Problématique des systèmes temps réel

LPSIL
Option Informatique Embarquée
et Réseaux sans Fil

Marie-Agnès Peraldi-Frati

Cours IREEL

Problématique des STR 15h

UML Temps réel 30h

Mise en œuvre des STR 30h
- Approche Asynchrone
- Ordonnancement TR
- RTLinux

Mise en œuvre des STR 30h
- L'Approche synchrone
- Vérification
- Simulation

Programmation JavaCArd 9h

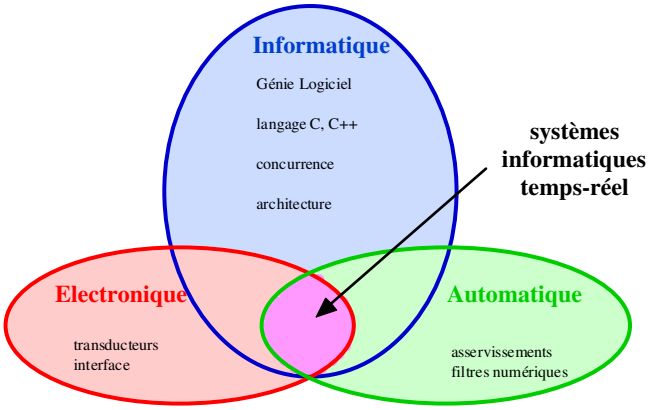
Administration de système linux 30h
- Installation
- configuration

OS embarqué 15h
- Windows CE ?
- Linux embarqué ?

Réseaux sans fil 30h
- Wan: GSM GPRS
- LAN : 802.11
- PAN Bluetooth

Réseaux filaires 15h
- FIP
- OSEK
- CAN

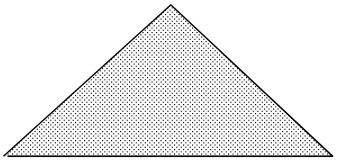
Où se situe le Temps Réel ?



Triangle de réussite

Equipe compétente, rigoureuse, bien coordonnée, à l'aise avec les **abstractions**, les **concepts**, tout en ayant les **connaissances approfondies** dans le **domaine de l'application**

HUMAIN



FORMATION
Méthodologie efficace produisant des modèles performants, fiables, évolutifs, faciles à maintenir. Maîtrise des fondements de la modélisation objet. Actualisation continue des connaissances sur les produits existants (réutilisation effective)

OUTIL et NOTATION
Outil de développement approprié, facilitant la mise à jour, l'exploitation, la compréhension intuitive, la communication, la documentatio

Plan du cours

- **Concepts de base pour le temps réel**
- Problématique par domaines d'applications
- Conception de systèmes temps réel
  - Commande numérique
  - Système logique
  - Entrées – Sorties - horloge
  - Approches Asynchrone /Synchrone
- Résumé
- Références

Définition du Temps réel

On qualifie de ***temps réel*** une application mettant en œuvre un système informatique dont le **comportement est conditionné** par **l'évolution dynamique** de l'état du **procédé** qui lui est **connecté**.
Ce système informatique est alors chargé de suivre ou de piloter ce procédé en **respectant des contraintes temporelles** définies dans le cahier des charges de l'application.

Jean-Pierre ELLOY, 1991

Définition du Temps réel

- La validité du traitement dépend
  - du résultat des calculs
  - des instants auxquels ces résultats sont attendus
  - du respect de la causalité

Attention !!
rapidité n'implique pas temps réel

Classification du temps réel: caractéristiques temporelles

Classification du temps réel : échéances (deadline)

Soft Realtime :
- le résultat du traitement a encore une utilité après l'échéance.

Hard Realtime :
- le résultat du traitement n 'a pas de validité après l 'échéance
- Le non respect des échéances peut avoir des conséquences catastrophiques pour l 'environnement (hommes, équipements ...).

Contraintes Temporelles Relatives

Temps réel à **contraintes relatives** « soft real-time »

• Le non-respect des contraintes temporelles ne conduit pas à un mauvais fonctionnement
• Il induit juste certaines **nuisances**
  •coût supplémentaire
  •agacement des opérateurs (temps de réponse) imprécisions admissibles…

Temps réel à **contraintes mixtes** « firm real-time »

Contraintes Temporelles Strictes

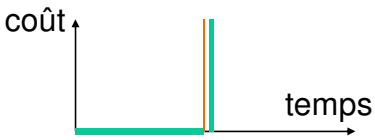
Temps réel à **contraintes strictes** « hard real-time »

• Le respect des contraintes temporelles est **indispensable** au bon fonctionnement.
• Le non-respect conduit
  • à des **résultats inexploitables**
  • à des **risques** pour la mission, le système ou son environnement

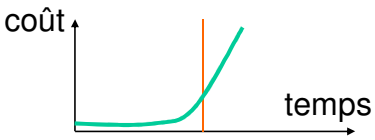
Contraintes strictes/relatives/fermes

Coût du dépassement de l'échéance :

- Résultat sans intérêt
- Dangereux



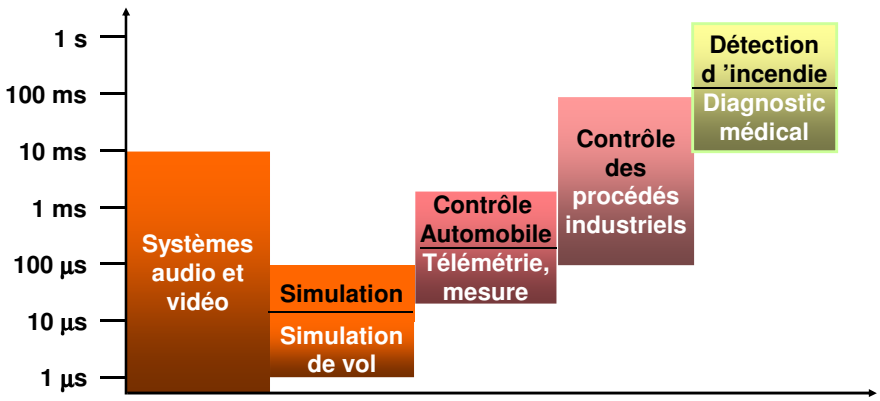
- Croissance du coût avec le retard



Analyse :

- Pire cas
- Performance en moyenne

Quantification des contraintes temporelles

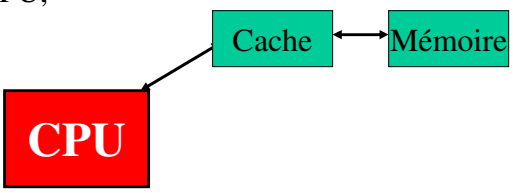


Mesure de performance : Vue classique

- Fonction de :
  - la performance CPU.

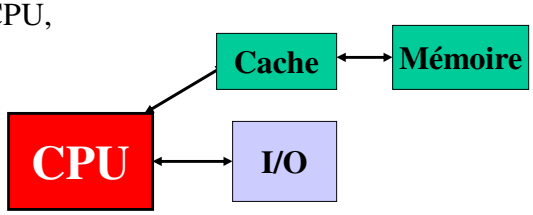
Mesure de performance : Vue par un ingénieur

- Fonction de :
  - la performance CPU,
  - les compilateurs



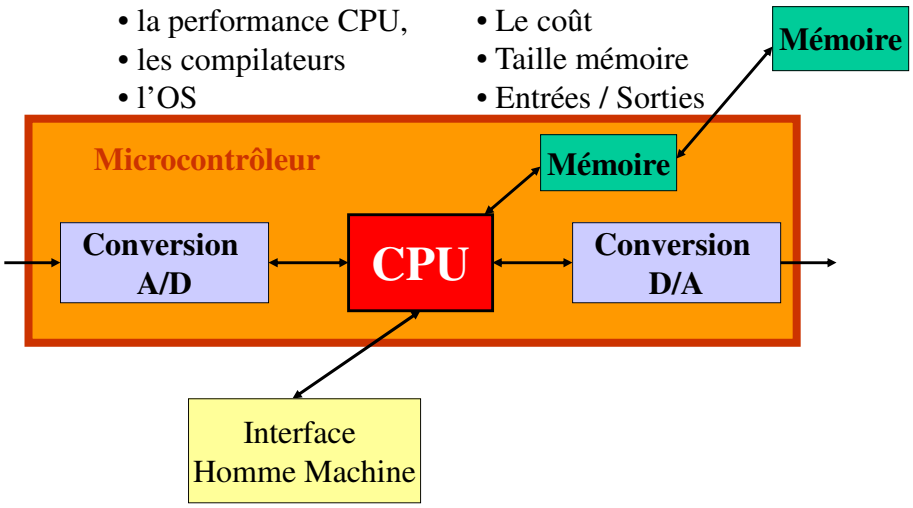
Mesure de performance :
Vue par un ingénieur spécialisé

- Fonction de :
  - la performance CPU,
  - les compilateurs
  - l'OS



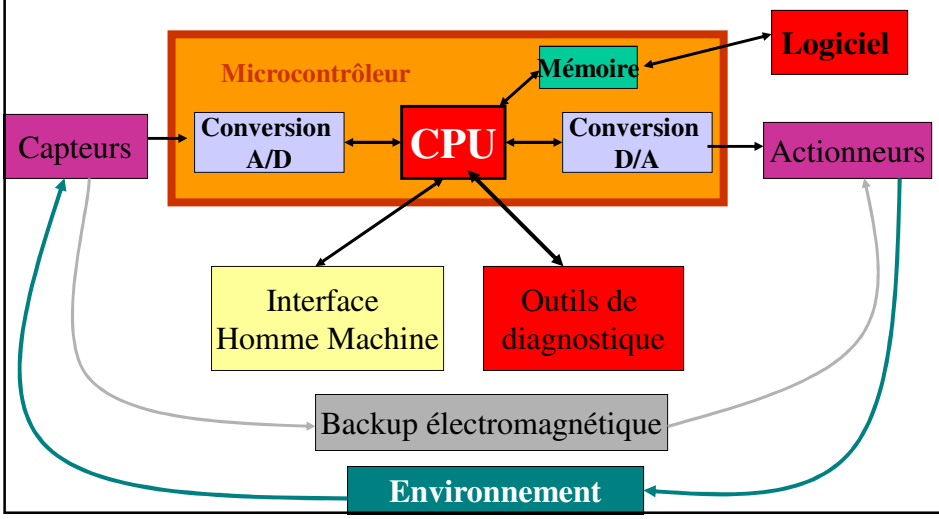
Mesure de performance :
Vue par un ingénieur en systèmes embarqués

- Fonction de :
  - la performance CPU,
  - les compilateurs
  - l'OS
- Le coût
- Taille mémoire
- Entrées / Sorties



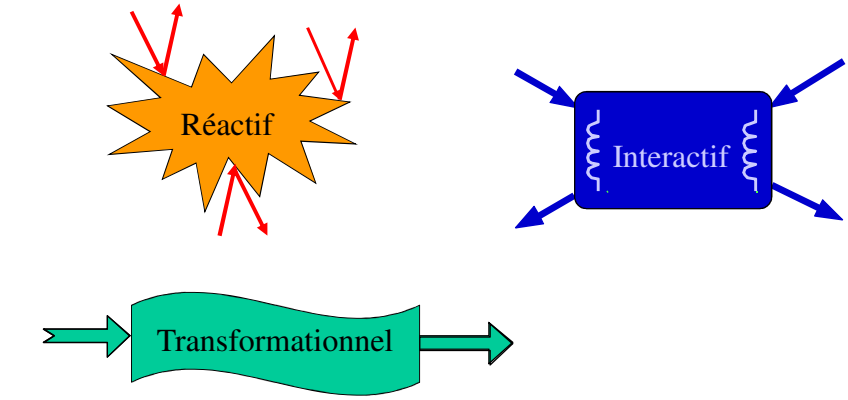
Mesure de performance :

Vue par un ingénieur en systèmes embarqués



Classification des systèmes temps réel

Caractéristique comportementale



Classification des systèmes temps réel

- **Réactifs** :
  - interactions permanentes avec l'environnement
  - En réponse aux stimuli le système provoque des réactions
  - Le système ne travaille que lors de l'élaboration des réactions
  - les instants de production des résultats sont contraints par la dynamique du procédé
- **Transformationnels**
  - Faible couplage avec l'environnement.
  - Les données en entrée sont prises à l'initiative du système.
  - Les résultats sont engendrés à l'initiative du système.
  - Les traitements internes sont généralement importants (traitements algorithmiques).
- **Interactifs**
  - les stimuli provoquent des réactions
  - leur prise en compte reste à l'initiative du système

Nature des traitements

- Sporadiques
  - Arrêt d'urgence
  - Changement de mode de fonctionnement
  - Traitement par interruption
- Périodiques
  - Acquisition de capteurs (image, poids, température)
  - Calcul de position
  - Calcul de consigne

Interactions Procédé Environnement

- Informations échangées entre procédé et système reflétant l'état du procédé et acquises sur ordre du système informatique de pilotage.
  - les événements : grandeurs signalant des transitions d'état (au sens discret du terme) du procédé et émises
  - les mesures : grandeurs par le procédé. Déclencheurs de réactions de la part du système
  - les commandes : grandeurs émises par le système de pilotage.

Exigences des systèmes temps réel

- Contraintes temporelles *(timing constraints)*
- Parallélisme *(concurrency)*
- Prévisibilité des comportements *(predictability)*
- Sûreté de fonctionnement *(dependability)*

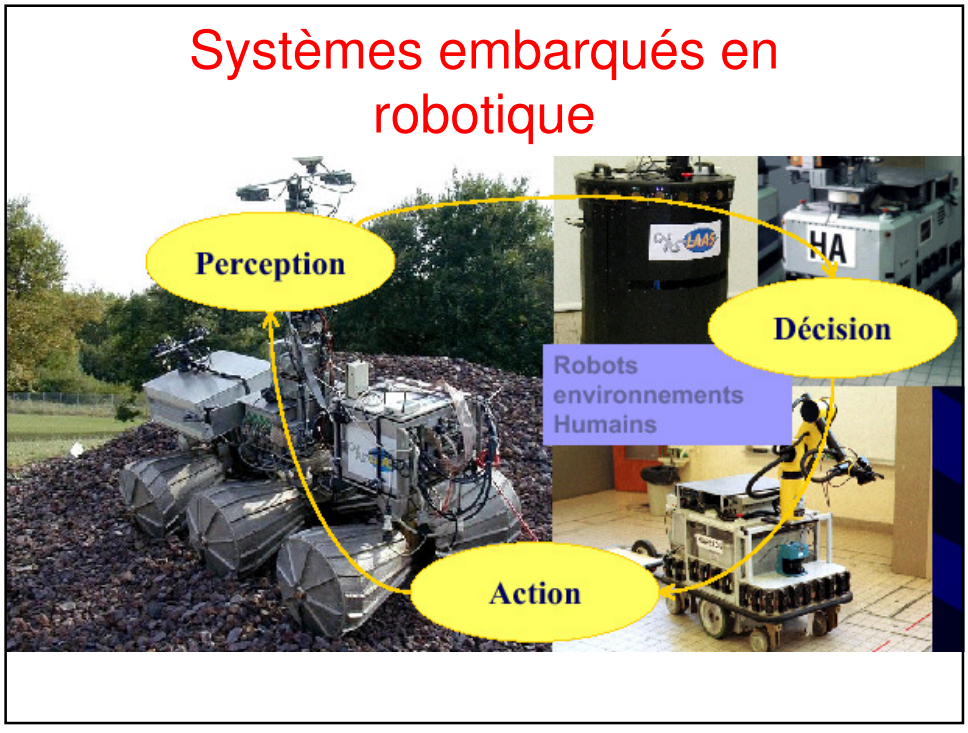
Exigences des systèmes embarqués

- Taille mémoire
  - Applications doivent être peu gourmandes en ressources mémoire
  - Applications reconfigurables
  - Optimisation du code
- Consommation
  - Équipements autonomes PDA, portables …
  - Différent niveau de gestiuon de la consommation
  - Mise en veille du matériel (ctrl de bus, accès mémoire , accès réseau…)

Plan du cours

- Concepts de base pour le temps réel
- **Problématique par domaines d'applications**
- Conception de systèmes temps réel
- Résumé
- Réfèrences


Systèmes embarqués :
Ordinateur + Application
dans un produit


Systèmes embarqués en robotique
Perception
Décision
Action
Robots
environnements
Humains

Systèmes robotique : caractéristiques

- Embarqués
- En boucle fermée avec l'environnement et l'utilisateur
- Autonomie décisionnelle

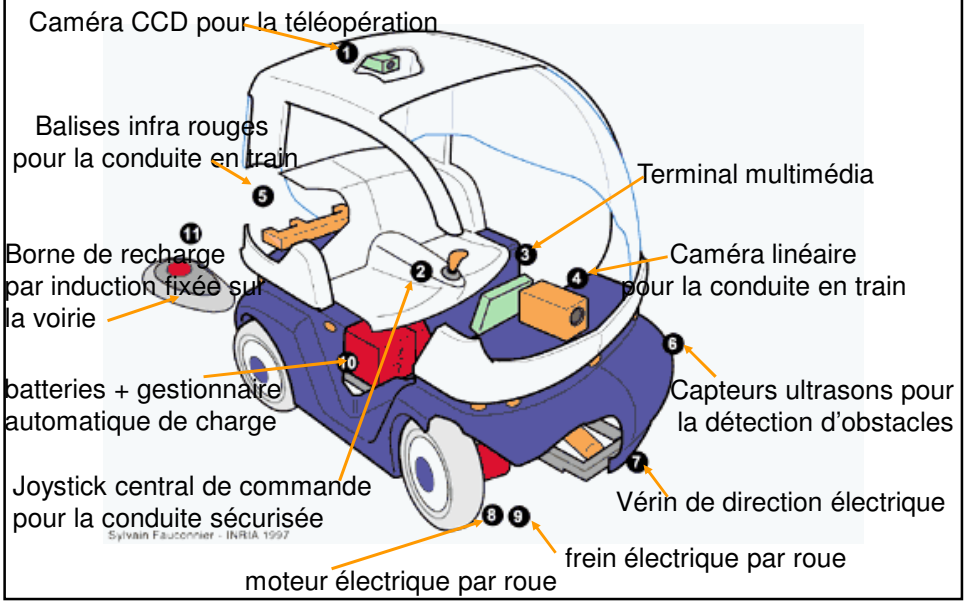
Interface avec l'environnement

- Nombreux capteurs et actionneurs
  - Dynamique variable
  - Organisation des boucles de contrôle commande
    - Synchronisation
    - Latence entre les traitements
    - Control Flow et Data Flow
- Interaction avec humains

Exemple d'application robotique : le cycab

- Véhicule électrique
- conçus pour les zones à circulation restreinte
  - hyper-centre urbain
  - gare/aérogare
  - campus universitaire
  - site touristique
- Véhicules conçus pour être utilisés de façon sûre et simple :
  - accès par carte à puce
  - conduite sécurisée par joystick
  - parking et recharge automatiques
  - disponible en porte à porte
  - terminal multimédia d'information

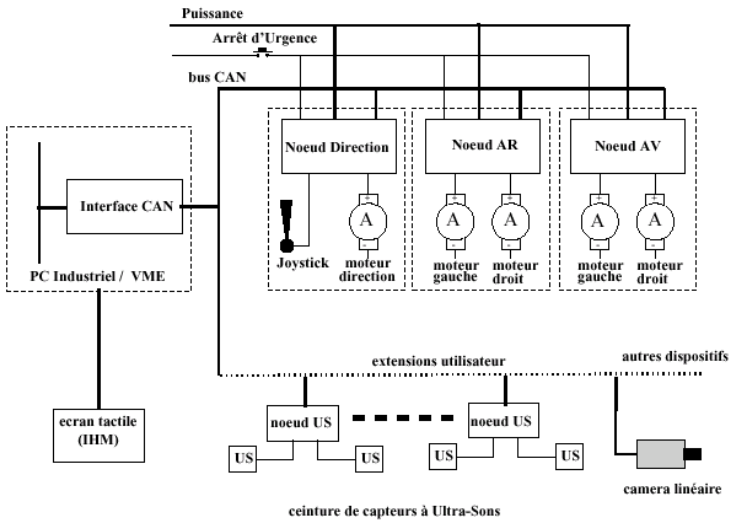
Equipements du Cycab



Architecture du Cycab :description

- Pour contrôler et commander ces 9 moteurs (4 de traction, 1 de direction et 4 de frein),
- Une architecture matérielle distribuée : 3 noeuds intelligents gérant 2 moteurs.
  - Le ***noeud de direction***,gère le joystick et contrôle le vérin motorisé de direction.
  - Deux ***noeuds de traction***, contrôlent les deux moteurs de traction.
  - La commande de frein de parking est unique pour les quatre moteurs.
- Un PC industriel dont le rôle essentiel est de gérer l'écran tactile de l'IHM, et une carte Motorola MVME162 sous VxWorks complète cette architecture matérielle.
- Les trois noeuds, le PC et la carte Motorola communiquent entre eux via un ***bus de terrain Controller Area Network (CAN).***
- Sur ce même bus CAN seront connéctés ***les capteurs ultrasonores*** ainsi que la ***caméra linéaire.***

Architecture du Cycab : synoptique



Robotique : systèmes et OS

- OS temps réel
  - –VxWorks Qnx,LinxOS, Psos, iRMX
  - – Unix et Linux temps réel

Mélanges

« haut niveau » Unix/linux/NT ou autres

« bas niveau » OS et micro contrôleurs spécifiques sur cartes

Robotique : le temps

- Dynamique du domaine
  - –Boucle de contrôle et d'asservissement
  - –Temps de réaction aux aléas et contingences
  - –Temps des traitements
- Une partie de la dynamique dépend de notre propre rythme
  - – On règle la vitesse max en fonction du temps d'arrêt (et des traitements qui l'y amènent)

Systèmes embarqués en avionique

Hardware spécialisé ou banalisé ?

– Anciennes générations (…A340):
  • calculateurs spécifiques (technologies dédiées)
  • bus avioniques (norme ARINC 429)

–Nouvelles générations (B777, Rafale, A380)
  • calculateurs banalisés (étagère vue comme un pool de machines utilisables pour l 'exécution des fonctions avioniques)
  • bus ethernet … banalisé
  • capteurs et actionneurs spécifiques

Système embarqué en avionique: hardware

...hardware rarement du dernier cri !

–Intel 286 sur l 'A320
–Intel 486 sur l 'A340-600 (1er vol en 2001)
–PowerPC pour le Rafale
–…

• pour des raisons de fiabilité
• de longueur de développement
• de coûts de qualification

Système embarqué en avionique: software

Langage de développement ?

- spécifiques jusqu'à l 'A340 (SAO)
- standardisé depuis l 'A340-600 (Lustre,SLD, Esterel,...)

Remarque : l 'avionique civile est un domaine qui peut se contenter de langages simples

=> utilisation de langages simples,souvent maison,dans le but de réduire les coûts de développement et de certification.

Système embarqué en avionique: OS

Les OS en avioniques civils :

- très simples pour les systèmes critiques :séquenceurs statiques (cf. Esterel)
- préemptifs pour les fonctions non critiques

(par exemple LynxOS avec RMA ou EDF)

OS exclusivement centralisés

- pas d'OS réparti
- système devant être sûrs : « safe »

Système embarqué en avionique: OS

à deux niveaux

- niveau partition =séquencement statique des partitions sur chaque calculateur
- niveau intra partition = politique d'ordonnancement libre au choix du concepteur

=> partage statique de l 'UC entre les partitions

=> politique d 'ordonnancement des processus à l'intérieur d'une partition propre à chaque partition.

Système embarqué en avionique : le temps

Avionique civile :

- pas de référence temporelle globale
- système fortement asynchrone
- architecture peu couplée

Avionique militaire :

- besoin d 'une horloge globale précise
- systèmes fortement intégrés (ou couplés)

Système embarqué en avionique : le temps

Un système avionique est réactif :

- soumis aux contraintes de temps imposées par son environnement
- fréquences d 'asservissement imposées par la dynamique de l'avion
- sensibilités aux variations de latences et durées de traitements déterminées par l 'environnement
- temps de réaction (d 'une chaîne fonctionnelle) maximum imposé par la dynamique de l 'environnement …

⇒ problème de performances

⇒ problème de déterminisme

Système embarqué en avionique : la sûreté

Avionique civile :

- certification ⇒ besoin de sûreté évident
- architecture fortement redondante
- vers des techniques formelles (Lustre,vérification
- formelle …)

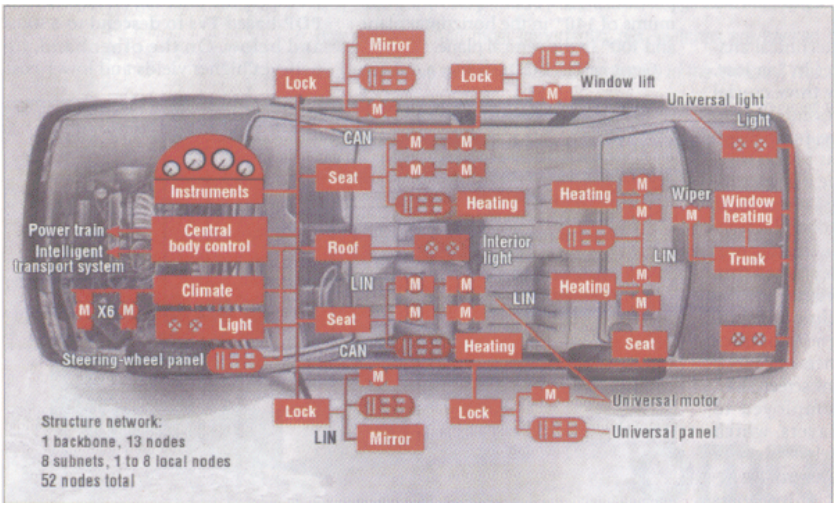
Avionique militaire :

- pas de certification mais besoin de sûreté aussi fort que dans le domaine civil
- architecture peu redondante mais reconfigurable
- vers des techniques formelles (Esterel,vérification formelle …)

Système embarqué en automobile

- Electronique représente 20% du prix du véhicule
- 80% des innovations se font grâce à l'apport de l'électronique
- Lois de commande de + en + complexes
  - ESP : stabilité du chassis
  - CGC : contrôle global du chassis , liaison au sol
  - ACC : Adaptation contrôle de vitesse
- Couplages entre les différents organes
  - Freins, suspensions, moteur …
- Commandes électroniques se substituent aux organes mécaniques
  - papillon motorisé,
  - X by Wire

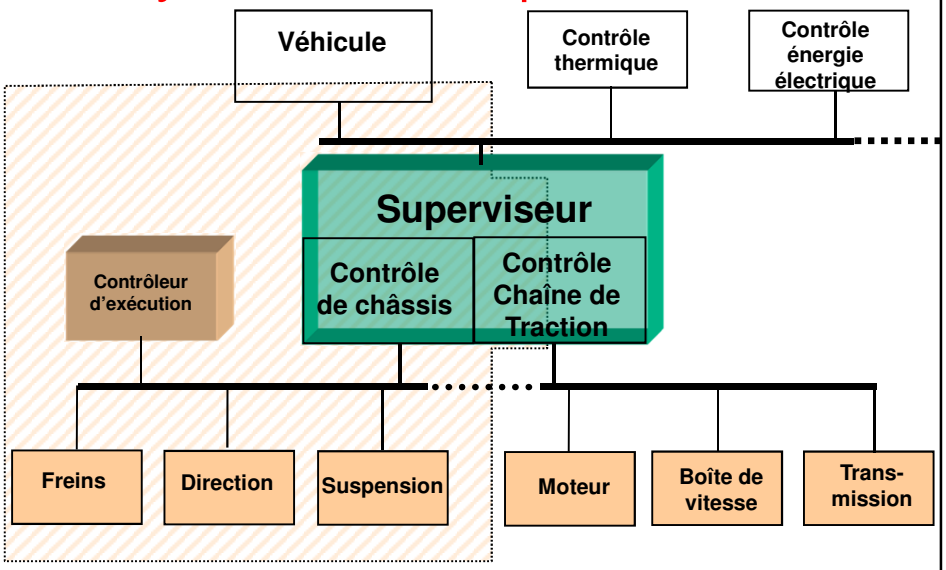
Système Embarqué en automobile



Exemple de fonctions embarquées sur un véhicule

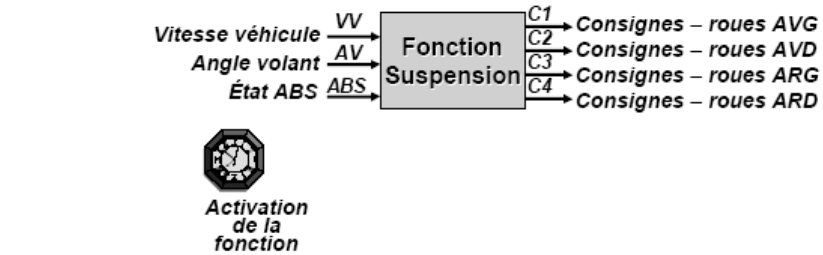
- Fonctions « sous capot »
  - Moteur - Frein
  - Direction - Boîte de Vitesse
  - Suspension - Embrayage
- Fonctions habitacle
  - les régulateurs d'allure
  - Détecteur de volume dans l'habitacle d'airbag
  - les systèmes d'éclairage intelligent
  - la direction assistée
- Fonctions de sécurité
  - la détection de l'état d'hypovigilance du conducteur
  - l'emploi de dispositifs détecteurs d'obstacles et d'aide au freinage

Système Embarqué distribué



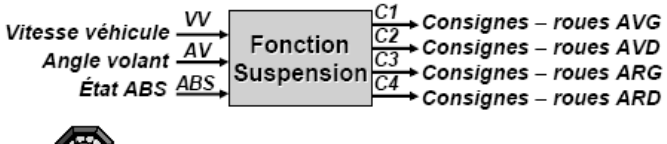
Système embarqué en automobile : les contraintes de temps

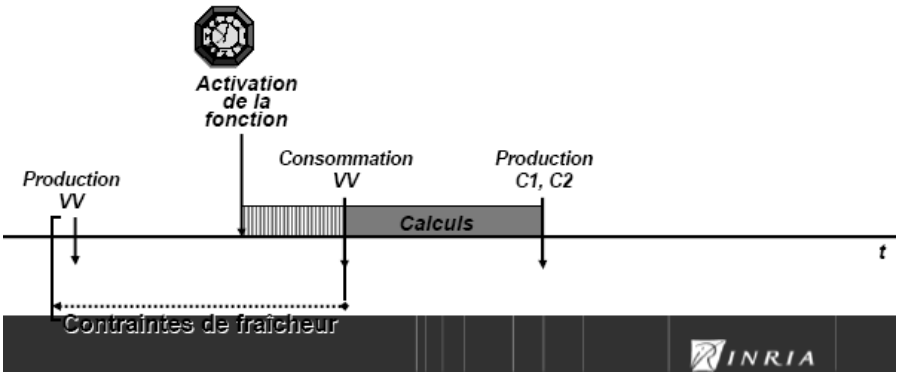
Exemple : supension pilotée



Système embarqué en automobile : les contraintes de temps

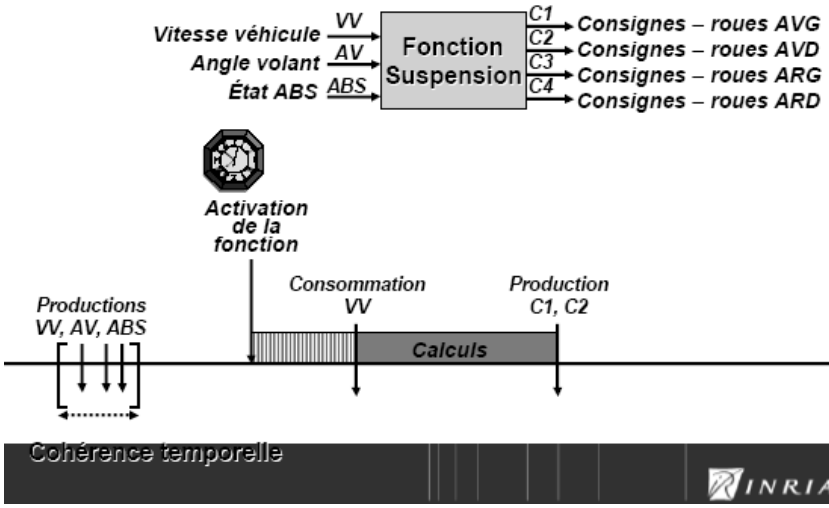
Exemple : supension pilotée





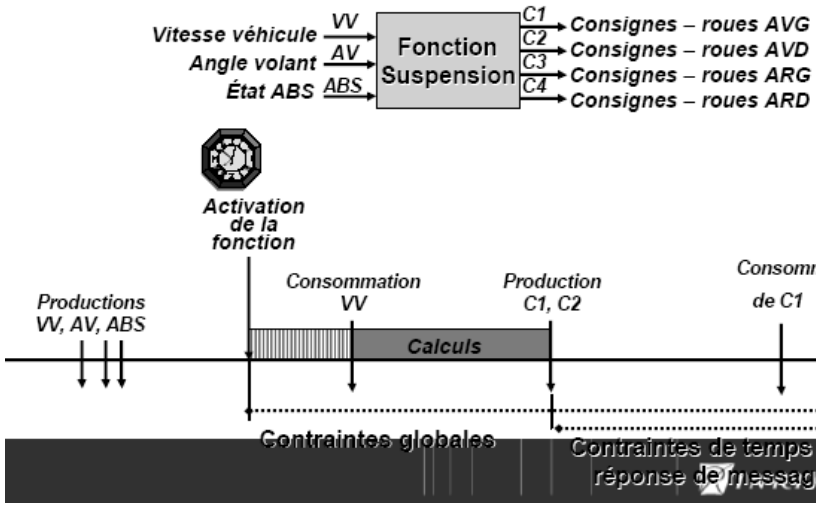
Système embarqué en automobile : les contraintes de temps

Exemple : supension pilotée



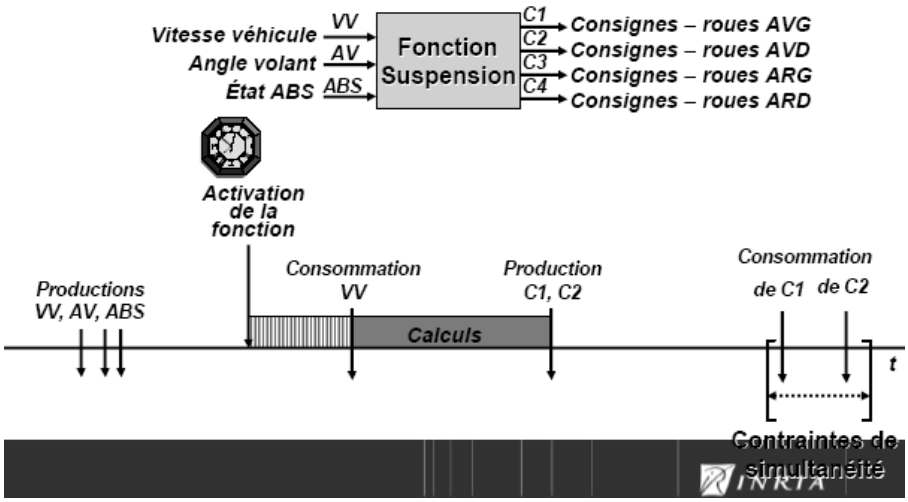
Système embarqué en automobile : les contraintes de temps

Exemple : supension pilotée



Système embarqué en automobile : les contraintes de temps

Exemple : supension pilotée



Système embarqué en automobile : la sûreté

- Indispensable pour le X by Wire
  - Sécurisation des architectures
  - Application des règles de Sûreté de Fonctionnement
    - FMEA Failure Mode Effect Analysis,
    - FTA (Fault Tree Analysis)
  - Modes de défaillances et de recouvrement d'erreurs

Autres domaines applicatifs

- Média (décodeur numérique audio vidéo…)
- Téléphonie (autocommutateurs, GSM…)
- Supervision médicale industrielle
- Système de production industrielle (centrale nucléaire, chaîne de montage, usine chimique…)
- Etc …

Résumé des contraintes des SE

- Hardware spécialisé:
  - cartes,bus,disques,driver ES
  - Peut être « durci » (rayonnement, thermique,vibration)
  - Bus et réseau spécifique
  - Rarement le dernier cri...
    - Spatial (première sparc)
    - Avionique civile (2-386)

Résumé des contraintes des SE

- Système et OS :
  - Habituellement temps réel
  - Parfois très simple
    - cadenceur/séquenceur TR plutôt que scheduleur
    - Avantage d'être plus déterministe
  - Offre des primitives TR (sémaphores, moniteurs,etc)

Résumé des contraintes des SE

- Contraintes temporelles :
  - Dynamique imposée par l'extérieur et les lois physiques
  - Temps de traitement et communication (et variations)
    - Simultanéité
    - Temps de réponse
    - Cohérence temporelle
    - Fraîcheur

Résumé des contraintes des SE

- Ressources limitées :
  - Espace, taille, forme
  - Énergie,puissance
  - Mémoire et processeur
    - Processus «créés » et mémoire « réservée » par avance

Résumé des contraintes des SE

- Interface avec l'environnement :
  - Nombreux capteurs et actionneurs
  - Pilotes logiciels
  - Dynamique du capteur et temps de latence
  - Synchronisme et synchronisation
  - Propriétés temporelles

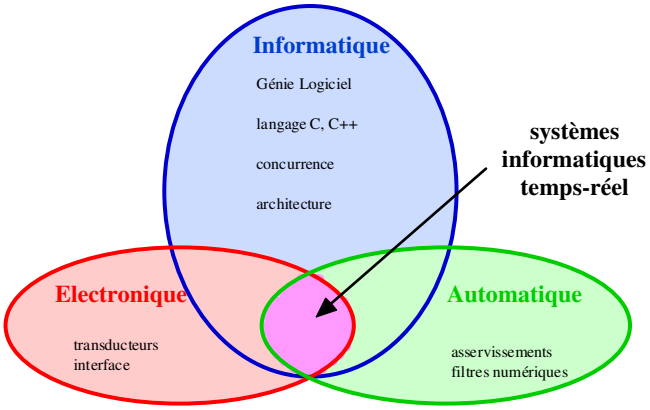
Résumé des contraintes des SE

- Sûreté des codes et des composants :
  - Redondance hard et soft
  - Robustesse des traitements
  - Reprise d 'erreurs
  - Modes dégradés lorsque nécessaire
  - Partition des traitements

Plan du cours

- Concepts de base pour le temps réel
- Problématique par domaines d'applications
- **Conception de systèmes temps réel**
  - Commande numérique
  - Système logique
  - Entrées – Sorties - horloge
  - Approches Asynchrone /Synchrone
- Résumé
- Réfèrences

Où se situe le Temps Réel ?



Triangle de réussite

Equipe compétente, rigoureuse, bien coordonnée, à l'aise avec les **abstractions**, les **concepts**, tout en ayant les **connaissances approfondies** dans le **domaine de l'application**

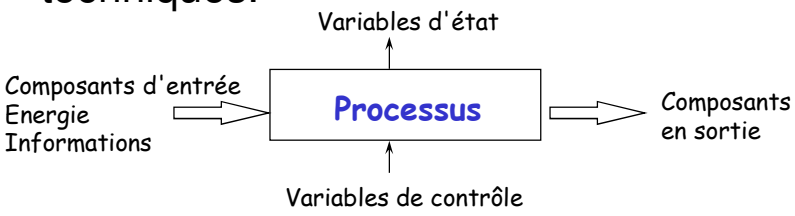
HUMAIN

FORMATION
Méthodologie efficace produisant des modèles performants, fiables, évolutifs, faciles à maintenir. Maîtrise des fondements de la modélisation objet. Actualisation continue des connaissances sur les produits existants (réutilisation effective)

OUTIL et NOTATION
Outil de développement approprié, facilitant la mise à jour, l'exploitation, la compréhension intuitive, la communication, la documentatio

Contrôle industriel

- **Processus Technique** : Un processus dont les paramètres physiques peuvent être acquis, et modifiés par des moyens techniques.

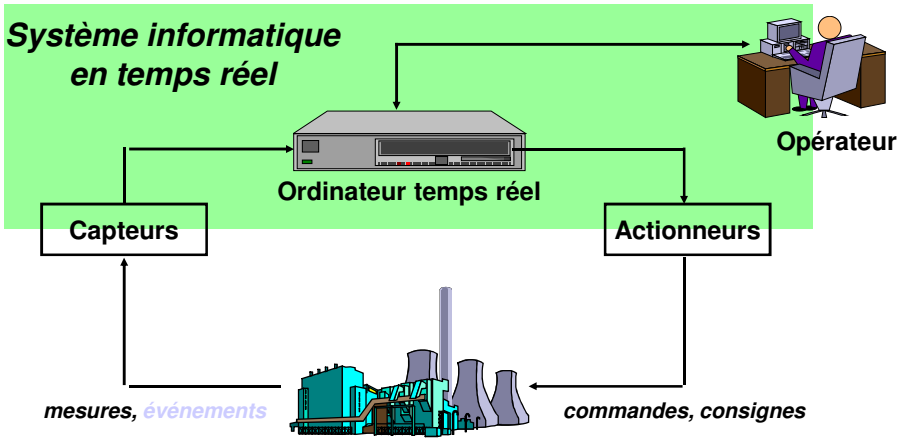


Objectif : Maintient du processus dans un état désiré

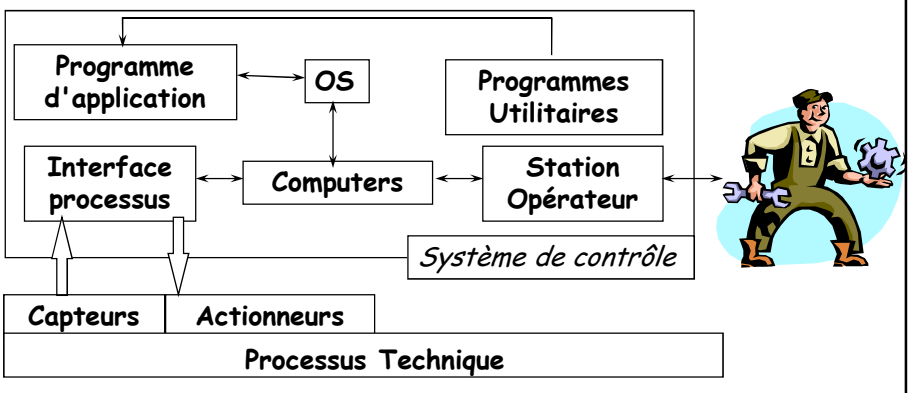
Couplages processus contrôleur

- Implantation d'un contrôleur
  - manuel (opérateur humain),
  - automatique (contrôle effectué par ordinateur)
  - ou mixte. (computer + opérateur)
- Le contrôleur doit être connecté au système à contrôler:
  - off-line
  - on-line en boucle ouverte
  - on-line en boucle fermée

Interactions



Architecture du contrôle



Fonctions à assurer

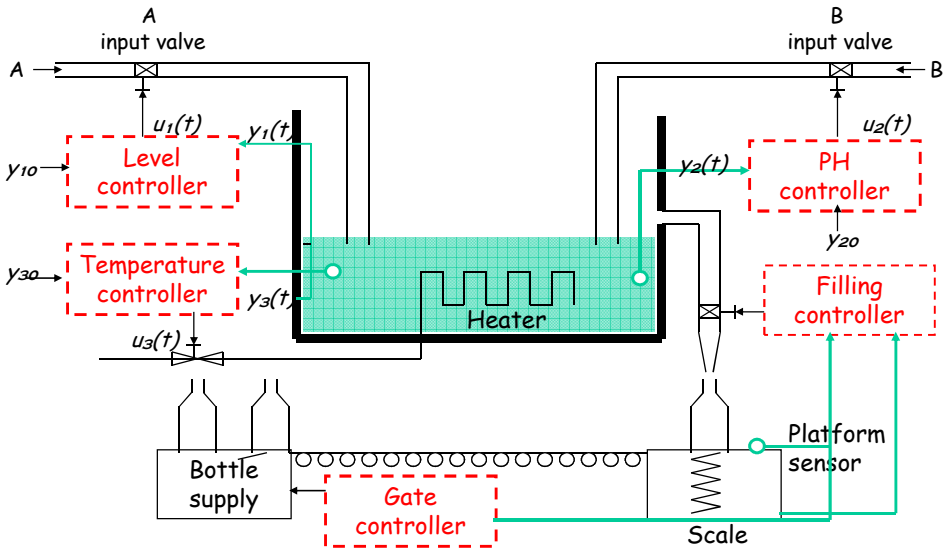
- Acquisition des données
- Contrôle séquentiel
- Boucle de contrôle
- Supervision
- Analyse des données
- Stockage des données
- Interface homme-machine

Exemple d'un système de contrôle:

un processus chimique

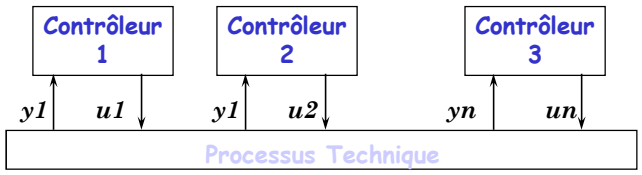
- 4 boucles de régulation
  - **Niveau de la cuve** est régulé à partir d'un capteur et d'une vanne A qui amène un produit A dans la cuve
  - **La température** à l'intérieur de la cuve régulée à partir d'une résistance et d'un capteur de température
  - Contrôle du **PH** par un capteur et régulation par ajustement de l'ingrédient B
  - **Remplissage** des bouteilles, capteurs de présence et de poids

Architecture du processus

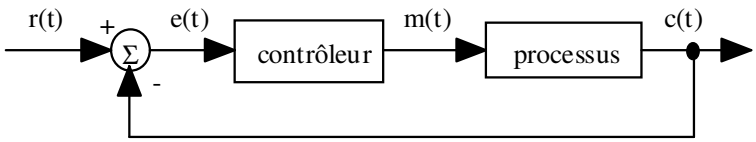


Structure de processus classique

- Boucles de contrôle indépendantes
  - 3 boucles de contrôle continu (évolution continue des variables d'état)
  - 1 boucle de contrôle liés à un état binaire (capteurs de présence)
- Chaque boucle :
  - Etat courant *y(t)*, Consigne *y0(t)*, Variable de contrôle *u(t)* (ajustement)
  - PID (Proportionnel intégrateur dérivateur) Contrôleur
  - Relais, portes logiques



Asservissement système continu



- Système bouclé
  - Consigne r
  - Variable de mesure c
  - Variable de commande m
  - Erreur e

Objectif : c doit suivre r $\forall t\ c(t) = r(t)$

Asservissement système continu

- Si e(t) = r(t) - c(t) >0 ⇒ augmenter m(t)

Asservissement système continu

- Si e(t) = r(t) - c(t) >0 ⇒ augmenter m(t)
- Si e(t) = r(t) - c(t) <0 ⇒ diminuer m(t)

Asservissement système continu

- Si e(t) = r(t) - c(t) >0 ⇒ augmenter m(t)
- Si e(t) = r(t) - c(t) <0 ⇒ diminuer m(t)

• Quelle proportion pour la commande ?

Asservissement système continu

- Si e(t) = r(t) - c(t) >0 ⇒ augmenter m(t)
- Si e(t) = r(t) - c(t) <0 ⇒ diminuer m(t)

- Quelle proportion pour la commande ?
  - Proportionnelle (P) : $m(t) = K_p * e(t)$ **Rapidité ++**

Asservissement système continu

- Si e(t) = r(t) - c(t) >0 ⇒ augmenter m(t)
- Si e(t) = r(t) - c(t) <0 ⇒ diminuer m(t)

- Quelle proportion pour la commande ?
  - Proportionnelle (P) : $m(t) = K_p * e(t)$ **Rapidité ++**
  - P Intégrale (PI) :

Précision ++

$$m(t) = K_p * [e(t) + \frac{1}{T_i} \int_0^t e(\tau)\, d\tau]$$

Asservissement système continu

- Si e(t) = r(t) - c(t) >0 ⇒ augmenter m(t)
- Si e(t) = r(t) - c(t) <0 ⇒ diminuer m(t)

- Quelle proportion pour la commande ?
  - Proportionnelle (P) : $m(t) = K_p * e(t)$ **Rapidité ++**
  - P Intégrale (PI) :

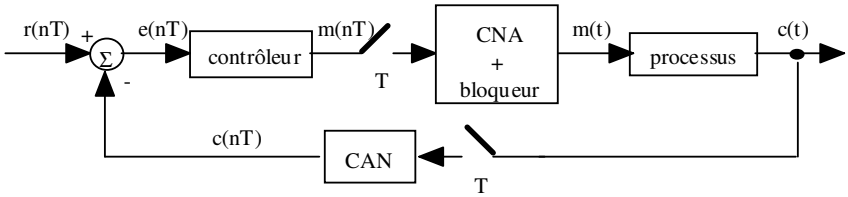
Précision ++ $$m(t) = K_p * [e(t) + \frac{1}{T_i} \int_0^t e(\tau)\, d\tau]$$

  - P Intégrale dérivée (PID) :

Stabilité ++ $$m(t) = K_p * [e(t) + \frac{1}{T_i} \int_0^t e(\tau)\, d\tau + T_d \frac{de(t)}{dt}]$$

Asservissement numérique

- Echantillonnage : temps discret
- Conversion signaux : CNA et CAN



$$m(k) = K_p . \left(e(k) + \frac{1}{T_i} \sum_{l=0}^{k-1} e(l).h + T_d . \frac{e(k) - e(k-1)}{h} \right)$$

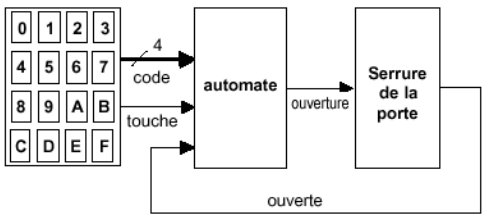
Asservissement numérique

Problème de programmation :

- Calculer m_n avec une période T_p
- Tâche périodique avec plusieurs paramètres à gérer :
  - Pi : Période de la tâche i. Comme l'algorithme PID doit être relancé périodiquement, il faut lui définir un intervalle de temps entre les exécutions successives de l'algo
  - Di :Temps limite pour la tâche i. Il s'agit du temps maximum alloué pour accomplir le travail. (exécution de la commande M)
  - Ci : Temps de calcul maximal pour la tâche i. Comme l'algorithme lui-même met un temps donné pour s'exécuter, il faut définir un temps maximal de calcul.

Système logique : automate

- Exemple : digicode constitué de :
  - un clavier à 16 touches qui constitue l'organe d'entrée
  - une commande l'ouverture d'une porte en sortie.
  - code secret fixé une fois pour toutes.



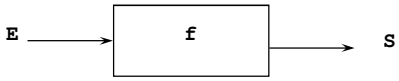
Système logique : automate

- Les automates représentent la solution d'un problème par une succession d'étapes bien définies, bien séparées les unes des autres.
- Chaque étape est représentée par un ensemble **fini** de valeurs de ce que l'on appelle l'état (interne) du système.
- Les entrées du système provoquent des **transitions** entre les états, une transition ne peut être franchie que si le système est dans l'état de départ de la transition concernée et qu'une condition extérieure est vérifiée.
- Si aucune des transitions issues d'un état n'est active, toutes les conditions étant fausses, le système reste dans l'état considéré, **par défaut il y a mémorisation de l'état présen**t.
- Les sorties du système dépendent de la valeur de l'état: plusieurs états peuvent conduire aux mêmes valeurs des sorties

Circuits combinatoires et Séquentiels

- Elément de l'algèbre de Boole => circuits combinatoires
  - circuits de base des ordinateurs
- Théorie des automates => circuits séquentiels
  - modèle de base pour le fonctionnement des circuits
- Signaux logiques et Analogiques
  - Traitement de l'information
    - traiter
    - mémoriser des signaux électriques
    - transférer

Circuits combinatoires



Système combinatoire =>

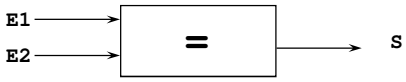
un ensemble fini d 'entrée E

un ensemble fini de sortie S

Sortie = f(Entrée)

Exemple :

- portes ET/OU
- Aiguillage d'info (multiplexeur demultiplexeur)



E1 /E2	0	1
0	1	0
1	0	1

Circuits séquentiels

Sortie = f(entrée; état interne)

Système séquentiel =>

- un ensemble fini d 'entrée E
- un ensemble fini d 'états Q
- une fonction de transition G :Q x E -> Q
- un ensemble fini de sortie S
- un état initial Q0
- une fonction de sortie

  Mealy F : Q x E -> S

  Moore F ': Q -> S

Circuits séquentiels

Sortie = f(entrée; état interne)

Exemple :

- Bascule SR
- Fonction de décalage
- Fonction de comptage...

Théorie des automates

Circuits séquentiels

- **Fonctionnement :**

Les circuits séquentiels sont conçus à partir de :

circuits combinatoires (algèbre de Boole)

\+ Etat interne

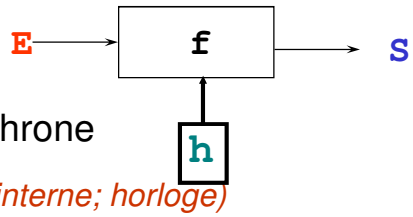
- **Utilisation :**
  - mémorisation de l'information
  - structure de contrôle
  - unité de commande
  - support théorique pour le développement (algo, langages, techniques de compilation)
  - modèle d'expression des systèmes d'état.
  - …

Circuits séquentiels

- Modèles pour la représentation de ces circuits
  - Machine de MOORE
    - q =G (q,e)
    - s = F (q)
    - la sortie est liée à l 'état. La durée de la sortie est égale au temps resté dans l 'état
  - Machine de MEALY
    - q=G(q,e)
    - s=F(q,e)
    - la sortie est liée à la transition. La durée de la sortie est égale au temps que dure l 'entrée.

Circuits séquentiels Asynchrone/Synchrone

- Modèles asynchrones :
  - le temps n'intervient pas explicitement,
  - problèmes de comportement (stabilité des entrées)
- Ajout d'une horloge => Circuit séquentiel synchrone
  - déclenchement des signaux
  - chronologie des actions
  - Exemple : séquenceur.



- Automate d'état fini synchrone

Sortie = f(entrée; état interne; horloge)

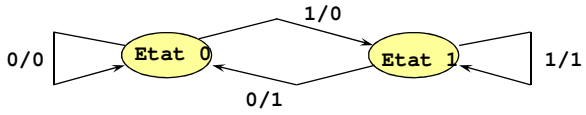
Description d 'un circuit séquentiel

- Mémoire d'une position binaire
  - une entrée **e** 0/1
  - deux états **q** 0/1
  - une sortie **s** 0/1
- l'**état** de l'automate q(t+1) dépend de e(t)
  - e(t) = 0 => q(t+1) = 0
  - e(t) = 1 => q(t+1) = 1
- la **sortie** de l'automate s(t+1) dépend de q(t)
  - q(t) = 0 => s(t+1) = 0
  - q(t) = 1 => s(t+1) = 1

Description d'un circuit séquentiel

- **Graphe** : Exemple de machine de Mealy

Rond : état
Arc : transition entre état
Label : entrée / Sortie



- **Table :**

Ligne : état courant
Colonne : entrée
Cellule : état suivant / Sortie

	0	1
Etat 0	Etat0/0	Etat1/0
Etat 1	Etat0/1	Etat1/1

- **Textuel (équations booléennes):**

si (Etat0 et 0) alors (0,Etat0)
si (Etat0 et 1) alors (0,Etat1)
Si (Etat1 et 0) alors (1,Etat0)
si (Etat1 et 1) alors (1,Etat1)

Mise en œuvre d 'un automate en C

```
int Etat ;
void Sortie0_0() {printf("0\n");}
void Sortie1_1() {printf(  »1\n");}
…..
```

Circuit séquentiels synchrones

- **Stimulus :** entrée E à l'instant t E(t)
- **Instant d'observation :** t, t+1, t+2, ...
- **Fonction de transition :**
  - Machine de MEALY

    q à l'instant t+1 q(t+1) = G[q(t), e(t)]

    s à l'instant t+1 s(t+1) = F[q(t), e(t)]

  - Machine de MOORE

    q à l'instant t+1 q(t+1) = G[q(t), e(t)]

    s à l'instant t+1 s(t+1) = F[q(t)]

Mise en œuvre d 'un automate en C

```
int Etat ;
void Sortie0_0() {printf("0\n");}
void Sortie1_1() {printf( »1\n");}
…..
void main() {
char Entree;
Etat=0 ; // état initial
do {









}while (1);
}/* end main */
```

Mise en œuvre d 'un automate en C

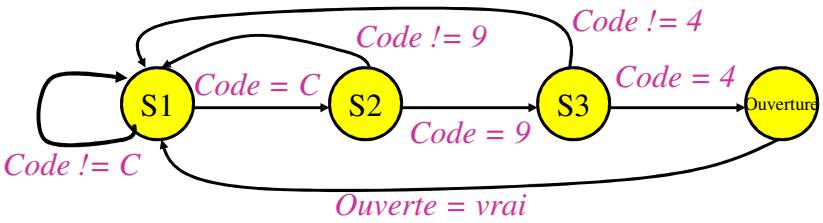
```
int Etat ;
void Sortie0_0() {printf("0\n");}
void Sortie1_1() {printf( »1\n");}
…..
void main() {
char Entree;
Etat=0 ; // état initial
do {
        printf("  Etat=%d\n",Etat);
        printf("  Commande==>");
        Commande=getche();
        switch(Etat) {




        }
}while (1);
}/* end main */
```

```
int Etat ;
void Sortie0_0() {printf("0\n");}
void Sortie1_1() {printf(  »1\n");}
…..
void main() {
char Entree;
Etat=0 ; // état initial
do {
        printf("  Etat=%d\n",Etat);
        printf("  Commande==>");
        Commande=getche();
        switch(Etat) {
                case 0 :
                        switch (Entree) {
                                case '0': Sortie0_0(); Etat=0; break;
                                case '1': Sortie0_1(); Etat=1; break;
                        }break;
                case 1 :
                        switch (Entree) {
                                case '0': Sortie1_0(); Etat=0; break;
                                case '1': Sortie1_1(); Etat=1; break;
                        }
        }
}while (1);
}/* end main */
```

Un automate de la serrure à code



Problème de programmation :

- ne pas perdre de franchissement
- programme près à réagir en permanence
- notion de tâche sporadique

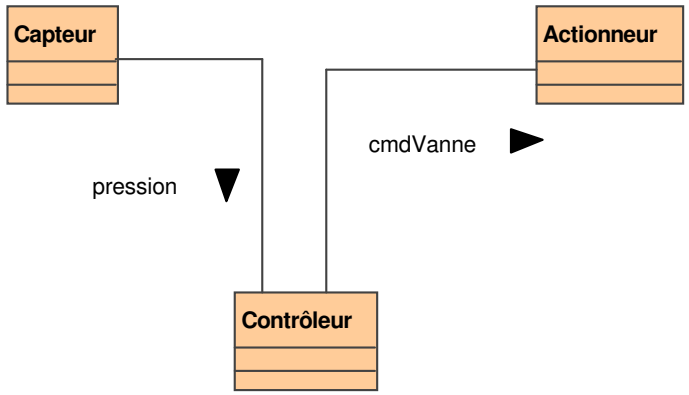
Plan du cours

- Concepts de base pour le temps réel
- Problématique par domaines d'applications
- **Conception de systèmes temps réel**
  - Commande numérique
  - Système logique
  - Entrées – Sorties - horloge
  - Approches Asynchrone /Synchrone
- Résumé
- Réfèrences

Entrées sorties

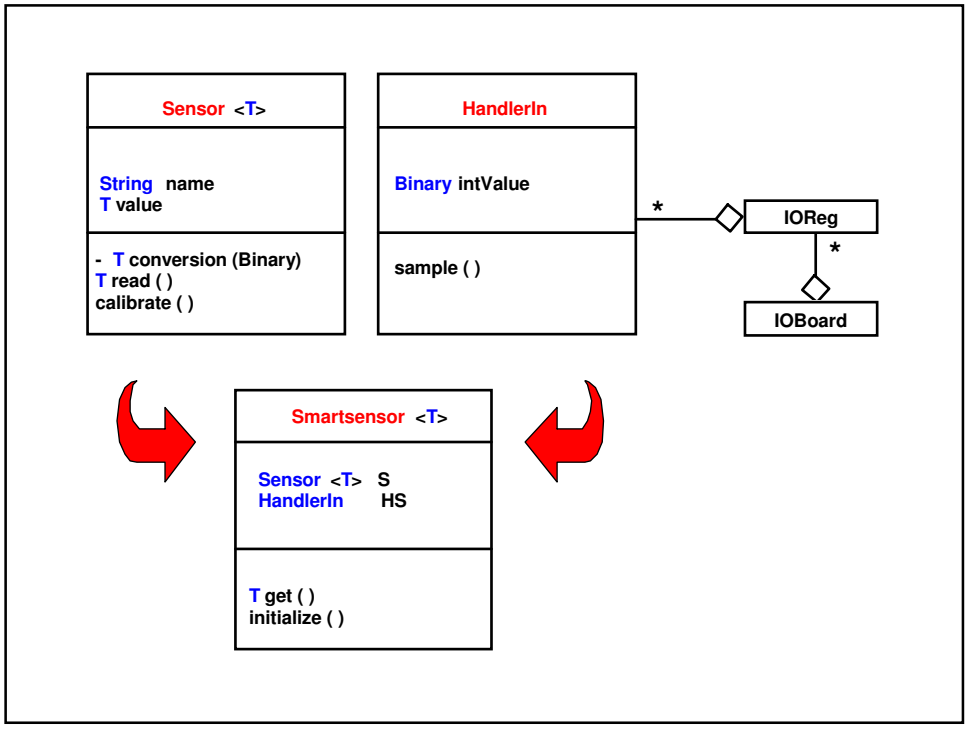
- Capteurs : sensors
  - Conversion Analogique Numérique (CAN)
  - Lecture sur un port d'E/S /Registre
  - Lecture dans une mémoire partagée /DMA
- Actionneurs : actuators
  - Écriture sur un port d'E/S /Registre
  - Écriture dans une mémoire partagée / DMA
  - Conversion Numérique Analogique (CNA)
- Horloge
  - Cadence le programme
  - Horloge locale au microprocesseur
  - Horloge externe

Objet Capteurs et Actionneurs



Types

```
#typedef unsigned Address;
 struct Sensor {
    char *name;
    T value;
    Address port;
    UByte mask;
};
 struct Actuator {
    char *name;
    T value;
    Address port;
    UByte mask;
};
```



Méthodes

// Capteurs (Sensors)

void sample(struct Sensor *); *// échantillonnage par le handler*

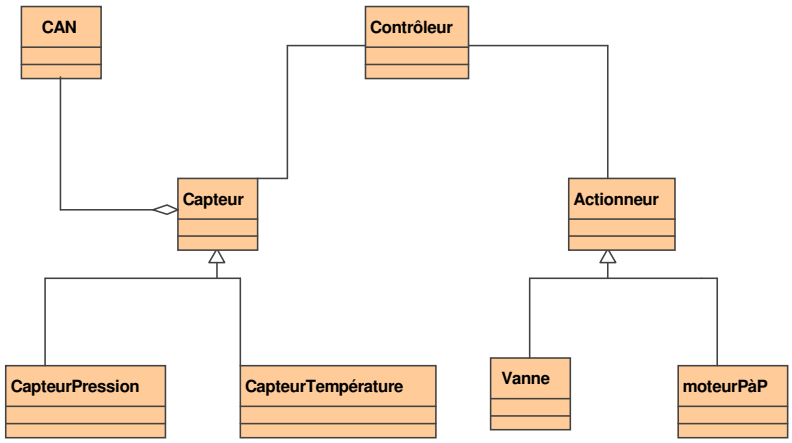
T readSensor(struct Sensor *); *// lecture par le programme*

// Actionneurs (Actuators)

void strobe(struct Actuator *); *// écriture par le programme*

void writeActuator(struct Actuator *,T); *// émission vers l'extérieur par le handler*

Objet (suite)



Scrutation (Polling)

loop

if cond **then**

action

end if

end loop

// Exemple

T getSensor(struct Sensor*); // Enchaîne l'échantillonnage puis lecture vers le programme

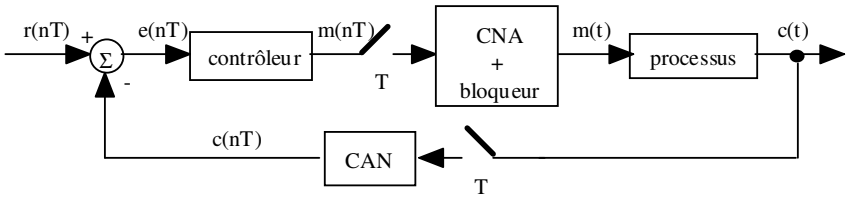
void putActuator(struct Actuator*,T); // Enchaîne opération d'écriture puis émission à l'extérieur

....

Scrutation (Polling)

- Solution la plus simple
- Possible dès lors qu'il s'agit d'un processeur dédié à un périphérique unique.
- Attente active

Exemple du PID



Scrutation (suite)

```
/* définir les structures (Sensor, Actuator, ....) */
    ....
int main() {
  /* déclarer les capteurs, actionneurs, pid */
  struct Sensor T;
    ....
  /* déclarer les autres variables locales */
  double e, r, c, m;
  /* initialisations */
  ....
```

Scrutation (fin)

```
// boucle sans fin
  while (true) {
    if (occurrence d'un top d'horloge) {
      c = getSensor(&T); // échantillonner les entrées
      r = getSensor(&C);
      e = r - c;
      m = controlPID(&P,e); // PID
      putActuator(&Q,m); // exercer la commande
    } // endif
  } // endwhile
}
```

Contrôle par phases

```
switch state
case S1:
        Phase1
        state -> S2;
        break;
case S2:
        Phase2
        state -> S3;
        break;
....
end switch
```

Contrôle par tables

```
....
#define badState 4
....
#include ....
int main() {
        // Instanciations des capteurs, actionneurs
        SmartSensor<Byte> pIn("control","pIn",....);
        SmartActuator<Byte> pOut("control","pOut",....);
        // Instanciations des tableaux
        Tableau etatSuivant(badState,4,4);
        Tableau sortie(badOutput,4,4);

        Byte state, nextState, i, o;
```

Contrôle par tables (suite)

```
// initialisations
  etatSuivant(0,0) = 0;
  ....
  // regime permanent
  while (1) {
        i = pIn.get();
        nextState = etatSuivant(state,i);
        if (nextstate == badState) break;
        o = sortie(state,i);
        if (o == badOutput) break;
        pOut.put(o);
        state = nextState;
  } // endwhile
}
```

Coroutines

- Multiplexage des exécutions
- Création d'un dispatcher chargé de donner la main aux tâches à tour de rôle (Round-robin)
- Une tâche qui n'a rien à faire rend la main immédiatement
- \+ : facile à implémenté, équité, tâches écrites indépendamment
- \- : difficulté à découper les tâches en phase, utilisation (dangereuse) des variables globales.

C'est le programme qui décide des interactions avec l'extérieur

Coroutines

```
tâche T1:                    tâche T2:
  loop                         loop
    switch state1                switch state2
      1: phaseT1.1                 1: phaseT2.1
      2: phaseT1.2                 2: phaseT2.2
        ....                         ....
      n: phaseT1.n                 m: phaseT2.m
    end switch                   end switch
  end loop                     end loop
```

Pilotage par les interruptions

- Amélioration de la réactivité
- L'environnement peut provoquer des exécutions : *Interrupt-driven system*
- Rappel sur les interruptions *:*
  - *Logicielles ou matérielles*
  - *interruptions autorisées (démasquées) ou pas (masquées)*
  - *Déroutement de l'exécution du programme vers une routine d'interruption (interrupt handler)*
  - *Sauvegarde au préalable du contexte du programme en cours et restauration à la fin de la routine d'interruption (Context switching)*
    - *Registres, compteur de programme, registres du coprocesseur pages mémoire, memory-mapped I/O, variables*

Pilotage par interruptions

- Principe de mise en œuvre :
  - Programme principal :
    - Initialise les interruptions (périodique, sporadique ou hybrides)
    - boucle sans fin qui ne fait rien

Pilotage par interruptions

main :

init

loop end loop

interrupt handler H1 :

save_context

<task_1>

restore_context

....

interrupt handler Hn :

save_context

<task_n>

restore_context

Interruptions : exemple

```
int main() {
   /* déclarations des capteurs, actionneurs, PIDs */
   ....
   /* initialisations */
   ....
   while (1) ; /* boucle sans fin */
}
```

Interruption : exemple

```
void handler_1( ) {
        double c,e,r,m;
        saveContext();
        c = getSensor(&sens1);/* échantillonner les entrées */
        r = getSensor(&cons1);
        e = r - c;
        m = controlPID(&pid,e);/* PID */
        putActuator(&act1,m);/* effectuer la commande */
        restoreContext();
}
```

Horloge

- L'horloge temps réel fournit une indication sur le temps qui s'écoule
- Fréquence généralement déterminée par un composant dédié
- Tick retourne l'heure courante
- On peut réaliser comme cela des exécutions périodiques par scrutation de l'horloge temps réel.

Horloge

```
struct Clock {
        char *name;
        unsigned pulsePerUnit;
        Time currentTime;
};

Time tick(struct Clock*);
```

Exécution périodique

```
#define per ....
....
int main() {
        /* déclarations des capteurs, actionneurs, PIDs */
         ....
        struct Clock RTClock;
        Time time, nextTime;
         /* initialisations */
         ....
         nextTime = tick(&RTClock) + per;
        while (1) { /* loop for ever */
                while ( (time=tick(&RTClock)) < nextTime );
                 .... /* get samples */
                 .... /* perform control */
                 .... /* set actuators */
                nextTime = time + per;
         } /* end loop */
}
```