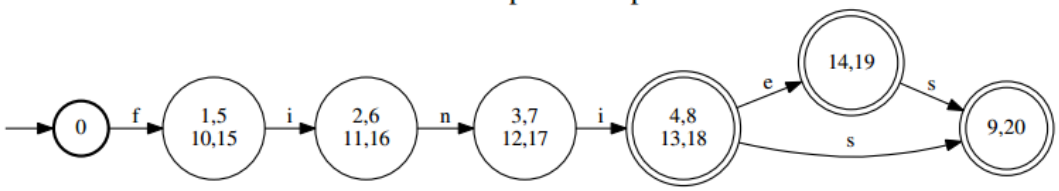


La construction de l'automate déterministe part de l'état initial, 0. Par `f`, on peut aller dans les états 1, 5, 10, et 15. On crée un nouvel état de nom `1,5,10,15`. De 1 par `i` on va en 2, de 5 en 6, de 10 en 11, de 15 en 16. On crée donc un nouvel état appelé `2,6,11,16`, avec une flèche de `1,5,10,15` vers `2,6,11,16`, étiquetée par `i`. Et ainsi de suite, ce qui donne le résultat suivant.



Propriété 14 *Il existe un algorithme appelé* algorithme de minimisation *qui pour tout automate fini* **déterministe** A *calcule un automate fini déterministe* A' *tel que* $L(A) = L(A')$ *et* A' *a un nombre d'état inférieur ou égal à tout autre automate définissant le langage* $L(A)$.

La minimisation de l'automate déterministe précédent permet de fusionner les états 9 et 20.



Nous allons maintenant présenter une variante des automates fini qui autorise l'utilisation de transitions sans étiquettes, qui ne lisent pas de symbole de la chaîne. On appelle ces transitions

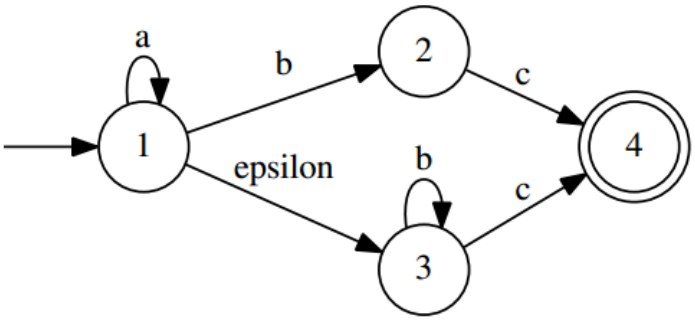
epsilon-transitions et on les note avec un epsilon en guise d'étiquette. Cette extension de la syntaxe des automates finis ne change rien à la puissance du formalisme. C'est à dire que les automates avec epsilon-transitions décrivent la même classe de langages que les automates sans epsilon-transition, c'est-à-dire les langages réguliers.

Définition 17 *Automate fini avec* ϵ*-transition*
Un automate fini est un quintuplet $A = (\Sigma, Q, \delta, i, F)$ *où :*

- Σ *est un ensemble fini de symboles appelé alphabet.*
- Q *est un ensemble fini dont les éléments sont appelés* états.
- δ *est une relation de* $Q \times (\Sigma \cup \{\epsilon\}) \times Q$ *appelée* transition *ou* ensemble des transitions *de* A.
- i *est un état de* Q *appelé* état initial.
- F *est un sous-ensemble de* Q *appelé* ensemble des états finals *de* A.

Propriété 15 *Il existe un algorithme dit* algorithme d'élimination des ϵ *qui pour tout automate* A *avec des* ϵ*-transitions calcule un automate* A' *sans* ϵ*-transition et tel que* $L(A) = L(A')$.

Le principe de l'algorithme consiste à remplacer chaque chemin de longueur 1 commençant par une epsilon-transition par une nouvelle transition qui décrit ce chemin. Prenons un exemple.

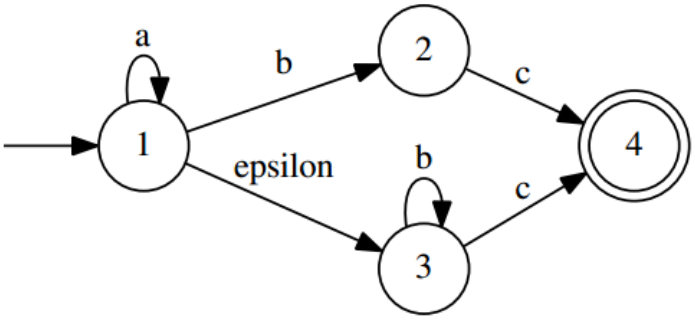


automate avec epsilon-transition

Il y a deux chemin de longueur 1 qui commencent par la transition sur epsilon :
- $(1, \epsilon, 3)(3, b, 3)$
- $(1, \epsilon, 3)(3, c, 4)$

On ajoute à l'automate deux nouvelles transitions qui résument ces deux chemins :
- $(1, b, 3)$
- $(1, c, 4)$

Et on supprime la transition $(1, \epsilon, 3)$. Il est facile de voir que pour tout succès dans l'ancien automate, il existe un succès dans le nouveau et réciproquement.

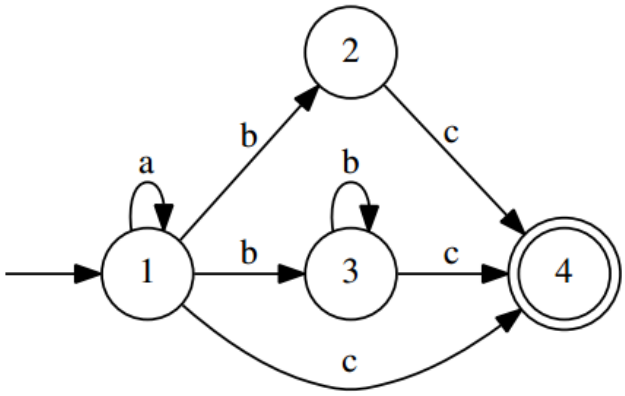


automate avec epsilon-transition

Il y a deux chemin de longueur 1 qui commencent par la transition sur epsilon :
- $(1, \epsilon, 3)(3, b, 3)$
- $(1, \epsilon, 3)(3, c, 4)$

On ajoute à l'automate deux nouvelles transitions qui résument ces deux chemins :
- $(1, b, 3)$
- $(1, c, 4)$

Et on supprime la transition $(1, \epsilon, 3)$. Il est facile de voir que pour tout succès dans l'ancien automate, il existe un succès dans le nouveau et réciproquement.



automate sans epsilon-transition

L'algorithme est un peu plus complexe dans les cas où plusieurs epsilon-transitions se suivent et si elles vont dans un état final, mais le principe général présenté ici reste valable.

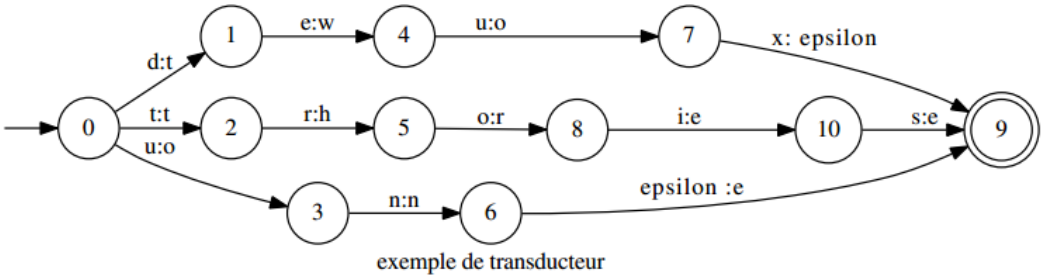
L'utilisation des epsilon-transitions permet parfois de décrire plus lisiblement un langage. Elle permet aussi de simplifier la description de certaines opérations (par exemple l'union ou la concaténation). D'un autre côté, l'existence d'epsilon-transitions rend généralement l'automate non-déterministe puisqu'on a toujours la possibilité d'emprunter une telle transition même quand il existe une transition ordinaire que l'on pourrait emprunter. Par exemple, sur notre exemple, depuis la configuration initiale $(1, abc)$, on peut par un pas de calcul aller aussi bien dans la configuration $(1, bc)$ en suivant la transition sur a que dans la configuration $(3, abc)$ en suivant la transition sur epsilon.

II.6 TRANSDUCTEURS FINIS A ETAT

Selon le dictionnaire Trésor de la Langue Française informatisé, un transducteur est un dispositif ou élément d'une chaîne de communication (mécanique, électrique, etc.) recevant un message sous une certaine forme et le transformant en une autre. Selon le Larousse, transducteur est un synonyme de traducteur.

Un transducteur fini à état est un automate fini qui non seulement reconnaît un ensemble régulier de chaînes, mais encore qui traduit chaque chaîne de cet ensemble dans une autre chaîne appartenant à un autre langage régulier. Concrètement, comme un automate fini, c'est un graphe orienté étiqueté, mais chaque transition est étiquetée par un (ou zéro) symbole à reconnaître et un (ou zéro) symbole utilisé pour construire la traduction, séparés par un point-virgule.

Voyons un exemple qui concerne la traduction de quelques chiffres du français à l'anglais (un→one, deux → two et trois→three).



exemple de transducteur

Définition 18 *Transducteur fini à état*
Un automate fini est un sextuplet $A = (\Sigma_i, \Sigma_o, Q, \delta, i, F)$ *où :*

- Σ_i *et* Σ_o *sont deux ensembles finis de symboles appelés respectivement alphabet d'entrée et alphabet de sortie.*
- Q *est un ensemble fini dont les éléments sont appelés* états.
- δ *est une relation de* $Q \times (\Sigma_i \cup \{\epsilon\}) \times (\Sigma_o \cup \{\epsilon\}) \times Q$ *appelée* transition *ou* ensemble des transitions *de* A.
- i *est un état de* Q *appelé* état initial.
- F *est un sous-ensemble de* Q *appelé* ensemble des états finals *de* A.

La notion de chemin succès est inchangée par rapport aux automates : c'est un chemin qui part de l'état initial et arrive dans un état final. Mais un tel chemin n'est plus associé à une chaîne mais à une paire de chaînes : une chaîne de symboles de Σ_i (ou entrée) et une de symboles de Σ_o.

Un transducteur ne défini pas un langage (ensemble de chaînes), mais une relation binaire (ensembles de paires de chaînes). Une relation définissable par un transducteur fini est appelée *relation régulière*.

Bien que la terminologie rappelle un peu celle des fonctions, il s'agit bien de relations, ce qui signifie qu'une chaîne peut avoir plusieurs traductions et deux chaînes différentes peuvent avoir la même traduction. Par exemple, dans une traduction français-anglais, on peut avoir la relation suivante : { `(un,a)`, `(un,one)`, `(gratuit,free)`, `(libre,free)` }.

Un transducteur peut donner lieu à différent types d'exécution :

- étant donné une chaîne entrée, calculer sa ou ses traductions en sortie (si cette chaîne appartient à une ou plusieurs paires de la relation).
- étant donné une chaîne en sortie, calculer la ou les entrées dont cette sortie est la traduction.
- étant donné une paire de chaînes, déterminer si cette paire appartient à la relation.
- exhiber une ou plusieurs paires de la relation.

Nous allons décrire le calcul pour le premier type d'exécution, que nous appellerons *exécution canonique* du transducteur.

Une configuration est un triplet avec un chemin du transducteur, une chaîne de Σ_i^* et une chaîne de Σ_o^*. Soit $T = (\Sigma_i, \Sigma_o, Q, \delta, i, F)$ un transducteur et w_i la chaîne de Σ_i^* à traduire. La configuration initiale est (i, w_i, ϵ). Un pas de calcul permet de passer d'une configuration à une autre en utilisant une transition $t = (q_i, x_i, x_o, q_f)$. $(q_1 \dots q_i, x_i v_i, v_o) \vdash^t (q_1 \dots q_i \rightarrow, v_i, v_o x_o)$
Une configuration $(q_0 \rightarrow \dots q_n, w_i, w_o)$ est finale si et seulement si q_n est final et $w_i = \epsilon$.

Les trasducteurs offrent un modèle plus puissant que les automates finis, mais ce surcroît de puissance se fait au prix de la perte de certaines propriétés des automates finis : la clôture sous intersection et différence, les propriétés d'optimisation automatique (déterminisation, minimisation).

II.7 LES COMPILATEURS

II.7.1 Introduction

Dans le sens le plus usuel du terme, la compilation est une transformation que l'on fait subir à un programme écrit dans un langage évolué pour le rendre exécutable. Fondamentalement, c'est une traduction : un texte écrit en Pascal, C, Java, etc., exprime un algorithme et il s'agit de produire un autre texte, spécifiant le même algorithme dans le langage d'une machine que nous cherchons à programmer.

En généralisant un peu, on peut dire que compiler c'est lire une suite de caractères obéissant à une certaine syntaxe, en construisant une (autre) représentation de l'information que ces caractères expriment. De ce point de vue, beaucoup d'opérations apparaissent comme étant de la compilation ; à la limite, la lecture d'un nombre, qu'on obtient en C par une instruction comme :

Sélectionnez
```
scanf("%f", &x);
```

est déjà de la compilation, puisqu'il s'agit de lire des caractères constituant l'écriture d'une valeur selon la syntaxe des nombres décimaux et de fabriquer une autre représentation de la même information, à savoir sa valeur numérique.

Bien sûr, les questions qui nous intéresseront ici seront plus complexes que la simple lecture d'un nombre.

Mais il faut comprendre que le domaine d'application des principes et méthodes de l'écriture de compilateurs contient bien d'autres choses que la seule production de programmes exécutables. Chaque fois que vous aurez à écrire un programme lisant des expressions plus compliquées que des nombres, vous pourrez tirer profit des concepts, techniques et outils expliqués dans ce cours.

II.7.2 Structure de principe d'un compilateur

La nature de ce qui sort d'un compilateur est très variable. Cela peut être un programme exécutable pour un processeur physique, comme un Pentium III ou un G4, ou un fichier de code pour une machine virtuelle, comme la machine Java, ou un code abstrait destiné à un outil qui en fera ultérieurement du code exécutable, ou encore le codage d'un arbre représentant la structure logique d'un programme, etc.

En entrée d'un compilateur on trouve toujours la même chose : une suite de caractères, appelée le texte source(1). Voici les phases en lesquelles se décompose le travail d'un compilateur, du moins d'un point de vue logique(2) (voyez la figure 1) :

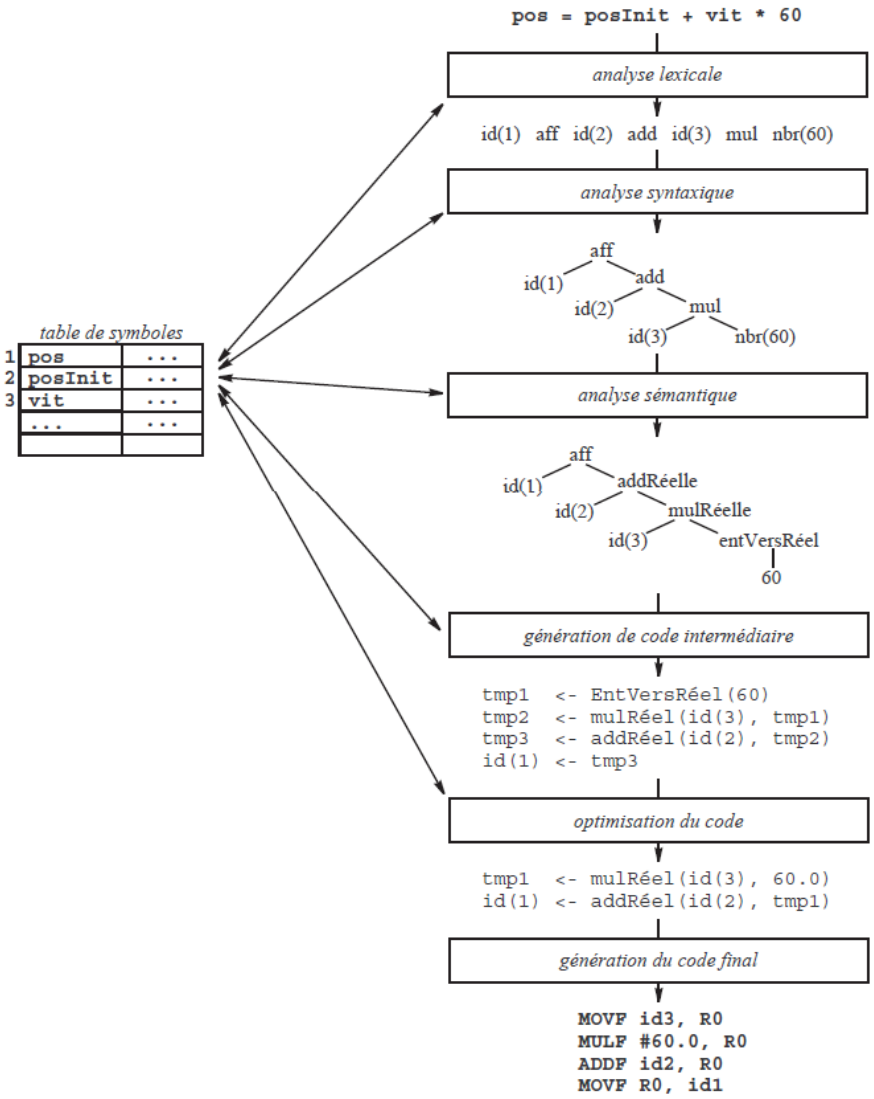


Figure 1 PHASE LOGIQUE DE LA COMPILATION D' UNE INSTRUCTION

Analyse lexicale Dans cette phase, les caractères isolés qui constituent le texte source sont regroupés pour former des unités lexicales, qui sont les mots du langage.

L'analyse lexicale opère sous le contrôle de l'analyse syntaxique ; elle apparaît comme une sorte de fonction de « lecture améliorée », qui fournit un mot lors de chaque appel.

Analyse syntaxique Alors que l'analyse lexicale reconnaît les mots du langage, l'analyse syntaxique en reconnaît les phrases. Le rôle principal de cette phase est de dire si le texte source appartient au langage considéré, c'est-à-dire s'il est correct relativement à la grammaire de ce dernier.

Analyse sémantique La structure du texte source étant correcte, il s'agit ici de vérifier certaines propriétés sémantiques, c'est-à-dire relatives à la signification de la phrase et de ses constituants :

- les identificateurs apparaissant dans les expressions ont-ils été déclarés ?
- les opérandes ont-ils les types requis par les opérateurs ?
- les opérandes sont-ils compatibles ? N'y a-t-il pas des conversions à insérer ?

- les arguments des appels de fonctions ont-ils le nombre et le type requis ?
- etc.

Génération de code intermédiaire Après les phases d'analyse, certains compilateurs ne produisent pas directement le code attendu en sortie, mais une représentation intermédiaire, une sorte de code pour une machine abstraite. Cela permet de concevoir indépendamment les premières phases du compilateur (constituant ce que l'on appelle sa face avant) qui ne dépendent que du langage source compilé et les dernières phases (formant sa face arrière) qui ne dépendent que du langage cible ; l'idéal serait d'avoir plusieurs faces avant et plusieurs faces arrière qu'on pourrait assembler librement(3).

Optimisation du code Il s'agit généralement ici de transformer le code afin que le programme résultant s'exécute plus rapidement. Par exemple :

- détecter l'inutilité de recalculer des expressions dont la valeur est déjà connue ;
- transporter à l'extérieur des boucles des expressions et sous-expressions dont les opérandes ont la même valeur à toutes les itérations ;
- détecter, et supprimer, les expressions inutiles ;
- etc.

Génération du code final Cette phase, la plus impressionnante pour le néophyte, n'est pas forcément la plus difficile à réaliser. Elle nécessite la connaissance de la machine cible (réelle, virtuelle ou abstraite), et notamment de ses possibilités en matière de registres, piles, etc.

II.7.3 Analyse lexicale

L'analyse lexicale est la première phase de la compilation. Dans le texte source, qui se présente comme un flot de caractères, l'analyse lexicale reconnaît des unités lexicales, qui sont les mots avec lesquels les phrases sont formées, et les présente à la phase suivante, l'analyse syntaxique.

Les principales sortes d'unités lexicales qu'on trouve dans les langages de programmation courants sont :

- les caractères spéciaux simples : +, =, etc. ;
- les caractères spéciaux doubles : <=, ++, etc. ;
- les mots-clés : if, while, etc. ;
- les constantes littérales : 123, -5, etc. ;
- et les identificateurs : i, vitesse_du_vent, etc.

À propos d'une unité lexicale reconnue dans le texte source on doit distinguer quatre notions importantes :

- l'unité lexicale, représentée généralement par un code conventionnel ; pour nos dix exemples +, =, <=, ++, if, while, 123, -5, i et vitesse_du_vent, ce pourrait être, respectivement(4) : **PLUS**, **EGAL**, **INFEGAL**, **PLUSPLUS**, **SI**, **TANTQUE**, **NOMBRE**, **NOMBRE**, **IDENTIF**, **IDENTIF** ;
- le lexème, qui est la chaîne de caractères correspondante. Pour les dix exemples précédents, les lexèmes correspondants sont : « + », « = », « <= », « ++ », « if », « while », « 123 », « -5 », « i » et « vitesse_du_vent » ;
- éventuellement, un attribut, qui dépend de l'unité lexicale en question, et qui la complète. Seules les dernières des dix unités précédentes ont un attribut ; pour un nombre, il s'agit de sa valeur (123, -5) ; pour un identificateur, il s'agit d'un

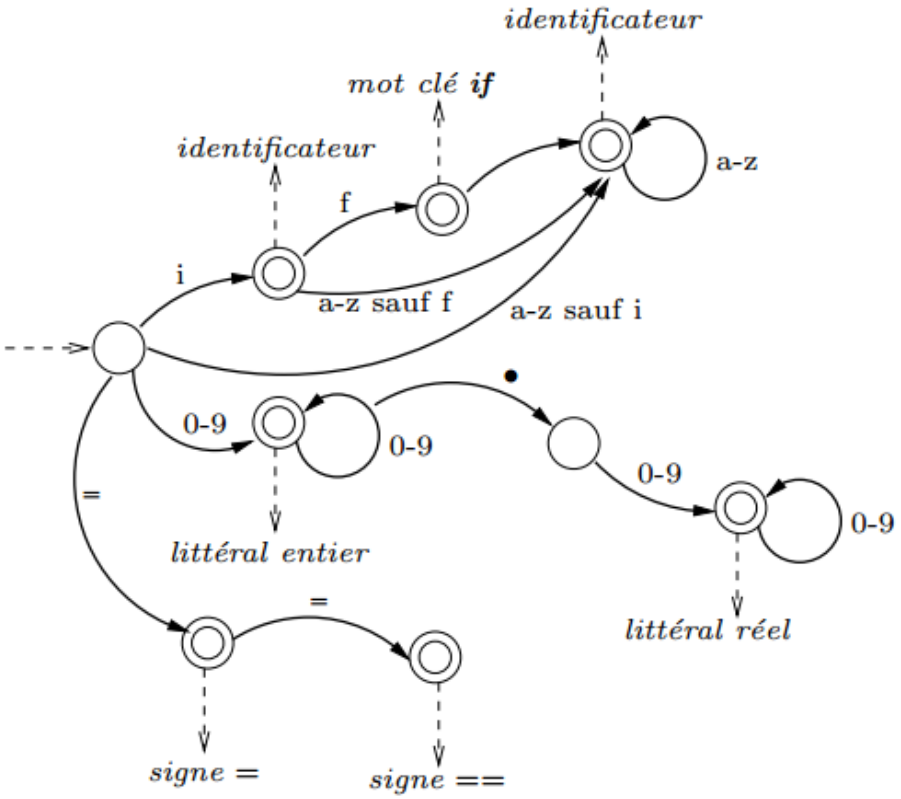
renvoi à une table dans laquelle sont placés tous les identificateurs rencontrés (on verra cela plus loin) ;
- le modèle qui sert à spécifier l'unité lexicale. Nous verrons ci-après des moyens formels pour définir rigoureusement les modèles ; pour le moment, nous nous contenterons de descriptions informelles comme :
    - pour les caractères spéciaux simples et doubles et les mots réservés, le lexème et le modèle coïncident,
    - le modèle d'un nombre est « une suite de chiffres, éventuellement précédée d'un signe »,
    - le modèle d'un identificateur est une suite de lettres, de chiffres et du caractère « _ », commençant par une lettre.

Outre la reconnaissance des unités lexicales, les analyseurs lexicaux assurent certaines tâches mineures comme la suppression des caractères de décoration (blancs, tabulations, fins de ligne, etc.) et celle des commentaires (généralement considérés comme ayant la même valeur qu'un blanc), l'interface avec les fonctions de lecture de caractères, à travers lesquelles le texte source est acquis, la gestion des fichiers et l'affichage des erreurs, etc.

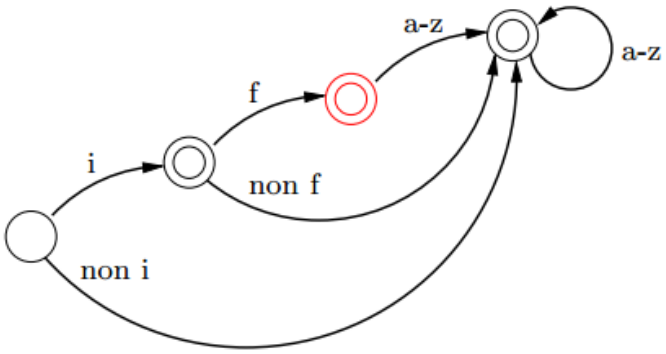
REMARQUE : la frontière entre l'analyse lexicale et l'analyse syntaxique n'est pas fixe. D'ailleurs, l'analyse lexicale n'est pas une obligation, on peut concevoir des compilateurs dans lesquels la syntaxe est définie à partir des caractères individuels. Mais les analyseurs syntaxiques qu'il faut alors écrire sont bien plus complexes que ceux qu'on obtient en utilisant des analyseurs lexicaux pour reconnaître les mots du langage.

Simplicité et efficacité sont les raisons d'être des analyseurs lexicaux. Comme nous allons le voir, les techniques pour reconnaître les unités lexicales sont bien plus simples et efficaces que les techniques pour vérifier la syntaxe.

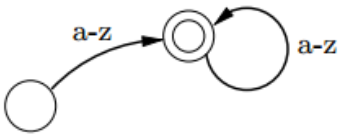
Fragment d'automate lexical



- la discrimination par l'automate entre les identificateurs et les mots-clés multiplie le nombre d'états



automate de discrimination entre `if` *et les identificateurs*



automate de reconnaissance des identificateurs

Reconnaissance des unités lexicales

Nous avons vu comment spécifier les unités lexicales ; notre problème maintenant est d'écrire un programme qui les reconnaît dans le texte source. Un tel programme s'appelle un analyseur lexical.

Dans un compilateur, le principal client de l'analyseur lexical est l'analyseur syntaxique. L'interface entre ces deux analyseurs est une fonction int unite Suivante (void)(8), qui renvoie à chaque appel l'unité lexicale suivante trouvée dans le texte source.

Cela suppose que l'analyseur lexical et l'analyseur syntaxique partagent les définitions des constantes conventionnelles définissant les unités lexicales. Si on programme en C, cela veut dire que dans les fichiers sources des deux analyseurs on a inclus un fichier d'en-tête (fichier « .h ») comportant une série de définitions comme(9) :

Sélectionnez
```
#define IDENTIF 1
#define NOMBRE 2
#define SI 3
#define ALORS 4
#define SINON 5
etc.
```

Cela suppose aussi que l'analyseur lexical et l'analyseur syntaxique partagent également une variable globale contenant le lexème correspondant à la dernière unité lexicale reconnue, ainsi qu'une variable globale contenant le (ou les) attribut(s) de l'unité lexicale courante, lorsque cela est pertinent, et notamment lorsque l'unité lexicale est **NOMBRE** ou **IDENTIF**.

On se donnera, du moins dans le cadre de ce cours, quelques « règles du jeu » supplémentaires :

- l'analyseur lexical est « glouton » : chaque lexème est le plus long possible(10) ;
- seul l'analyseur lexical accède au texte source. L'analyseur syntaxique n'acquiert ses données d'entrée autrement qu'à travers la fonction **uniteSuivante** ;
- l'analyseur lexical acquiert le texte source un caractère à la fois. Cela est un choix que nous faisons ici ; d'autres choix auraient été possibles, mais nous verrons que les langages qui nous intéressent permettent de travailler de cette manière.

Diagrammes de transition

Pour illustrer cette section nous allons nous donner comme exemple le problème de la reconnaissance des unités lexicales **INFEG**, **DIFF**, **INF**, **EGAL**, **SUPEG**, **SUP**, **IDENTIF**, respectivement définies par les expressions régulières **<=**, **<>**, **<**, **=**, **>=**, **>** et lettre(lettre|chiffre)*, lettre et chiffre ayant leurs définitions déjà vues.

Les diagrammes de transition sont une étape préparatoire pour la réalisation d'un analyseur lexical. Au fur et à mesure qu'il reconnaît une unité lexicale, l'analyseur lexical passe par divers états. Ces états sont numérotés et représentés dans le diagramme par des cercles.

De chaque état e sont issues une ou plusieurs flèches étiquetées par des caractères. Une flèche étiquetée par c relie e à l'état e_1 dans lequel l'analyseur passera si, alors qu'il se trouve dans l'état e, le caractère c est lu dans le texte source.

Un état particulier représente l'état initial de l'analyseur ; on le signale en en faisant l'extrémité d'une flèche étiquetée debut.

Des doubles cercles identifient les états finaux, correspondant à la reconnaissance complète d'une unité lexicale. Certains états finaux sont marqués d'une étoile : cela signifie que la reconnaissance s'est faite au prix de la lecture d'un caractère au-delà de la fin du lexème(11).

Par exemple, la figure 2 montre les diagrammes traduisant la reconnaissance des unités lexicales **INFEG**, **DIFF**, **INF**, **EGAL**, **SUPEG**, **SUP** et **IDENTIF**.

Un diagramme de transition est dit non déterministe lorsqu'il existe, issues d'un même état, plusieurs flèches étiquetées par le même caractère, ou bien lorsqu'il existe des flèches étiquetées par la chaîne vide ε. Dans le cas

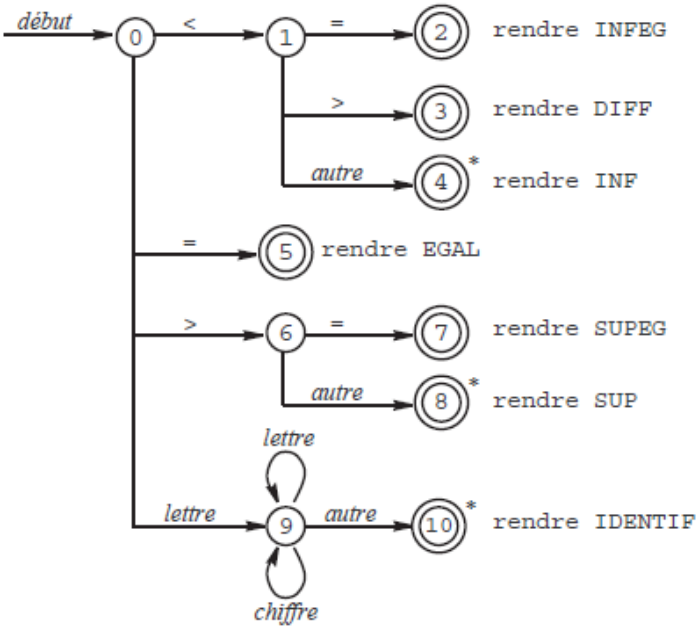


Fig. 2 - Diagramme de transition pour les opérateurs de comparaison et les identificateurs

contraire, le diagramme est dit déterministe. Il est clair que le diagramme de la figure 2 est déterministe. Seuls les diagrammes déterministes nous intéresseront dans le cadre de ce cours.

Analyseurs lexicaux programmés en dur

Les diagrammes de transition sont une aide importante pour l'écriture d'analyseurs lexicaux. Par exemple, à partir du diagramme de la figure 2 on peut obtenir rapidement un analyseur lexical reconnaissant les unités **INFEG**, **DIFF**, **INF**, **EGAL**, **SUPEG**, **SUP** et **IDENTIF**.

Auparavant, nous apportons une légère modification à nos diagrammes de transition, afin de permettre que les unités lexicales soient séparées par un ou plusieurs blancs(12). La figure 3 montre le (début du) diagramme modifié(13).

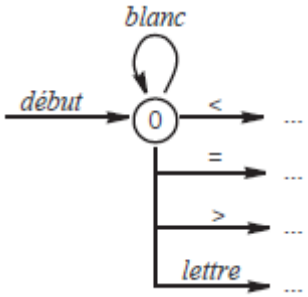


Fig. 3 - Ignorer les blancs devant une unité lexicale

Et voici le programme obtenu :

Sélectionnez

```
int uniteSuivante(void) {
    char c;
    c = lireCar(); /* etat = 0 */
    while (estBlanc(c))
        c = lireCar();
    if (c == '<') {
        c = lireCar(); /* etat = 1 */
        if (c == '=')
            return INFEG; /* etat = 2 */
        else if (c == '>')
            return DIFF; /* etat = 3 */
        else {
            delireCar(c); /* etat = 4 */
            return INF;
        }
    }
    else if (c == '=')
        return EGAL; /* etat = 5 */
    else if (c == '>') {
        c = lireCar(); /* etat = 6 */
        if (c == '=')
            return SUPEG; /* etat = 7 */
        else {
            delireCar(c); /* etat = 8 */
            return SUP;
        }
    }
    else if (estLettre(c)) {
        lonLex = 0; /* etat = 9 */
        lexeme[lonLex++] = c;
        c = lireCar();
        while (estLettre(c) || estChiffre(c)) {
            lexeme[lonLex++] = c;
            c = lireCar();
        }
        delireCar(c); /* etat = 10 */
        return IDENTIF;
    }
    else {
        delireCar(c);
        return NEANT; /* ou bien donner une erreur */
    }
}
```

Dans le programme précédent on utilise des fonctions auxiliaires, dont voici une version simple :

Sélectionnez
```
int estBlanc(char c) {
    return c == ' ' || c == '\t' || c == '\n';
}

int estLettre(char c) {
    return 'A' <= c && c <= 'Z' || 'a' <= c && c <= 'z';
}

int estChiffre(char c) {
    return '0' <= c && c <= '9';
}
```

NOTE. On peut augmenter l'efficacité de ces fonctions, au détriment de leur sécurité d'utilisation, en en faisant des macros :

Sélectionnez
```
#define estBlanc(c) ((c) == ' ' || (c) == '\t' || (c) == '\n')
#define estLettre(c) ('A' <= (c) && (c) <= 'Z' || 'a' <= (c) && (c) <= 'z')
#define estChiffre(c) ('0' <= (c) && (c) <= '9')
```

Il y plusieurs manières de prendre en charge la restitution d'un caractère lu en trop (notre fonction **delireCar**).

Si on dispose de la bibliothèque standard C on peut utiliser la fonction **ungetc** :

Sélectionnez
```
void delireCar(char c) {
    ungetc(c, stdin);
}

char lireCar(void) {
    return getc(stdin);
}
```

Une autre manière de faire permet de se passer de la fonction **ungetc**. Pour cela, on gère une variable globale contenant, quand il y a lieu, le caractère lu en trop (il n'y a jamais plus d'un caractère lu en trop). Déclaration :

Sélectionnez
```
int carEnAvance = -1;
```

avec cela nos deux fonctions deviennent

Sélectionnez
```
void delireCar(char c) {
    carEnAvance = c;
}
```

```
char lireCar(void) {
    char r;
    if (carEnAvance >= 0) {
        r = carEnAvance;
        carEnAvance = -1;
    }
    else
        r = getc(stdin);
    return r;
}
```

RECONNAISSANCE DES MOTS RÉSERVÉS. Les mots réservés appartiennent au langage défini par l'expression régulière lettre(lettre|chiffre)*, tout comme les identificateurs. Leur reconnaissance peut donc se traiter de deux manières :

- soit on incorpore les mots réservés aux diagrammes de transition, ce qui permet d'obtenir un analyseur très efficace, mais au prix d'un travail de programmation plus important, car les diagrammes de transition deviennent très volumineux(14) ;
- soit on laisse l'analyseur traiter de la même manière les mots réservés et les identificateurs puis, quand la reconnaissance d'un « identificateur-ou-mot-réservé » est terminée, on recherche le lexème dans une table pour déterminer s'il s'agit d'un identificateur ou d'un mot réservé.

Dans les analyseurs lexicaux « en dur » on utilise souvent la deuxième méthode, plus facile à programmer.

On se donne donc une table de mots réservés :

Sélectionnez

```
struct {
    char *lexeme;
    int uniteLexicale;
} motRes[] = {
    { "si", SI },
    { "alors", ALORS },
    { "sinon", SINON },
    ...
};

int nbMotRes = sizeof motRes / sizeof motRes[0];
```

puis on modifie de la manière suivante la partie concernant les identificateurs de la fonction **uniteSuivante** :

Sélectionnez

```
...
else if (estLettre(c)) {
    lonLex = 0; /* etat = 9 */
    lexeme[lonLex++] = c;
    c = lireCar();
    while (estLettre(c) || estChiffre(c)) {
        lexeme[lonLex++] = c;
        c = lireCar();
    }
    delireCar(c); /* etat = 10 */
```

```
    lexeme[lonLex] = '\0';
    for (i = 0; i < nbMotRes; i++)
        if (strcmp(lexeme, motRes[i].lexeme) == 0)
            return motRes[i].uniteLexicale;
    return IDENTIF;
}
```

II.7.4 Analyse syntaxique

Grammaires non contextuelles

Les langages de programmation sont souvent définis par des règles récursives, comme : « on a une expression en écrivant successivement un terme, '+' et une expression » ou « on obtient une instruction en écrivant à la suite **si**, une expression, **alors**, une instruction et, éventuellement, **sinon** et une instruction ». Les grammaires non contextuelles sont un formalisme particulièrement bien adapté à la description de telles règles.

Définitions

Une grammaire non contextuelle, on dit parfois grammaire BNF (pour Backus-Naur form(23)), est un quadruplet $G = (V_T , V_N, S_0, P)$ formé de

- un ensemble V_T de symboles terminaux ;
- un ensemble V_N de symboles non terminaux ;
- un symbole $S_0 \in V_N$ particulier, appelé symbole de départ ou axiome ;
- un ensemble P de productions, qui sont des règles de la forme $S \rightarrow S_1S_2 \ldots S_k$ avec $S \in V_N$ et $S_i \in V_N \cup V_T$.

Compte tenu de l'usage que nous en faisons dans le cadre de l'écriture de compilateurs, nous pouvons expliquer ces éléments de la manière suivante :

- Les symboles terminaux sont les symboles élémentaires qui constituent les chaînes du langage, les phrases. Ce sont donc les unités lexicales, extraites du texte source par l'analyseur lexical (il faut se rappeler que l'analyseur syntaxique ne connaît pas les caractères dont le texte source est fait, il ne voit ce dernier que comme une suite d'unités lexicales).
- Les symboles non terminaux sont des variables syntaxiques désignant des ensembles de chaînes de symboles terminaux.
- Le symbole de départ est un symbole non terminal particulier qui désigne le langage en son entier.
- Les productions peuvent être interprétées de deux manières :
    1. comme des règles d'écriture (on dit plutôt de réécriture), permettant d'engendrer toutes les chaînes correctes. De ce point de vue, la production $S \rightarrow S_1S_2 \ldots S_k$ se lit « pour produire un S correct [sous-entendu : de toutes les manières possibles] il fait produire un S_1 [de toutes les manières possibles] » suivi d'un S_2 [de toutes les manières possibles] suivi d'un… suivi d'un S_k [de toutes les manières possibles],
    2. comme des règles d'analyse, on dit aussi reconnaissance. La production $S \rightarrow S_1S_2 \ldots S_k$ se lit alors « pour reconnaître un S, dans une suite de terminaux donnée, il faut reconnaître un S_1 suivi d'un S_2 suivi d'un… suivi d'un S_k ».

La définition d'une grammaire devrait donc commencer par l'énumération des ensembles V_T et V_N. En pratique on se limite à donner la liste des productions, avec une convention typographique pour distinguer les symboles terminaux des symboles non terminaux, et on convient que :

- V_T est l'ensemble de tous les symboles terminaux apparaissant dans les productions ;
- V_N est l'ensemble de tous les symboles non terminaux apparaissant dans les productions ;
- le symbole de départ est le membre gauche de la première production.

En outre, on allège les notations en décidant que si plusieurs productions ont le même membre gauche

- $S \rightarrow S_{1,1}\ S_{1,2}\ \ldots\ S_{1,k1}$
- $S \rightarrow S_{2,1}\ S_{2,2}\ \ldots\ S_{2,k2}$
- . . .
- $S \rightarrow S_{n,1}\ S_{n,2}\ \ldots\ S_{n,kn}$

alors on peut les noter simplement

- $S \rightarrow S_{1,1}\ S_{1,2}\ \ldots\ S1_{,k1} \mid S_{2,1}\ S_{2,2}\ \ldots\ S_{2,k2} \mid \ldots \mid S_{n,1}\ S_{n,2}\ \ldots\ S_{n,kn}$

Dans les exemples de ce document, la convention typographique sera la suivante :

- une chaîne en italique représente un symbole non terminal ;
- une chaîne en **caractères_télétype** ou « entre guillemets », représente un symbole terminal.

À titre d'exemple, voici la grammaire G_1 définissant le langage dont les chaînes sont les expressions arithmétiques formées avec des nombres, des identificateurs et les deux opérateurs + et *, comme 60 * vitesse + 200 . Suivant notre convention, les symboles non terminaux sont expression, terme et facteur ; le symbole de départ est expression :

Sélectionnez

```
expression → expression "+" terme | terme
terme → terme "*" facteur | facteur (G1)
facteur → nombre | identificateur | "(" expression ")"
```

II.8 PROGRAMMATION D'UN AUTOMATE FINI

Un automate fini est défini par la donnée de

- un ensemble fini d'états E ;
- un ensemble fini de symboles (ou alphabet) d'entrée Σ ;
- une fonction de transition, transit : E x Σ → E ;
- un état ε_0 distingué, appelé état initial ;
- un ensemble d'états F, appelés états d'acceptation ou états finaux.

Un automate peut être représenté graphiquement par un graphe où les états sont figurés par des cercles (les états finaux par des cercles doubles) et la fonction de transition par des flèches étiquetées par des caractères : si transit(e_1, c) = e_2 alors le graphe a une flèche étiquetée par le caractère c, issue de e_1 et aboutissant à e_2.

Un tel graphe est exactement ce que nous avons appelé diagramme de transition à la section 2.2.1Diagrammes de transition (voir la figure 2). Si on en reparle ici c'est qu'on peut en déduire un autre style d'analyseur lexical, assez différent de ce que nous avons appelé analyseur programmé en dur.

On dit qu'un automate fini accepte une chaîne d'entrée s = $c_1c_2 \ldots c_k$ si, et seulement si, il existe dans le graphe de transition un chemin joignant l'état initial e0 à un certain état final ek, composé de k flèches étiquetées par les caractères c_1, $c_2 \ldots c_k$.

Pour transformer un automate fini en un analyseur lexical il suffira donc d'associer une unité lexicale à chaque état final et de faire en sorte que l'acceptation d'une chaîne produise comme résultat l'unité lexicale associée à l'état final en question.

Autrement dit, pour programmer un analyseur il suffira maintenant d'implémenter la fonction **transit** ce qui, puisqu'elle est définie sur des ensembles finis, pourra se faire par une table à double entrée. Pour les diagrammes des figures 2 et 3 cela donne la table suivante (les états finaux sont indiqués en gras, leurs lignes ont été supprimées) :

	' '	'\t'	'\n'	'<'	'='	'>'	lettre	chiffre	autre
0	0	0	0	1	5	6	9	erreur	erreur
1	4*	4*	4*	4*	2	3	4*	4*	4*
6	8*	8*	8*	8*	7	8*	8*	8*	8*
9	10*	10*	10*	10*	10*	10*	9	9	10*

On obtiendra un analyseur peut-être plus encombrant que dans la première manière, mais certainement plus rapide puisque l'essentiel du travail de l'analyseur se réduira à répéter « bêtement » l'action etat = transit[etat][lireCar()] jusqu'à tomber sur un état final. Voici ce programme :

Sélectionnez

```
#define NBR_ETATS ...
#define NBR_CARS 256

int transit[NBR_ETATS][NBR_CARS];
int final[NBR_ETATS + 1];

int uniteSuivante(void) {
    char caractere;
    int etat = etatInitial;
    while ( ! final[etat]) {
        caractere = lireCar();
        etat = transit[etat][caractere];
    }
    if (final[etat] < 0)
        delireCar(caractere);
    return abs(final[etat]) - 1;
}
```

Notre tablefinalement, indexé par les états, est défini par

- final[e] = 0 si e n'est pas un état final (vu comme un booléen, final[e] est faux),

- final[e] = U + 1 si e est final, sans étoile et associé à l'unité lexicale U (en tant que booléen, final[e] est vrai, car les unités lexicales sont numérotées au moins à partir de zéro),
- final[e] = i(U + 1) si e est final, étoilé et associé à l'unité lexicale U (en tant que booléen, final[e] est encore vrai).

Enfin, voici comment les tableaux transit et final devraient être initialisés pour correspondre aux diagrammes des figures 2 et 3(15) :

Sélectionnez

```
void initialiser(void) {
    int i, j;

    for (i = 0; i < NBR_ETATS; i++) final[i] = 0;
    final[ 2] = INFEG + 1;
    final[ 3] = DIFF + 1;
    final[ 4] = - (INF + 1);
    final[ 5] = EGAL + 1;
    final[ 7] = SUPEG + 1;
    final[ 8] = - (SUP + 1);
    final[10] = - (IDENTIF + 1);
    final[NBR_ETATS] = ERREUR + 1;

    for (i = 0; i < NBR_ETATS; i++)
        for (j = 0; j < NBR_CARS; j++)
            transit[i][j] = NBR_ETATS;

    transit[0][' '] = 0;
    transit[0]['\t'] = 0;
    transit[0]['\n'] = 0;

    transit[0]['<'] = 1;
    transit[0]['='] = 5;
    transit[0]['>'] = 6;

    for (j = 'A'; j <= 'Z'; j++)   transit[0][j] = 9;
    for (j = 'a'; j <= 'z'; j++)   transit[0][j] = 9;

    for (j = 0; j < NBR_CARS; j++) transit[1][j] = 4;
    transit[1]['='] = 2;
    transit[1]['>'] = 3;
    for (j = 0; j < NBR_CARS; j++) transit[6][j] = 8;
    transit[6]['='] = 7;

    for (j = 0; j < NBR_CARS; j++) transit[9][j] = 10;
    for (j = 'A'; j <= 'Z'; j++) transit[9][j] = 9;
    for (j = 'a'; j <= 'z'; j++) transit[9][j] = 9;
    for (j = '0'; j <= '9'; j++) transit[9][j] = 9;
}
```

CHAP.III LES SYSTEMES LINAIERES ET LOGIQUES

CHAP.II LES SYSTEMES AUTOMATISES LINEAIRES ET LOGIQUES

II.1. INTRODUCTION AUX SYSTEMES AUTOMATISES LINEAIRES ANALOGIQUE

II.1.1 INTRODUCTION

Partout, l'homme rencontre dans sa vie plusieurs opérations qui sont trop répétitives et dénuées d'intérêt (exemple tri de courrier) ou trop complexes, pénibles ou délicates, qui ne peuvent pas être confiées à lui (exemple régulation de la fréquence du réseau électrique, radar et poursuite automatique). Pour ces raisons, il fallait substituer la machine à l'homme dans de telles opérations. Cette machine doit être conçue pour assurer des tâches avec une intervention minimale de l'homme ou même sans aucune intervention. On dit qu'on est devant un problème *d'automatisation des systèmes*. Automatiser un système, c'est le transformer en vu d'y réduire ou d'y supprimer toute intervention de l'être humain et toute influence indésirable de tout autre élément externe.

L'automatique est une science qui fournit la théorie et les méthodes pour concevoir et réaliser les commandes automatiques de systèmes ou procédés. Ainsi, un système automatique est capable de fonctionner d'une manière autonome.

II.1.2 NOTIONS DE SYSTEME

1. CARACTERISTIQUES DES SYSTEMES

Les systèmes automatiques sont caractérisés par leurs modes de fonctionnement (en boucle ouverte ou fermée) ainsi que la nature du commande (commande continue ou discrète).

- Les systèmes automatiques peuvent fonctionner selon l'une des deux configurations suivantes : en boucle ouverte ou en boucle fermée.
  - ***Les systèmes en boucle ouverte*** : Ce sont des systèmes dont l'état de la sortie à un instant donnée t ne dépend que de la nature du système et de l'état de l'entrée. Le positionnement manuel d'une antenne de réception représente un exemple typique de système en boucle ouverte. Cependant, la position de l'antenne peut être modifiée à cause du vent. Dans ce cas, il faut la rétablir par l'intervention de l'opérateur.
  - ***Les systèmes en boucle fermée*** : Ce sont des systèmes dont l'état de sortie à un instant donnée t dépend de la nature du système, de l'état d'entrée et des états antérieurs de la sortie. La poursuite automatique du radar représente un exemple qui fonctionne en boucle fermée. En effet, le radar poursuit la cible sans aucune intervention de l'opérateur. On parle dans ce cas d'un système régulé ou asservi.

Le comportement d'un système asservi est comparable à celui de l'être humain dans son travail.

- Un système commandé est un système qui possède une ou plusieurs variables de sortie commandées à partir d'un ensemble de variables *d'action* ou *d'entrée*. Selon le type de variables, on peut distinguer deux types de commande :
  - ***La commande continue*** : si les variables sont continues dans le temps.
  - ***La commande discrète*** : si les variables sont de type binaire, il s'agit alors d'automatismes logiques ou séquentiels.

2. DEMARCHE D'ANALYSE DES SYSTEMES LINEAIRES

On considère un système qui représente une installation de chauffage central d'une maison, chaque radiateur possède un robinet qui permet de modifier le débit d'eau chaude et par conséquent la température dans la pièce.

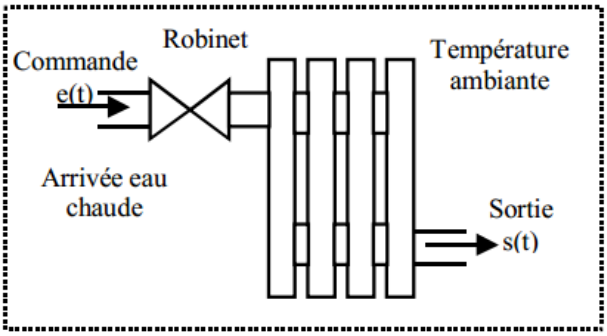


Figure 1.1 : Radiateur de chauffage central

Ce système physique est un système commandé qui possède une entrée, la commande *e(t)* d'ouverture du robinet, une sortie qui représente la température $\theta(t)$ de la pièce. Notons F la fonction qui relie la sortie *s(t)* à l'entrée *e(t)* à chaque instant. Le système physique de la figure 1.1 se représente sous la forme du diagramme fonctionnel de la figure 1.2. Un tel système possède plusieurs propriétés :

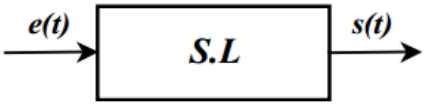


Figure 1.2 : Schéma fonctionnel d'un système linéaire

- C'est un système ***monovariable*** : il ne possède qu'une seule entrée et une seule sortie.
- C'est un système ***continu*** : la fonction F est une fonction continue du temps.
- C'est un système ***linéaire*** : la fonction F est linéaire. Ainsi, pour toute entrée $e_1(t)$ et $e_2(t)$, $\lambda_1,\lambda_2 \in \Re$: $F(\lambda_1 e_1(t)+\lambda_2 e_2(t)) = \lambda_1 F(e_1(t))+\lambda_2 F(e_2(t))$ (1.1)
- C'est un système ***stationnaire*** ou ***temporellement invariant*** : à chaque fois qu'on applique un signal e(t) à l'entrée, on obtient la même sortie s(t) ; (F est une fonction indépendante du temps).

L'approche d'un système qui consiste à l'isoler et à en identifier les entrées et les sorties est typique de la démarche de l'automaticien. Son but est de déterminer le signal de commande optimal à appliquer au système qui permet d'atteindre des objectifs fixés pour la sortie. Pour cela, il est nécessaire de connaître le comportement du système. Plusieurs démarches sont possibles. L'automaticien peut détailler le comportement interne du système en traduisant sous forme d'équations les différents phénomènes physiques mis en jeu. Il peut aussi se contenter de soumettre le système à un signal d'entrée connu et analyser sa sortie. Quelle que soit la démarche utilisée, il obtiendra un modèle mathématique qui lui permettra de prévoir l'évolution du système soumis à une entrée quelconque.

L'exemple de la figure 1.1 montre un système non linéaire puisque l'entrée x ne peut pas prendre de valeurs négatives. Néanmoins, nous considérerons que ce système est linéaire dans son domaine de fonctionnement normal. De plus, ce système n'est pas stationnaire puisque la température dans la pièce est influencée par d'autres facteurs que l'ouverture du robinet, par exemple, la température de l'eau, le rayonnement du soleil, la température extérieur… Ces différents facteurs correspondent à des perturbations qui vont influencer notre système et entraîner une valeur de la sortie différente de celle prévue. C'est une caractéristique générale des systèmes commandés que d'être sensibles aux perturbations.

Remarque : on considère le système représenté par la figure 1.3.

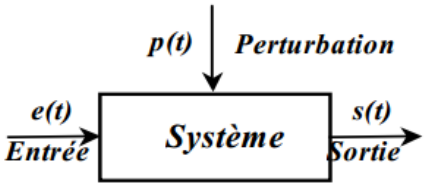


Figure 1.3 : Schéma fonctionnel d'un système linéaire soumis à une perturbation

Ce système est considéré comme monovariable, c'est-à-dire une seule variable d'entrée peut être manipulée (e(t) ou p(t)) et une seule variable est observée ou mesurée , c'est la variable de sortie s(t).

La modélisation mathématique s'exprime sous forme d'une relation entre l'évolution temporelle de la sortie s(t) en fonction de celle de l'entrée e(t) ou de celle d'une perturbation p(t). L'hypothèse de linéarité permet de lever toute difficulté éventuelle. En effet, la sortie s(t) est la somme des réponses à e(t) d'une part et p(t) d'autre part. On peut donc traiter séparément la réponse à chacune des entrées. Le système est alors monovariable, soit avec l'entrée e(t), soit avec l'entrée p(t). Si besoin, on fait la somme des réponses.

II.1.3 NOTIONS D' ASSERVISSEMENTS LINEAIRES ANALOGIQUES

Dans de nombreux processus industriels, il est indispensable de maîtriser un certain nombre de grandeurs physiques de type:

- Courant ou tension d'une source.
- Vitesse de rotation d'un moteur.
- Température d'un four.

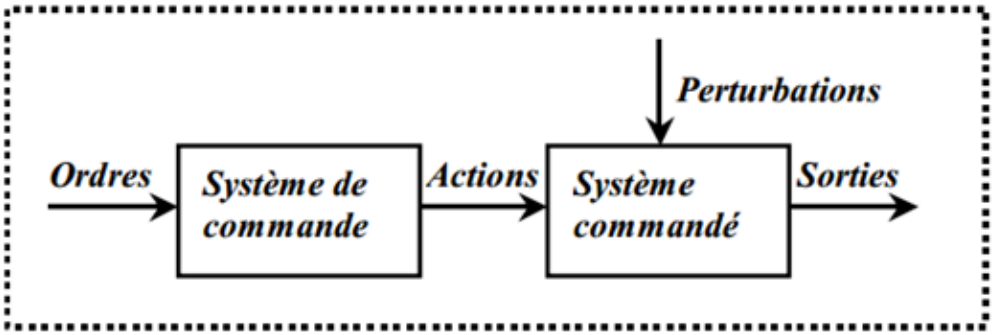


Figure 1.4 : Système commandé

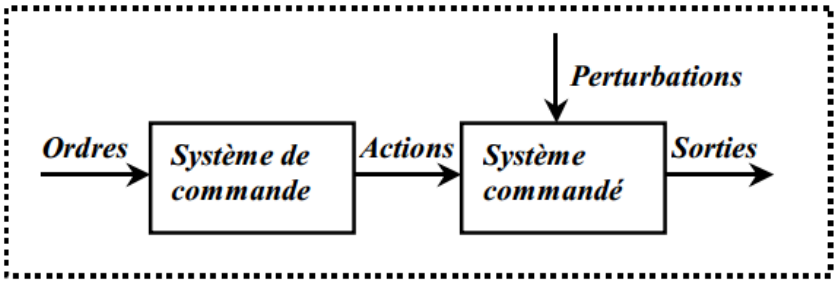


Figure 1.4 : Système commandé

Les paramètres d'un système de commande sont composés par :

- **Ordre** : C'est la consigne qui traduit un but fixé. Par exemple, on peut se décider de fixer la température d'une pièce à 37 °c ou fixer une trajectoire d'un avion.
- **Action de commande** : Action susceptible de changer l'état du système à commander. Elle est élaborée en fonction des ordres.
- **Perturbations** : Variable aléatoire dont on ne connaît pas l'origine.
- **Sortie** : Variable à contrôler.

II.1.4. Schéma fonctionnel général d'un système d'asservissement

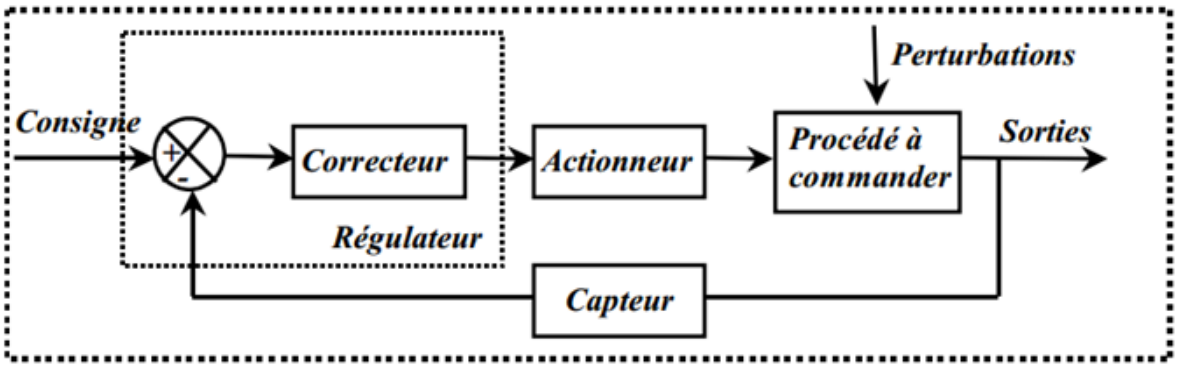


Figure 1.9 : Schéma fonctionnel général d'un système asservi

Ce système est constitué par les éléments de base suivants :

- ***Régulateur*** : C'est l'organe intelligent qui élabore l'ordre de commande
- ***Actionneur*** : C'est l'élément qui apporte la puissance nécessaire pour la réalisation de la tâche.
- ***Procédé à commander*** : Appelé aussi le système dynamique, il représente la partie qui évolue selon l'actionneur suivant les lois physiques qui lui sont propres.
- ***Capteur*** : C'est l'élément qui délivre une grandeur physique caractérisant l'observation. On cite à titre d'exemple le capteur de température PT100.

Dans le cas où la consigne est constante, on parle de régulation (exemple : régulation de température).

II.1.5 Asservissement et régulation

Deux situations peuvent se présenter dans la pratique :

A/ Cas où la consigne est **constante** et le système doit compenser l'effet des perturbations : on parle alors de **régulation** exemples :

- Régulation de niveau d'eau dans un réservoir.
- Régulation de température d'un four.
- Régulation de vitesse d'un moteur.

B/ Cas où la consigne **évolue** avec le temps, on parle alors de **poursuite** ou d'**asservissement**. Le système est dit ***suiveur.***

Exemples : une machine outil qui doit usiner une pièce selon un profil donné, un missile qui poursuit une cible, Asservissement de position, asservissement de vitesse, Asservissement de débit, etc.

Pour évaluer les objectifs cités, on procède en pratique à la mesure d'indices dits de performances : il s'agit de la stabilité, de la précision et de la rapidité. Nous découvrirons dans la suite le sens et l'importance de ces indices.

I.1.4.6 Propriétés et performance d'un système asservi

Stabilité

On dit qu'un système est stable su pour une entrée, la sortie reste constante quelle que soient les perturbations.

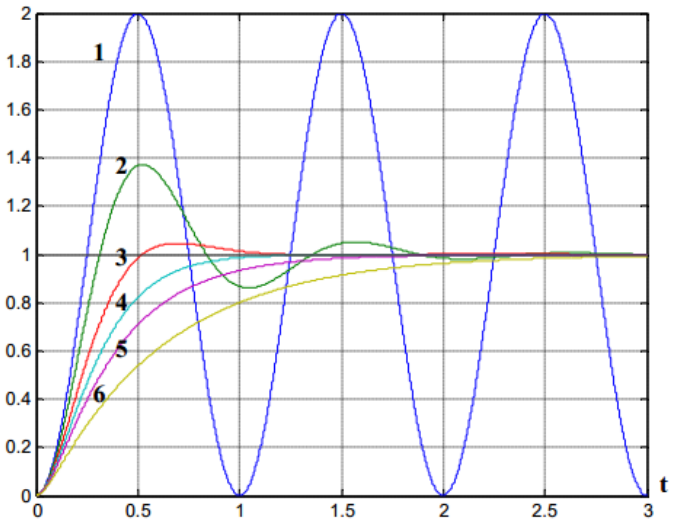


Figure 1.10 : Réponse d'un système asservi

- Les courbes 1 à 6 représentent la réponse d'un système.
- Les courbes de 2 à 6 sont caractéristiques de la réponse d'un système stable, pour une entrée constante, la sortie évolue vers une sortie constante.
- La courbe 1 est caractéristique d'un système instable, la sortie diverge.

- On s'aperçoit en comparant les réponses 2 à 6 que le critère de stabilité n'est pas un critère judicieux de réglage d'un système asservi. En effet, est-il envisageable qu'un système atteigne sa position définitive après un grand nombre d'oscillation ?

Dépassement

Un critère important de la stabilité est le dépassement. Ce critère permet de définir la notion de stabilité relative.
Le dépassement est mesuré par le taux de dépassement. On définit le premier dépassement par :

$$D_1\,\% = \frac{S(t_1) - S_\infty}{S_\infty} = \frac{\Delta_1}{S_\infty}$$

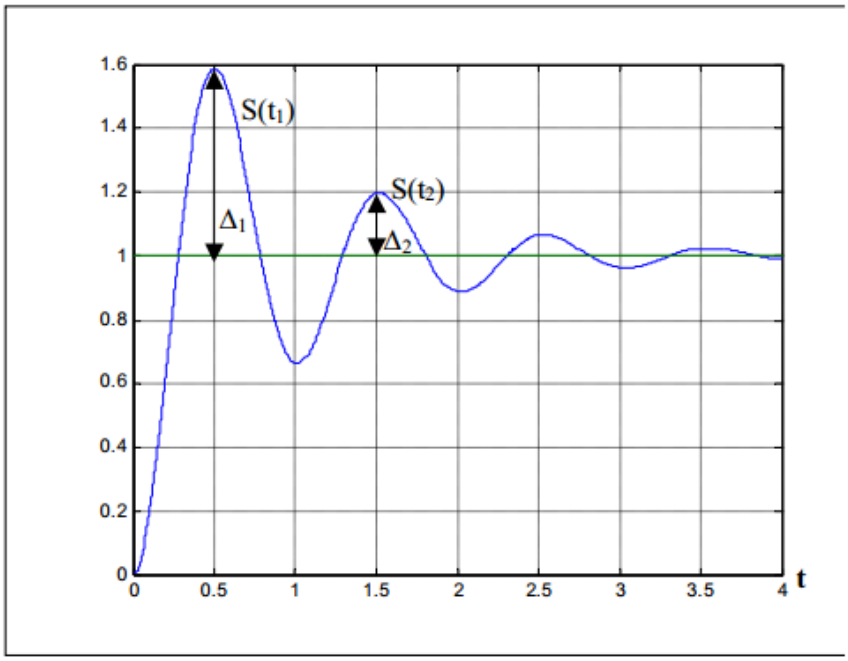


Figure 1.11 : Dépassement dans un système asservi

Avec :

- S_∞ : La valeur finale de la sortie.
- $S(t_1)$: La valeur de la sortie à l'instant du premier dépassement.
- On définit de même le deuxième dépassement.

Rapidité

Un Système a une rapidité satisfaisante s'il se stabilise à son niveau constant en temps jugé satisfaisant.

La rapidité caractérise le temps mis par le système pour que la sortie atteigne se nouvelle valeur. On définit pour caractériser la rapidité, le temps de réponse à 5% ($t_{r5\%}$), c'est le temps mis par le système pour atteindre sa valeur finale à 5% près.

Précision

La précision est caractérisée par l'écart entre la consigne et la sortie.

Précision statique

On appelle précision statique, l'écart entre la sortie et l'entrée lorsque le système est stabilisé.

II.1.2 INTRODUCTION AUX SYSTEMES AUTOMATISES LINEAIRES NUMERIQUES(OU ASSERVISSEMENTS NUMERIQUES)

II.1.2.1 GENERALITES

Depuis l'apparition et le développement des systèmes informatiques (micro-processeurs, micro-ordinateurs, micro-contrôleurs), leur utilisation en commande et régulation des systèmes industriels ne cesse de s'accroître.
Ce développement résulte essentiellement de la souplesse de réalisation des régulateurs numériques : la mise au point et le réglage consistent essentiellement à déterminer les coefficients d'une équation dite équation récurrente, qui constitue le cœur d'un programme de calcul exécuté en boucle par un processeur. Le coût de développement et de maintenance d'un tel régulateur est donc nettement plus avantageux que la réalisation de cartes analogiques spécifiques, nécessaires à la réalisation de régulateurs analogiques.
Cependant, le traitement numérique présente quelques différences importantes par rapport au traitement analogique:
• les valeurs des grandeurs physiques constituant les signaux analogiques doivent être représentées par des nombres;
• les opérations numériques réalisées par le processeur ne se font pas instantanément: il faut donc introduire la prise en compte de la durée du calcul.
En fait, vu d'un calculateur numérique, le temps ne peut pas s'écouler de façon continue tel qu'on le perçoit dans le monde physique. Le temps se définit sur un ensemble discret (ensemble dénombrable, isomorphe à l'ensemble des entiers) d'instants : les instants d'échantillonnage, séparés par un intervalle de temps régulier : la période d'échantillonnage.
Il est donc nécessaire de définir des outils mathématiques nouveaux, adaptés au temps discret, pour représenter ces signaux et systèmes échantillonnés, puis d'adapter les outils et méthodes de l'automatique analogique (à temps continu) à la conception de régulateurs numériques.

II.1.2.2 MISE EN ŒUVRE DES ASSERVISSEMENTS NUMERIQUES

Le schéma général d'un asservissement analogique est représenté en figure 1.11, sa transposition en commande numérique est représentée en figure 1.12.

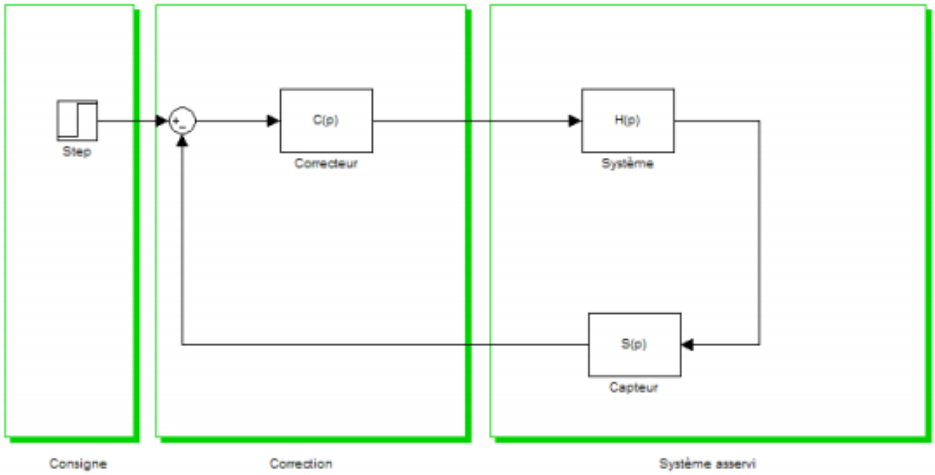


FIG. 1.11 – Système asservi linéaire continu.

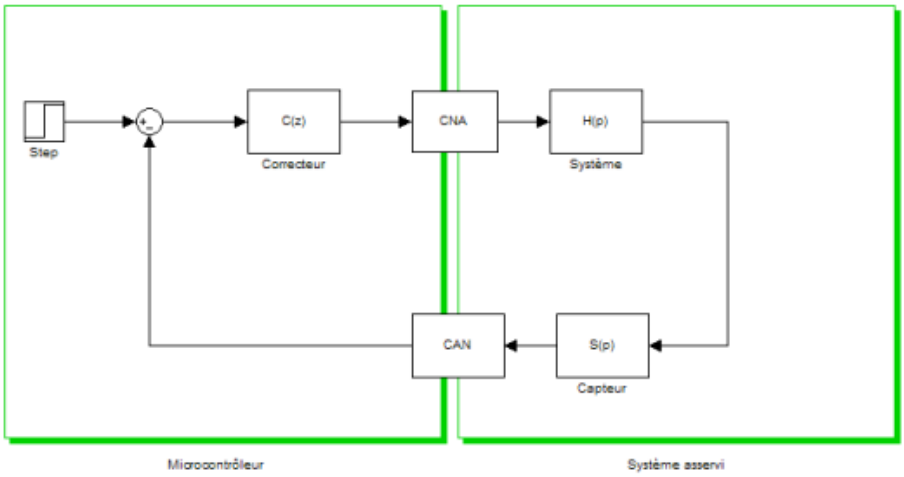


FIG. 1.12 – Système asservi linéaire échantillonné. (Ne cherchez pas les boites CAN et CNA sous simulink, elles n'existen pas ! CAN est intrinsèque en passant de continu à échantillonné, CNA s'appelle en fait ZOH pour "Zero Order Hold".)

Les avantages de l'asservissement numérique sont nombreux, en voici quelques uns.

- Réalisation aisée de régulateurs complexes, lois de commande raffinées.
- Facilité de mise en oeuvre de commandes anticipatrices (compensation par rapport à la consigne ou à certaines perturbations).
- Mise en oeuvre d'algorithmes de régulation sans équivalent analogique.
- Insensibilité de la caractéristique entrée-sortie du régulateur aux parasites, aux variations de température, au vieillissement, etc.
- Pas de dispersion des paramètres du régulateur en cas de fabrication en série.
- Prise en compte de défauts, des limites et comportements particuliers du système à régler (non linéarités, saturations) par simple programmation.
- Linéarisation autour d'un point de fonctionnement ajustable.
- Générateur de trajectoires intégré.
- Changement de correcteur souple et rapide.
- Interface utilisateur conviviale.
- Plusieurs systèmes corrigés par un seul microprocesseur.

Il y a aussi quelques inconvénients qu'il convient de connaître pour mieux les contourner. Notamment l'observation discontinue de la grandeur réglée, le système est en boucle ouverte entre deux instants d'échantillonnage. Sans précautions particulières, le bouclage numérique insère des non linéarités dans la boucle de régulation dues :

- à la quantification des convertisseurs,
- à la précision de calcul finie du microprocesseur,
- au procédé d'échantillonnage (recouvrement spectral).

Ces non linéarités peuvent avoir un effet déstabilisant (cycles limites) et introduisent des bruits supplémentaires, voire des battements. Un autre phénomène important est l'insertion de retards purs dans la boucle de régulation dus au temps de conversion analogique-numérique (A/N) et numérique-analogique (N/A) et au temps d'exécution de l'algorithme de régulation, or les retards purs sont très déstabilisants.

II.1.2.3 NUMERISATION DES SIGNAUX ANALOGIQUES

1. ECHANTILLONNAGE ET QUANTIFICATION

La figure 1.21 montre la différence fondamentale entre le signal analogique et le signal échantillonné et quantifié que le microcontrôleur "perçoit". Ce signal est en fait une suite de nombres codés sur n bits (typiquement 8 à 16 bits)

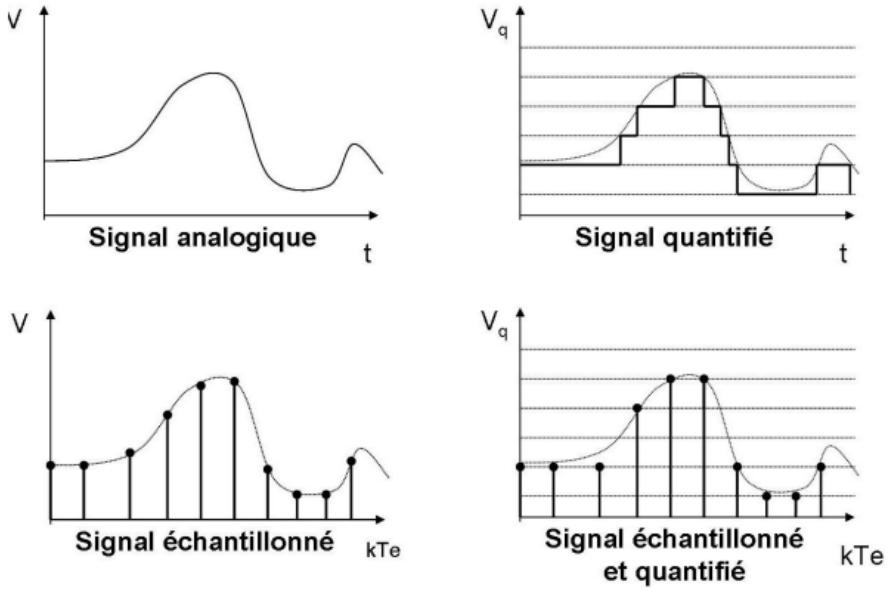


FIG. 1.21 – Echantillonnage et quantification d'un signal.

En observant le signal échantillonné et quantifié (en bas à droite de la figure 1.21), la tentation est grande de vouloir s'approcher du signal analogique en essayant de faire du "quasi-continu" par un surdimensionnement des deux paramètres principaux que sont la fréquence d'échantillonnage et le pas de quantification. Les deux parties suivantes, je l'espère, vous en dissuaderont.

2. BRUIT DE QUANTIFICATION

Pour un signal sinusoïdal d'amplitude égale à la dynamique du convertisseur, le rapport signal à bruit SNR[3] est de :

$$SNR = 6\ n - 4,76$$

où n est le nombre de bits du convertisseur
En pratique on choisit n tel que le rapport signal à bruit de l'ensemble ne soit pas trop diminué par la quantification. Il existe des convertisseurs de : 8, 10, 12, 16, 24, 40 bits. Il existe aussi des convertisseurs ayant une caractéristique logarithmique.

3. PROBLEME D'ECHANTILLONNAGE

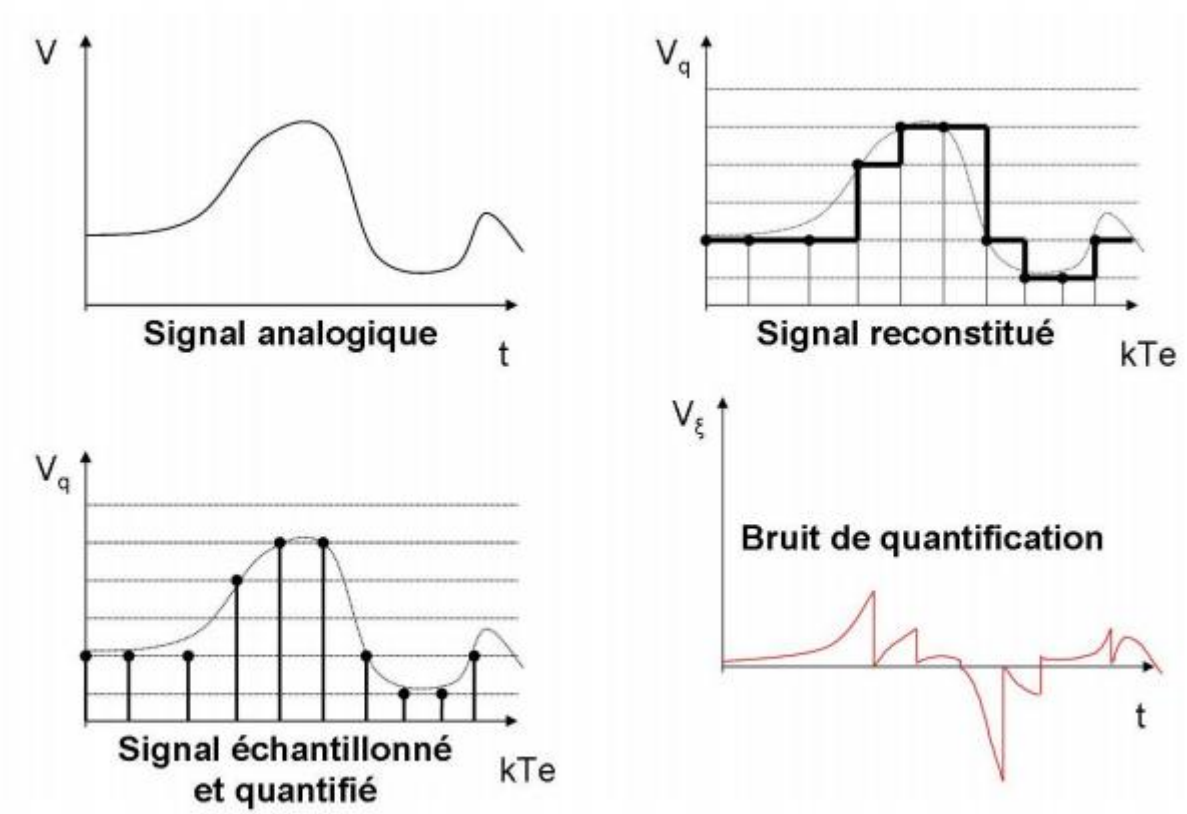


FIG. 1.22 – Bruit de quantification d'un signal.

La figure 1.23 montre un même signal échantillonné à plusieurs fréquences, lorsque la période d'échantillonnage T_e est égale à la moitié de la fréquence du signal, on voit qu'il devient impossible de reconstituer le signal original.
En augmentant encore la période d'échantillonnage on tombe sur des aberrations comme illustré en figure 1.24.

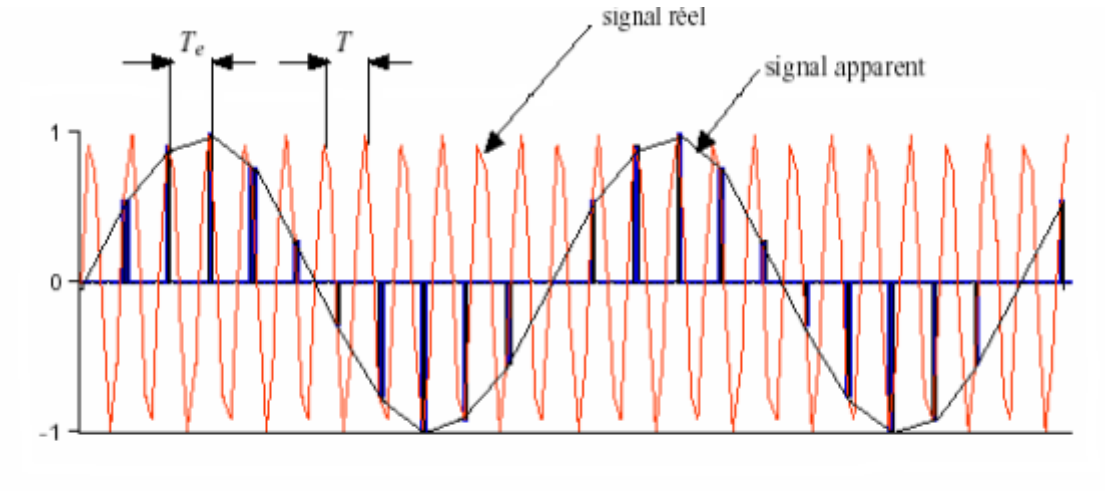


FIG. 1.24 – Signal ostensiblement sous-échantillonné.

II.1.2.4 TECHNOLOGIE DES ASSERSSIMENTS NUMERIQUES

I

Le choix du support matériel d'un asservissement numérique est évidemment fonction de l'objet contrôlé. La gamme de supports est vaste, pour fixer les idées voici trois systèmes complets.
Dans le cas de systèmes très complexes, nécessitant plusieurs asservissements simultanés, des interfaces utilisateurs conviviales, le choix se porte sur des calculateurs industriels qui se présentent sous la forme d'un rack dans lequel on insère des cartes à microprocesseur et des cartes d'entrées sorties. La puissance de calcul n'est pas limitée.

FIG. 1.13 – Kit de développement sur Bus VME.

FIG. 1.14 – Carte d' acquisition haute densité. 96 voies analogiques 12 bits 3 Méch./s.

La programmation de ces systèmes fait appel à un système d'exploitation multitâche temps réel.
A l'inverse, pour des systèmes simples, un microcontrôleur[1] suffit. La programmation se fait alors en C ou en Assembleur, les microcontrôleurs plus puissants acceptent sans problème un OS temps réel.

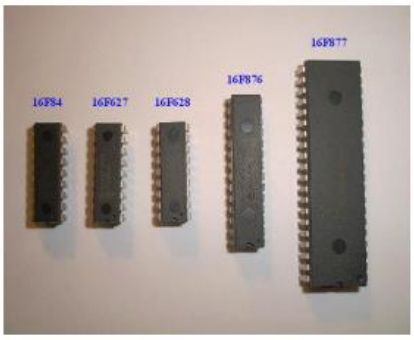


FIG. 1.15 – Microcontrôleur PIC.

Entre les deux, il existe des cartes de puissance intermédiaire telle que la SSV DIL/NetPC DNP/5280. Celle-ci possède 8Mo de ROM et 16 Mo de RAM (à comparer aux 8192 mots de ROM et 368 octets de RAM d'un PIC

[1]Microcontrôleur = sur la même puce microprocesseur+ périphériques : horloges, entrées sorties binaires et analogiques, liaisons numériques, ...

16F877) ainsi que pratiquement tous les bus classiques (I2C, SPI, CAN et une interface Ethernet)

FIG. 1.16 – Carte microcontrôleur SSV DIL/NetPC DNP/5280.

Enfin, il existe les DSP : Digital Signal Processing, dédiés au calcul rapide et flottant, plutôt utilisés dans des applications nécessitant du traitement du signal conséquent.
Et pour finir, sachez que les PID industriels soit disant analogiques cachent un microcontrôleur, cette technologie permettant de réaliser des constantes de temps impossibles à réaliser en analogique. Mais aussi, de gérer des alarmes, des saturations, des modes de défaut, etc.

FIG. 1.17 – Régulateur "analogique" de type PID.

CAN

Afin de faire traiter le signal par un microprocesseur, il faut le lui rendre accessible, c'est le rôle du convertisseur analogique numérique. Plusieurs méthodes de conversion existent[2] .

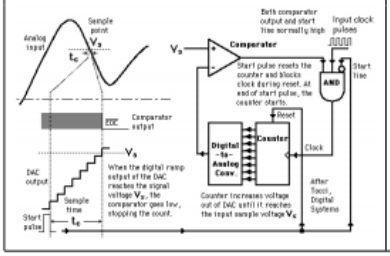
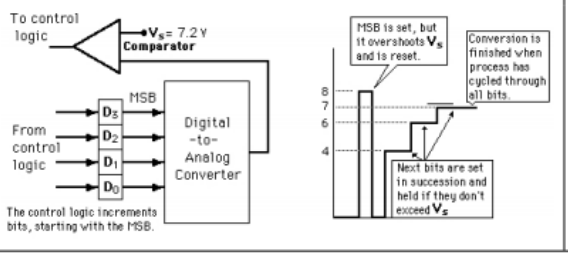
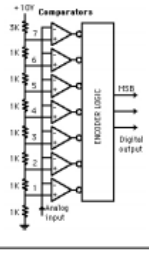
Convertisseur à rampe numérique	Convertisseur à pesées successives	Convertisseur flash
lent, précis	⟹	rapide, moins précis
		

FIG. 1.18 – Principes de conversion analogique numérique.

CNA

Le convertisseur numérique analogique le plus connu est fondé sur le principe du réseau R-2R comme illustré sur la figure 1.19. Ce principe est très consommateur de surface de silicium, aussi la grande majorité des microcontrôleurs n'en possèdent pas. Une autre méthode de conversion consiste à utiliser le principe de modulation en largeur d'impulsion (MLI, en anglais PWM). Le principe est de faire varier la valeur moyenne du signal de sortie binaire (0 - 5V) en faisant varier la largeur de l'impulsion. Un filtre RC filtrera les variations rapides, ne laissant passer que les variations lentes soit le signal désiré. Ce principe présente l'énorme avantage de ne demander qu'un bit de sortie du microcontrôleur par rapport à 16 bits pour un CNA classique !

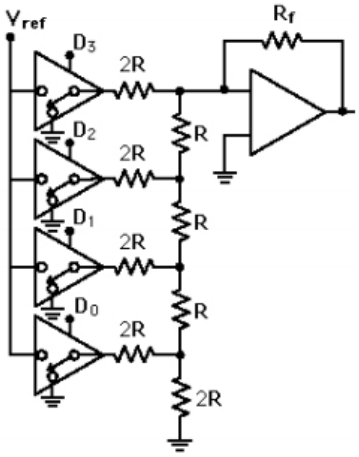


FIG. 1.19 – Convertisseur numérique analogique de type réseau R-2R.

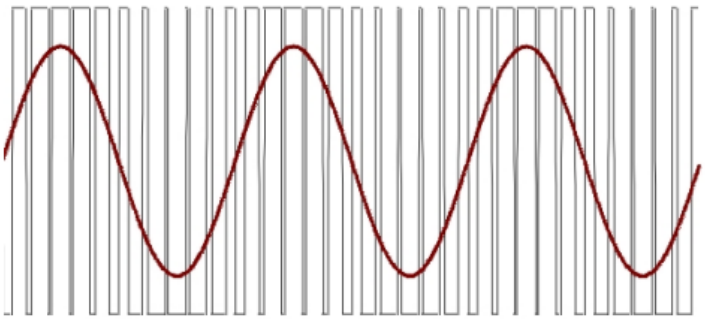


FIG. 1.20 – Signal modulé en modulation de largeur d'impulsion avant et après filtrage haute fréquence.

II.2 LES SYSTEMES AUTOMATISES LINEAIRES

II.2.1 MODELISATION DES SYSTEMES LINEAIRES

On distingue trois phases dans la modélisation :

1. Isoler le système étudié en positionnant la frontière et en recensant les entrées sorties.
2. Effectuer une décomposition en sous-systèmes plus facilement exploitable.
3. Établir un modèle de connaissance ou de comportement pour chaque sous-système.

Un **modèle de connaissance** est un **modèle obtenu à partir de lois physiques**. Cette modélisation est **analytique** et possède un **sens physique fort**.

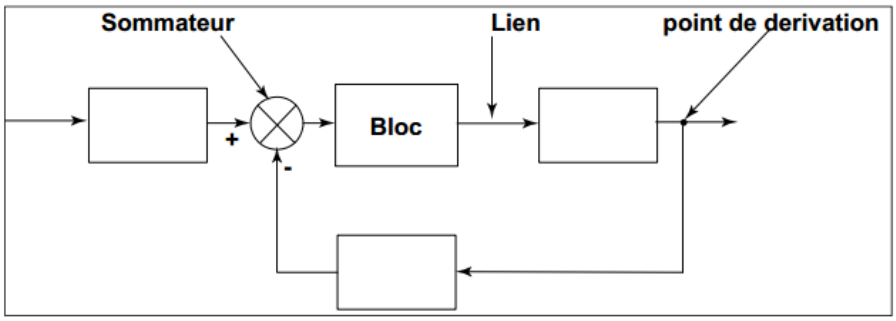
Un **modèle de comportement** est un modèle dans lequel le sous-système est remplacé par une boîte noire. Le **comportement réel est identifié au mieux à partir de résultats expérimentaux.**

La relation de comportement d'un système linéaire peut se mettre sous la forme d'une équation différentielle linéaire à coefficients constants.

Modélisation des systèmes linéaires continus invariants

Représentation par schémas blocs

Un système sera représenté par un schéma bloc ou (schéma bloc fonctionnel), dans lequel on pourra distinguer :



Les blocs :

Chaque sous-système est représenté par une boîte noire (bloc fonctionnel). Chaque bloc fonctionnel possède une seule entrée et une seule sortie (système monovariable).

A chaque bloc fonctionnel correspond **une équation différentielle linéaire à coefficients constants** :

$$a_n \frac{d^{(n)}s(t)}{dt^n} + \ldots + a_1 \frac{ds(t)}{dt} + a_0 s(t) = b_m \frac{d^{(m)}e(t)}{dt^m} + \ldots + b_1 \frac{de(t)}{dt} + b_0 e(t)$$

encore appelée **forme canonique de la fonction de transfert** :

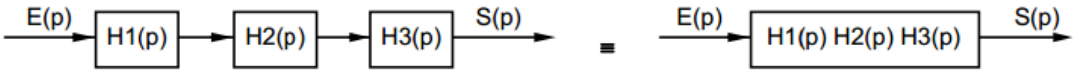
$$H(p)=\frac{S(p)}{E(p)}=\frac{K}{p^{\alpha}}\frac{1+b_1'p+\ldots}{1+a_1'p+\ldots}$$

On définit :

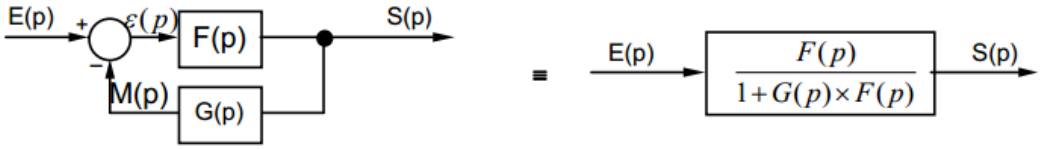
Les **pôles** : les racines du dénominateur
Les **zéros** : les racines du numérateur
α classe du système : si α≠0 alors p=0 est un pôle du dénominateur. On dit que le système comporte α intégrateurs.
K gain

Opérations sur les schémas blocs

Transmittances en série



Structure en boucle fermée



Fonction de transfert en boucle fermée

$$FTBF(p)=\frac{S(p)}{E(p)}=\frac{F(p)}{1+G(p)\times F(p)}$$

Afin de déterminer les fonctions de transferts en chaîne directe et en boucle ouverte, la boucle de retour est coupé.

Fonction de transfert en boucle ouverte :

$$FTBO(p)=\frac{M(p)}{\varepsilon(p)}=F(p)\times G(p)$$

Cas des systèmes avec perturbation :

Il faut alors réaliser une étude en poursuite (système suiveur) et une étude en régulation (système régulateur). Il y a deux fonctions de transfert à calculer

II.2.2 ANALYSE DES SYSTEMES LINEAIRES

TRANSFORMEE DE LAPLACE.

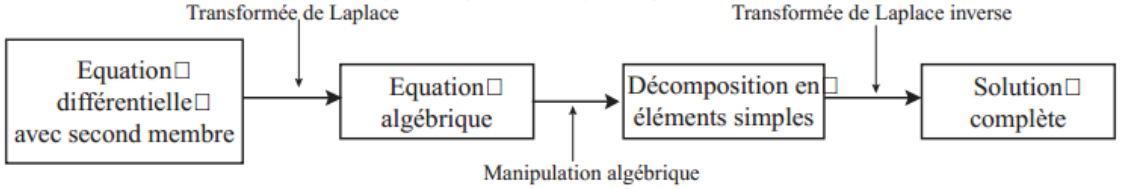
Utilisation des transformées de Laplace :

La transformée de Laplace permet de :

Transformer une équation différentielle linéaire en une équation algébrique plus facilemer exploitable.

De revenir dans le domaine temporel par la transformée de Laplace inverse (cas simple)

De donner les caractéristiques principales du système en termes de performances san calculer la réponse temporelle (cas + complexes)



Définition

$$F(p) = \mathcal{L}(f(t)) = \int_0^{+\infty} e^{-p.t} \cdot f(t)\, dt$$

p est un nombre complexe

La transformée de Laplace inverse sera notée : $f(t) = \mathcal{L}^{-1}(F(p))$

PROPRIÉTÉS

Dérivation

Ce théorème très important pour le traitement, par la transformée de Laplace, des systèmes linéaires continus permet la linéarisation des équations différentielles. La transformée de Laplace de $\dot{f}(t)$ vaut :

$$L(\dot{f}(t)) = p.F(p) - f(0) \quad \text{avec} \quad f(0) = \lim_{t \to 0^+} f(t)$$

Si toutes les conditions initiales sont nulles (on parle de fonction causale) : f(0)= f'(0) =... = $f^{(n-1)}(0)$= 0, l'expression précédente se réduit à :

$$\mathcal{L}\left(f^{(n)}(t)\right) = p^n . F(p)$$

intégration

$$\mathcal{L}\left(\int_0^t f(\tau) . d\tau\right) = \frac{F(p)}{p}$$

valable uniquement pour une intégration de 0 à t

Théorème de la valeur initiale et finale

Lorsque vous désirez obtenir des informations sur la fonction originale f(t) (au voisinage de t = 0 et t = infini) sans calculer la transformée de Laplace inverse de F(p),vous pouvez utiliser les théorèmes suivants :

$$\lim_{t\to 0} f(t) = \lim_{p\to+\infty} p.F(p) \quad \text{et} \quad \lim_{t\to+\infty} f(t) = \lim_{p\to 0} p.F(p)$$

Théorèmes valables dans la mesure où la limite existe

Retard

$$\forall\ \tau \in \Re \quad \mathcal{L}\left((f(t-\tau)\right) = e^{-p.\tau}.F(p)$$

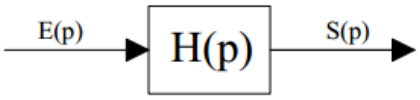
TRANSFORMEE DE LAPLACE INVERSE

Cette opération consiste à rechercher la fonction temporelle qui correspond à une expression F(p) donnée. Pour cela, il faut décomposer la fonction F(p) en éléments simples (éléments pour lesquels on connaît les transformées de Laplace inverses).

Il faut donc identifier les pôles de F(p), factoriser la fonction, puis déterminer les différentes constantes relatives à chaque élément simple :

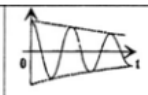
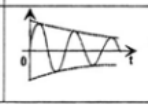
$$F(p) = \frac{N(p)}{D(p)} = A_0 + \sum_{i=1}^{n}\sum_{j=1}^{m} \frac{A_{ij}}{(p+\alpha_i)^j} + \sum_{k=1}^{o}\sum_{l=1}^{p} \frac{(A_{kl}.p + B_{kl})}{((p+\beta_k)^2+\omega_k^2)^l}$$

En SI, le but est de déterminer les réponses temporelles s(t) de système à partir de la connaissance de sa fonction de transfert H(p).



Ainsi, le calcul algébrique est toujours effectué sur la fonction S(p) = H(p) .E(p)
E(p) est dépendant du type de réponse que l'on souhaite déterminée.
Pour une **réponse impulsionnelle** l'entrée E(p) est une impulsion (Dirac) donc **E(p)=1**
Pour une **réponse indicielle**, l'entrée E(p) est un échelon unitaire donc **E(p)=1/p**

La fraction rationnelle associée à S(p) étant décomposée en éléments simples, il s'agit d'utiliser le tableau des transformées usuelles.

Cosinus amorti		$e^{-at}\cos(\omega t)U(t)$	$\frac{p+a}{(p+a)^2+\omega^2}$	$-a \pm j.\omega$
Sinus amorti		$e^{-at}\sin(\omega t)U(t)$	$\frac{\omega}{(p+a)^2+\omega^2}$	$-a \pm j.\omega$

Exemple : cas de racine complexe

$$S(p) = \frac{1}{p+1} - \frac{p-2}{p^2+p+1}$$

Il faut faire apparaitre les formes du cosinus et sinus amorti :

$$S(p) = \frac{1}{p+1} - \frac{p-2}{p^2+p+1} = \frac{1}{p+1} - \left[\frac{p-2}{\left(p+\frac{1}{2}\right)^2+\frac{3}{4}}\right]$$

$$S(p) = \frac{1}{p+1} - \left[\frac{p+1/2}{\left(p+\frac{1}{2}\right)^2+\frac{3}{4}} - \frac{5/2}{\left(p+\frac{1}{2}\right)^2+\frac{3}{4}}\right]$$

$$S(p) = \frac{1}{p+1} - \left[\frac{p+1/2}{\left(p+\frac{1}{2}\right)^2+\frac{3}{4}} - \frac{5}{\sqrt{3}}.\frac{\sqrt{3}/2}{\left(p+\frac{1}{2}\right)^2+\frac{3}{4}}\right]$$

$$s(t) = (e^{-t} - e^{-t/2}[\cos\left(\frac{\sqrt{3}}{2}t\right) - \frac{5}{\sqrt{3}}.\sin\left(\frac{\sqrt{3}}{2}t\right)]).u(t)$$

REPONSE TEMPORELLE DES SYSTEMES DU PREMIER ET DU SECOND ORDRE.

Système du 1ier ordre

Définition

Un système du premier ordre est un système où la relation entre l'entrée et la sortie peut se mettre sous la forme d'une équation différentielle du premier ordre :

$$s(t) + \tau.\dot{s}(t) = K.e(t)$$

K est le **gain statique**, **τ** est la **constante de temps** du système.

La **fonction de transfert** de ce système est donc : $H(p) = \frac{S(p)}{E(p)} = \frac{K}{1+\tau.p}$

Réponses temporelles aux signaux tests

Réponse indicielle : $e(t) = e_0 u(t)$

Réponse temporelle : La transformée de LAPLACE de e(t) est égale à E(p) = $\frac{e_0}{p}$. La sortie du processus est donc égale à : S(p) = $\frac{Ke_0}{p(1+\tau p)}$. En décomposant en éléments simples, on obtient : $S(p) = Ke_0\left(\frac{1}{p} - \frac{1}{(\frac{1}{\tau}+p)}\right)$	

En déterminant la fonction originale de S(p), on en déduit la réponse indicielle d'un système du premier ordre : $\boxed{s(t) = K \cdot e_0 \cdot \left(1 - e^{-t/\tau}\right) \cdot u(t)}$

Propriétés remarquables :

- **La valeur à convergence vaut :** $\lim_{t\to\infty} s(t) = K.e_0$
- **Le temps de réponse à 5% est obtenu pour un temps** $t = 3\tau$: $s(3\tau) = 0{,}95.K.e_0$
- Pour le temps $t = \tau$, on obtient : $s(\tau) = 0{,}63.K.e_0$
- La pente à l'origine vaut $K.e_0/\tau$

Système du 2nd ordre

Définition

L'équation différentielle d'un second ordre est de la forme :

$$\boxed{\ddot{s}(t) + 2\,.\,\xi.\,\omega_0\,.\,\dot{s}(t) + \omega_0^2\,.\,s(t) = K\,.\,\omega_0^2\,.\,e(t)}$$

K : gain statique, ξ : coefficient d'amortissement,
ω_0 : pulsation propre non amortie du système.

La fonction de transfert s'écrit donc : $\boxed{H(p) = \dfrac{K}{1 + \dfrac{2.\xi.p}{\omega_0} + \dfrac{p^2}{\omega_0^2}}}$

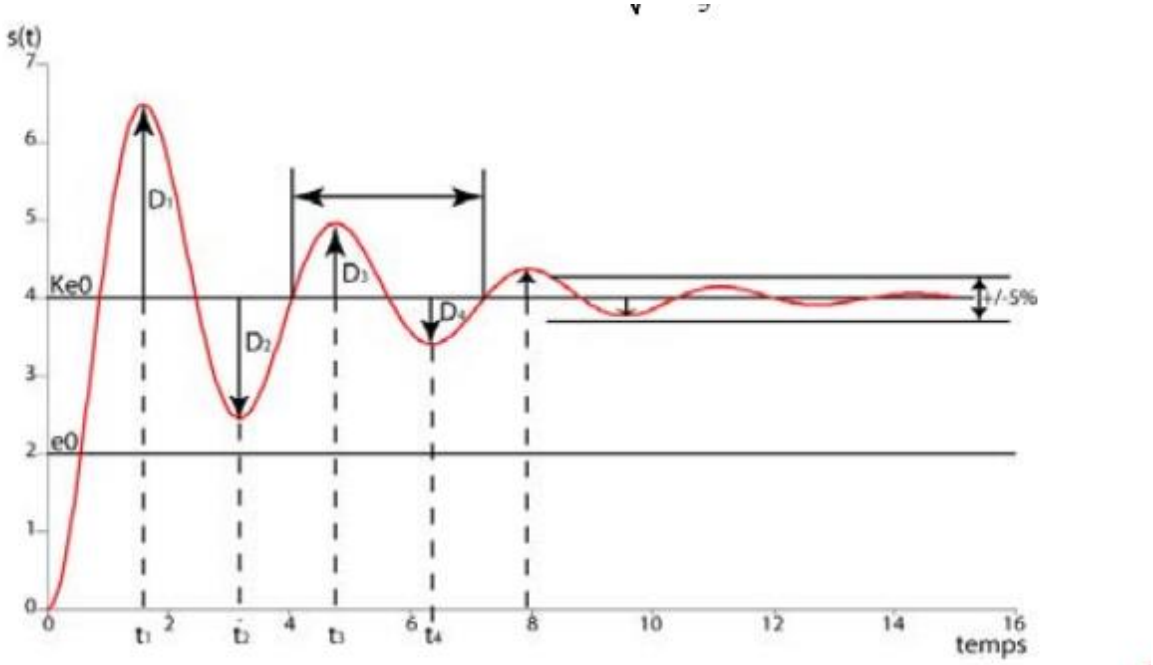
Pour déterminer la sortie d'un système du second ordre, il faut distinguer trois cas en fonction de la nature des pôles de H(p) ($\Delta = \dfrac{4}{\omega_0^2}\left(\zeta^2 - 1\right)$). H(p) possède :

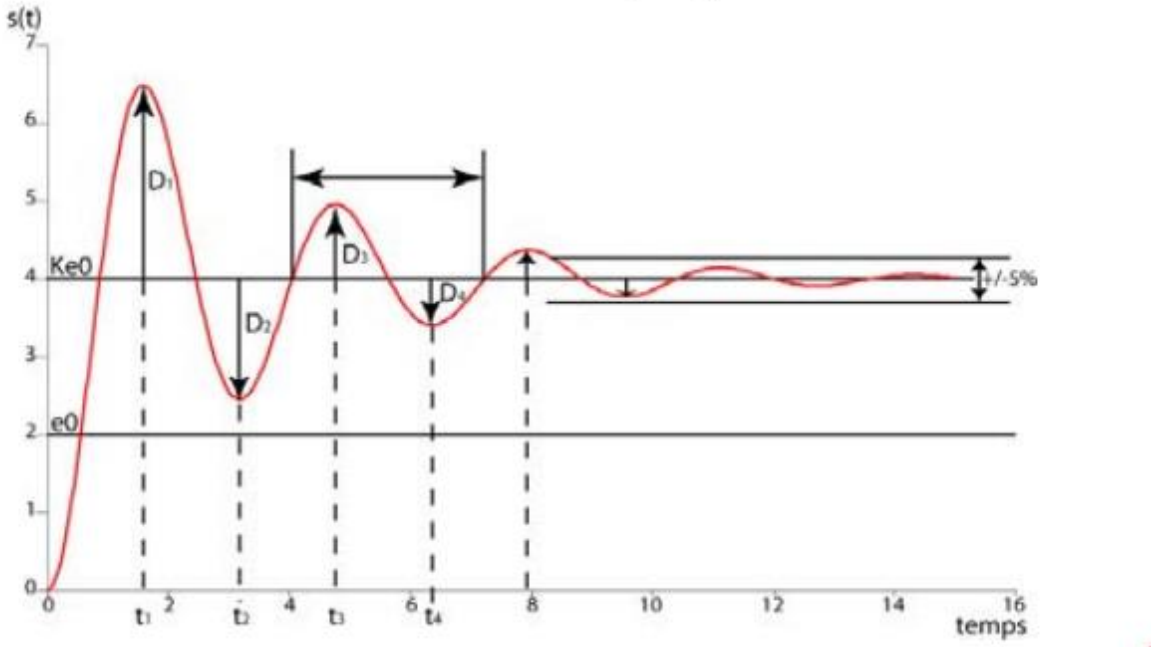
deux pôles complexes conjugués (ξ<1), on dira que l'on est en régime pseudopériodique.
deux pôles réels distincts (ξ>1), on dira alors que l'on est en régime apériodique,
un pôle double (ξ=1), on dira que l'on est en régime apériodique critique,

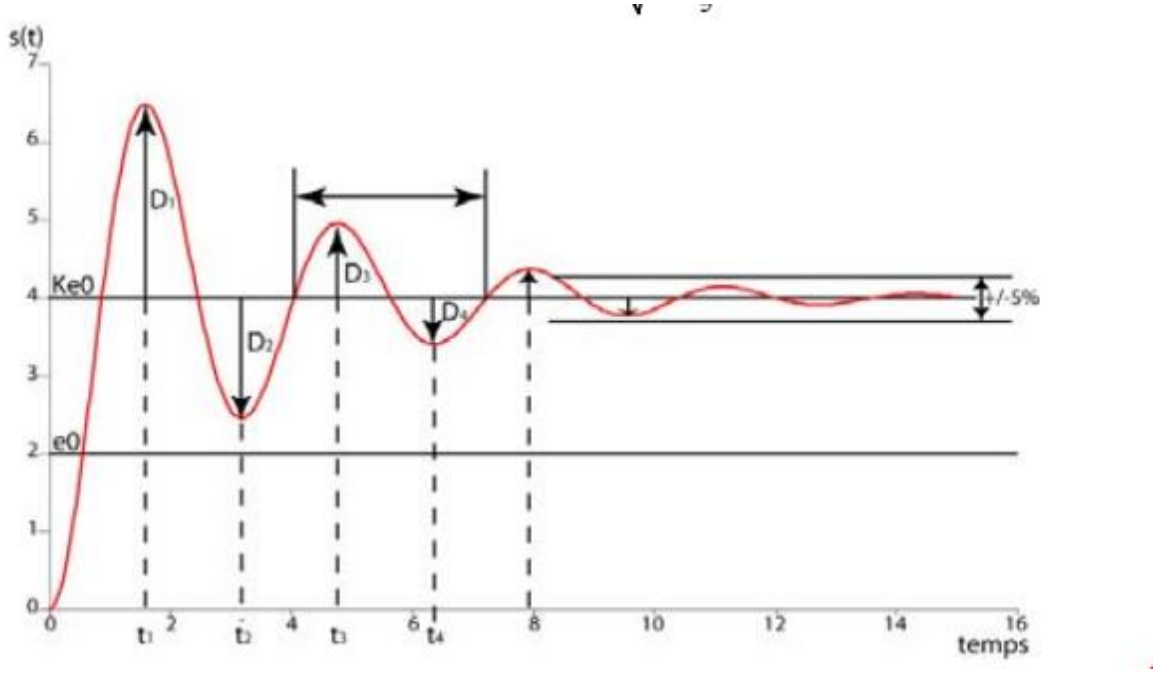
Réponse indicielle : $e(t) = e_0 u(t)$

Régime pseudopériodique (0 < ξ < 1)

$$s(t) = K\,e_0\left[1 - e^{-\omega_0 \xi t}\left(\cos\left(\sqrt{1-\xi^2}\;\omega_0\,t\right) + \frac{\xi}{\sqrt{1-\xi^2}}\sin\left(\sqrt{1-\xi^2}\;\omega_0\,t\right)\right)\right]u(t)$$


s(t)
Ke0
e0
D1
D2
D3
D4
+/-5%
t1
t2
t3
t4
temps

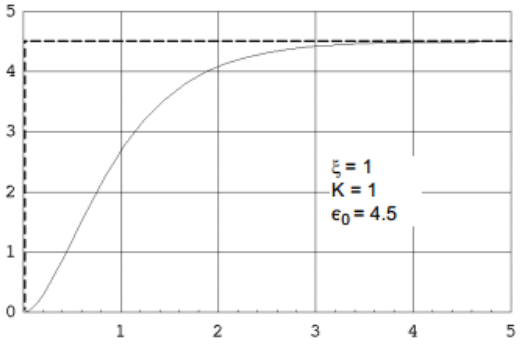

s(t)
Ke0
e0
D1
D2
D3
D4
+/-5%
t1
t2
t3
t4
temps



Propriétés remarquables :
La tangente à l'origine est horizontale. La valeur à convergence permet d'identifier le gain statique K. Le dépassement D_1 permet d'identifier le coefficient d'amortissement ζ tandis que temps de dépassement t_1 permet d'identifier la pulsation non amortie ω_0.

Temps du k^{ème} dépassement	$t_k = \dfrac{k.\pi}{\omega_0\sqrt{1-\zeta^2}}$
Amplitude du k^{ème} dépassement	$D_k = K.e_0.\exp\left(\dfrac{-k\pi\zeta}{\sqrt{1-\zeta^2}}\right)$
Pseudo pulsation	$\omega_p = \omega_0\sqrt{1-\zeta^2}$
Pseudo période	$T = \dfrac{2\pi}{\omega_p} = \dfrac{2\pi}{\omega_0\sqrt{1-\zeta^2}}$

Régime apériodique critique (ξ =1)



$$s(t) = K.e_0.\{1-(1+\omega_0.t).e^{-\omega_0.t}\}\ .\ u(t)$$

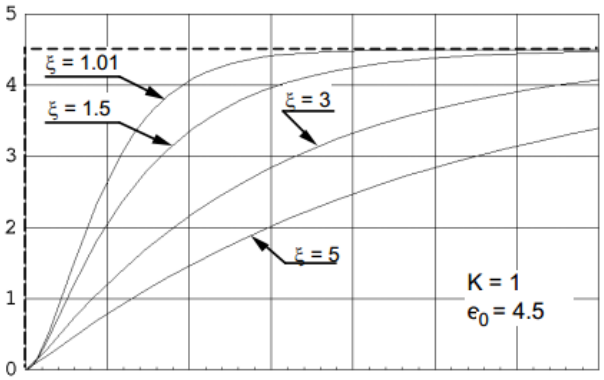
Propriétés :

Pente à l'origine nulle,
aucun dépassement.

Régime apériodique (ξ >1)

La transmittance H(p) s'écrit donc sous la forme : $H(p) = \dfrac{K\omega_0^2}{\left(p+\dfrac{1}{T1}\right)\left(p+\dfrac{1}{T2}\right)}$

$s(t) = K\, e_0\, [1 - \frac{1}{2}(1 + a)\, e^{-t/T_1} - \frac{1}{2}(1-a)\, e^{-t/T_2}]u(t)$

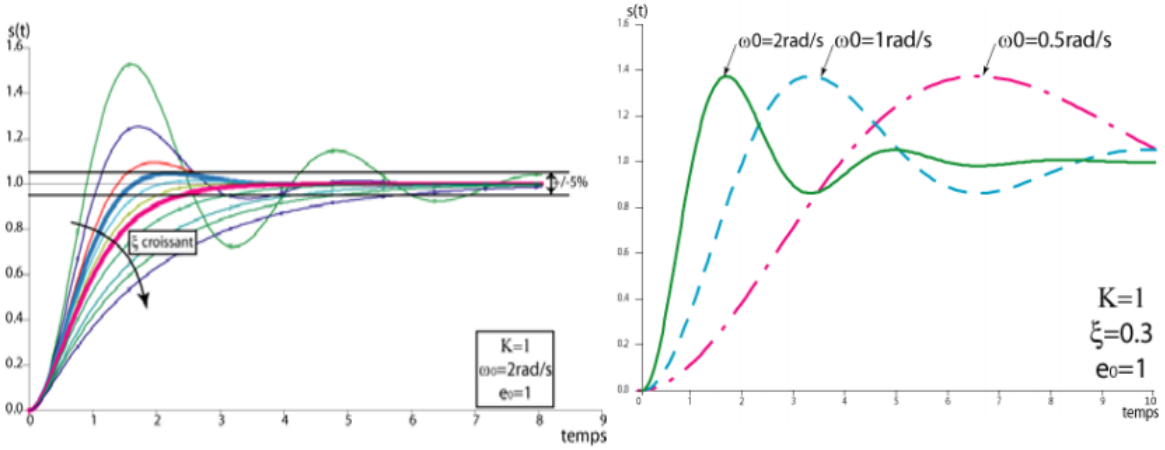


Plusieurs cas sont à examiner suivant la valeur des racines T_1 et T_2.
Par exemple si $T_2 \approx 100\ T_1$, on obtient : $\omega_0 = \dfrac{1}{10T_1} \Rightarrow \xi \approx 5{,}05$

Par contre un amortissement $\xi = 1{,}01$ correspond à un rapport des racines $\dfrac{T_2}{T_1} = 1{,}04$

Propriétés : Lorsque les deux constantes de temps T_1 et T_2 sont très différentes (rapport de l'ordre de 3) alors le système peut être assimilé à un premier ordre de constante de temps T_2 éventuellement retardé de T_1.

Influence des paramètres caractéristiques ω_0 et ζ



La réponse temporelle présente des dépassements pour $\zeta > 1$ (pseudo périodique). Les courbes en gras représentent les cas particuliers suivants :

- $\zeta = 1$ réponse apériodique critique
- $\zeta \approx 0,7$ réponse optimisant le temps de réponse à 5 %(le premier dépassement vaut 1,05.K.e_0)

La pulsation ω ne modifie pas l'amplitude des dépassements.

REPONSE FREQUENTIELLE DES SYSTEMES DU PREMIER ET DU SECOND ORDRE.

DEFINITION

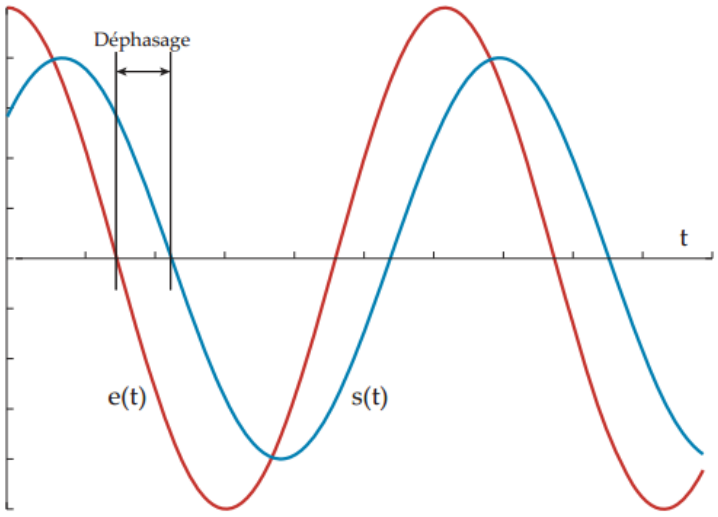
L'analyse harmonique d'un système consiste à le soumettre à une entrée sinusoïdale et à étudier sa sortie en régime permanent en fonction de la pulsation du signal d'entrée.

Considérons un système linéaire soumis à une entrée sinusoïdale :

$$e_{(t)} = e_0 \sin(\omega t)$$

On montre qu'après extinction du régime transitoire, la réponse s(t) est sinusoïdale de même pulsation, mais avec une amplitude s_ω et une phase φ_ω fonctions de la pulsation :

$$s_{(t)} = s_\omega \sin(\omega t + \varphi_\omega)$$



Pour chaque valeur de ω, on va définir les deux quantités suivantes :

le gain $G_\omega = \frac{s_\omega}{e_0}$

la phase φ_ω

On caractérise ainsi l'évolution du gain et de la phase en fonction de la pulsation.

FONCTION DE TRANSFERT EN REGIME HARMONIQUE

Soit H(p) la fonction de transfert d'un système linéaire.
On peut démontrer qu'en remplaçant la variable p de Laplace par jω, cette fonction de transfert s'écrit :

H(jω) est un nombre complexe :

de module $|H(j\omega)| = G_\omega$

d'argument $\arg(H(j\omega)) = \varphi_\omega$

$$H(j\omega) = \frac{S_\omega}{e_0} e^{j\varphi_\omega}$$

REPRESENTATION DU COMPORTEMENT HARMONIQUE

Ordres de grandeurs et modes de représentation

On utilise des échelles logarithmiques (logarithme décimal, noté : log) pour graduer les axes ou les courbes. Les échelles de variation de ces grandeurs sont si étendues qu'il est préférable de faire figurer les logarithmes de ces quantités. On définit alors le **gain en décibels** (dB) par

$$G_{dB} = 20 \log |H(j\omega)| \text{ en dB}$$

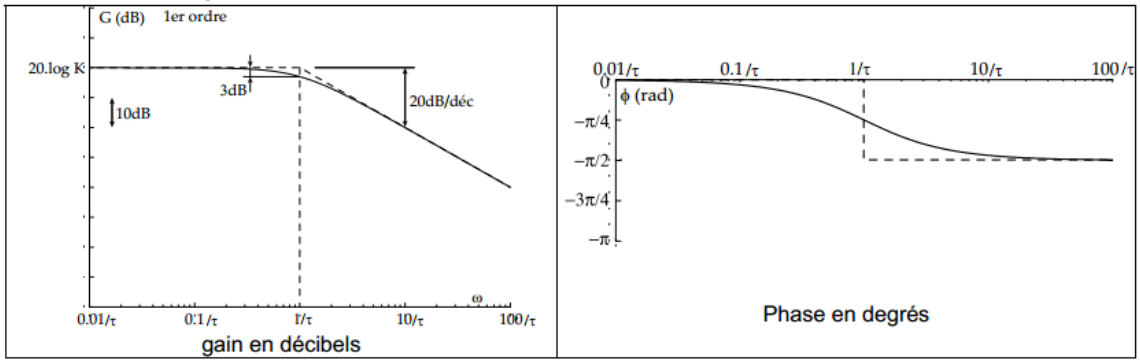
Afin de représenter H(jω) , on utilise principalement les représentations de BODE, de NYQUIST et de BLACK.

Diagramme de BODE

Représentation d'un système du premier ordre

La transmittance s'écrit sous la forme : $H(p) = \frac{K}{1+\tau p} \Rightarrow H(j\omega) = \frac{K}{1+j\tau\omega}$

$$\begin{cases} |H(j\omega)| = \dfrac{K}{\sqrt{1+\tau^2\omega^2}} \\ \varphi_\omega = \arg(H(j\omega)) = -\arctan(\tau\omega) \ et \ \varphi_\omega \in \left[-\dfrac{\pi}{2}, 0\right] \ car \ \tau\omega > 0 \end{cases}$$



La valeur $\omega_0 = 1/\tau$ de la pulsation est appelée pulsation de cassure; la courbe de phase présente un point d'inflexion pour cette valeur

Le diagramme réel est distinct du diagramme asymptotique. L'écart maximal entre les deux diagrammes est de 3 dB à la pulsation de cassure. Ces écarts relativement faibles font que l'on peut se satisfaire pour la plupart des applications du seul diagramme asymptotique.

Représentation d'un système du second ordre

$$H(p) = \frac{K}{1 + \dfrac{2\xi p}{\omega_0} + \dfrac{p^2}{\omega_0^2}} \Rightarrow H(j\omega) = \frac{K}{(1 - \dfrac{\omega^2}{\omega_0^2}) + \dfrac{2j\xi\omega}{\omega_0}}$$

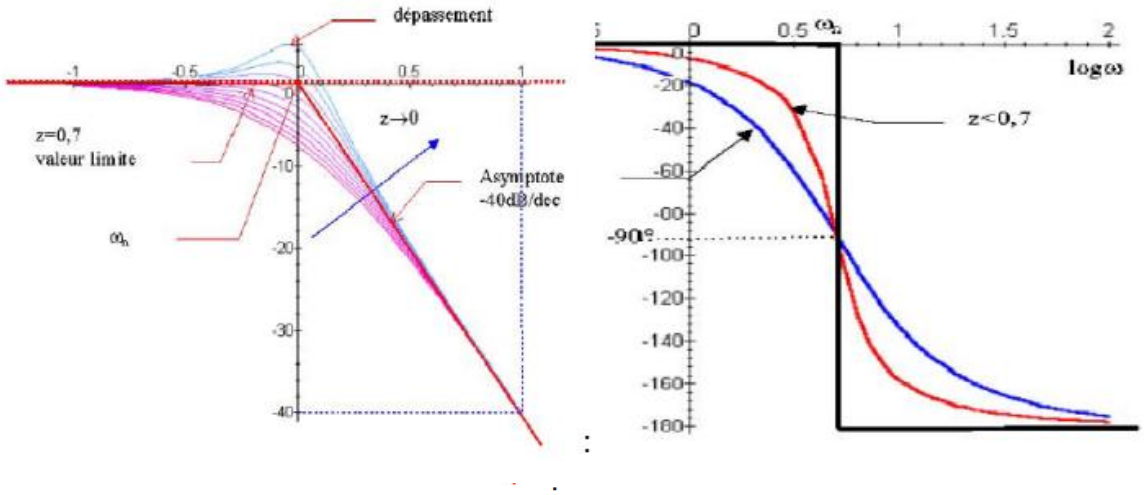
$$\begin{cases} |H(j\omega)| = \dfrac{K}{\sqrt{(1 - \dfrac{\omega^2}{\omega_0^2})^2 + \dfrac{4\xi^2\omega^2}{\omega_0^2}}} \\ \varphi_\omega = -\arg((1 - \dfrac{\omega^2}{\omega_0^2}) + \dfrac{2j\xi\ \omega}{\omega_0}) = -\arctan\left(\dfrac{2\xi\omega_0\omega}{(\omega_0^2 - \omega^2)}\right) \ et \ \varphi_\omega \in [-\pi, 0] \end{cases}$$

Propriétés et tracé de la courbe asymptotique : ***Pour 0<ξ≤1***

Si $\xi < \sqrt{2}/2$, alors la dérivée $\dfrac{d|H(j\omega)|}{d\omega}$ s'annule en $\boxed{\omega_n = \omega_0\sqrt{1-2\xi^2}}$, il y a donc un maximum en ω_n **qui est appelé pulsation de résonnance** et la valeur de la résonnance en gain vaut : $G_{dB} = 20\log\left(\dfrac{K}{2\xi\sqrt{1-\xi^2}}\right)$

Si $\xi > \sqrt{2}/2$ (fortement amorti), il n'y a pas de résonnnace.

Allure des diagrammes :



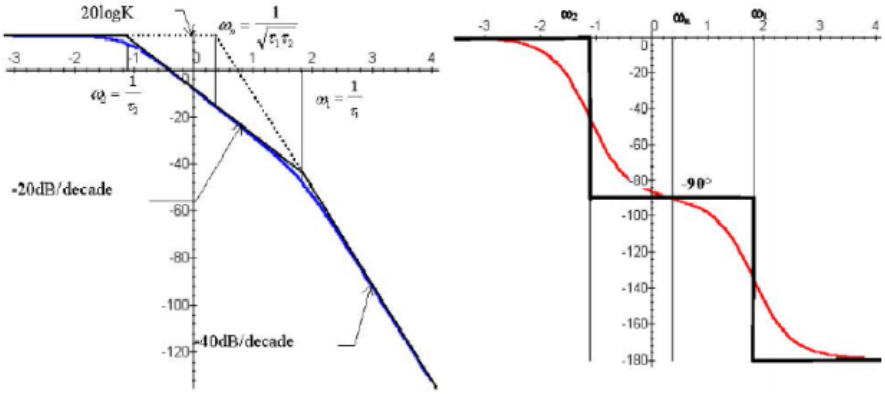
Pour ξ>1

les deux pôles sont réels, la fonction de transfert peut se mettre sous la forme :

$$H(j\,\omega) = \frac{K}{[1 + j\,T_1\,\omega]\,[1 + j\,T_2\,\omega]}$$

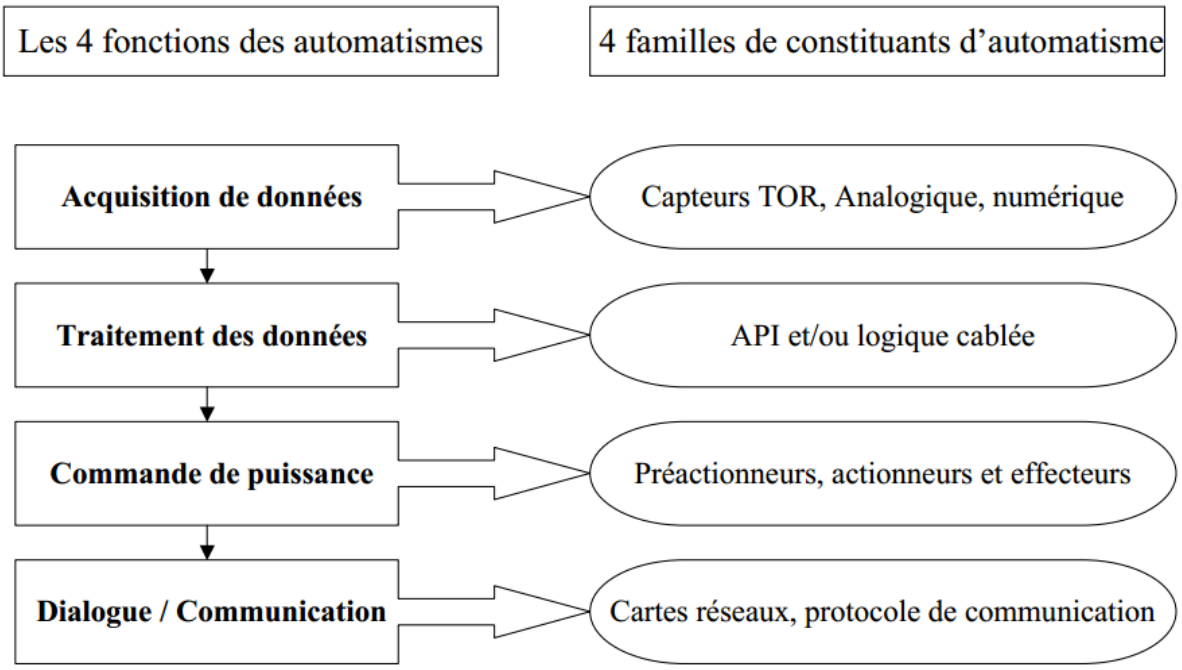
La fonction de transfert est la multiplication de deux premiers ordres de constante de temps T_1 et T_2 avec $\left(\frac{1}{T_2} < \frac{1}{T_1}\right)$.

Allure des diagrammes :

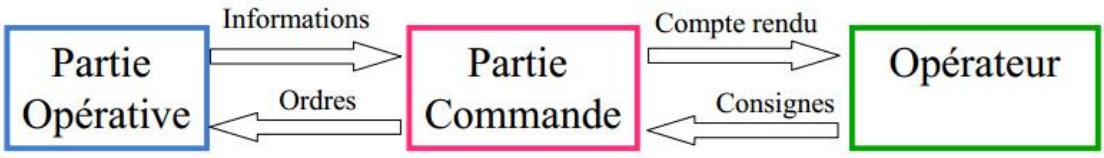


II.3 LES SYSTEMES AUTOMATISES LOGIQUES

II.3.1 LES FAMILLES DES CONSTITUANTS D'AUTOMATISME



II.2.2 FONCTIONS TRAITEMENTS DE DONNEES



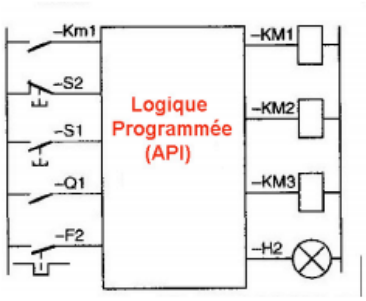
Logique programmée

API : Automate Programmable Industriel

Logique câblée

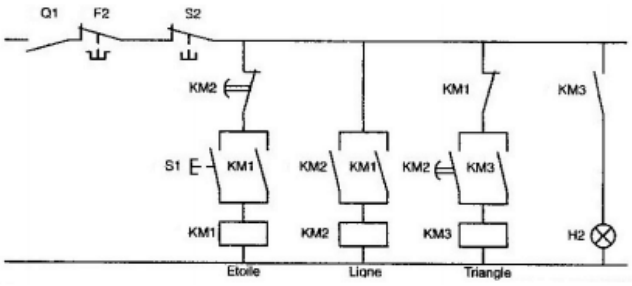
Relais de commande et déclinaisons auxiliaires

Logique programmée vs Logique câblée



Entrées
Km1 : contact de KM1
S2 : BP arrêt
S1 : BP marche
Q1 : sectionneur fermé
F2 : contact associé au relais thermique

Sorties
KM1 : contacteur étoile
KM2 : contacteur ligne
KM3 : contacteur triangle
H2 : voyant fin du démarrage

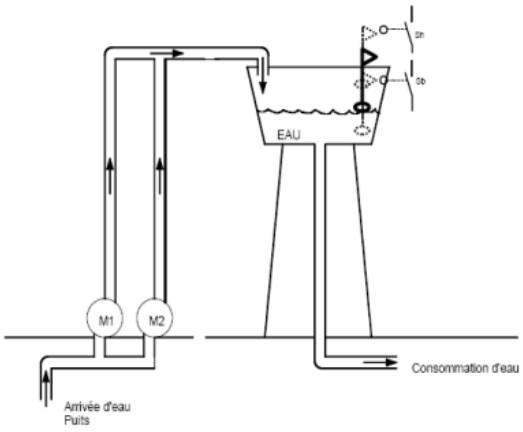


Avantages

Inconvénients

Logique programmée	Logique câblée
■Souplesse et adaptabilité de l'installation (*Remplacement des fonctions combinatoires et séquentielles par un programme).* ■Solution plus compacte	■*Automatisme simple et rapide à mettre en oeuvre* ■Obligatoire pour le traitement d'arrêt d'urgence et de sécurité.
■Plus cher. ■Compatilité entre familles d'automates. ■Pérennité d'une installation…	■Solution rigide et rapidement volumineuse.

II.3.2 EXEMPLE DE LA LOGIQUE CABLEE



Commander 2 pompes de remplissage d'un réservoir de sorte que:

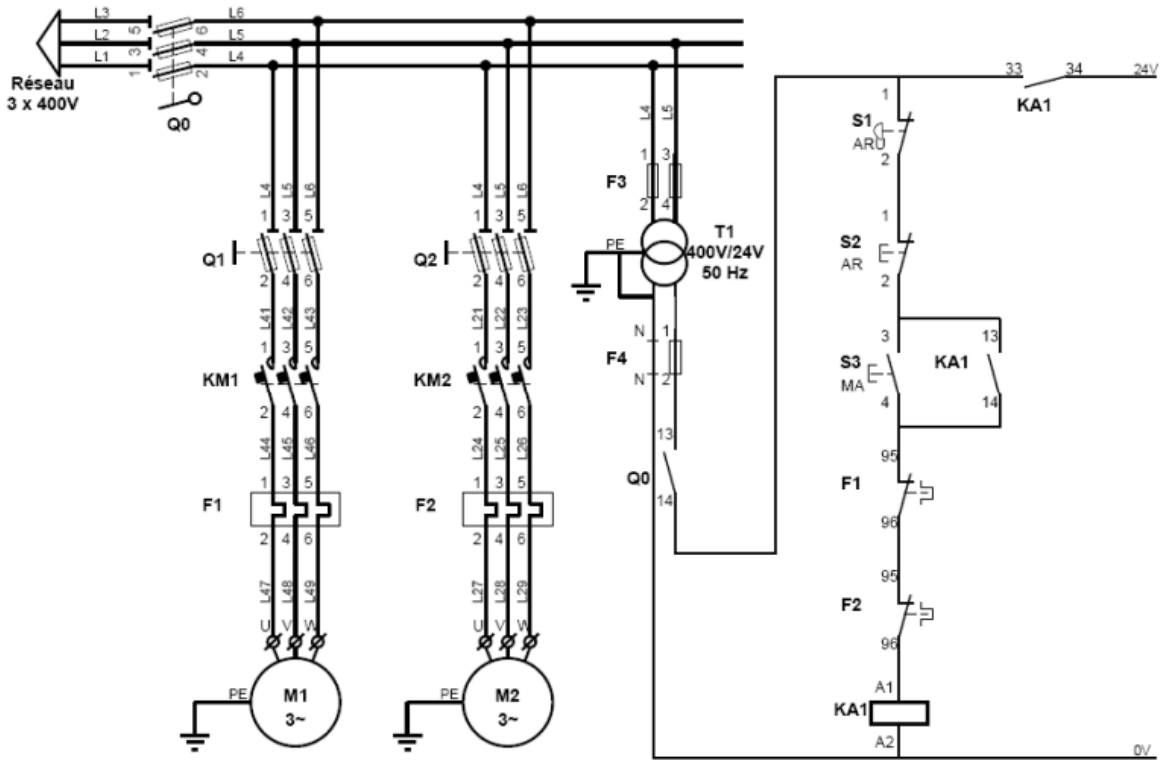
1. Quand la cuve est pleine (Sh=1 et Sb=1), aucune pompe ne fonctionne.
2. Quand la cuve est vide (Sh=0 et Sb=0), les 2 pompes fonctionnent
3. Quand le cuve est à moitié vide (ou pleine..) (Sh=0 et Sb=1), une seule pompe fonctionne. Le choix se fait à l'aide d'un commutateur C=1 alors le pompe M1 fonctionne.

Il vient intuitivement

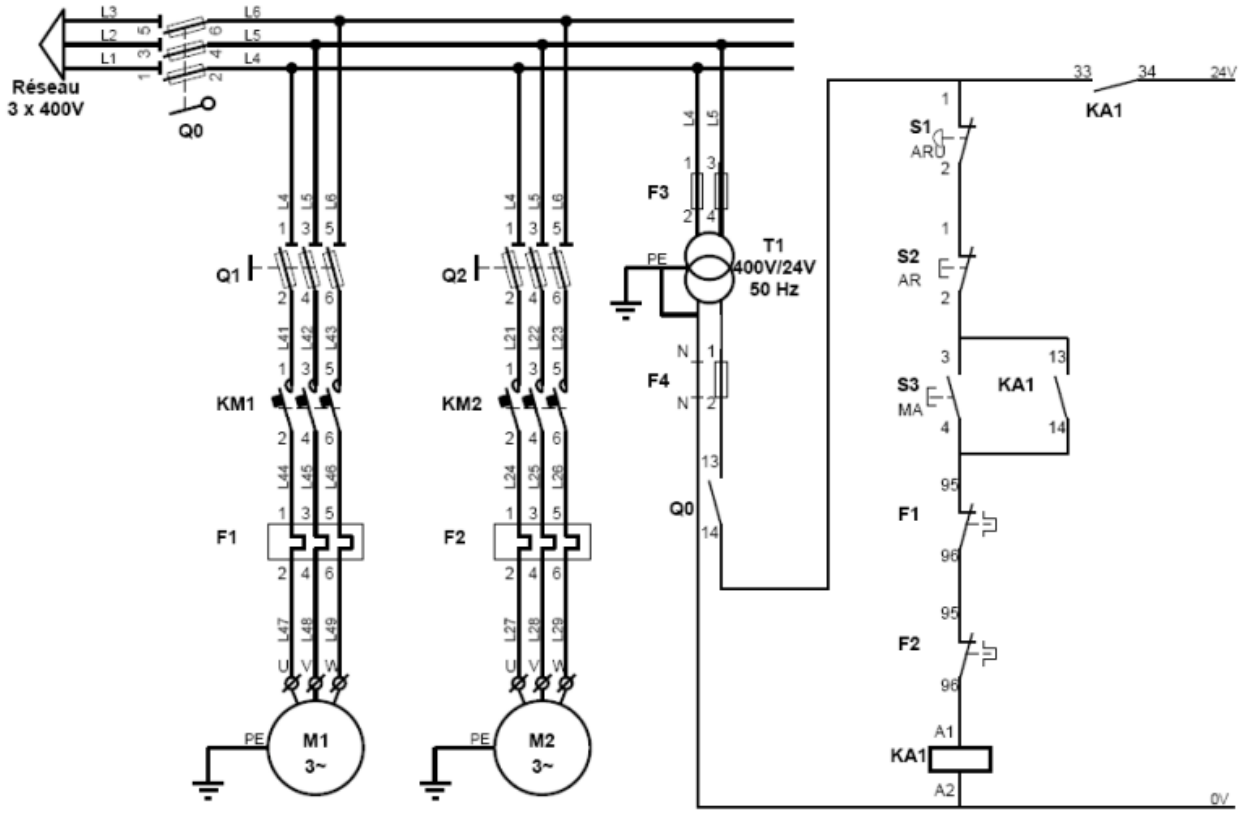
$$\begin{cases} KM1 = \overline{Sh} \cdot \overline{Sb} + C \cdot \overline{Sh} \\ KM2 = \overline{Sh} \cdot \overline{Sb} + \overline{C} \cdot \overline{Sh} \end{cases}$$

$$\begin{cases} KM1 = \overline{Sh} \cdot \left(\overline{Sb} + C \right) \\ KM2 = \overline{Sh} \cdot \left(\overline{Sb} + \overline{C} \right) \end{cases}$$

Logique programmée vs Logique câblée: Schéma de puissance de l'installation

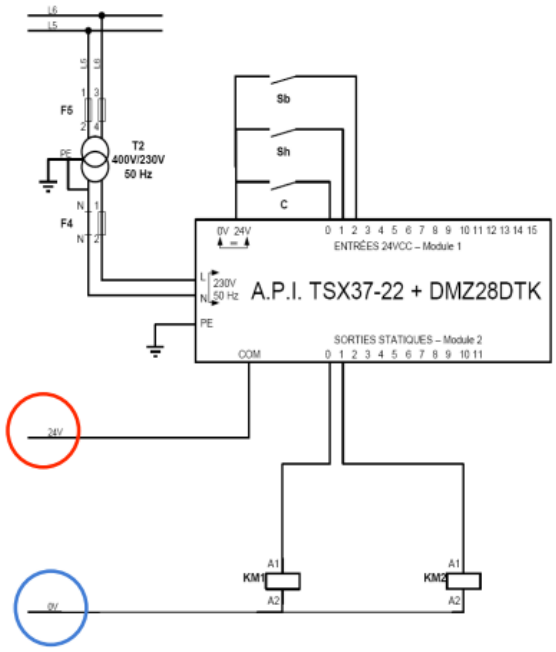


Logique programmée vs Logique câblée: Schéma de puissance de l'installation

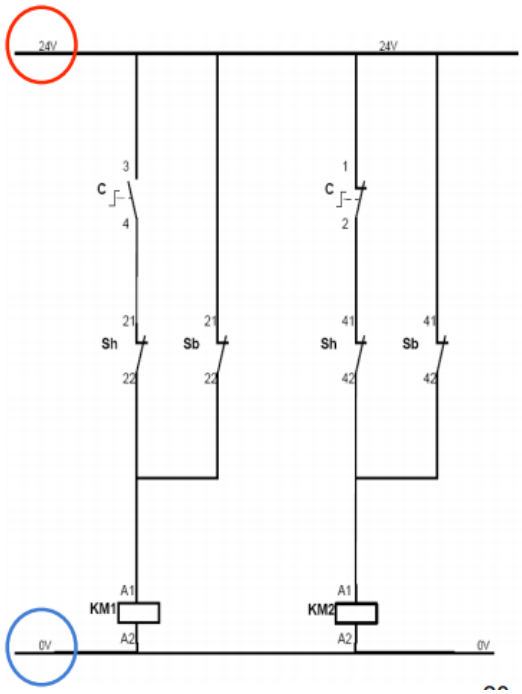


Logique programmée vs Logique câblée: Comparatif schéma de commande (3)

En logique programmée

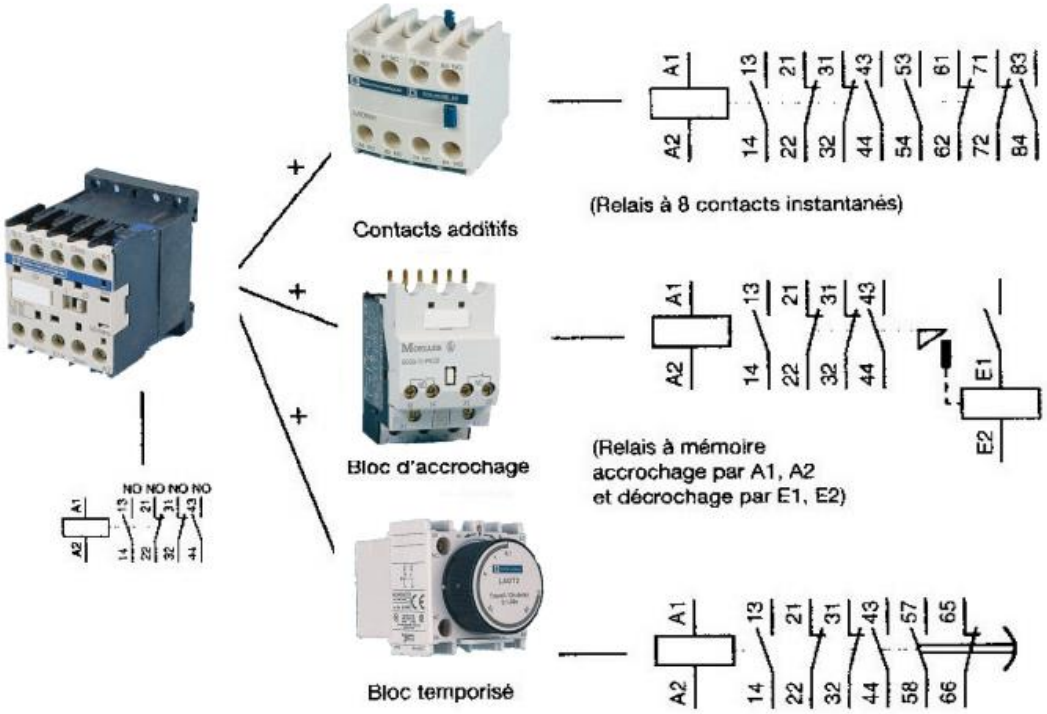


En logique câblée



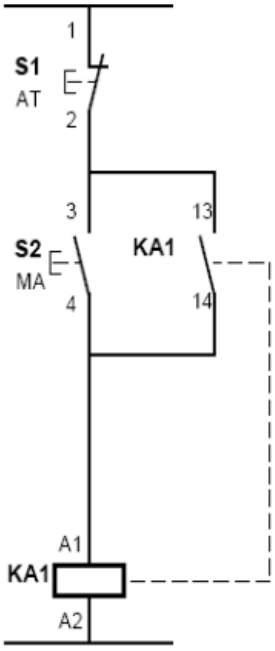
II.3.3 ELEMENTS DE LA LOGIQUE CABLEE ET DE LA LOGIQUE PROGRAMMEE

Élément de base des automatismes câblés: Le relais



Fonction Mémoire en logique câblée

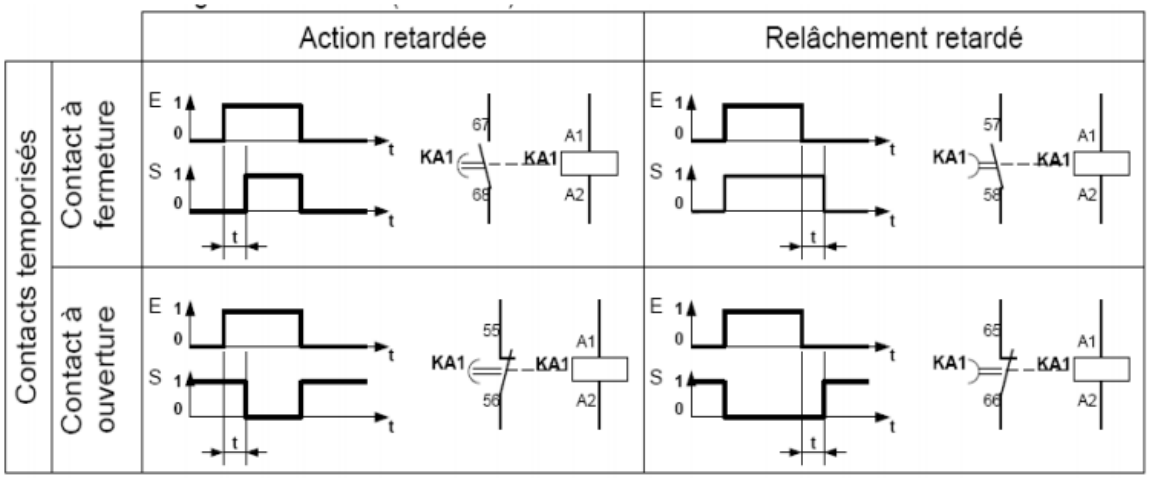
Schéma important: A comprendre et à retenir



Le relais KA1:
1°) « colle » par appui sur le BP NO S2.
2°) s'auto-maintient.
3°) « décolle » par appui sur le BP NF S1

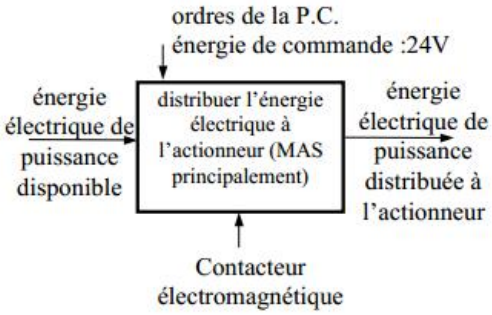
$$KA1=\overline{S1}\cdot(S2+KA1)$$

Les relais temporisés



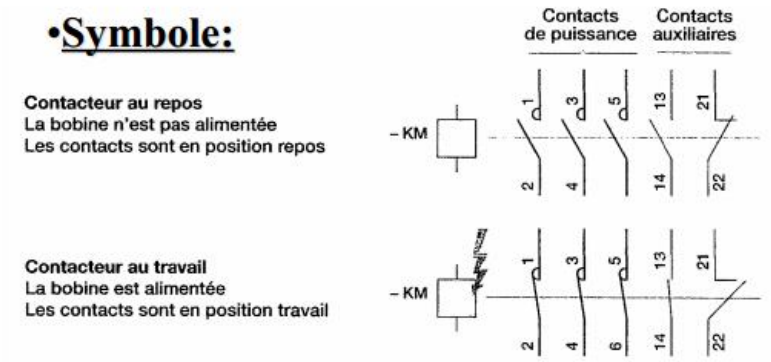
Pré-actionneurs électriques: les contacteurs

• aspect fonctionnel



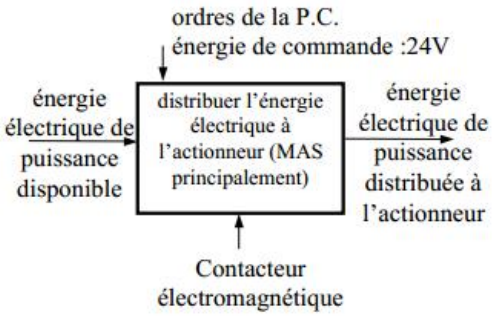
• Principe: électromagnétique

•Symbole:



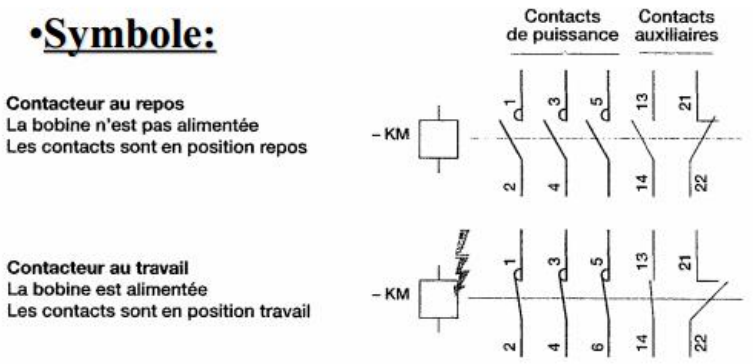
Pré-actionneurs électriques: les contacteurs

• aspect fonctionnel

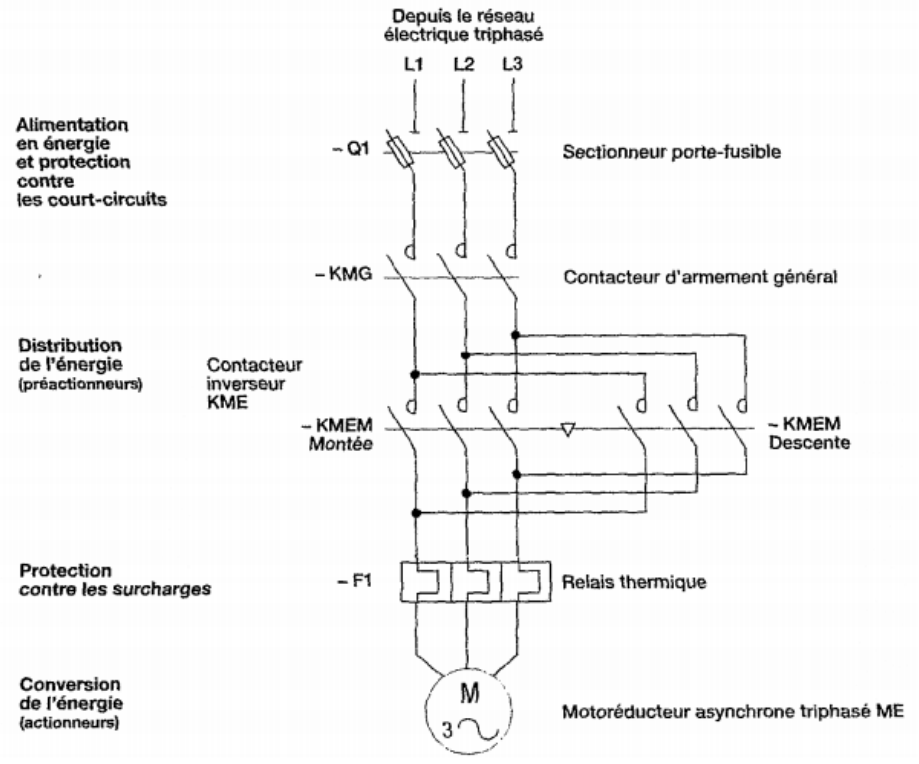


• Principe: électromagnétique

•Symbole:



Exemple de circuit de puissance d'un actionneur électrique



II.3.3 PRINCIPES DE LA LOGIQUE CABLEE

On traduit le fonctionnement souhaité de l'installation par des équations booléennes.

Exemple: $KM1 = Q1 \cdot \overline{F2} \cdot \overline{S2} \cdot \overline{KM2(T)} \cdot [S1 + KM1]$

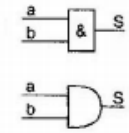
➡ **Les équations combinatoires se réalisent en langages à contact câblés**

⇨ **Les équations séquentielles se réalisent à partir du schéma classique de la mémoire à relais**

➡ **Les temporisations se réalisent à l'aide de relais temporisés**

Le langage à contacts

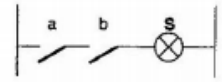
• **Opérateur logique ET**



a	b	S
0	0	0
0	1	0
1	0	0
1	1	1

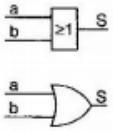
$S = a \bullet b$

S égal a ET b



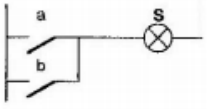
2 contacts en série

• **Opérateur logique OU**



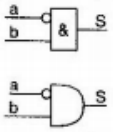
a	b	S
0	0	0
0	1	1
1	0	1
1	1	1

$S = a + b$



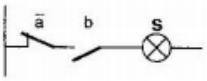
2 contacts en parallèle

• **Opérateur logique INHIBITION**



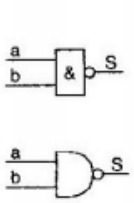
a	b	S
0	0	0
0	1	1
1	0	0
1	1	0

$S = \bar{a} \bullet b$



2 contacts en série

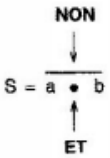
• **Opérateur logique NAND (NON ET)**



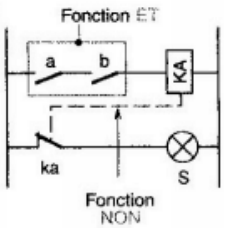
a	b	S
0	0	1
0	1	1
1	0	1
1	1	0

$S = \bar{a}\bar{b} + \bar{a}b + a\bar{b}$

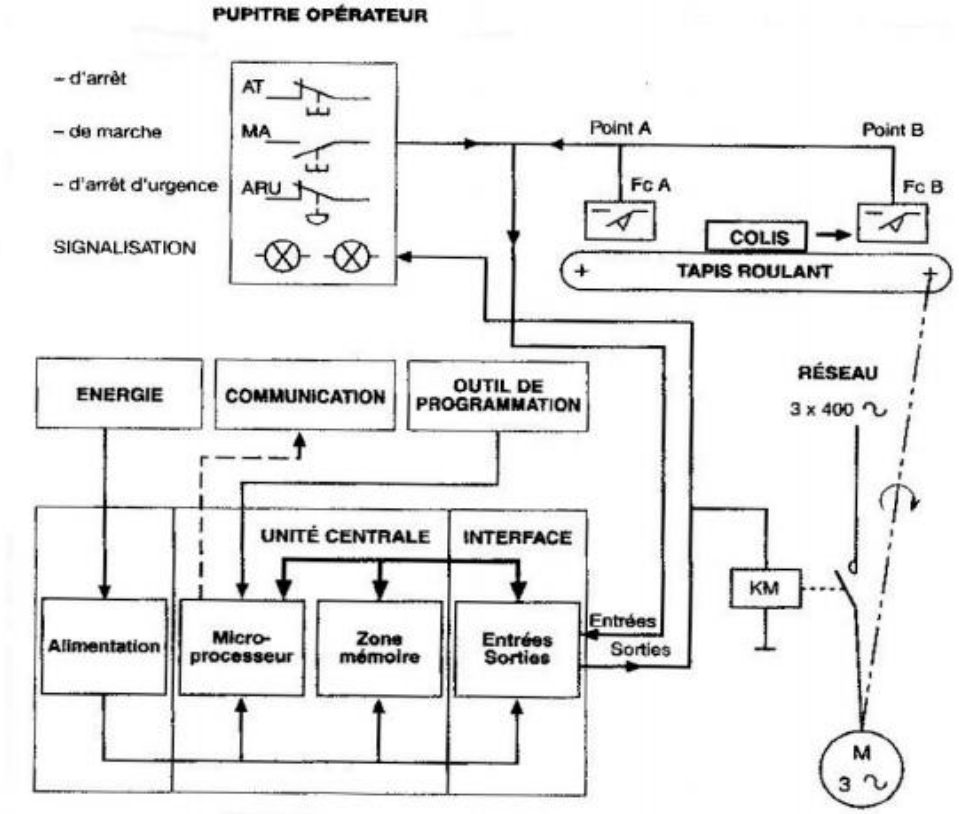
$S = \bar{a} + \bar{b}a \Rightarrow$



$S = \bar{a} + \bar{b}$



Synoptique d'une installation automatisée



CHAP.IV LES AUTOMATES PROGRAMMABLES INDUSTRIELS

Les Automates Programmables Industriels (API)

1- Introduction

Les **Automates Programmables Industriels (API)** sont apparus aux Etats-Unis vers 1969 où ils répondaient aux désirs des industries de l'automobile de développer des chaînes de fabrication automatisées qui pourraient suivre l'évolution des techniques et des modèles fabriqués.

Un Automate Programmable Industriel (**API**) est une machine électronique programmable par un personnel non informaticien et destiné à piloter en ambiance industrielle et en temps réel des procédés industriels. Un **automate programmable** est adaptable à un maximum d'application, d'un point de vue traitement, composants, language. C'est pour cela qu'il est de construction modulaire.

Il est en général manipulé par un personnel électromécanicien. Le développement de l'industrie à entraîner une augmentation constante des fonctions électroniques présentes dans un automatisme c'est pour ça que l'API s'est substitué aux armoires à relais en raison de sa souplesse dans la mise en œuvre, mais aussi parce que dans les coûts de câblage et de maintenance devenaient trop élevés.

2- Pourquoi l'automatisation ?

L'automatisation permet d'apporter des éléments supplémentaires à la valeur ajoutée par le système. Ces éléments sont exprimables en termes d'objectifs par :

- Accroître la productivité (rentabilité, compétitivité) du système
- Améliorer la flexibilité de production ;
- Améliorer la qualité du produit
- Adaptation à des contextes particuliers tel que les environnements hostiles pour l'homme (milieu toxique, dangeureux.. nucléaire...), adaptation à des tâches physiques ou intellectuelles pénibles pour l'homme (manipulation de lourdes charges, tâches répétitives parallélisées...),

- Augmenter la sécurité, etc...

Figure 4.1 : Automate SIEMENS S5-95U

3– Structure générale des API :

Les caractéristiques principales d'un automate programmable industriel (**API**) sont :
coffret, rack, baie ou cartes

- Compact ou modulaire
- Tension d'alimentation
- Taille mémoire
- Sauvegarde (EPROM, EEPROM, pile, …)
- Nombre d'entrées / sorties
- Modules complémentaires (analogique, communication,..)
- Langage de programmation

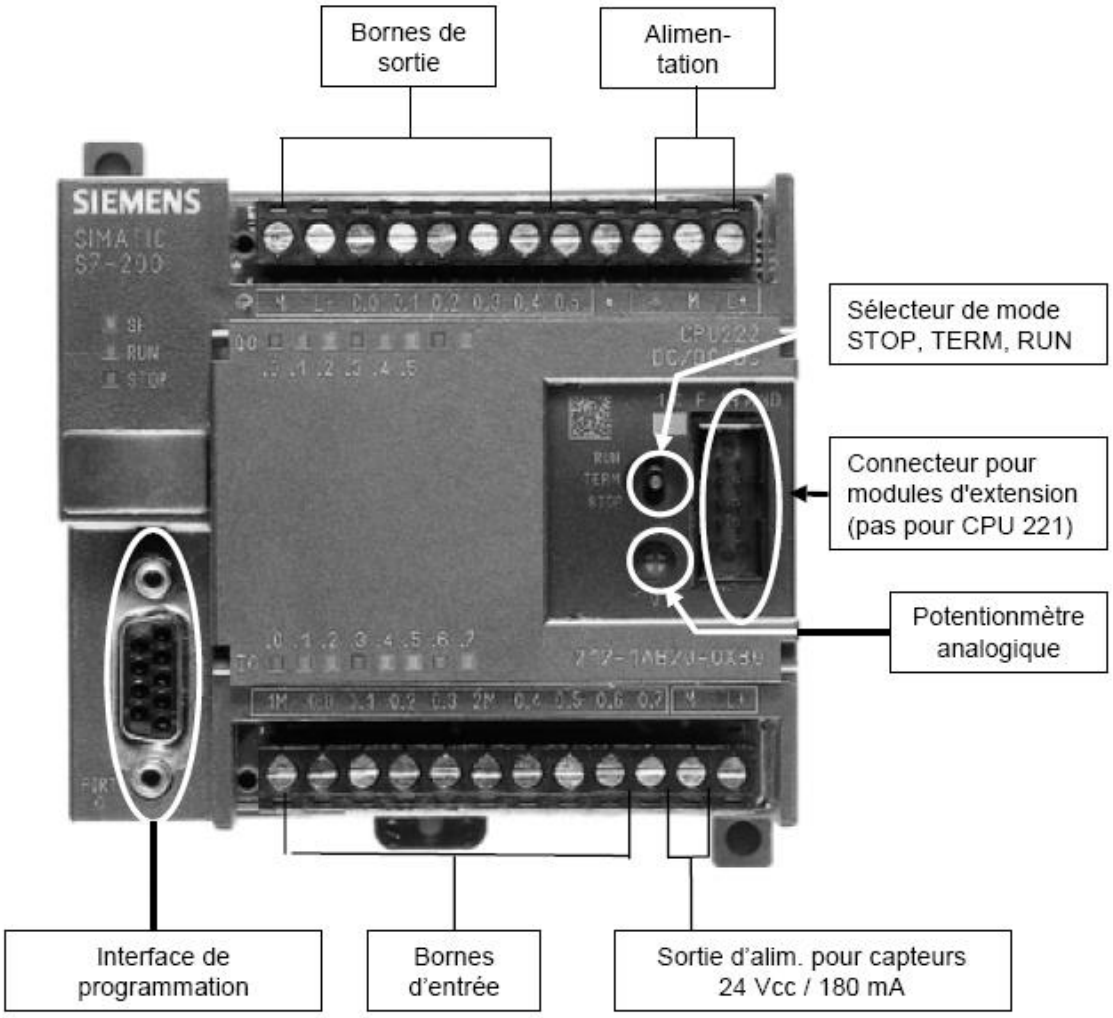


Figure 4.2 : Aspect extérieur d'un automate S7-200 CPU222

Des API en boîtier étanche sont utilisés pour les ambiances difficiles (température, poussière, risque de projection ...) supportant ainsi une large gamme de température, humidité ... L'environnement industriel se présentent sous trois formes :

- environnement physique et mécanique (poussières, température, humidité, vibrations);
- pollution chimique ;
- perturbation électrique. (parasites électromagnétiques)

Figure 4.3 : Automate Modulaire

4- Structure interne d'un automate programmable industriel (API) :

Les API comportent quatre principales parties (Figure 4.4) :

- Une unité de traitement (un processeur CPU);
- Une mémoire ;
- Des modules d'entrées-sorties ;
- Des interfaces d'entrées-sorties ;
- Une alimentation 230 V, 50/60 Hz (AC) - 24 V (DC).

La structure interne d'un **automate programmable industriel** (API) est assez voisine de celle d'un système informatique simple, L'unité centrale est le regroupement du processeur et de la mémoire centrale. Elle commande l'interprétation et l'exécution des instructions programme. Les instructions sont effectuées les unes après les autres, séquencées par une horloge.
Deux types de mémoire cohabitent :
- **La mémoire Programme** où est stocké le langage de programmation. Elle est en général figée, c'est à dire en lecture seulement. (ROM : mémoire morte)
- **La mémoire de données** utilisable en lecture-écriture pendant le fonctionnement c'est la RAM (mémoire vive). Elle fait partie du système entrées-sorties. Elle fige les valeurs (0 ou 1) présentes sur

les lignes d'entrées, à chaque prise en compte cyclique de celle-ci, elle mémorise les valeurs calculées à placer sur les sorties.

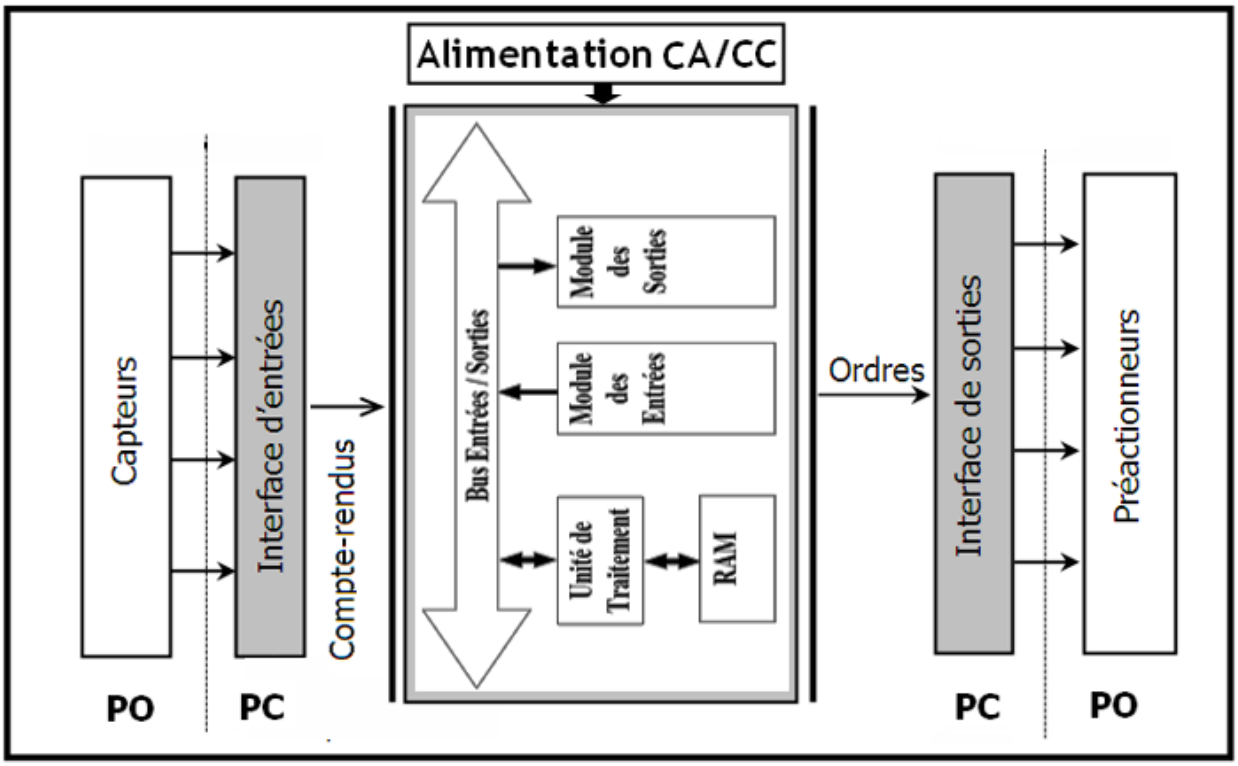


Figure 4.4 : Structure interne d'un automates programmables industriels (API)

5- Fonctionnement :

L'automate programmable **reçoit** les informations relatives à l'état du système et puis **commande** les pré-actionneurs suivant le programme inscrit dans sa mémoire.
Généralement les automates programmables industriels ont un fonctionnement cyclique (Figure 4.5). Le **microprocesseur** réalise toutes les fonctions logiques ET, OU, les fonctions de temporisation, de comptage, de calcul... Il est connecté aux autres éléments (mémoire et interface E/S) par des liaisons **parallèles** appelées ' **BUS** ' qui véhiculent les informations sous forme binaire.. Lorsque le fonctionnement est dit synchrone par rapport aux entrées et aux sorties, le cycle de traitement commence par la prise en compte des entrées qui sont figées en mémoire pour tout le cycle.

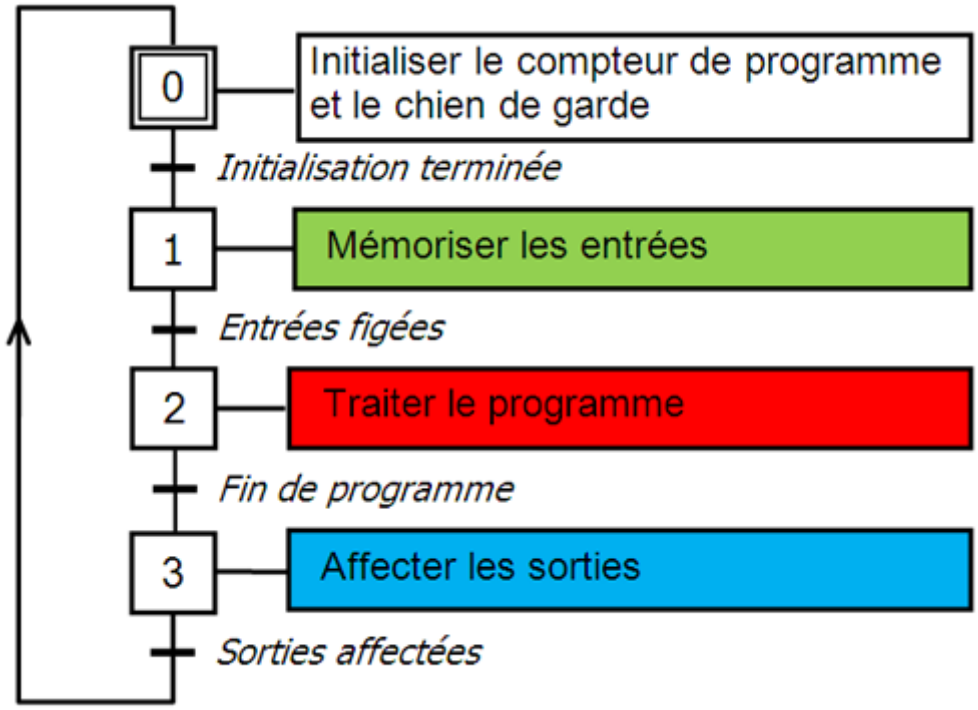


Figure 4.5 : Fonctionnement cyclique d'un API

Le processeur exécute alors le programme instruction par instruction en rangeant à chaque fois les résultats en mémoire. En fin de cycle les sorties sont affectées d'un état binaire, par mise en communication avec les mémoires correspondantes. Dans ce cas, le temps de réponse à une variation d'état d'une entrée peut être compris entre un ou deux temps de cycle (durée moyenne d'un temps de cycle est de 5 à 15 ms Figure 4.6).

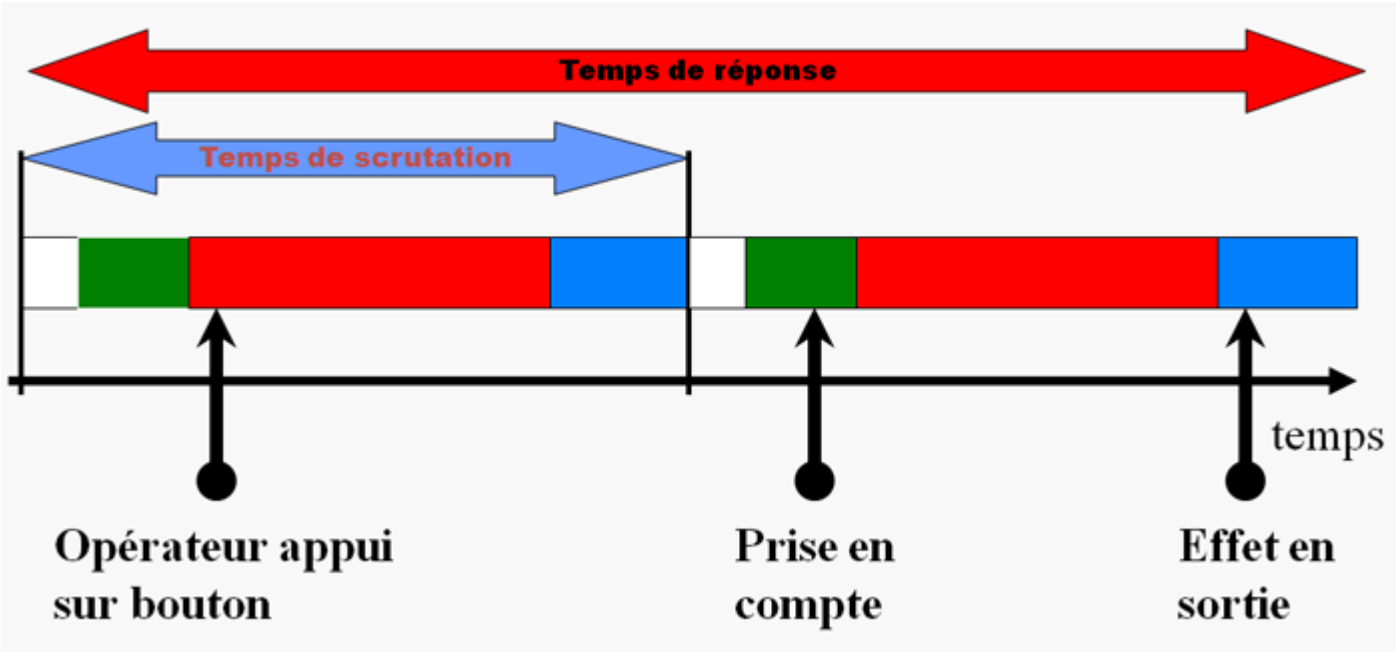


Figure 4.6 : Temps de scrutation vs Temps de réponse

Il existe d'autres modes de fonctionnement, moins courants :

- synchrone par rapport aux entrées seulement ;
- asynchrone.

6– Description des éléments d'un API :

6.1- La mémoire :

Elle est conçue pour recevoir, gérer, stocker des informations issues des différents secteurs du système que sont le terminal de programmation (PC ou console) et le processeur, qui lui gère et exécute le programme. Elle reçoit également des informations en provenance des capteurs.

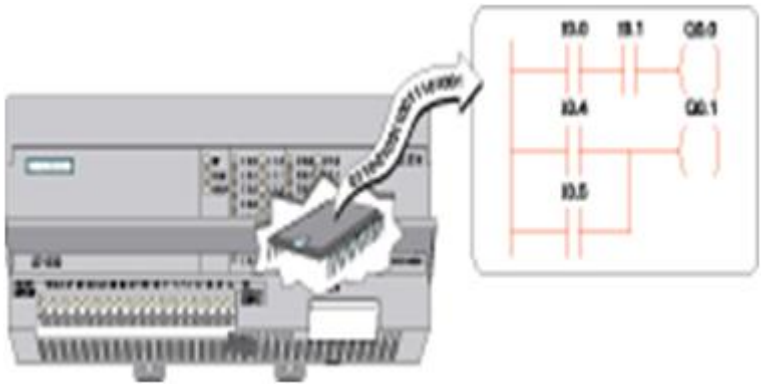


Figure 4.7 : La mémoire

Il existe dans les automates deux types de mémoires qui remplissent des fonctions différentes :
- La mémoire Langage où est stocké le langage de programmation. Elle est en général figée, c'est à dire en lecture seulement. (ROM : mémoire morte)
- La mémoire Travail utilisable en lecture-écriture pendant le fonctionnement c'est la RAM (mémoire vive). Elle s'efface automatiquement à l'arrêt de l'automate (nécessite une batterie de sauvegarde).

Répartition des zones mémoires :

- Table image des entrées
- Table image des sorties
- Mémoire des bits internes
- Mémoire programme d'application

6.2- Le processeur :

Son rôle consiste d'une part à organiser les différentes relations entre la zone mémoire et les interfaces d'entrées et de sorties et d'autre part à exécuter les instructions du programme.

6.3- Les interfaces et les cartes d'Entrées / Sorties:

L'interface d'entrée comporte des adresses d'entrée. Chaque capteur est relié à une de ces adresses. L'interface de sortie comporte de la même façon des adresses de sortie. Chaque préactionneur est relié à une de ces

adresses. Le nombre de ces entrées est sorties varie suivant le type d'automate. Les cartes d'E/S ont une modularit´e de 8, 16 ou 32 voies. Les tensions disponibles sont normalisées (24, 48, 110 ou 230V continu ou alternatif ...).

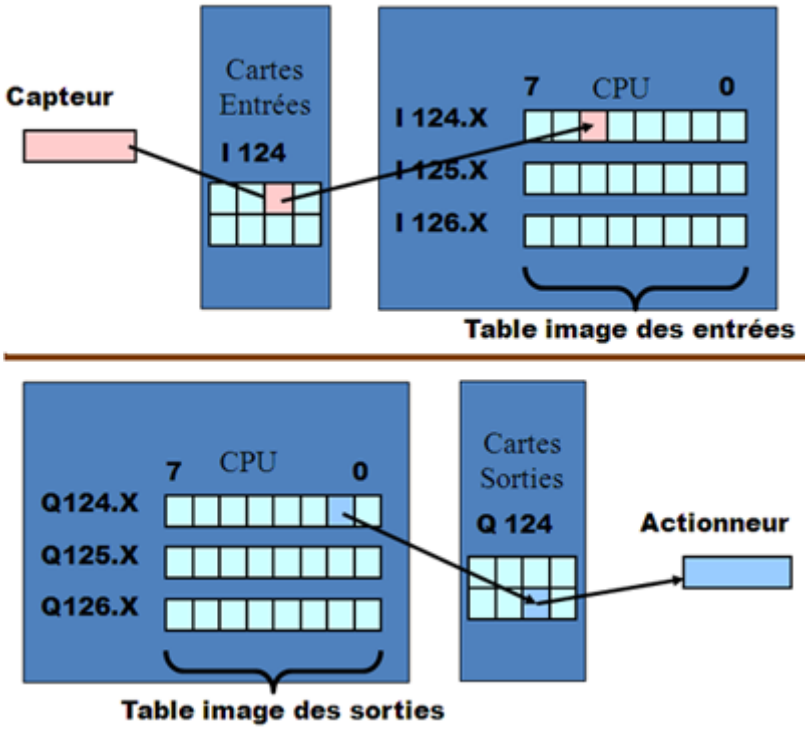


Figure 4.8 : Les interfaces d'entrées/sorties

6.3.1- Cartes d'entrées :

Elles sont destinées à recevoir l'information en provenance des capteurs et adapter le signal en le mettant en forme, en éliminant les parasites et en isolant électriquement l'unité de commande de la partie opérative.

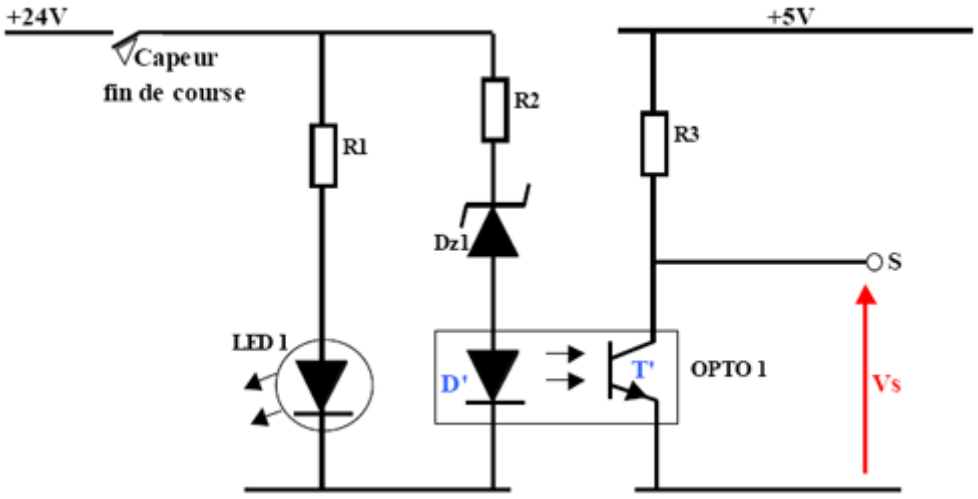


Figure 4.9: Exemple d'une carte d'entrées typique d'un API

6.3.1- Cartes de sorties:

Elles sont destinées à commander les pré-actionneurs et éléments des signalisations du système et adapter les niveaux de tensions de l'unité de commande à celle de la partie opérative du système en garantissant une isolation galvanique entre ces dernières

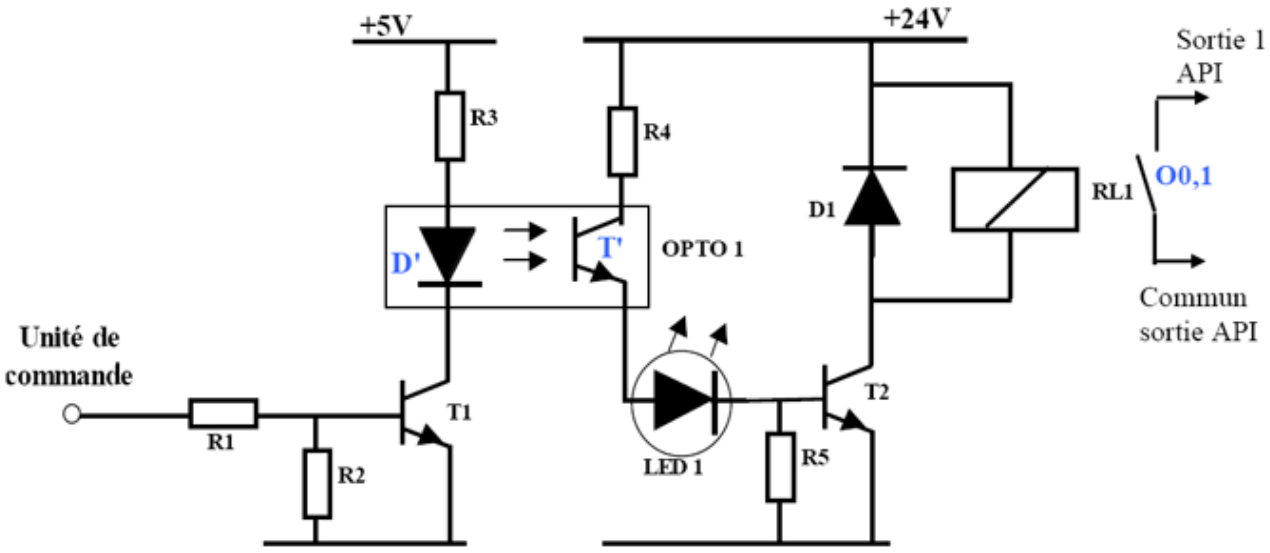


Figure 4.10: Exemple d'une carte de sortie typique d'un API

6.4- Exemple de cartes:

- **Cartes de comptage rapide** : elles permettent d'acquérir des informations de fréquences élevées incompatibles avec le temps de traitement de l'automate. (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. Les grandeurs analogique sont normalisées : 0-10V ou 4-20mA.
- **Cartes de régulation PID**
- **Cartes de pesage**
- **Cartes de communication** (RS485, Ethernet ...)
- **Cartes d'entrées / sorties déportées**

6.5- L'alimentation électrique :

Tous les automates actuels sont équipés d'une alimentation 240 V 50/60 Hz, 24 V DC. Les entrées sont en 24 V DC et une mise à la terre doit également être prévue.

7- Jeu d'instructions :

Le processeur peut exécuter un certain nombre d'opérations logiques; l'ensemble des instructions booléennes des instructions complémentaires de gestion de programme (saut, mémorisation, adressage ...) constitue un jeu d'instructions.

Chaque automate possède son propre jeux d'instructions. Mais par contre, les constructeurs proposent tous une interface logicielle de programmation répondant à la norme CEI1131-3. Cette norme définit cinq langages de programmation utilisables, qui sont :

- *Les langages graphiques :*
  - **LD** : **L**adder **D**iagram (Diagrammes échelle)
  - **FBD** : **F**unction **B**lock **D**iagram (Logigrammes)
  - **SFC** : **S**equential **F**unction **C**hart (Grafcet)
- *Les langages textuels :*
  - **IL** : **I**nstruction **L**ist (Liste d'instructions).
  - **ST** : **S**tructured **T**ext (Texte structuré).

Le langage à relais (Ladder Diagram) est basé sur un symbolisme très proche de celui utilisé pour les schémas de câblage classiques. Les symboles les plus utilisés sont donnés au tableau suivant :

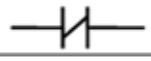
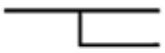
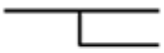
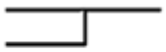
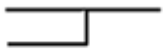
Fonction	**Symbole**	
	Européen	Américain
Contact ouvert au ropos	---o o---	
Contact fermé au ropos	---o o---	
Début de branchement		
Fin de branchement		
Affectation	---( )---	---( )

Figure 4.11: Symboles usuels en langages LD

8- Sécurité :

Les systèmes automatisés sont, par nature, source de nombreux dangers (tensions utilisées, déplacements mécaniques, jets de matière sous pression ...).
Placé au coeur du système automatisé, l'automate se doit d'être un élément fiable car un dysfonctionnement de celui-ci pourrait avoir de graves répercussions sur la sécurité des personnes, de plus les coûts de réparation et un arrêt de la production peuvent avoir de lourdes conséquences sur le plan financier.

Aussi, l'automate fait l'objet de nombreuses dispositions pour assurer la sécurité :
- **Contraintes extérieures** : l'automate est conçu pour supporter les différentes contraintes du monde industriel et à fait l'objet de nombreux tests normalisés.
- **Coupures d'alimentation** : l'automate est conçu pour supporter les coupures d'alimentation et permet, par programme, d'assurer un fonctionnement correct lors de la réalimentation (reprises à froid ou à chaud)
- **Mode RUN/STOP** : Seul un technicien peut mettre en marche ou arrêter un automate et la remise en marche se fait par une procédure d'initialisation (programmée)
- **Contrôles cycliques** :

  - Procédures d'autocontrôle des mémoires, de l'horloges, de la batterie, de la tensions d'alimentation et des entrées / sorties
  - Vérification du temps de scrutation à chaque cycle appelée ***Watchdog*** *(chien de garde),* et enclenchement d'une procédure d'alarme en cas de dépassement de celui-ci (réglé par l'utilisateur)

- **Visualisation** : Les automates offrent un écran de visualisation où l'on peut voir l'évolution des entrées / sorties

Les normes interdisent la gestion des arrêts d'urgence par l'automate ; celle-ci doit être réalisée en technologie câblée.

8- Réseaux d'automates

8.1- Principe

Avec le développement des systèmes automatisés et de l'électronique, la recherche de la baisse des coûts et la nécessité actuelle de pouvoir gérer au mieux la production et a partir du moment où tous les équipements sont de type informatique, il devient intéressant de les interconnecter à un mini-ordinateur ou à un automate de supervision (Figure 4.12).

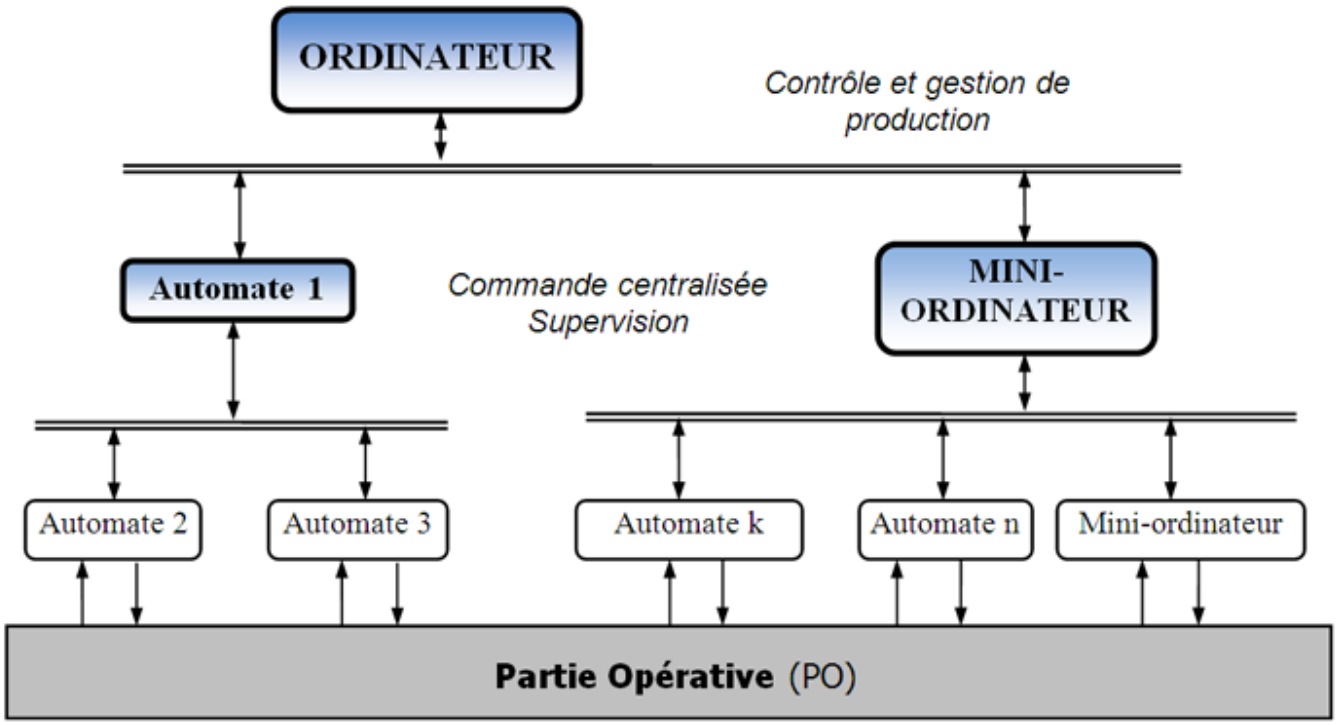


Figure 4.12: Exemple d'une structure de contrôle et gestion de production

L'interconnexion entre deux automates peut être réalisée très simplement en reliant une ou plusieurs sorties d'un automate à des entrées de l'autre et vice-versa (Figure 4.13).

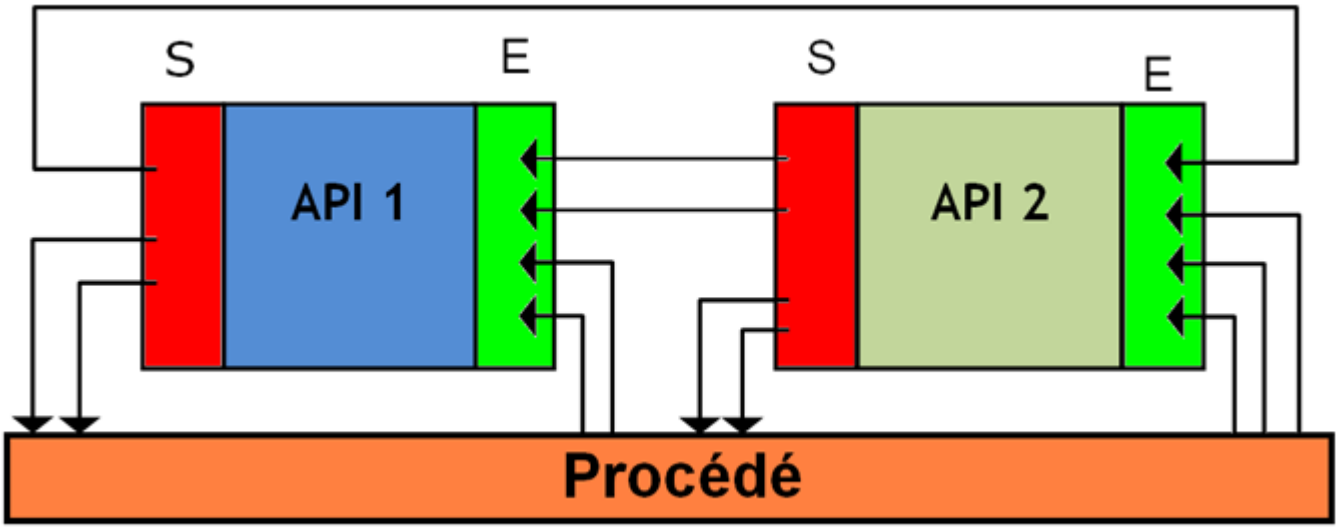


Figure 4.13: Interconnexion simple (Entrées/Sorties) entre deux automates (API)

Cette méthode ne permet pas de transférer directement des variables internes d'un automate sur l'autre, de sorte que celles-ci doivent être converties par programme en variables de sortie avant leur transfert. Elle devient coûteuse en nombre d'entrées/sorties mobilisé pour cet usage et lourde du point de vue du câblage, lorsque le nombre de variables qui doivent être échangées devient important.

8.2- Bus de terrain

Pour diminuer les coûts de câblage des entrées / sorties des automates, sont apparus les bus de terrains. L'utilisation de blocs d'entrées / sorties déportés à permis tout d'abord de répondre à cette exigence.

Les interfaces d'entrées/sorties sont déportées au plus près des capteurs. Avec le développement technologique, les capteurs, détecteurs ... sont devenus intelligents" et ont permis de se connecter directement à un bus.

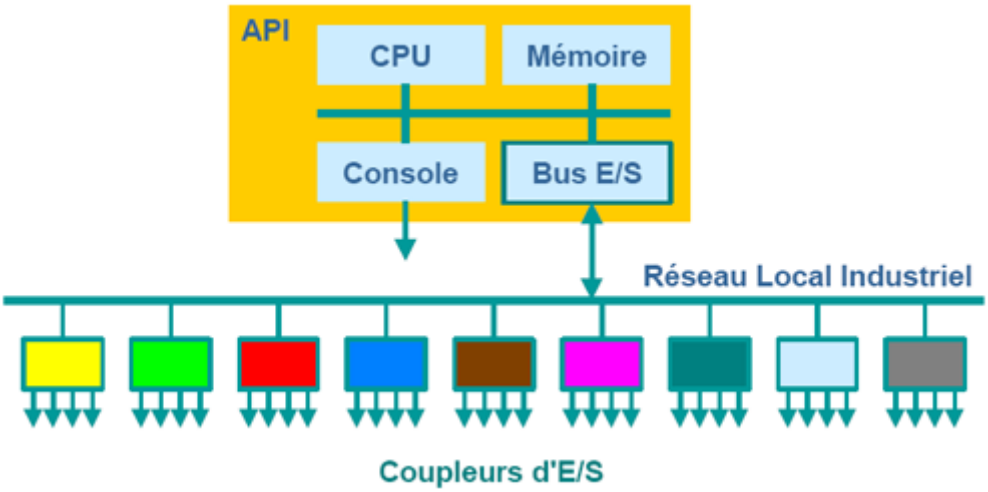


Figure 4.14: Interconnexion par entrées/sorties déportées

Plusieurs protocoles de communication et des standards sont apparus pour assurer le "multiplexage" de toutes les informations en provenance des capteurs / préactionneurs apr exemple le bus ASi (Actuators Sensors interface) est un bus de capteurs/actionneurs de type Maître / Esclave qui permet de raccorder 31 esclaves (capteurs ou préactionneurs) sur un câble spécifique (deux fils) transportant les données et la puissance.
Ce bus est totalement standardisé et permet d'utiliser des technologies de plusieurs constructeurs

Avantages des bus de terrain :

- Réduction des coûts de câblage et possibilité de réutiliser le matériel existant
- Réduction des coûts de maintenance

Inconvénients des bus de terrain :

- Taille du réseau limitée

- Latence dans les applications à temps critique
- Coût global

8.3- Différents types de réseaux d'automates :

8.3.1- Réseau en étoile :

Un centre de traitement commun échange avec chacune des autres stations. Deux stations ne peuvent pas échanger directement entre elles (Figure 4.15). Exemple le réseau de terrain BITBUS de la société INTEL

Avantages :

- Grande vitesse d'échange.
- Différent types de supports de transmission.
- Pas de gestion d'accès au support.

Inconvénients :

- Coût global élevé.
- Evolutions limitées.
- Tout repose sur la station centrale.

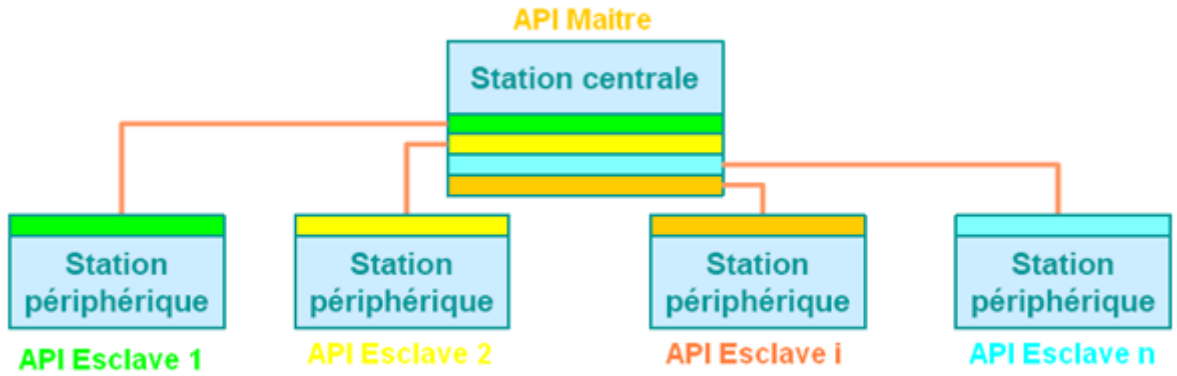


Figure 4.15: Interconnexion par entrées/sorties déportées

8.3.2- Réseau en anneau :

Chaque station peut communiquer avec sa voisine. Cette solution est intéressante lorsqu'une station doit recevoir des informations de la station précédente ou en transmettre vers la suivante (Figure 4.16).

Figure 4.16: Topologie Anneau

Avantages :

- Signal régénéré donc fiable.
- Contrôle facile des échanges (le message revient à l'émetteur).

Inconvénients :

- Chaque station est bloquante.
- Une extension interrompe momentanément le réseau.

8.3.3- Réseau hiérarchisé :

C'est la forme de réseaux la plus performante. Elle offre une grande souplesse d'utilisation, les informations pouvant circuler entre-stations d'un même niveau ou circuler de la station la plus évoluée (en général un calculateur) vers la plus simple, et réciproquement (Figure 4.17).

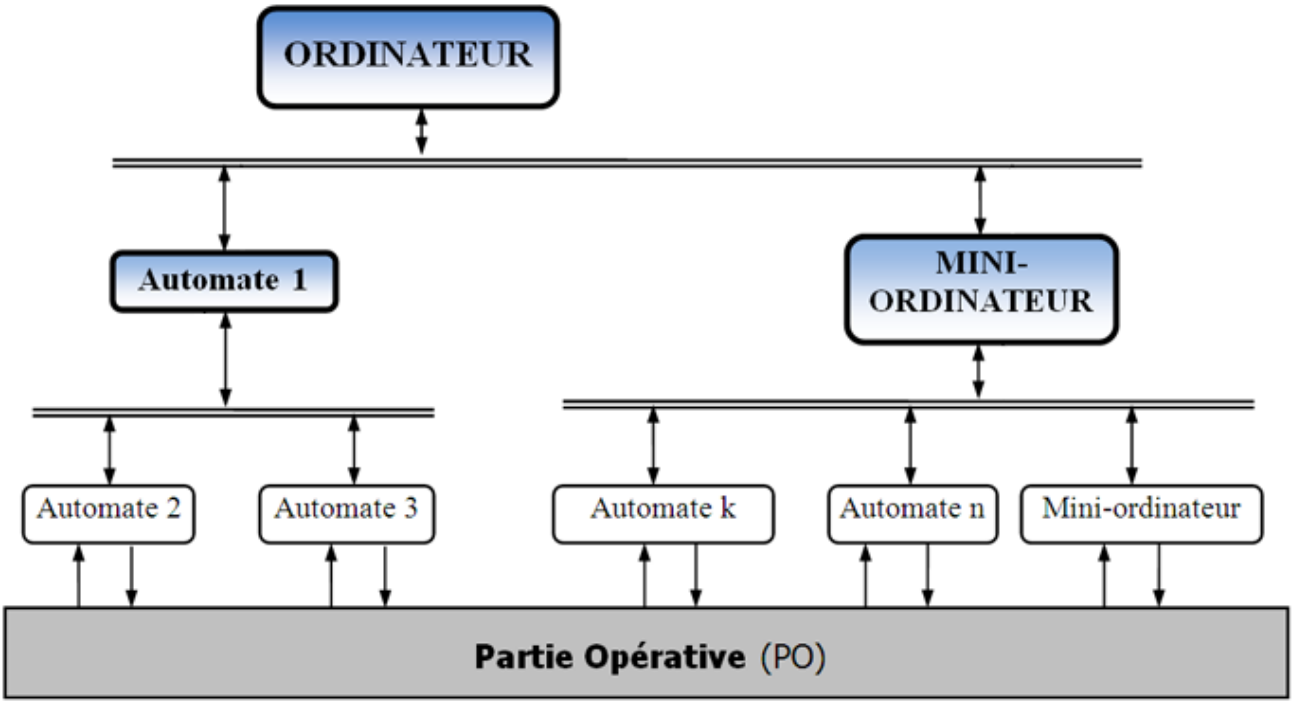


Figure 4.17: Réseau hiérarchisé

9- Critères de choix d'un automate

Le choix d'un automate programmable est généralement basé sur :
- Nombre d'entrées / sorties : le nombre de cartes peut avoir une incidence sur le nombre de racks dès que le nombre d'entrées / sorties nécessaires devient élevé.
- Type de processeur : la taille mémoire, la vitesse de traitement et les fonctions spéciales offertes par le processeur permettront le choix dans la gamme souvent très étendue.

- Fonctions ou modules spéciaux : certaines cartes (commande d'axe, pesage ...) permettront de "soulager" le processeur et devront offrir les caractéristiques souhaitées (résolution, ...).
- Fonctions de communication : l'automate doit pouvoir communiquer avec les autres systèmes de commande (API, supervision ...) et offrir des possibilités de communication avec des standards normalisés (Profibus ...).

10- Mise en œuvre et diagnostic d'un API :

10.1: Vérification du fonctionnement

Lors de sa première mise en œuvre il faut réaliser la mise au point du système.

- Prendre connaissance du système (dossier technique, des GRAFCETS et du GEMMA, affectation des entrées / sorties, Les schémas de commande et de puissance des entrées et des sorties).
- Lancer l'exécution du programme (RUN ou MARCHE)
- Visualiser l'état des GRAFCET, des variables...

Il existe deux façons de vérifier le fonctionnement :

- En simulation (sans Partie Opérative).
- En condition réelle (avec Partie Opérative).

Simulation sans Partie opérative	**Simulation avec Partie opérative** (*Conditions réelles*)
Le fonctionnement sera vérifié en simulant le comportement de la Partie Opérative, c'est à dire l'état des capteurs, en validant uniquement des entrées. - Valider les entrées correspondant à l'état initial (position) de la Partie Opérative. - Valider les entrées correspondant aux conditions demarche du cycle. - Vérifier l'évolution des grafcets (étapes actives). - Vérifier les ordres émis (Leds de sorties). - Modifier l'état des entrées en fonction des ordres émis (état transitoire de la P.O.). - Modifier l'état des entrées en fonction des ordres émis (état final de la	Le fonctionnement sera vérifié en suivant le comportement de la P.O. - Positionner la P.O. dans sa position initiale. - Valider les conditions de marche du cycle. - Vérifier l'évolution des grafcets et le comportement de la P.O. Toutes les évolutions du GEMMA et des grafcets doivent être vérifiées.

P.O.). Toutes les évolutions du GEMMA et des grafcets doivent être vérifiées.	

10.2 : Recherche des dysfonctionnements

Un dysfonctionnement peut avoir pour origine :

- Un composant mécanique défaillant (préactionneur, actionneur, détecteur,...).
- Un câblage incorrect ou défaillant (entrées, sorties).
- Un composant électrique ou électronique défectueux (interface d'entrée ou de sortie).
- Une erreur de programmation (affectation d'entrées-sorties, ou d'écriture).
- Un système non initialisé (étape, conditions initiales...).
- .....

Méthode de recherche de pannes:

Méthode de recherche de pannes:

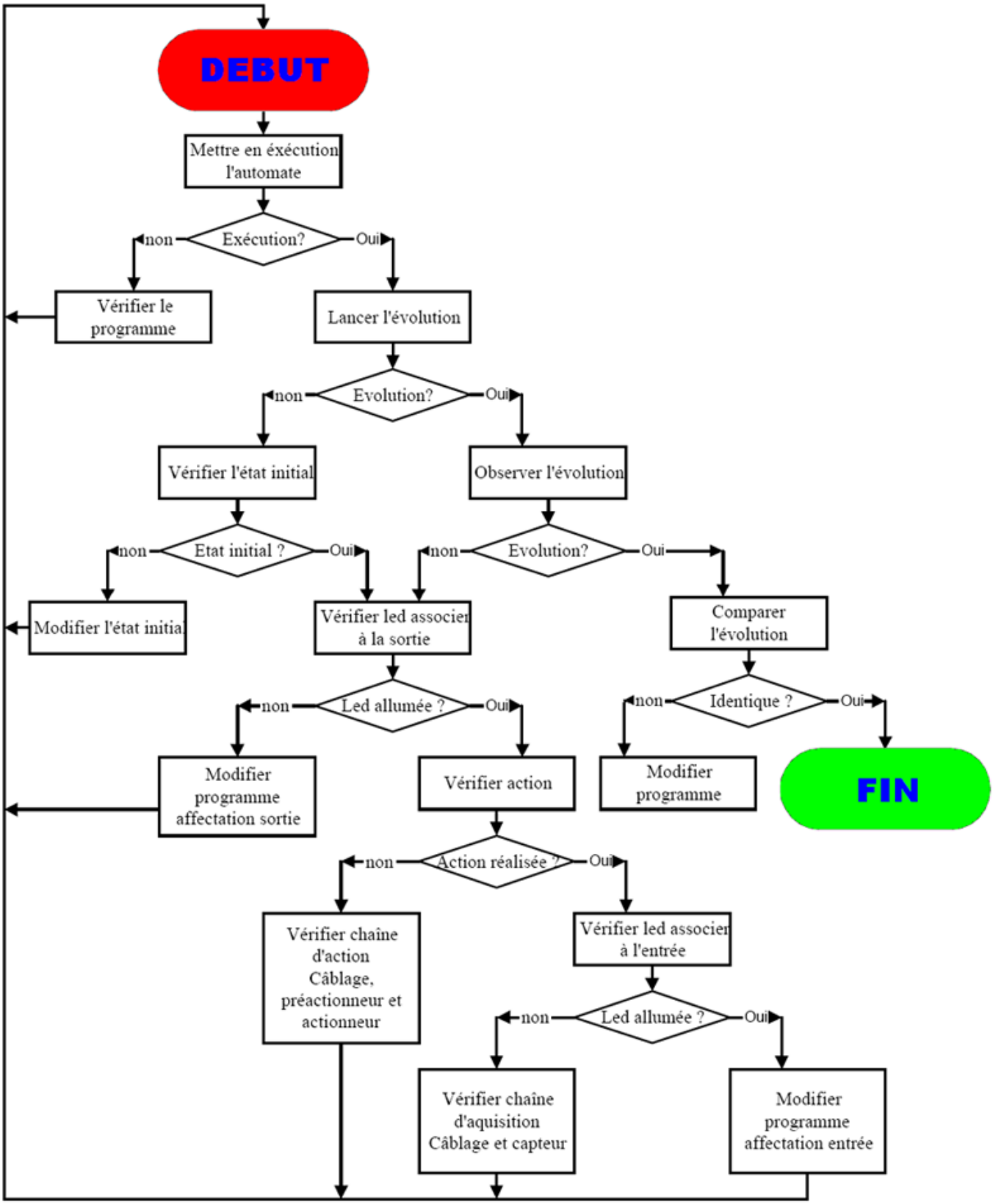
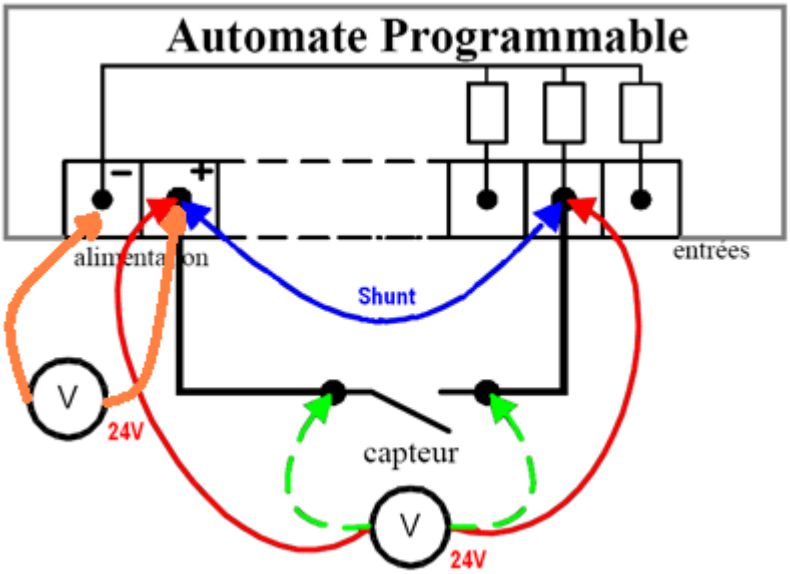
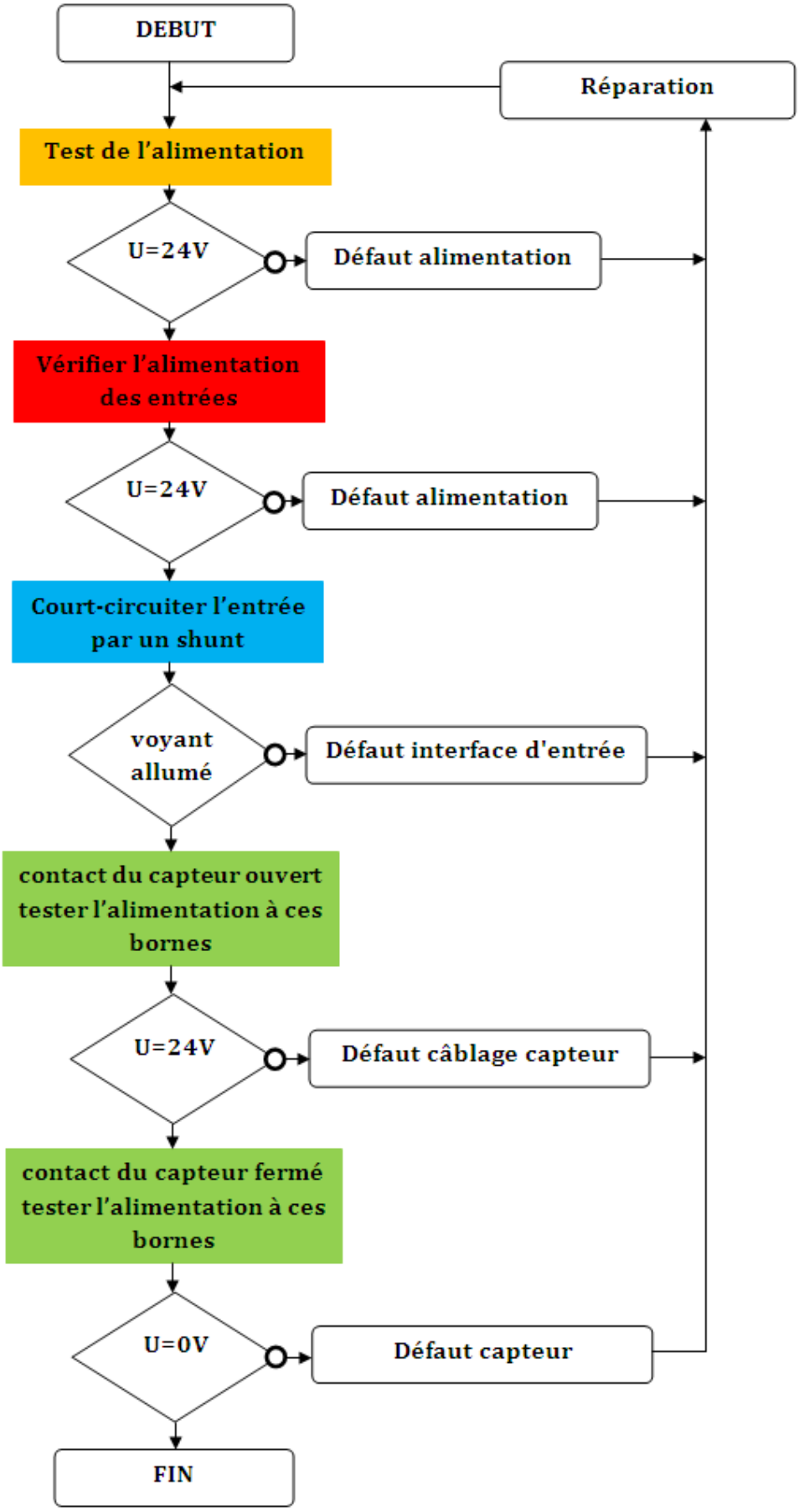


Figure 4.17: Méthode de recherche de pannes et Diagnostic d'un API

Méthode de vérification du câblage d'une entrée à masse commune :

Cette vérification se réalise à l'aide d'un voltmètre-ohmètre et d'un shunt (morceau de fil électrique).




DEBUT
Réparation
Test de l'alimentation
U=24V
Défaut alimentation
Vérifier l'alimentation des entrées
U=24V
Défaut alimentation
Court-circuiter l'entrée par un shunt
voyant allumé
Défaut interface d'entrée
contact du capteur ouvert tester l'alimentation à ces bornes
U=24V
Défaut câblage capteur
contact du capteur fermé tester l'alimentation à ces bornes
U=0V
Défaut capteur
FIN

11- Principaux automates programmables industriels :

La programmation de ces automates se fait soit à partir de leur propre console, soit à partir du logiciel de programmation propre à la marque.

OMRON :

- **CQM1 – CPU 11/21/41**
  - E - 192 Entrées/Sorties (à relais, à triac, à transistors ou TTL) ;
  - 32 K RAM data on Board ;
  - structure multifonction ;
  - structuration multitâche ;
  - SYSWIN 3.1, 3.2 … 3.4 et CX_Programmer (Littéral, Ladder) ;
  - comunication sur RS 232 – C ;
  - programmation sur IBM PC/PS.

TELEMECANIQUE :

- **TSX 17/20 :**
  - Nombre d'entrées et de sorties variable : 20 à 160 E/S.
  - microprocesseur 8031.
  - langage de programmation PL7.2.
- **TSX 67.20 :** La compacité d'un automate haut de gamme, à E/S déportables par fibre optique:
  - 1024 E/S en six bacs de huit modules;
  - extension de bacs à distance par fibre optique à 2000 m;
  - 16 coupleurs intelligents;
  - 24 K RAM data on Board;
  - 32 K RAM / EPROM cartouche utilisateur;
  - structure multifonctions;
  - structuration multitâche;
  - langage PL7.3 (Grafcet, Littéral, Ladder);
  - programmation sur IBM PC/PS.
- **FESTO :** Architecture modulaire : carte de base; carte processeur; carte de mémorisation; carte E/S.
  - FPC 202 :
  - 16 entrées 24 V DC;
  - 16 sorties 24 V DC - 1 A;
  - 8 K RAM, 8 K EPROM;
  - interface série, 20 mA boucle de courant pour imprimante;
  - console de programmation externe : console ou IBM PC;
  - programmation : grafcet, langage Festo, schéma à relais.

SIEMENS:

- **S7 – 200.**

- 64 entrées 24 V DC;
- 64 sorties 24 V DC - 1 A ;
- 8 Entrées anlogiques AEW0
- AEW14 ; - 8 Sorties anlogiques AAW0
- AAW6 ; - interface série,
- console de programmation externe : PG 702;

programmation STEP7: schéma à relais , Ladder.

CHAP.V AUTOMATION ET REGULATION

VOIR LES ANNEXES DU SYLLABUS

CHAP.VI INTRODUCTION AUX RESEAUX D'AUTOMATES ET A L'INTERNET INDUSTRIEL

VI.1 INTRODUCTION

L'histoire des réseaux locaux industriels remonte à la fin des années 70, avec l'apparition des équipements industriels numériques intelligents et des réseaux informatique de bureaux. Leur apparition est venue répondre,

- premièrement, à la demande croissante de productivité dans le domaine industriel par l'automatisation de la communication entre les différents équipements industriels (de contrôle et de mesure) de façon à éliminer les pertes de temps et les risques d'erreurs dus aux interventions humaines,

- deuxièmement, au besoin d'interconnexion des équipements industriels informatisés hétérogènes qui ont été introduits dans le milieu industriel d'une manière anarchique, c'est-à-dire en résolvant chaque problème à part sans prendre en compte l'intégrité de tout le système industriel.

Les réseaux locaux industriels ont été donc introduits petit à petit dans les systèmes automatisés, à des stades divers selon les domaines d'application. Ils sont nés avec le développement de l'électronique et des matériels numériques programmables. L'apparition des régulateurs numériques et des automates programmables a conduit les offreurs à mettre sur le marché des réseaux pour les interconnecter et rapatrier à moindre coût de câblage les informations nécessaires à la conduite par les opérateurs dans les salles de commande

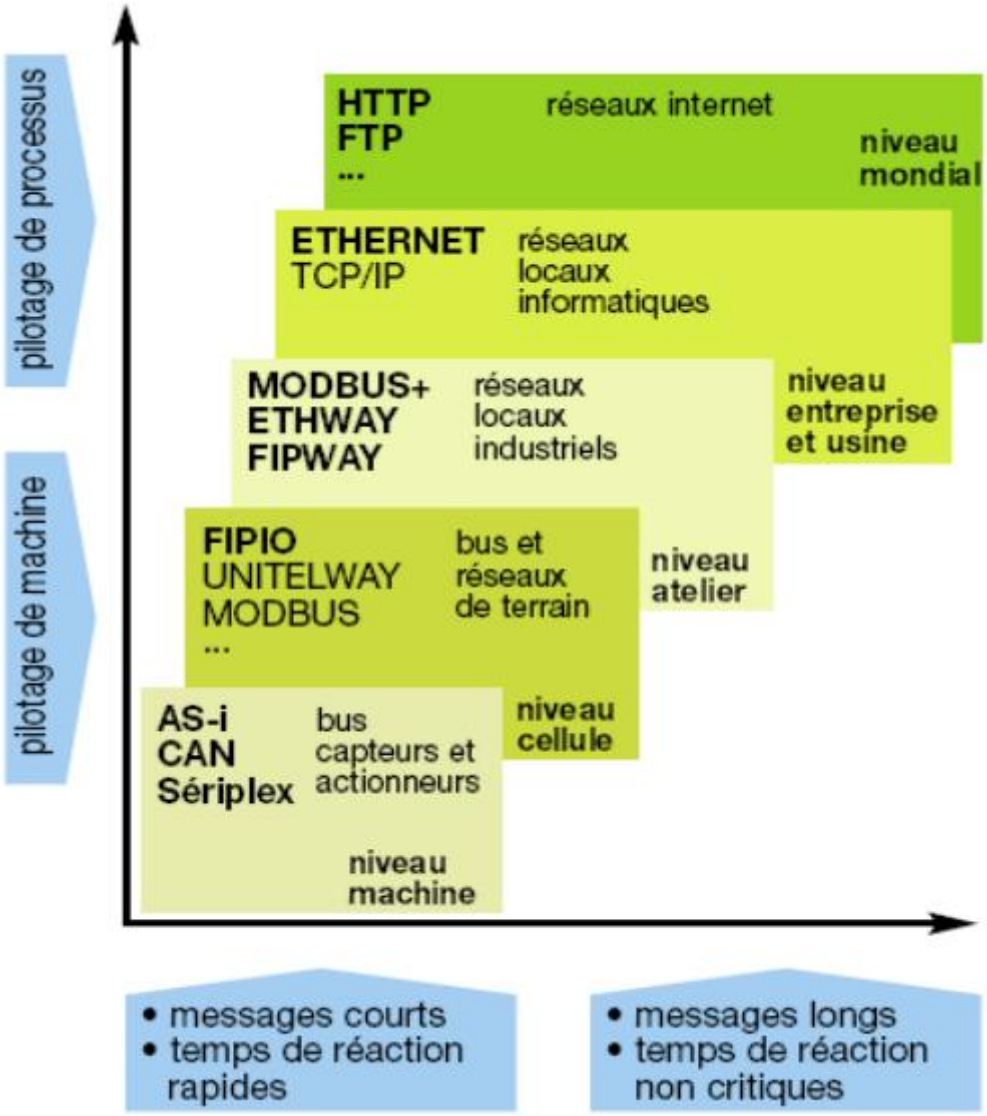
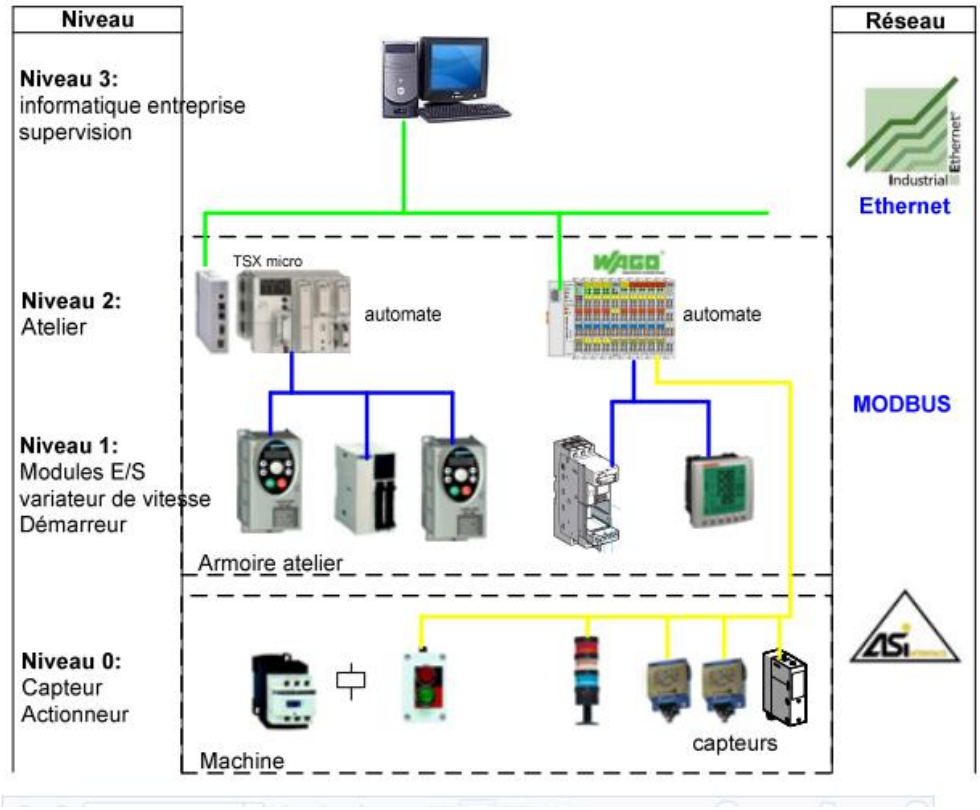


Figure 2 NIVEAUX ET CARACTERISTIQUES RESEAUX DES ORGANES D'AUTOMATISMES ET DE CONTROLE

Structure d'une installation automatisée:

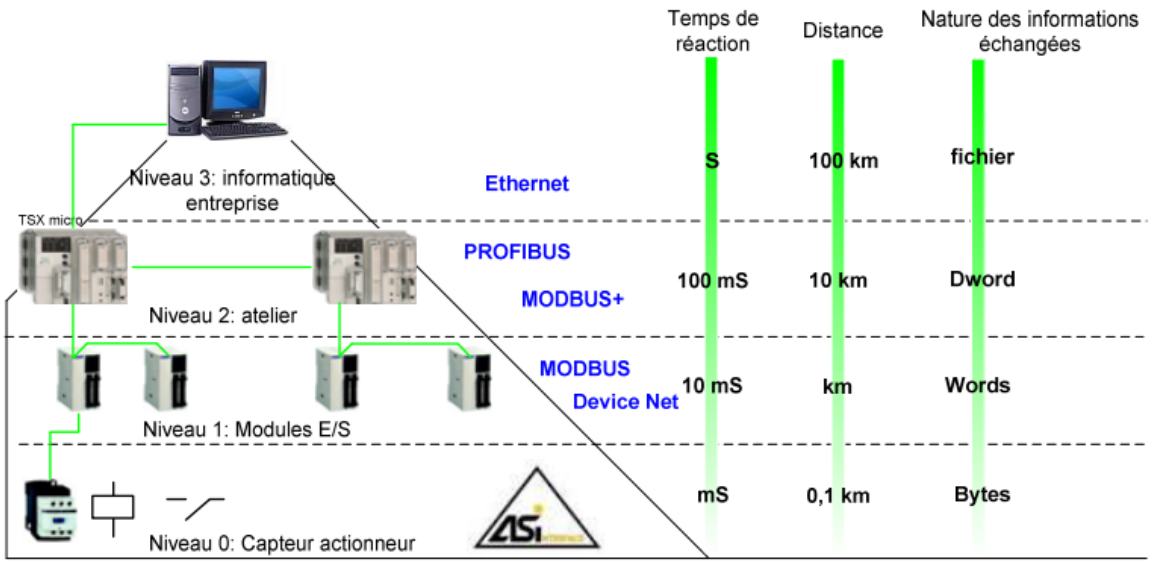


L'environnement industriel englobe tous les équipements qui participent à la chaîne de production que ce soit pour la fabrication, le contrôle ou la maintenance. Ces équipements peuvent être des :

- machines à outils,
- robots,
- contrôleurs à logique programmée (PLC),
- capteurs,
- actionneurs,
- stations de supervision,
- …etc.

Ainsi que les moyens nécessaires à leur interconnexion tel que les câbles, les passerelles, les routeurs,… etc.

Dans les environnements industriels d'aujourd'hui, la plus part des tâches se font d'une façon automatique ce qui maximise les taux de production, garantit une meilleure sécurité du personnel, et augmente la rentabilité de l'industrie en générale.



L'automatisation de l'industrie permet d'atteindre des objectifs très intéressants :

- Commercialiser rapidement les nouveaux produits,
- Réagir à court terme et avec souplesse aux exigences du marché,
- Réduire le temps de mise sur marché,
- Produire de manière efficace et économique,
- Exploiter de façon optimale les capacités des machines,
- Minimiser les temps improductifs,
- …etc.

De telles objectifs ne sont parfaitement atteints que si toutes les machines d'une installation sont complètement automatisées et fonctionnent en parfaite interaction, ce qui peut-être atteint par :

- l'utilisation de machines automatisées,
- L'utilisation des PLC (contrôleurs à logique programmée) qui permettent d'automatiser l'utilisation de certains équipement non automatisés.
- L'utilisation des robots pour automatiser les tâches « intelligentes « telles que la soudure, le montage, assemblage, … etc.

- L'utilisation des réseaux informatiques industriels pour garantir l'interopérabilité des équipements automatisés.

Les installations industrielles, permettent de mettre en œuvre un grand nombre de fonctions qui sont largement interdépendantes et qui peuvent être organisé hiérarchiquement en quatre niveaux d'abstraction :

1. **Le niveau Entreprise (niveau 3)**

   On trouve à ce niveau des services de gestion tel que :

   La gestion commerciale,

   La gestion du personnel,

   La gestion financière, …

2. **Le niveau usine (niveau 2)**

   Ce niveau englobe des tâches de gestion de la production tel que :

   La GPAO : gestion de production assistée par ordinateur,

   La CFAO : Contrôle de fabrication assisté par ordinateur,

   La CAO : Conception assisté par ordinateur,

   Des services de transport,

   Le contrôle de qualité,….

3. **Le niveau atelier ou cellule (niveau 1)**

Contient plusieurs îlots de fabrication, de vision, de supervision, des robots, des automates, …etc.

4. **Le niveau terrain (niveau 0)**

C'est le niveau le plus bas, qui contient les équipements de fabrication proprement dite tel que :

- Les machines automatisées de production qui sont des machines programmables qui peuvent selon le programme chargé exécuter des tâches complexes sans intervention humaine,
- Les capteurs qui sont des instruments de mesure qui peuvent fournir à des machines intelligentes (tel que les ordinateurs ou les contrôleurs) des informations telle que la température, la pression, la tension, la couleur, les variations, …etc
- Les actionneurs qui sont des instruments qui peuvent être activés par des machines intelligentes tel que les vannes, les interrupteurs, les alarmes,…etc

Parmi toutes les composantes d'une installation industrielle, les réseaux de communication jouent un rôle central dans les solutions automatisées, ils permettent essentiellement :

- un flux d'information continu depuis le niveau capteurs/actionneur jusqu'au niveau gestion de l'entreprise
- la disponibilité des informations en tout point de l'installation
- échange rapide des informations entre les différentes parties de l'installation
- un diagnostic, et une maintenance efficaces
- des fonction de sécurités intégrées empêchant les accès non autorisés
- … etc.

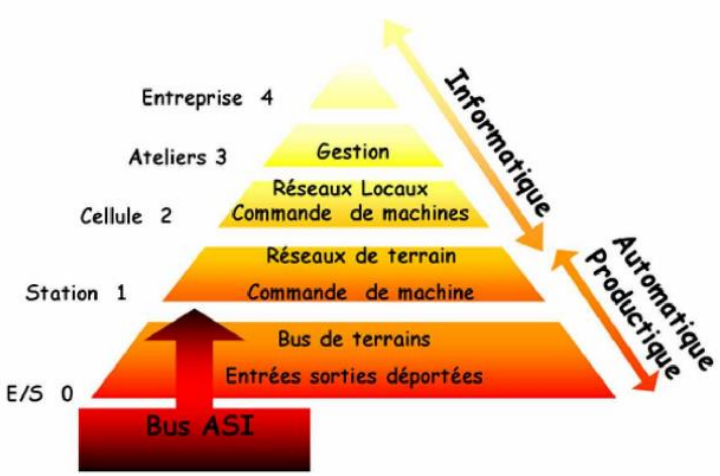
VI.2 LE BUS DES CAPTEURS ET DES ACTIONNEURS : NIVEAU 0

VI.2.1 PRESENTATION DU BUS DE CAPTEURS ET ACTIONNEURS (ASI)

Les bus de capteurs et des actionneurs sont connus surtout avec leur appellation en anglais Actuator Sensor Interface (ASI) :

En 1991, un groupe de 11 sociétés spécialisées dans les capteurs/actionneurs a défini un **bus de terrain** afin de pouvoir interconnecter facilement les capteurs et les actionneurs, ainsi est né le concept ASI (**A**ctuator **S**ensor **I**nterface).

Depuis 1992 une association ASI, qui compte à ce jour 50 membres, est chargée de coordonner, certifier et standardiser le bus.

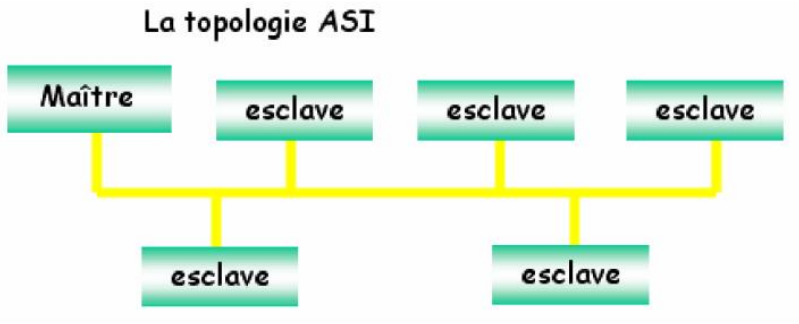


Caractéristique du bus ASI :

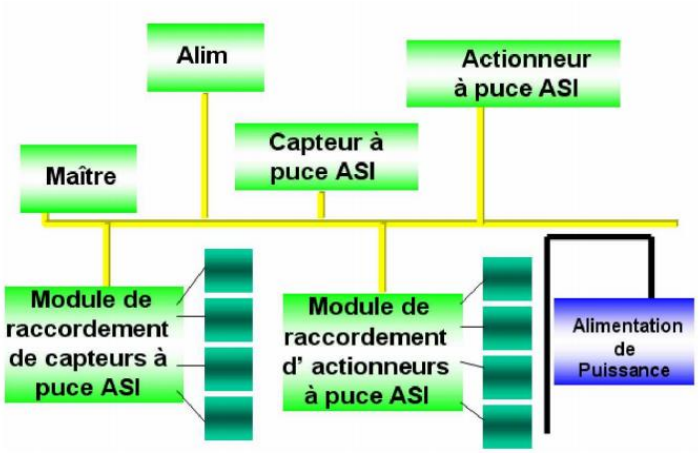
- Nature d'information échangé : bytes ;
- Débit : quelques bits / seconde ;
- Distance : maximum 100 m ;
- Temps de réaction : 10^{-3} secondes ;
- Fabriquant: AS-I; PROFIBUS PA; CAN; SERIPLEX; …..

VI.2.2 ARCHITECTURE DU BUS DE CAPTEURS ET ACTIONNEURS (ASI)

ASI permet d'interconnecter des modules Esclaves à un Maître (API).

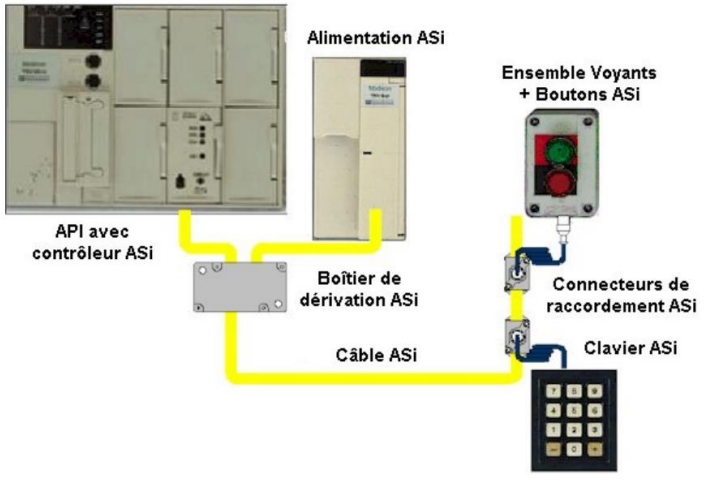


Pour 1seul maître, on peut connecter jusqu'à 31 esclaves et pour chacun d'eux on peut piloter 4 entrées et ou 4 sorties par esclave en disposant, en plus, de 4 bits de paramétrage. Les modules d'E/S permettent de recevoir 1 à 8 capteurs/actionneurs soit actifs soit passifs (intégrant un chip ASI).

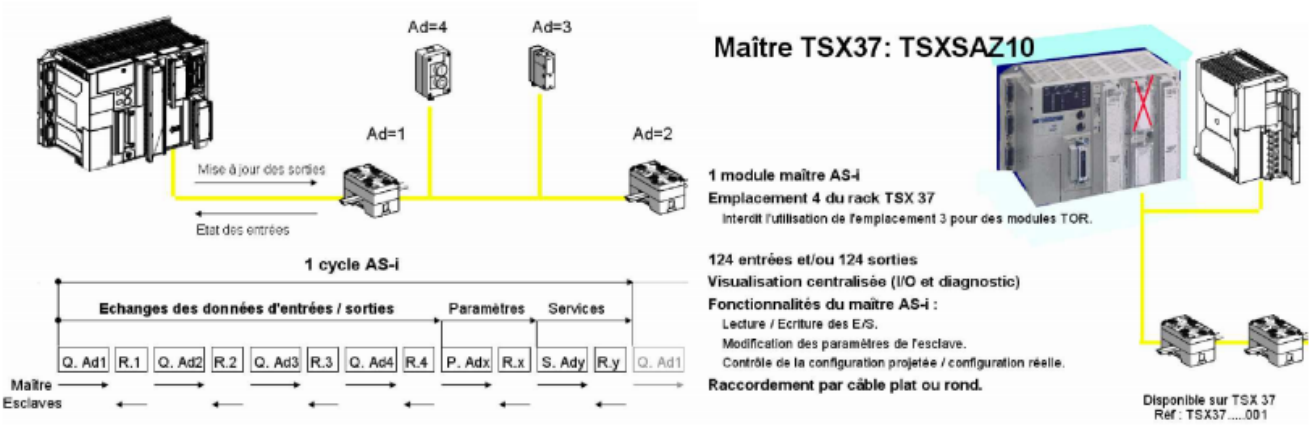


Cela correspond ainsi à un maximum de 248 capteurs ou actionneurs TOR. La topologie est totalement libre (étoile, bus, arbre), mais ne doit pas dépasser 100m.

VI.2.2 MISE EN OEUVRE DU BUS DE CAPTEURS ET ACTIONNEURS (ASI)



Un exemple : l'automate TSX37, équipé d'un module maître ASI, échange des données avec les périphériques esclaves repérés par leur adresse.



On retrouve ci-dessus les équipements minima :

- Un automate avec le coupleur maître ASI
- Un module d'alimentation
- Des périphériques

Les services du coupleur maître TSXSAZ10 :

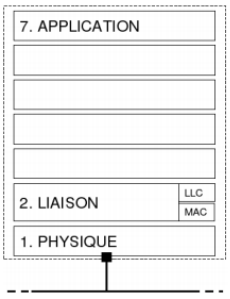
- Paramétrage des esclaves (Catalogue de produits =S=)
- Adressage géographique des E/S dans le programme
- Téléchargement, contrôle et gestion des profils
- Visualisation des voies E/S et des équipements (présents, absents ou défectueux)
- Adressage automatique des équipements au fur et à mesure de leur connexion
- Réadressage automatique des équipements remplacés

VI.3 BUS ET RESEAUX DE TERRAIN

VI.3.1 BUS DE TERRAIN (FIELDBUS) : Niveau 1

Le bus de terrain ou Fieldbus est un terme générique qui désigne un support de communication numérique qui remplace progressivement dans l'industrie le concept de transmission analogique appelé couple 4-20 mA par un bus série numérique bidirectionnel susceptible de relier les dispositifs indépendants sur terrain tels que les périphéries d'automatismes (variateur de vitesse, démarreur, robot…), le coupleur (module entrée-sortie), les capteurs, les actionneurs et les contrôleurs.

Un bus de terrain est un système de communication numérique dédié qui respecte le modèle d'interconnexion des systèmes ouverts (OSI) de l'Organisation de Standardisation Internationale (ISO 7498 – 1983).



Un bus de terrain est basé sur la restriction du <u>modèle OSI à 3 couches</u> :

- *Couche Application*
- *Couche Liaison*
- *Couche Physique*

*C'est un réseau **bidirectionnel, sériel**, **multibranche** (multidrop), reliant différents types d'équipements : E/S déportées, Capteur / Actionneur, Automate programmable (API), CNC, Calculateur, PC Industriel, ...*

• Terrain : espace géographique limité.

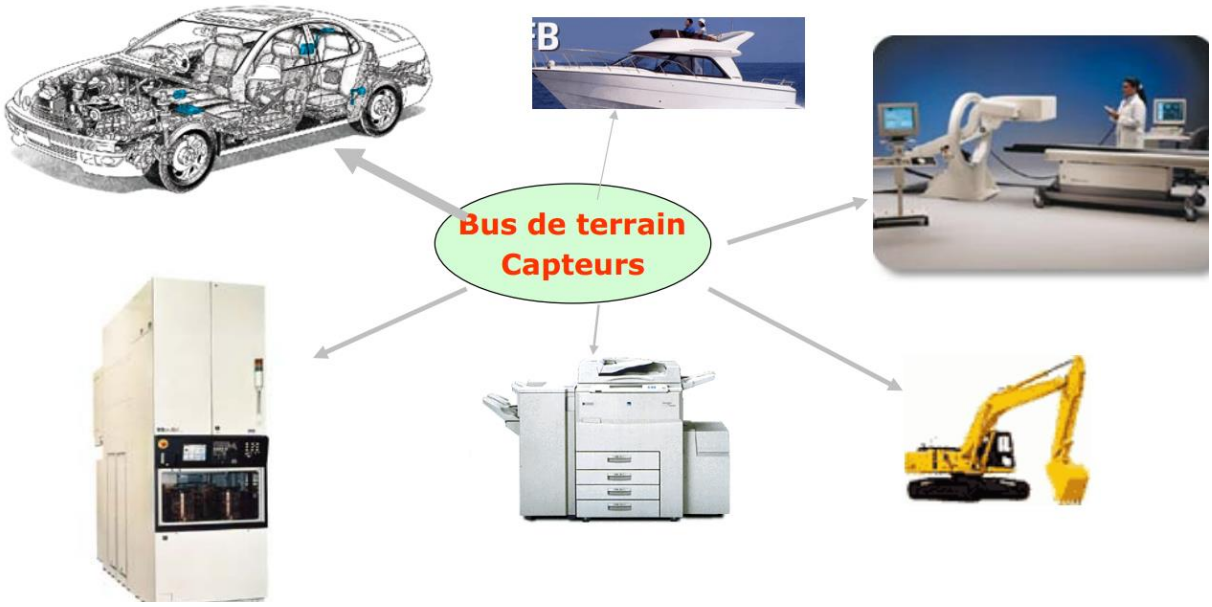
• Bus : ensemble de conducteurs commun à plusieurs circuits permettant d'échanger des données. Les échanges sont régis par un protocole.

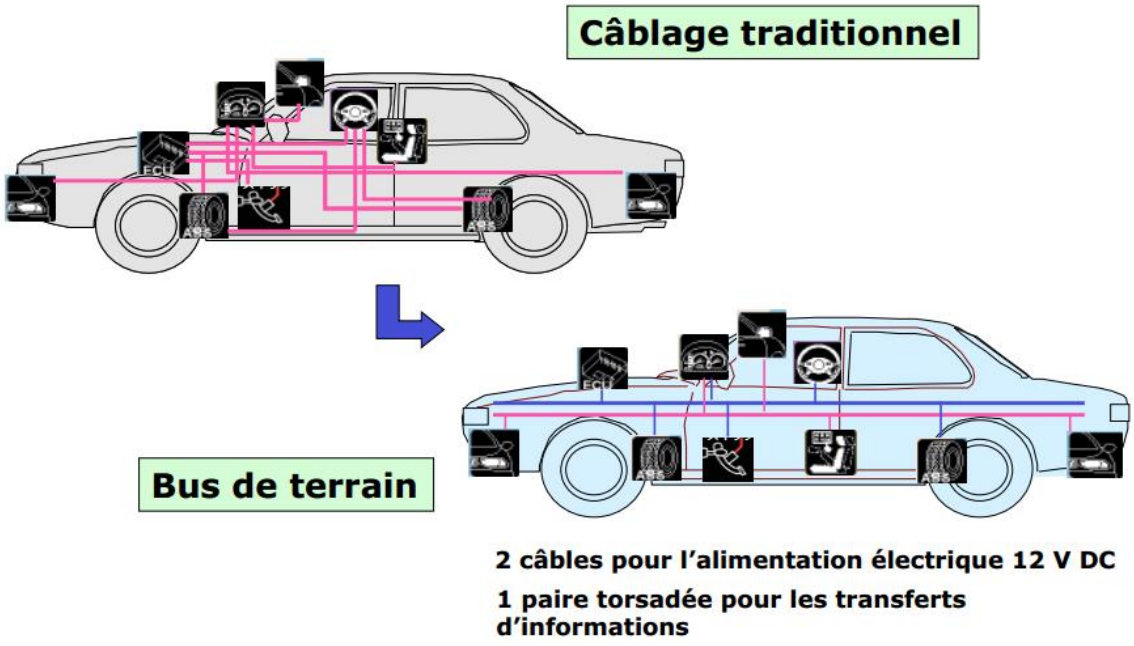
• Réseau : bus ou ensemble de bus répartis sur un terrain.

• Un bus de terrain est donc un système de communication entre plusieurs ensembles communiquants dans une zone géographique limitée (capteurs, calculateurs, automates, actionneurs, ...).

Caractéristique du bus terrain :

- Nature d'information échangé : mots (word) ;
- Débit : quelques octets / seconde ;
- Distance : maximum 1000 m ;
- Temps de réaction : 10^{-2} secondes ;
- Fabriquant: FIPIO; PROFIBUS DP; MODBUS ; UNTEL WAY ;….


Bus de terrain
Capteurs


Câblage traditionnel
Bus de terrain
2 câbles pour l'alimentation électrique 12 V DC
1 paire torsadée pour les transferts d'informations

VI.3.2 RESEAU DE TERRAIN (DEVICE BUS) : Niveau 2

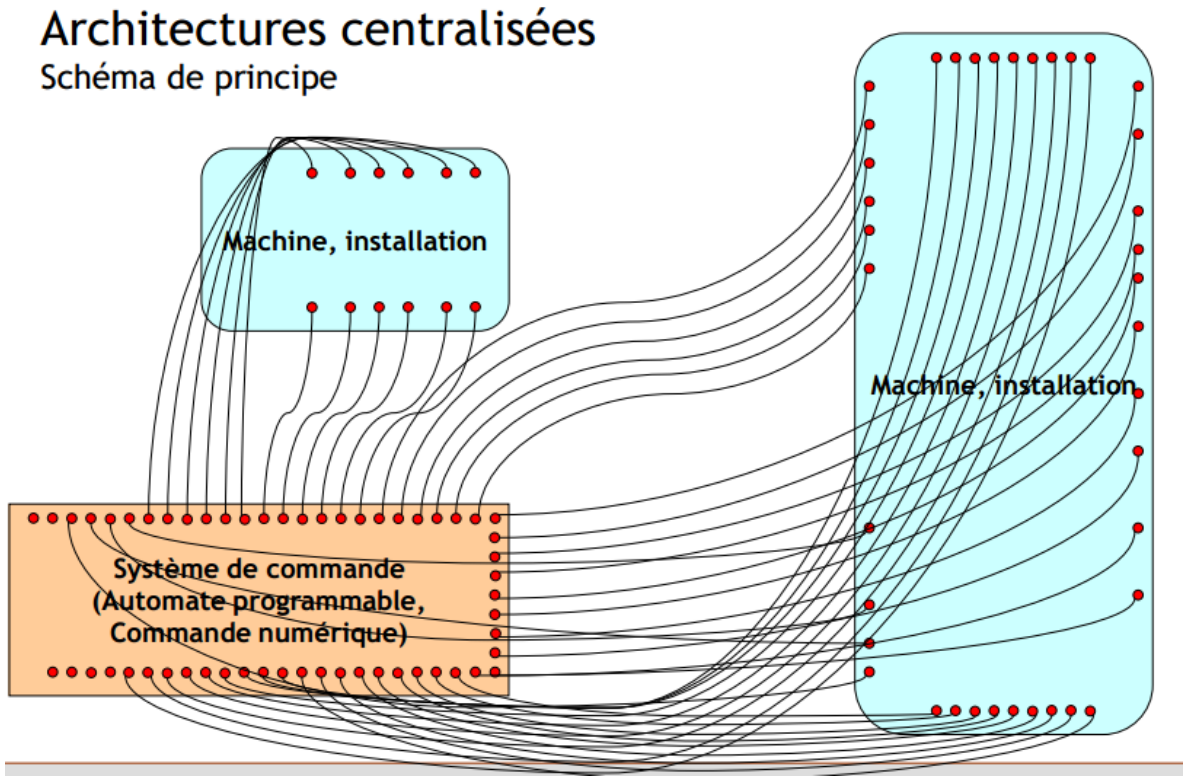
VI.3.2.1 DEFINITION DU RESEAU DE TERRAIN (BUS EQUIPEMENT)

Le bus d'équipement (*Device Bus*) permet de raccorder les organes de commande tels que les APIs, PCs, CNNs (Contrôle Commande Numérique), aux blocs de capteurs et actionneurs (modules déportés), aux lecteurs code barre, aux cellules photoélectriques, aux variateurs de vitesse, etc. La position de ce type de bus le situe entre les bus capteurs/actionneurs et les réseaux de terrain dits généralistes.

VI.3.2.2 CARACTERISTIQUE DU RESEAU DE TERRAIN :

- Nature d'information échangé : messages ;
- Débit : quelques kbits / seconde ;
- Distance : maximum 10000 m ;
- Temps de réaction : 10^{-1} secondes ;
    - Fabriquant: FIPWAY; PROFIBUS DP; MODBUS +; …..

VI.3.2.3 ARCHITECTURES CENTRALISEES ET DECENTRALISES DE RESEAU DE TERRAIN

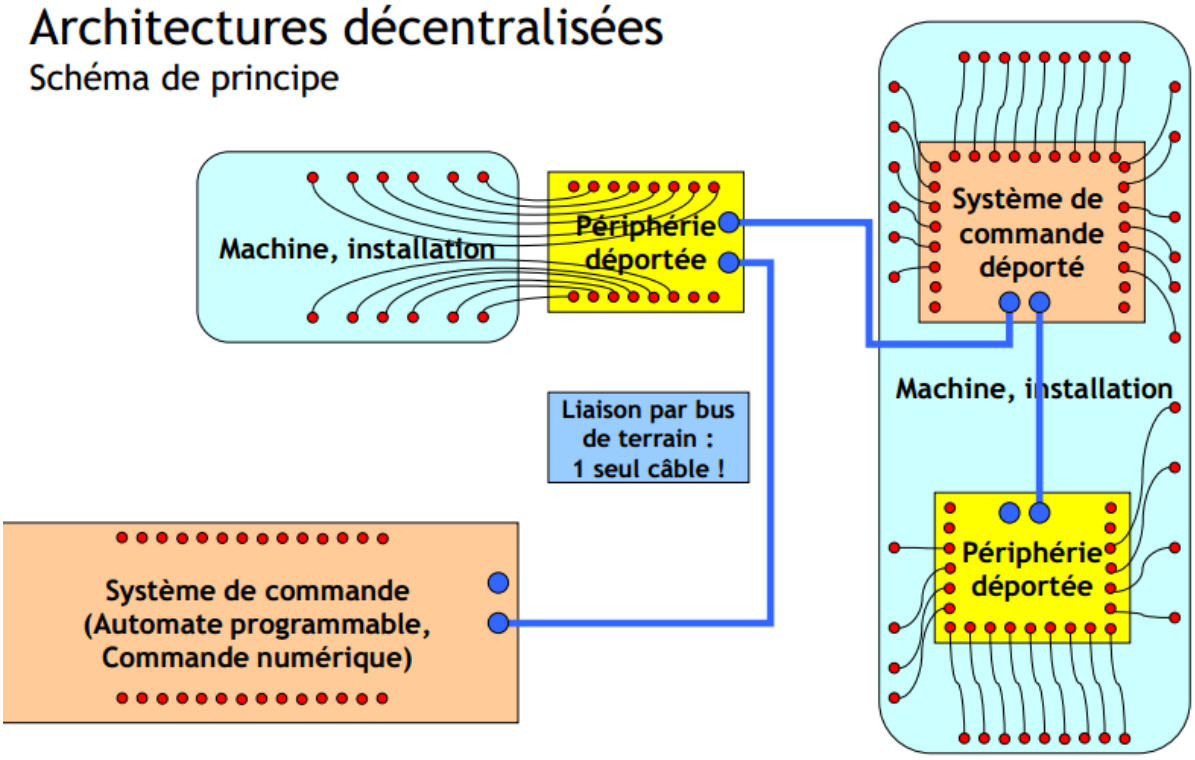


Architectures centralisées

Mise en œuvre pratique

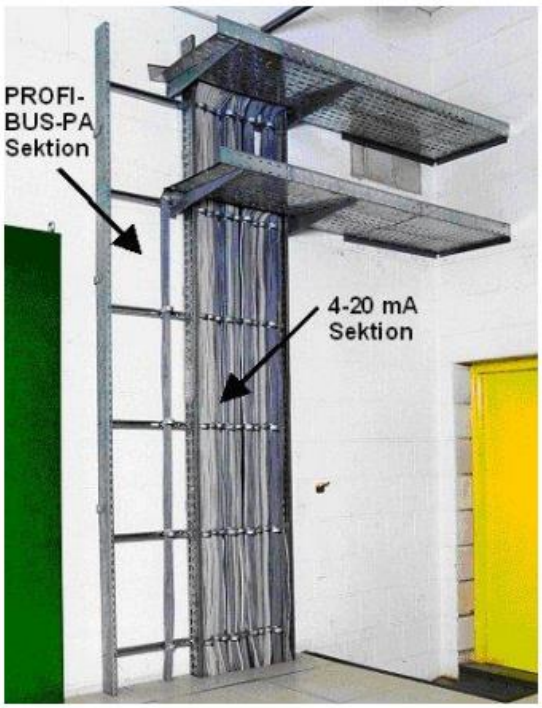
Architectures décentralisées

Schéma de principe



Architectures centralisées et décentralisées

Comparaison en pratique



VI.4 RESEAUX LOCAUX INDUSTRIELS : NIVEAU 3 (SYSTEME NUMERIQUE DE CONTROLE ET COMMANDE SNCC)

VI.4.1 EVOLUTION DE LA REGULATION PNEUMATIQUE AU SNCC

Années 1970-1985

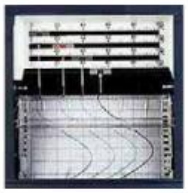
De la régulation pneumatique …

A la régulation électronique …

Puis à la régulation informatique!

Années 1970-1985

- Situation en place
  - Tableaux de contrôle au plus près des équipements
  - Enregistreurs papiers disséminés
  - Personnel réparti sur tout le site



- Création de nouveaux besoins ← complexité du procédé
  - Centralisation de l'information
  - Centralisation du personnel opérationnel
  - Pilotage intégré
    - *Vue d'ensemble de tous les processus*
    - *Meilleure réactivité en cas d'incidents*
    - *Simplification du diagnostic*

Système Numérique de Contrôle-Commande

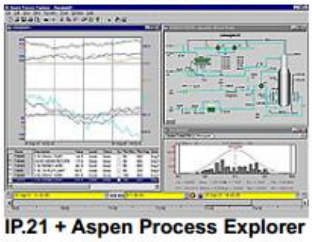
- Centralisation des systèmes de régulation dans une salle de contrôle

- Visualisation des informations sur des écrans

Années 1985-2000 - SNCC

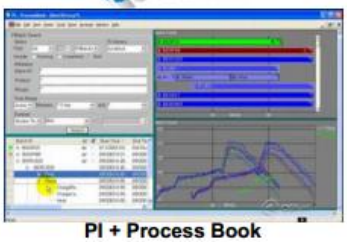
- Besoin de conserver l'histoire des données
  - Analyse a posteriori d'incident
  - Compréhension de phénomènes évanescents
- Apparition des « Data Historian » en complément des SNCC

aspentech



IP.21 + Aspen Process Explorer

OSIsoft.



PI + Process Book

- Intégration transverse
  - Connectivité avec les différents SNCC présents sur le site
  - Interface Homme-Machine orienté analyse
- Intégration temporelle
  - Le passé est utilisé pour optimiser le futur
  - Possibilité de corréler deux signaux

Années 1985-2000 – API + Supervision

- Besoin de visualiser l'état des automatismes

  - Connexion
    - *Ligne série RS-232, RS-485*
    - *Protocoles propriétaires : Unitelway, Modbus, Sinec-L1, DH+, …*

- Pilotage visuel d'une installation de production
  - Vue graphique « artistique » du procédé
  - Vue d'alarmes
  - Journal d'événements

PcVue Panorama CIMVIEW INDUSCREEN TopKapi Wizcon Telemecanique WinCC InTouch BSView32

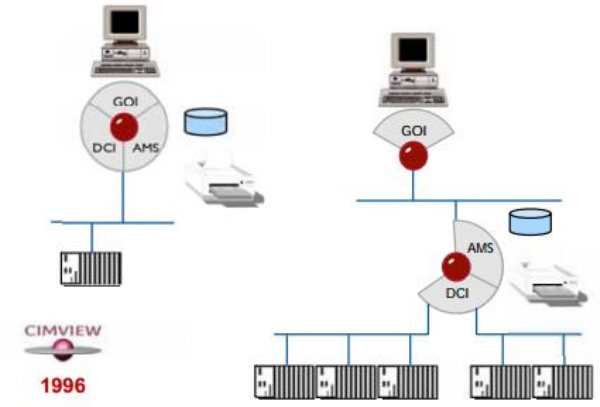
- Matériel standard

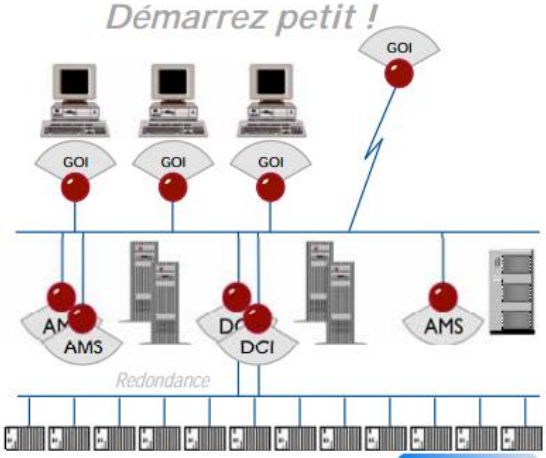
Années 1985-2000 – API + Supervision

- Supervision intégrée – un seul outil par atelier
  - Montée en puissance des réseaux informatiques
  - Généralisation de Ethernet TCP/IP
- Centralisation du pilotage

Pensez grand ,

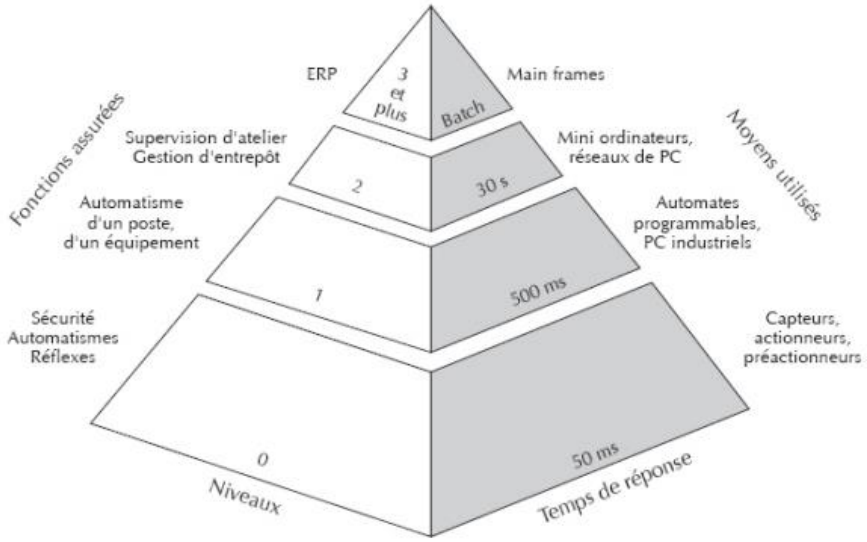
Démarrez petit !





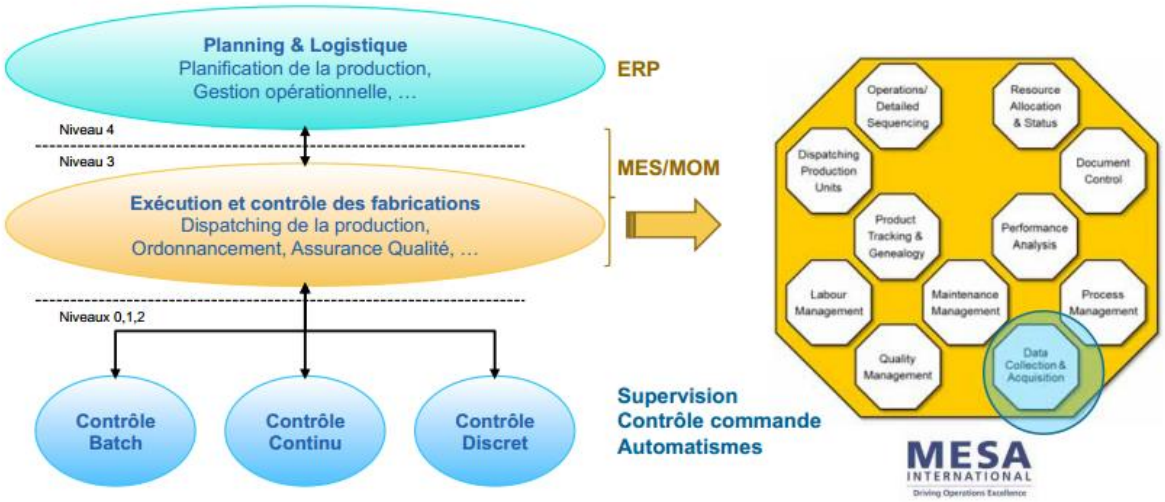
Années 1985-2000 : Concept d'Intégration Verticale

❑La pyramide du CIM



Années 2000 : intégration verticale

❑Du CIM au MES



Années 2000 : équipements communicants

- Automates, vannes, pompes, régulateurs, compteurs, …

- Connexion directe à partir de n'importe quel poste de travail
  - Maintenance plus aisée
  - Accès possible hors de l'entreprise (télé-diagnostique)

- Facteur de dé-intégration ?

Années 2000 : équipements communicants

- Automates, vannes, pompes, régulateurs, compteurs, …

- Connexion directe à partir de n'importe quel poste de travail
  - Maintenance plus aisée
  - Accès possible hors de l'entreprise (télé-diagnostique)

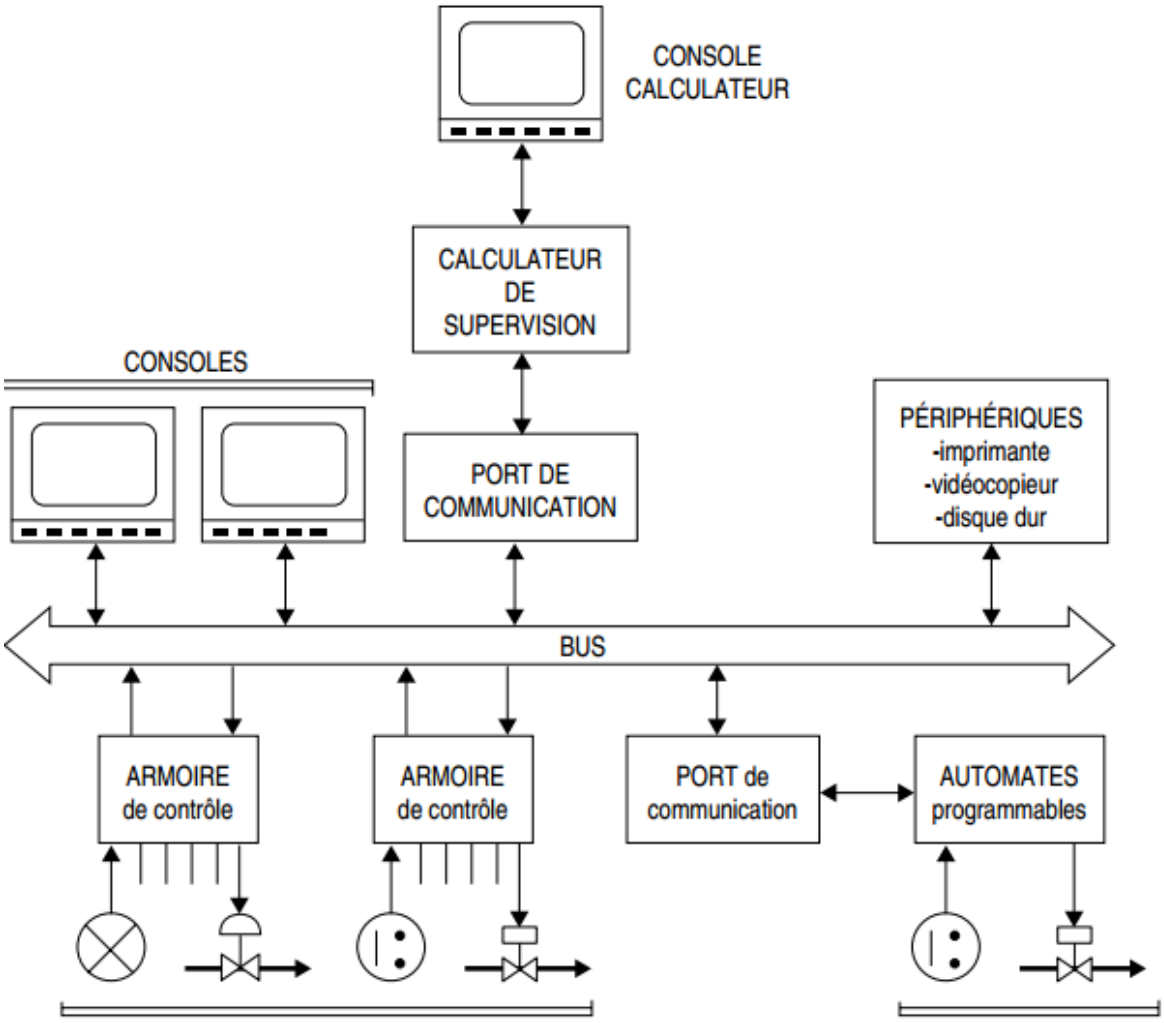
- Facteur de dé-intégration ?

VI.4.2 GENERALITES SUR SNCC

STRUCTURE GÉNÉRALE D'UN S.N.C.C.

Les Systèmes Numériques de Contrôle Commande (S.N.C.C.) ont pris leur essor dans les années 1970 avec la baisse du prix des microprocesseurs (la présentation des premiers systèmes date de 1975) ; celle-ci a permis, par utilisation de plusieurs unités de traitement à microprocesseurs, de disposer d'une grande puissance de traitement répartie en **sous-ensembles indépendants,** fournis, pour chacun d'entre eux, avec un certain nombre de fonctions ; ces fonctions, élémentaires au départ (régulation PID, et/ou acquisition, ...) sont devenues au cours du temps de plus en plus élaborées ce qui a largement augmenté les possibilités offertes par les systèmes.

La structure générale d'un S.N.C.C. est représentée ci-dessous.

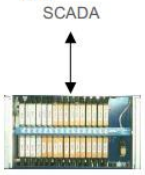


II.4.3 ARCHITECTURE MATERIELLE SNCC




DCS (DISTRIBITED CONTROL SYSTEM) =SNCC

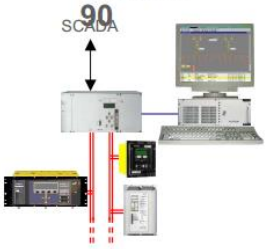
- **Années 80**



RTU

Supervision
Contrôle simple

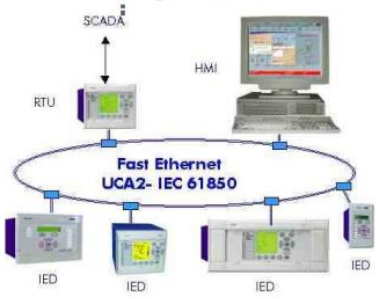
- **Années 90**



DCS
Ethernet/Protocole propriétaire
ou
type 103, MODBUS

Supervision
Protection
Automatismes
lents

- **Aujourd'hu**

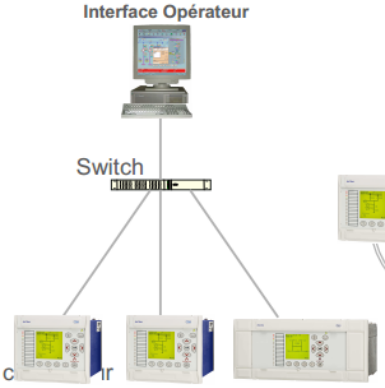


DCS
Ethernet/Protocole standardisé
IEC 61850

Supervision / Protection
Automatismes rapides
Ingénierie standardisée

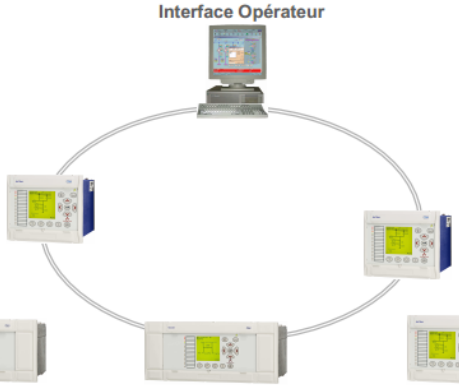
- **Différentes architectures basées sur le protocole IEC61850**

Étoile



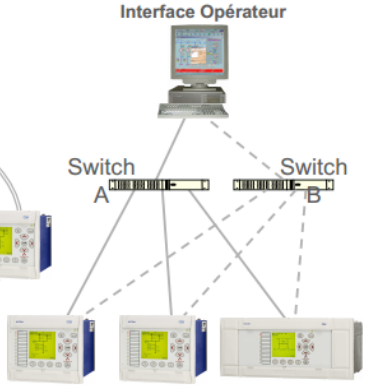
Architecture simple :
en cas de défaut sur le Switch Ethernet on perd la communication avec tous les calculateurs

Anneau

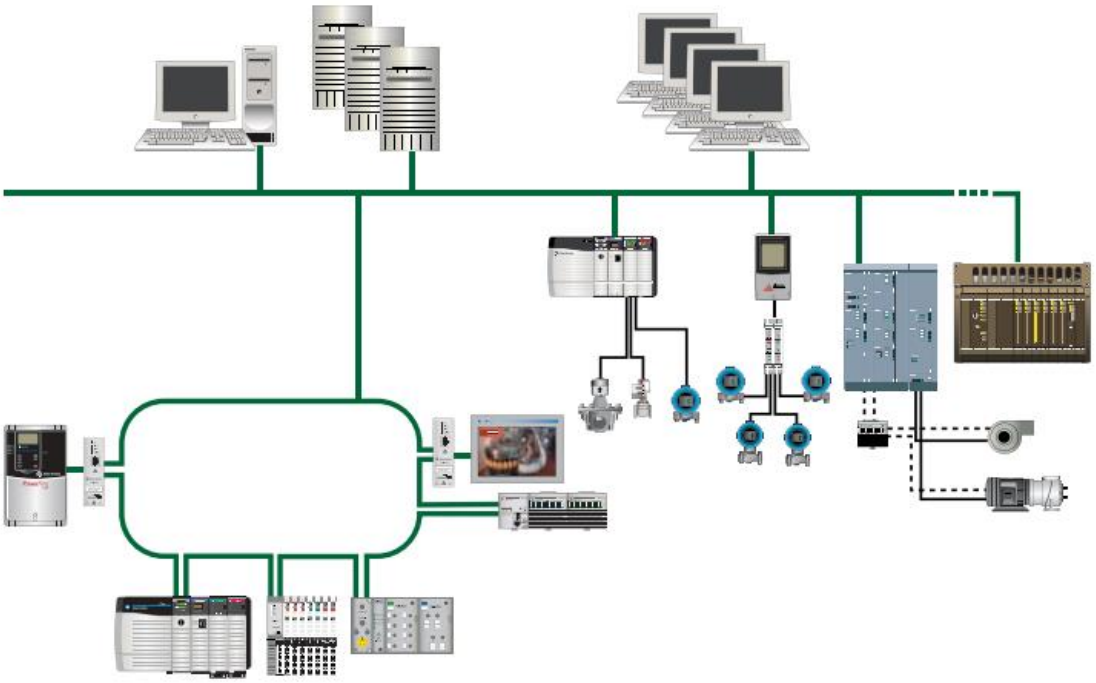


La solution la plus répandue :
protégé contre une rupture de communication
bon rapport qualité prix
(économie du Switch Ethernet)

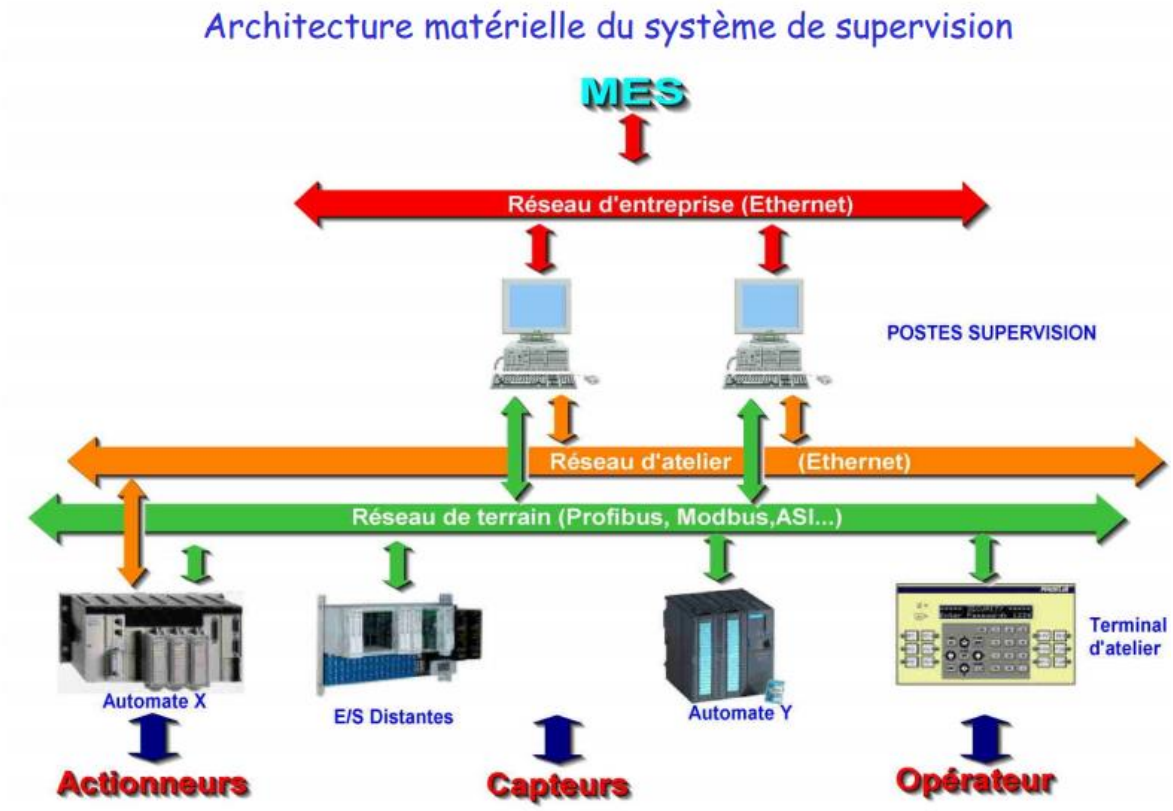
Double étoile



Architecture la plus sûre car elle tolère davantage de défaillance (excepté la perte simultanée des 2 Switch)



Un système d'automatisation moderne fournit une intégration étroite parmi différents automates et systèmes informatiques, ce qui permet une surveillance et un contrôle intégrés de l'ensemble de l'installation.



VI.4.3 COMPOSITION DU SNCC

Celui-ci est composé :

- de plusieurs sous-ensembles à microprocesseurs assurant chacun une partie du traitement et dotés des entrées-sorties industrielles nécessaires pour, essentiellement, la régulation continue et les séquences d'opération
- de périphériques de dialogue permettant :
  - le suivi de la marche des unités sur écrans vidéos couleurs
  - la commande par claviers des actionneurs (vannes, moteurs électriques, ...)
- de modules complémentaires permettant :
  - la réalisation d'historiques
  - l'optimisation par supervision des boucles de régulation de base
  - la mise en sécurité (automate programmable)
  - la sortie de journaux (imprimante)
  - l'archivage (disques magnétiques)
- d'un câble coaxial appelé bus faisant la liaison entre les éléments précédents, et dont la longueur permet leur dissémination dans l'usine :
  - armoires de contrôle près des unités de fabrication
  - périphériques de dialogue et modules complémentaires en salle de contrôle

VI.5 RESEAUX D'ENTREPRISE ET PYRAMIDE CIM : NIVEAU 4 (GESTION ENTREPRISE)

VI.5.1 GENERALITES SUR LA PYRAMIDE CIM

*La **pyramide du CIM** (Computer Integrated Manufacturing) est une méthode largement généralisée qui représente 4 niveaux auxquels correspondent des niveaux de décision. Plus on s'élève dans la pyramide du CIM, plus le niveau de décision est important et plus la visibilité est globale. Un niveau supérieur décide ce qu'un niveau inférieur exécute.*

On distingue généralement les niveaux suivants :

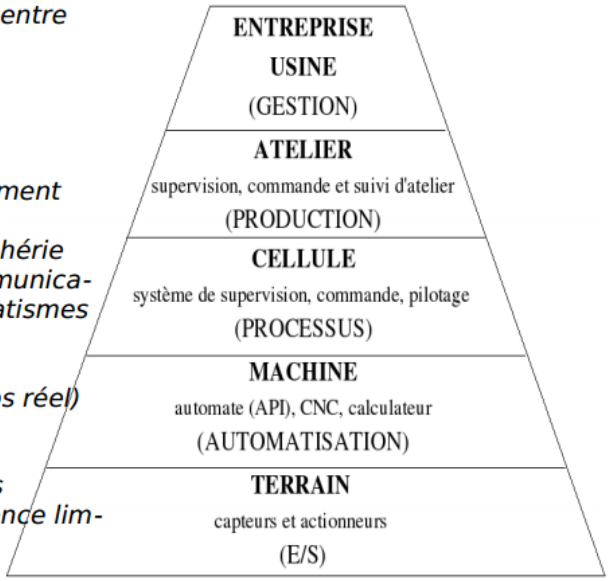
- au niveau 3 : la gestion des produits et des stocks, la gestion des approvisionnements, la gestion des clients, des commandes et de la facturation (gérés par les ERP (Enterprise Resources Planning))
- au niveau 2 : la localisation des produits en stocks, les mouvements physiques et la gestion des lots (géré par le système de gestion d'entrepôt)
- au niveau 1 : les automatismes
- au niveau 0 : les capteurs et actionneurs

Ainsi à chaque niveau, correspond un bus ou un réseau adapté aux besoins:

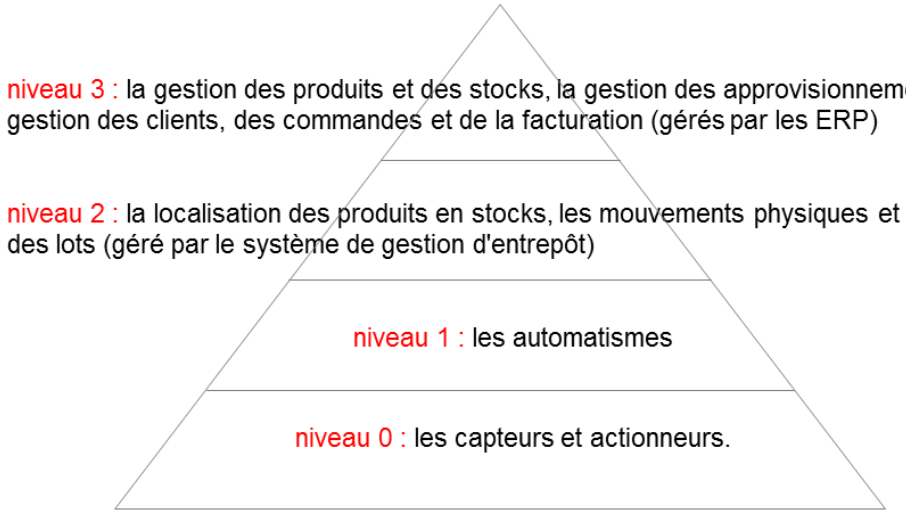
***Réseaux locaux industriels** (**data bus**): communication entre l'automatisme et le monde informatique*
→ Quantités importantes d'information (messages, fichiers)
→ Temps de réaction de 1 s à 10 s (temps non critique)
→ Longue distance possible

***Bus de terrain**: (**field bus**) réseaux entre unités de traitement (automates programmables, superviseurs, commandes numériques ...), (**device bus**) bus et réseaux pour la périphérie d'automatisme (variateurs, robots, axes ...) permet la communication d'unités de traitement pour la coordination des automatismes distribués*
→ Quantité relativement faible de données < 256 octets
→ Temps de réaction < 100 ms (notion d'événements temps réel)
→ Distance < 1 km

***Bus capteur/actionneur** (**sensor bus**): interface avec les capteurs/actionneurs, relie entre eux des noeuds à intelligence limitée ou nulle*
→ Niveau bits
→ Temps de réaction < 10 ms (contrainte temps réel)
→ Distance < 100 m



Le **Computer Integrated Manufacturing** (CIM) est un concept décrivant l'automatisation complète des processus de fabrication. C'est-à-dire que tous les équipements de l'usine fonctionnent sous le contrôle permanent des ordinateurs, automates programmables et autres systèmes numériques.

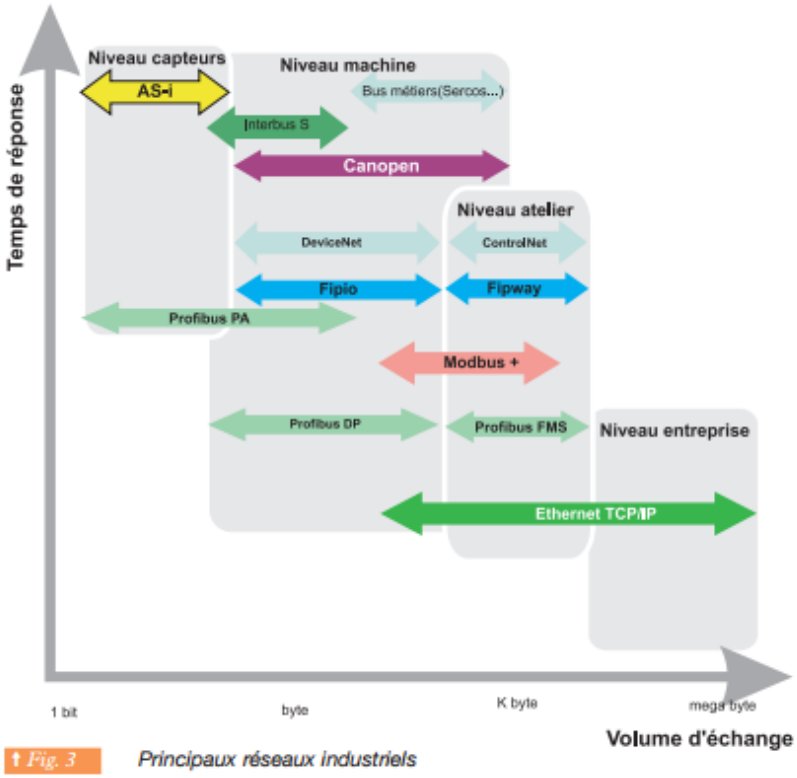


14

Nous pouvons, en première approche, retenir les deux principaux axes de ce tableau de besoins :

- le nombre d'informations à transmettre,
- le temps de réponse nécessaire.

Ceci nous permet de positionner les principaux réseaux commercialisés *(⇨ Fig.3)*.

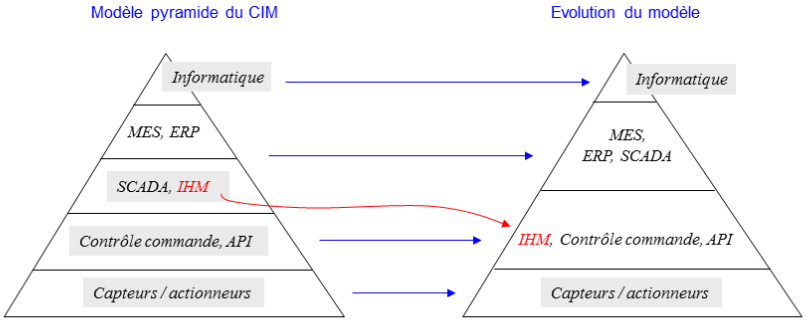


↑ *Fig. 3* *Principaux réseaux industriels*

VI.5.2 EVOLUTION DE L'ARCHITECTURE LOGICIELLE DE LA PYRAMIDE CIM

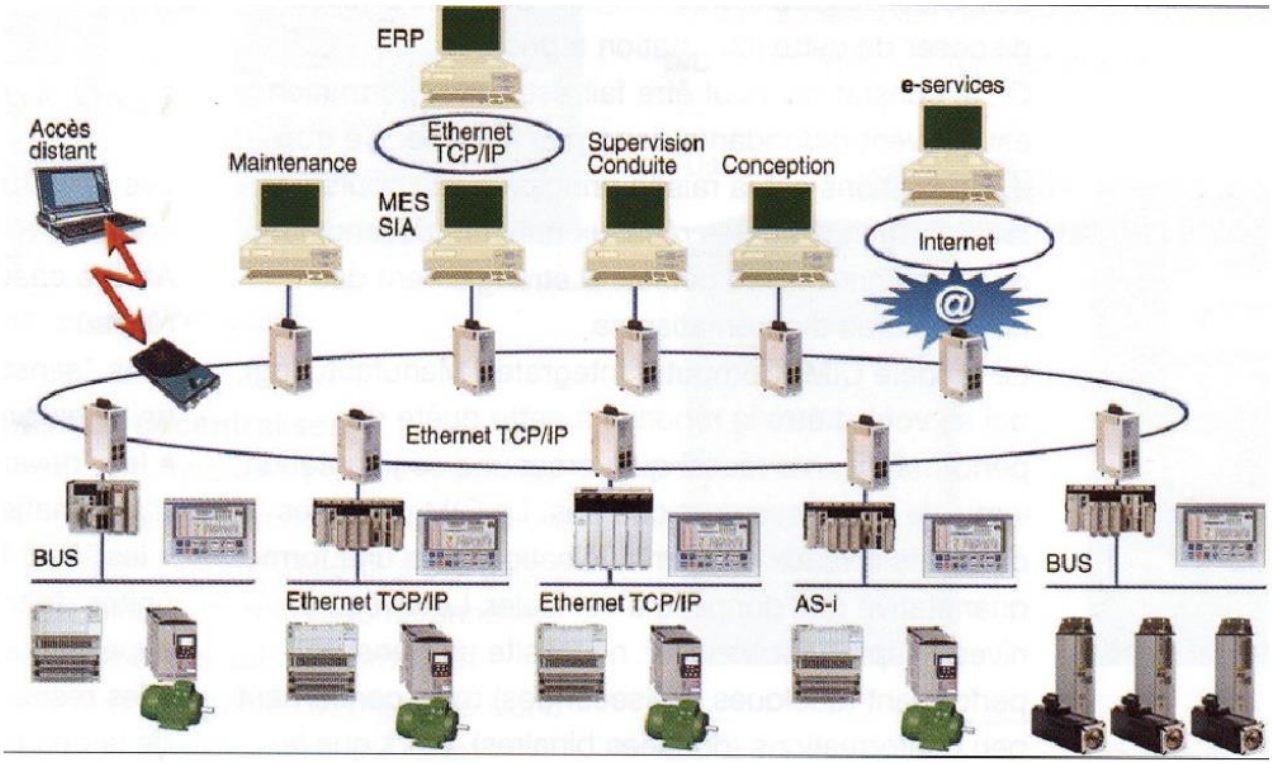
Un nouveau modèle de pyramide aplanie

5



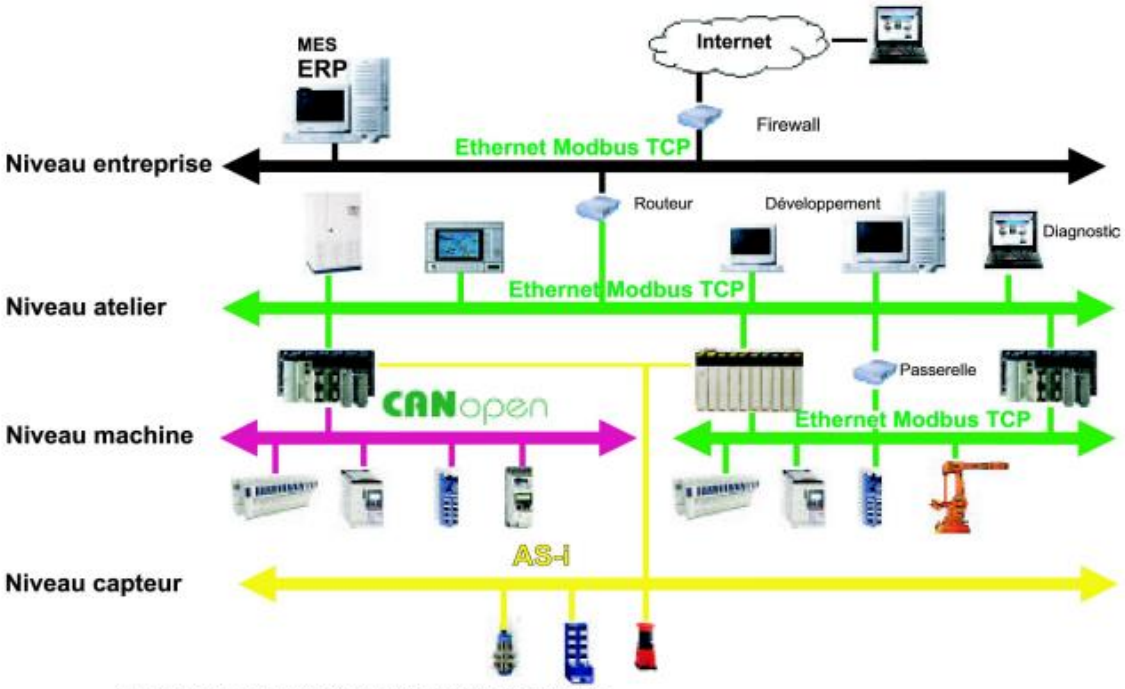
MES	Manufacturing Execution System
ERP	Entreprise Ressource Planning
SCADA	Superviseur
IHM	Interface Homme Machine
API	Automate Programmable

VI.5.3 ARCHITECTURE MATERIELLE DE LA PYRAMIDE CIM APLANIE



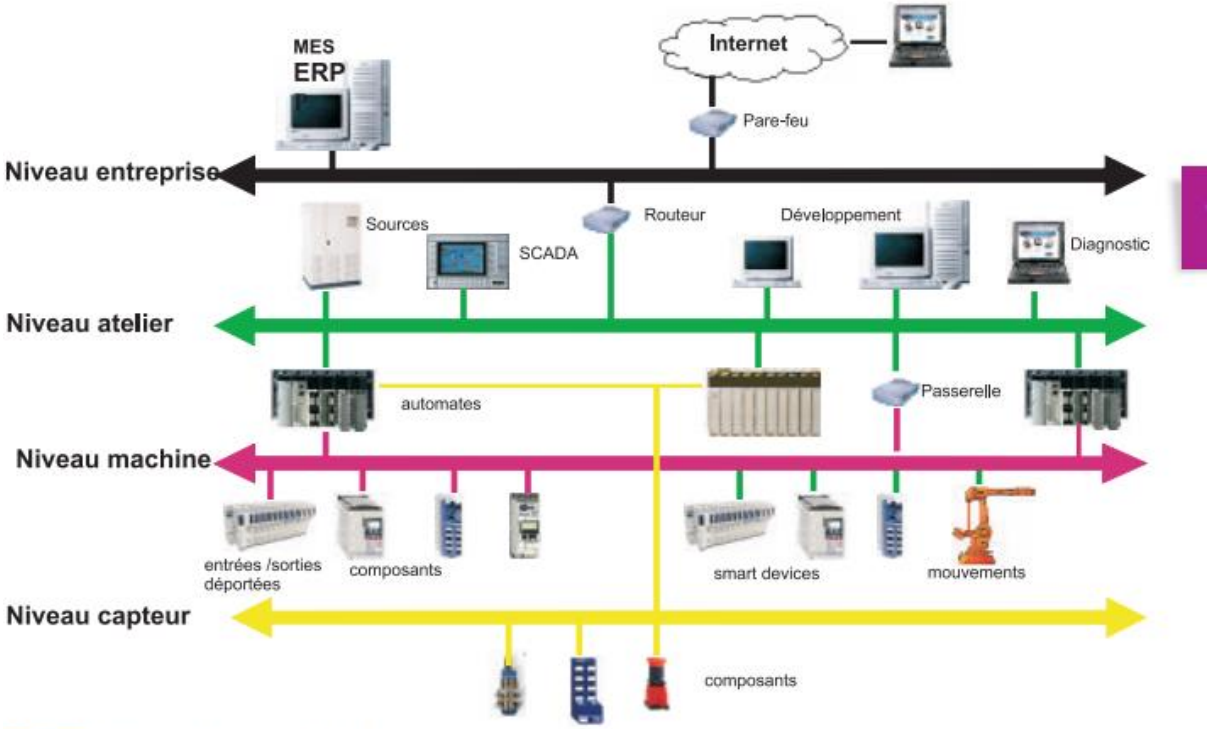
VI.5.4 MISE EN ŒUVRE DE LA PYRAMIDE CIM

EXEMPLE1 : SCHENEIDER ELECTRIC.VOIR LA PAGE SUIVANTE



Les niveaux de communication retenus par Schneider Electric

Sous l'effet conjugué des contraintes des utilisateurs, des technologies et des standards, les architectures actuelles se structurent en quatre niveaux distincts et interconnectés par des réseaux *(⇨ Fig. 1)*.



↑ *Fig. 1* *Exemple de niveaux d'architecture*

EXEMPLE2 : SIMATIC PCS 7 DE SIEMENS

SIMATIC PCS 7 DE SIEMENS