ALASCA — Architecture logicielles avancées pour les systèmes cyber-physiques autonomiques

© Jacques Malenfant

Master informatique, spécialité STL – UFR 919 Ingénierie

Sorbonne Université
Jacques.Malenfant@lip6.fr

Cours 9

De l'entité de contrôle — principes et conception

Objectifs pédagogiques du cours 9

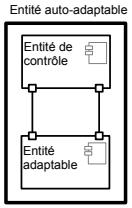
- Situer l'entité de contrôle dans les systèmes cyber-physiques et les systèmes autonomiques.
- Étudier les différents types de contrôleurs et de contrôle pouvant servir de fondements au développement des entités de contrôle.
- Comprendre les approches pour implanter les contrôleurs réactifs typiques en automatique, et en tirer les leçons pour la mise en œuvre des entités de contrôle.
- Étudier l'expérience acquise et les propositions architecturales faites en robotique autonome, domaine ayant des hypothèses et contraintes similaires à celles des systèmes auto-adaptables.
- Synthétiser les différentes architectures génériques possibles pour les entités de contrôle et quelques règles permettant de juger de leur adéquation avec les différents besoins des systèmes auto-adaptables.

Plan

Entité de contrôle

- Dans les systèmes cyber-physiques :
  - les entités logicielles de base de la partie « cyber » sont en contact avec le monde physique à travers des capteurs et des actionneurs ;
  - ces entités de base sont donc effectivement des entités de contrôle vis-à-vis d'entités situées dans le monde physique.
- Dans les systèmes autonomiques :
  - les éléments gérés ne sont pas nécessairement connectés au monde physique mais ils sont gérés (au sens d'IBM) via des capteurs et des actionneurs ;
  - les entités de contrôle gérant/contrôlant les éléments gérés sont appelées *gestionnaires autonomiques*.
- Ainsi entités de base connectées au monde physique ou gestionnaires autonomiques exercent un contrôle et emploient les mêmes techniques de contrôle issues de l'automatique.

Entité auto-adaptable = entité de contrôle ○ entité adaptable



- En s'inspirant de l'architecture de référence d'IBM, l'*entité auto-adaptable* est décomposée en une *entité adaptable*, avec ses capteurs et actionneurs, et une *entité de contrôle* conférant l'autonomie de décision et d'adaptation.
- Ceci assure une meilleure *modularité* et une meilleure *séparation des préoccupations*.
- Pour IBM, l'entité auto-adaptable est dénommée *élément autonomique*.
- Par contre, dans les systèmes cyber-physiques, le couple entité adaptable/ entité de contrôle est rarement identifié en tant que tel mais simplement vu comme le système lui-même.
- Par ailleurs, rappelons que dans les systèmes cyber-physiques *et* autonomiques, il y a bel et bien deux niveaux de contrôleurs :
  1. l'élément géré est un contrôleur qui agit sur le monde physique et
  2. le gestionnaire autonomique est un contrôleur vis-à-vis de l'élément géré.

Boucle de contrôle des systèmes continus

- Problème : passage d'un modèle de contrôle en temps continu *i.e.*, par équations, à un contrôleur numérique.
- Une petite leçon d'humilité : historiquement tout analogique !

Régulateur de Watt : asservissement de l'entrée de vapeur à la rotation de la machine à vapeur ; plus le moteur tourne vite, plus les boules s'écartent par effet centrifuge, plus le haut du pantographe descend pour fermer le soupape, et vice versa.
Les paramètres de la loi de contrôle se retrouvent dans la taille des engrenages et celle du pantographe ainsi que dans le poids des boules...

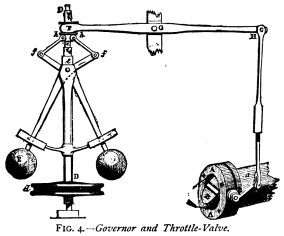


FIG. 4.—*Governor and Throttle-Valve.*

- Avec l'arrivée du contrôle numérique, il fallait passer de la composition de signaux à un calcul sur ordinateur, par nature un phénomène en temps discret sur des données discrètes, avec conversions analogique/numérique puis numérique/analogique.

Boucle de contrôle numérisée

- Contrôler revient alors à intervenir régulièrement sur le système contrôlé pour récupérer les valeurs de capteurs, calculer l'action selon la loi de commande puis appliquer cette action.
- Passer à un contrôleur numérique supose donc :
  1. de programmer l'intervention sous la forme d'une tâche et
  2. l'ordonnnancer pour exécution de telle manière à respecter toutes les contraintes imposées par le système contrôlé, délais de réaction ou accès en exclusion mutuelle à des ressources.
- Au cœur d'un contrôleur, il y a donc des tâches à exécuter dans un mode temps réel ayant donc une durée d'exécution bornée pour pouvoir garantir les délais de réaction.
  - Si certains calculs nécessaires au contrôle ne peuvent pas être bornés en temps d'exécution, ils doivent être effectués en dehors des tâches temps réel, leurs résultats étant rendus accessibles aux tâches temps réel de manière asynchrone.
  - Important pour des contrôles « intelligents » qui requièrent des temps d'exécution non bornables.

Types de contrôleurs (rappels)

On distingue classiquement trois grands types de contrôleurs :

1. Réactif : se base (presqu')uniquement sur les données recueillies via les capteurs au moment de la décision
   $\Rightarrow$ *i.e., contrôleurs de l'automatique.*
2. Délibératif : utilise un état (lourd) construit sur l'historique des données recueillies et des données préalables que viennent compléter les données des senseurs les plus récentes
   $\Rightarrow$ *i.e., contrôleurs « intelligents » apparus en IA/robotique.*
3. Hybride : combine réactif et délibératif selon différents modes
   - en contrôle hiérarchique : le contrôleur délibératif peut intervenir d'autorité, directement sur l'entité adaptable, par-dessus la tête du contrôleur réactif ;
   - en contrôle adaptatif : le contrôleur délibératif adapte le contrôleur réactif pour ajuster ses réactions à la situation courante.
   - un mélange des deux précédentes approches.

Types de contrôle

- On peut aussi distinguer deux grands modes de contrôle :
  1. à fréquence fixe (périodique) : le contrôleur supervise l'entité adaptable à *intervalle régulier*, décide si une adaptation est nécessaire, et si oui exécute une action d'adaptation sur cette dernière (variante : selon un plan irrégulier) ;
  2. par événements (apériodique) : le contrôleur est *passif* et *réagit aux événements* produits par l'entité adaptable ou l'environnement à des instants discrets.

  Nota : concept *indépendant* de la technique d'implantation des capteurs ; on peut échantillonner à intervalle régulier pour détecter des événements, et on peut générer des événements à intervalle régulier.

Voyons maintenant les inspirations et leçons à tirer de la mise en œuvre des contrôleurs réactifs (ou avec un état très simple), qu'ils soient à fréquence fixe ou par événements.

Plan

Contrôleur réactif de base

- Contrôleur « SISO » simple : *Single Input, Single Output*, c'est-à-dire une seule entrée, une seule sortie.
- Chaque réaction implique quatre étapes :
  1. Obtenir la valeur du capteur.
  2. Calculer, à partir de cette valeur, celle de la commande en appliquant une loi de commande.
     Exemple : avec la nouvelle valeur v du capteur et les précédentes, un contrôleur PID calcule l'intégrale i et la dérivée d du signal, puis le résultat de $K_p v + K_i i + K_d d$.
  3. Appeler l'actionneur avec la valeur calculée.
  4. Attendre que l'effet de l'action se soit fait sentir et que le capteur soit à nouveau disponible pour recommencer.
- Par rapport à un contrôleur analogique, l'entrée et la sortie prennent des valeurs discrètes à des *instants ponctuels*.
  - Selon l'application elles sont interprétées en continu comme valides ponctuellement (*i.e.*, fonctions par impulsions) ou jusqu'à l'obtention d'une nouvelle valeur (*i.e.*, fonctions en escalier).

Réalisation périodique du contrôleur de base

- En contrôle *périodique*, la « réaction » est exécutée à intervalle régulier, selon un cycle de durée fixe prédéterminée couvrant :
  1. l'exécution de l'appel du capteur, plutôt en mode « pull »,
  2. le calcul de la commande puis
  3. l'exécution de l'appel de l'actionneur avec la commande calculée.
     - En temps réel dur, il est *crucial* de disposer d'un système d'exploitation *temps réel* pour activer la tâche périodique avec grande précision temporelle.
- La réalisation informatique du contrôleur rend visible les *durées* des trois opérations :

← P →							
Appel capteur	Calcul commande	Appel actionneur	Attente (Libre)	Appel capteur	Calcul commande	Appel actionneur	Attente (Libre)
←	T	→					

T : cycle de calcul, P : période d'activation

La partie « Attente » donne le temps nécessaire pour que les effets de l'actionneur se réalisent et deviennent visibles à la lecture suivante du capteur (inertie de l'actionneur et du capteur) ... On pourra y caser d'autres calculs.

Ordonnançabilité et support système

- Définition : sur un système temps réel, capacité à ordonnancer l'ensembles des tâches à exécuter en respectant toutes leurs contraintes temps réel.
- Dans un contrôleur SISO périodique, la condition d'ordonnançabilité est très simple : $T < P$.
- Pour que cette condition soit valide, il faut estimer le *temps d'exécution en pire cas (WCET)* du cycle de calcul, incluant
  - l'exécution du capteur (*i.e.*, le délai d'obtention de la valeur),
  - le calcul de la commande (*i.e.*, durée du calcul),
  - et l'exécution de l'actionneur (*i.e.*, le temps de réalisation de la commande).

  ce qui n'est pas si simple, même pour la partie calcul, vu la complexité du fonctionnement des processeurs aujourd'hui !

Nota : si $T < P$ et qu'il n'y a qu'un calcul simple en une seule tâche à faire, point n'est besoin d'un matériel et d'un système d'exploitation complexes : une simple horloge temps-réel fonctionnant comme une interruption et transférant l'exécution à la tâche est suffisant, d'où les *micro-contrôleurs industriels*.

Résumé des premières leçons I

- La période du contrôleur ne peut être inférieure à la durée du cycle de lecture, calcul, actionnement pour que chaque cycle se termine complètement avant le début du cycle suivant (un « petit » débordement peut souvent être toléré).
- La mise en œuvre d'un contrôleur réactif périodique exige de pouvoir *estimer et borner* les durées d'exécution des appels aux capteurs et aux actionneurs ainsi que la durée du calcul de la loi de commande elle-même.
- Dans un contexte d'architecture logicielle, ces durées doivent être estimées dans le *contexte du déploiement*.
  - *D'où l'intérêt de définir ces éléments parmi les méta-propriétés des capteurs et actionneurs de l'entité adaptable et de les rendre profilables au déploiement... (§Cours 7 et 8).*
- Par ailleurs, la période doit aussi être déterminée par rapport à la dynamique de l'application elle-même :

Résumé des premières leçons II

  - à une période donnée et un taux de variation des valeurs des capteurs correspond l'*ampleur* de la variation *observable par cycle* ;
  - à cette période et selon les capacités de l'actionneur, limitées par la durée du cycle également, correspond une variation pouvant être *effectivement contrôlée* par cet actionneur.

  Le contrôleur doit être suffisamment rapide (i.e., période assez courte) et puissant pour « absorber » la dynamique de l'application.
- D'où l'intérêt pour le contrôleur de *borner la dynamique* qu'il sera capable d'absorber : taux et amplitude de variation de l'entrée en fonction de la durée du cycle et de la puissance des actionneurs. Tout dépassement de ces limites devra être traité par une technique *ad hoc* indépendante.
  - Ex.: laisser tomber des requêtes pour éviter l'écroulement si les ressources ne peuvent être augmentées assez rapidement ; faire du délestage dans les réseaux électriques.

Contrôleur à entrées et sorties multiples

- Rares sont les applications à entrée et sortie uniques ; souvent plusieurs actionneurs doivent être pilotés à partir de plusieurs capteurs avec des contraintes de temps pouvant être *différentes* pour chaque capteur, actionneur et phénomène contrôlé.
- Dans le contexte d'un contrôleur réactif périodique sur *mono-processeur/mono-cœur*, l'ensemble des cycles de calcul associés à chacun de ces contrôles vont devoir être *sérialisés* pour être exécutés séquentiellement.
- Cette séquentialisation mène à un *ordonnancement* des différents calculs au sein d'un même cycle global.
  - Sur multi-processeurs ou multi-cœurs, il y aurait plusieurs cycles *parallèles*, cas que nous ne regardons pas ici.
- Deux cas importants se présentent :
  1. contrôleur à cycle (période) unique, ou
  2. contrôleur à cycles (périodes) multiples *séquentialisés*.

Contrôleur à cycle (période) unique I

- Cas où une période unique P d'un *macro-cycle* peut être définie où toutes les tâches pourront être exécutés.

←		P	→				
C_1	...	C_n	Libre	C_1	...	C_n	Libre

C_i : tâches, P : période d'activation

- Soit $T_i = \text{durée}(C_i)$, la condition d'ordonnançabilité assez simple est : $\sum_{i=1}^{n} T_i \leq P$.
- Points délicats potentiels :
  - partage des valeurs de capteurs, surtout si ces derniers ne peuvent être appelés plusieurs fois pendant le même cycle ;
  - dépendances des valeurs mémorisées entre les calculs et les cycles, qui peuvent demander de fixer un ordre dans lequel les procédures de calcul C_1, ..., C_n sont appelées ;
  - conflits sur les actionneurs si des contrôles multiples impactent potentiellement les mêmes actionneurs.

Il faut alors coordonner les différents contrôles...

Contrôleur à cycle (période) unique II

- L'implantation la plus simple rassemble dans une même procédure tous les calculs, ce qui revient à *fondre* tous les contrôles à réaliser en *un unique modèle de contrôle*.
  - le plus efficace et le plus sûr, car trivial à ordonnancer et résoud manuellement tous les conflits potentiels...
  - ...mais produit des contrôleurs enchevêtrés difficiles à mettre au point et ensuite à maintenir.
- Une alternative plus souvent utilisée conserve des tâches séparées et les ordonnance dans un seul cycle, quitte à expliciter les dépendances temporelles et de production/consommation de données entre les tâches pour pouvoir en tenir compte.
  - Ex.: si on définit un ordre partiel $\prec$ entre les tâches, *i.e.*, $C_i \prec C_j$ indique que la tâche C_i doit précéder C_j dans le cycle, alors un tri topologique des tâches produit un ordonnancement correct (si bien sûr $\sum_{i=1}^{n} T_i \leq P$).

Contrôleur à cycles (périodes) multiples I

- Cas se produisant lorsque les dynamiques des différents processus à contrôler imposent différentes périodes de contrôle.
- Soient $(T_i, P_i), 1 \leq i \leq n$ les durées des micro-cycles et leurs périodes, avec des P_i entiers (sans perte de généralité), on peut calculer une période de base P_b et une super-période P_s :

$$P_b = PGCD[P_1, ..., P_n]$$
$$P_s = PPCM[P_1, ..., P_n]$$

  et utiliser ces valeurs pour construire un *macro-cycle*.
- Trois stratégies de réalisation du macro-cycle :
  1. Si $\sum_{i=1}^{n} T_i \leq P_b$, un ordonnancement hétérogène peut être produit sur la super-période P_s où chacun des cycles n'exécute que les C_i nécessaires selon leurs périodes $P_i = k_i P_b$ respectives.

Ex. : $(T_1, P_1) = (0.2, 2)$, $(T_2, P_2) = (0.3, 3)$, $PGCD[P_1, P_2] = 1$, $PPCM[P_1, P_2] = 6$, $T_1 + T_2 = 0.5 < 1$

C_1, C_2		C_1	C_2	C_1	

Contrôleur à cycles (périodes) multiples II

2. Si $\sum_{i=1}^{n} T_i > P_b$, on peut subdiviser statiquement chacun des C_i à période $P_i > P_b$ en parties $C_{i1} \ldots C_{ik_i}$ exécutées dans chaque cycle de base sur les k_i cycles de base de leur période propre P_i (sous la condition $\sum_{i=1}^{n} T_{ik} \leq P_b, 1 \leq k \leq P$ où $T_{ik} = \text{durée}(C_{ik})$)).

   Ex. : $((T_{11}, T_{12}), P_1) = ((0.5, 0.5), 2)$, $(T_2, P_2) = (0.3, 3)$,
   $PGCD[P_1, P_2] = 1$, $PPCM[P_1, P_2] = 6$, $T_{11} + T_{12} + T_2 = 1.3 > 1$

C_{11}, C_2	C_{12}	C_{11}	C_{12}, C_2	C_{11}	C_{12}

3. La troisième stratégie consiste à s'en remettre à l'ordonnanceur d'un système d'exploitation temps réel pour (1) activer cycliquement les C_i et (2) subdiviser dynamiquement les C_i trop longs par leur suspension et le changement de contexte.
   - *Nota : l'ordonnaçabilité dynamique et les différentes politiques d'ordonnancement capables d'exécuter tous les contrôles sous contraintes est un sujet qui a été énormément étudié pour les systèmes temps réel mais que nous ne couvrirons pas ici.*

- Les points délicats du cas SISO demeurent, surtout avec la stratégie dynamique dont les ordonnanceurs ont des difficultés à assurer le respect des dépendances entre les calculs.

Leçons complémentaires

1. Le partage potentiel de capteurs entre les contrôles montrent la nécessité de tenir compte des appels multiples pouvant être faits à des intervalles très courts sur un même capteur ; lorsque les capteurs ne le permet pas, il est important de pouvoir planifier ces appels et mémoriser les résultats.
2. L'architecture interne des calculs des différents contrôles doit être prévue pour les faire cohabiter dans les mêmes cycles, et leurs dépendances (temporelles, accès aux ressources, etc.) être explicitées pour les prendre en compte dans l'ordonnancement.
3. Les réalisations les plus simplistes de contrôleurs à périodes multiples exigent généralement l'indépendance entre les actionneurs appelés par les différents contrôles ; sans cette indépendance, une coordination devra s'exercer entre les différentes actions déclenchées sur les mêmes actionneurs.

Contrôleurs réactifs par événements

- Deuxième grande approche de contrôle, applicable lorsque les capteurs produisent des données à des instants imprévisibles, d'où une *inversion de contrôle* logique par rapport à l'approche périodique.
  - *Nota : C'est souvent une vision abstraite de la réalité, car dans le contexte informatique un événement ne se déclenche pas tout seul, il doit être détecté par un processus qui vérifie à une certaine fréquence si un événement s'est produit pour déclencher sa notification. Il y a donc généralement un processus périodique masqué dans le contrôle par événements discrets.*
- Les modèles du contrôle par événements discrets sont le plus souvent fondés sur les *automates*, traitant les événements comme les entrées d'un système à base d'états et de transitions.
  - Ex.: Statecharts (automates d'états finis parallèles et hiérarchiques), langages synchrones (produisant des automates ou un code équivalent par compilation).

Principe général de réalisation et problèmes engendrés

- Sa dynamique modélisée par automate, le contrôleur reçoit un événement, déclenche la transition exécutant les calculs de contrôle et produisant la sortie actionneur calculée, passe dans l'état suivant et recommence avec l'événement suivant.
  - C'est un modèle *temporel* (et non *temporisé*) sans explicitation du temps réel, ni dans les instants d'occurrences, ni dans les durées d'exécution des mesures, des calculs et des actions.
- La conversion d'un tel modèle temporel en une implantation *temporisée* présente deux difficultés théoriques et pratiques qui sont les deux faces d'une même médaille :
  1. l'impossibilité pour un contrôleur *réel* de traiter une quantité non bornée d'événements se produisant en un temps fini ;
  2. la possibilité *théorique* pour un contrôleur d'exhiber un comportement dit « *Zeno* » consistant à faire une quantité non bornée de transitions dans un temps aussi petit que l'on veut.
- Ce passage nécessite donc de *borner (inférieurement) le délai entre deux occurrences successives d'événements ou (supérieurement) le nombre d'événements pouvant se produire au même instant.*

Du modèle temporel à une implantation temporisée

- S'il est possible d'imposer une borne inférieure T_{inf} entre deux occurrences successives d'événements e_i, e_j, *peu importe leur type*, telle que tous les temps d'exécution en pire cas des calculs T_i déclenchés sont inférieurs à T_{inf}.
  Alors, chaque événement peut être traité avant le suivant.
- Comment satisfaire cette contrainte ?
  - Par la nature de l'application, qui permettrait de prouver l'existence de cette borne inférieure et la calculer.
  - En fixant une période d'échantillonnage des événements P égale ou supérieure à T_{inf} et en utilisant un tampon pour conserver les événements occurant durant chaque intervalle P et les exécuter selon un ordre de priorité respectant leurs contraintes de réaction respectives (ordonnancement).
- Sinon, avec des mécanismes d'interruption avec priorité, le traitement d'un événement moins prioritaire pouvant être suspendu le temps de traiter un événement plus prioritaire.

Implantation des contrôleurs par événements discrets I

- Stratégie dirigée par le temps (*time-triggered*) :
  - Considérant la nécessité d'un processus d'échantillonnage périodique pour détecter les occurrences d'événements, on détermine une période d'échantillonnage qui fixe un cycle de contrôle.
  - Devient similaire au contrôle périodique, ce qui en fait une bonne candidate pour des contrôleurs mixtes de systèmes hybrides contrôlant périodiquement les phénomènes continus et échantillonnant en même temps les occurences d'événements.
- Stratégie dirigée événements (*event-triggered*) « pure » :
  - Chaque contrôleur, pour chaque type d'événements, est programmé séparément, et invoqué par le système d'exploitation temps réel lorsque les événements sont notifiés ;

Implantation des contrôleurs par événements discrets II

  - Cette approche est similaire à une des approches multi-périodiques qui s'en remet au système pour ordonnancer correctement et jouer sur les interruptions et changements de contexte pour réagir aux événements.
  - Elle peut aussi se combiner avec un contrôleur périodique en considérant les tops d'horloge comme des événements.
- Globalement, le principal problème de ces stratégies d'implantation demeure la difficulté de garantir une réaction aux évenements respectant les délais maximaux pour les différents types d'événements, d'où :
  - des approches temps réel dures, où les contraintes temporelles de réaction *doivent absolument* être observées, et
  - des approches temps réel dites *molles*, où les contraintes peuvent être ponctuellement violées si statistiquement elles sont observées en moyenne.

Difficulté sémantique

- Une autre difficulté, de nature plus sémantique, lors du passage d'un modèle temporel à une implantation temporisée est l'ordre de traitement des événements détectés simultanément.
- Pour les systèmes dont l'état dépend de l'ordre de traitement de ces événements, un indéterministe lié aux conditions de compétition (*race conditions*) entre événements va apparaître.
- Première idée : n'utiliser que des contrôleurs *confluents*.
  - Confluence : soient e_1 et e_2 deux évenements et s_0 l'état courant du contrôleur, ce dernier est confluent si et seulement si le traitement de e_1 puis de e_2 donne le même état s_1 que le traitement de e_2 puis de e_1.

  L'ordre d'occurrence devient ainsi sans importance.
- Sinon, il faudra imposer une confluence artificielle en fixant un ordre de traitement par un mécanisme de priorités, par exemple.

Leçons à tirer

- Malgré leur simplicité apparente, les contrôleurs par événements discrets soulèvent des problèmes profonds quand le délai entre deux occurrences successives d'événements entre en conflit avec la durée d'exécution complète des calculs associés à chaque événements.
- *Il faut éviter la politique de l'autruche !*
  - Au minimum, il faut vérifier si l'hypothèse sous-jacente que le délai est borné inférieurement est justifiée par l'application.
  - Si la réponse est non, il faut adopter une stratégie d'implantation garantissant une borne inférieure *ad hoc*, tout en recherchant la propriété de confluence du contrôleur (et on a donné des pistes...).
- Qu'on adopte une stratégie d'implantation dirigée par le temps ou par les événements, il reste possible de marier un contrôle par événements avec un contrôle périodique.

Passage au contrôle réparti

- La plupart des architectures auto-adaptables sont naturellement réparties, ce qui suppose donc d'étendre la modèle de contrôle au cas du contrôle réparti.
- Mais le passage au réparti aggrave les problèmes rencontrés dans les contrôleurs centralisés :
  1. la variabilité dans les durées des calculs et des réactions, et
  2. l'ordre d'occurrence ou de détection des événements,

  ceci à cause :
  1. de la variabilité des durées de communication, et
  2. des dérives entre les horloges locales des différents nœuds.

Implantation des contrôleurs répartis I

- Deux principales stratégies d'implantation :
  1. À horloges locales :
     - chaque nœud de contrôle exécute un contrôleur périodique (multi-périodique ou par événements), et
     - ce contrôleur échantillonne son environnement constitué du monde physique et des événements produits par les autres nœuds selon son horloge locale.

     Le grand avantage de cette stratégie est sa simplicité d'implantation, n'utilisant que des procédés déjà évoqués pour les contrôleurs centralisés, mais sa limite apparaît lorsque les dérives et délais entre les nœuds se rapprochent voire dépassent les périodes des contrôleurs et qu'il devient quasi-impossible de garantir les contraintes temps réel des contrôles.

Implantation des contrôleurs répartis II

  2. À horloge globale :
     - un bus diffuse un signal d'horloge globale à tous les nœuds,
     - toutes les communications sont planifiées à des instants précis, garanties sans collision et donc sans délai imprévisible et
     - tous les nœuds de contrôle exécutent un contrôleur périodique non-préemptif selon l'horloge globale.

     Le grand avantage de cette stratégie est son déterminisme, mais elle requiert des ressources importantes pour la synchronisation d'horloge, elle est moins flexible et l'expérience montre qu'elle ne passe pas à l'échelle au-delà de quelques dizaines de nœuds.
- Une stratégie mixte se dessine, composant des sous-systèmes à horloges globales à l'intérieur de chaque sous-système, mais locales lorsque vues entre sous-systèmes (vision GALS).

Leçons

- Le contrôle réparti exacerbe les problèmes rencontrés car garantir les propriétés du contrôle réclame un grand déterminisme dans les durées des communications, ce qui est souvent impossible dans les systèmes autonomiques sous réseau IP.
- Une heuristique évidente mais dangereuse consiste à s'assurer que les délais de communication sont petits par rapport à la période des contrôleurs (et au délai minimal entre les occurrences d'événements).
- Sinon, le système tombe dans le domaine du *contrôle en réseau*, domaine ouvert qui fait encore aujourd'hui l'objet de nombreux travaux de recherche.
- Une autre approche consiste à trouver une solution du point de vue architectural (type GALS) : assurer la colocalisation des contrôleurs à haute fréquence et de l'entité contrôlée, quitte à recourir à des contrôleurs hybrides délibératifs/réactifs (voir ci-après).

En résumé : conception d'un contrôleur

1. Définir le modèle de contrôle
2. Déterminer le type de contrôle à effectuer, les événements le cas échéant, et les contraintes temporisées.
3. Analyser le modèle pour déterminer les plages de validité de ses propriétés (stabilité, etc.)
4. Programmer le contrôleur.
5. Tests et validation.
6. Déploiement : identification, profilage et configuration finale du contrôleur.

Plan

Introduction

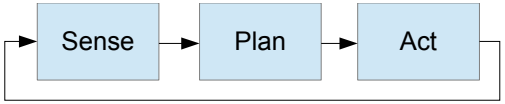
- Terme apparemment dû à Genesereth et Nilsson en 1987 qui l'ont utilisé pour parler des capacités de raisonnement logique des *agents* appelés *délibératifs*.
- Son utilisation s'est ensuite étendue en *robotique autonome* par opposition aux architectures *réactives* mais dans la suite des architectures précédentes dites *hiérarchiques*.
- Deux visions développées en parallèle :
  - dans le monde des agents, qui s'est principalement intéressé à des systèmes où le raisonnement intelligent prime sur la notion de contrôle et sur les contraintes temporelles de réaction ;
  - en robotique, où les travaux liés à une réalisation cyber-physique, ont dû prendre directement en compte ces aspects contrôle.
- Malgré l'origine « agents » du terme, les architectures pour la robotique autonome ont plus de leçons à apporter sur la conception d'architectures de contrôle en général.

Primitives de la robotique autonome

- Pour mieux comprendre les différentes architectures génériques proposées historiquement en robotique autonome, il faut d'abord s'intéresser aux principales fonctions d'un robot autonome.
- Elles sont résumées par trois primitives ou catégories de fonctions de base :
  1. **Percevoir** (*sense*), la fonction sensorielle par laquelle le robot acquiert la connaissance du monde et de son environnement.
  2. **Planifier** (*plan*), la fonction décisionnelle par laquelle le robot décide de l'enchaînement de ses actions sous la forme d'un plan, sur la base des connaissances acquises.
  3. **Agir** (*act*), la fonction produisant les commandes sur les actionneurs, à partir, du plan faisant agir le robot dans son environnement.
- C'est autour de l'organisation de ces trois fonctions que sont déclinées les principales architectures génériques de la robotique autonome.

Les architectures hiérarchiques

- Prévalentes entre 1960 et 1990, ces architectures proposent un comportement cyclique enchaînant simplement les trois primitives : percevoir, planifier, agir.



- Elles s'inspirent de notre compréhension du comportement conscient de l'humain, propulsant au premier rang le *raisonnement* comme réalisation de la fonction de *planification*.
- Caractéristiques essentielles de l'approche hiérarchique :
  - les données des capteurs servent d'abord à créer et alimenter le « modèle » du monde (ex.: carte du l'environnement du robot) grâce à une algorithmique souvent lourde et complexe ;
  - toute action est planifiée (décidée) à partir de ce modèle.

Exemples de fonctions intégrées à la planification

- Objectif : construction d'un « modèle » du monde monolithique, sous la forme d'une structure de données globale comprenant :
  - des connaissances génériques *a priori*, considérées comme fixes (ex.: cartes et plans, si connus à l'avance) ;
  - des connaissances provenant de la fonction de perception (ex.: position courante, présence d'obstacles) ;
  - des connaissances opérationnelles sur la tâche à réaliser (ex.: position des points de collecte et de livraison).
- Principales fonctions de haut niveau de la partie planification :
  - Traitement des perceptions (dont la vision).
  - Allocation des ressources (calcul mais aussi actions).
  - Localisation et cartographie.
  - Navigation et planification des chemins.
  - Planification des missions.
- Le nom « hiérarchique » vient de l'organisation de la planification en couches interconnectées, du plus concret (localisation relative) au plus abstrait (planification des missions).

Limites de l'approche hiérarchique et première leçon

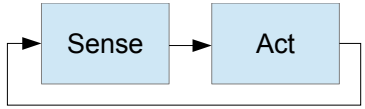
- Longtemps portée par l'intelligence artificielle, l'approche hiérarchique en a subi les contrecoups lorsque les procédés de raisonnement se sont heurtés à de graves difficultés :
  - hypothèse du monde fermé : le robot a une connaissance complète de son monde, mais cela nécessite un modèle difficile à construire et réclamant une quantité de calcul déraisonnable ;
  - problème du contexte (*frame problem*) : comment modéliser les changements dans le modèle du monde en fonction du temps et des actions de manière totale et complète ?
- Pour tout robot réaliste, l'approche hiérarchique exige des temps de calcul incompatibles avec le fonctionnement en *temps réel*.

Première leçon

Le contrôle d'un système cyber-physique ou autonomique sous contrainte de temps est incompatible avec une approche nécessitant, pour prendre chaque décision, le traitement d'une quantité de données importantes dont le durée ne peut être bornée.

Le paradigme réactif : agir promptement

- Inspiré des travaux en éthologie : le comportement animal est souvent fondé sur des *réflexes*, sans raisonnement mais simplement une *réaction* à un *stimuli*.
- Idée : aller au bout de cette logique, en bannissant complètement toute planification au profit de *réflexes*.



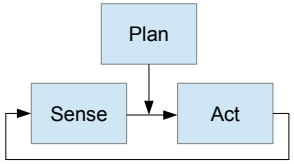
- Le modèle du monde a aussi été supprimé : *le monde est sa meilleure représentation qu'il suffit de « lire » par les capteurs*.
- À chaque réaction, le contrôleur n'utilise que les dernières informations reçues des capteurs, et ne conserve (en théorie) aucun état d'une réaction à l'autre.
- *Le fonctionnement d'un robot réactif est tout à fait similaire à celui des contrôleurs réactifs industriels !*

Naissance, succès et déclin de l'approche réactive

- Programmés selon des techniques qui auraient pu être utilisées dans des contrôleurs industriels, les robots réactifs ont rapidement démontré des capacités étonnantes et inédites dans les années 1990.
- L'ère de la robotique autonome devenait réalité, et elle allait produire ses premiers grands succès culminant aujourd'hui avec les robots aspirateurs qui se vendent par millions dans le monde.
- Néanmoins, l'absence de modèle du monde et de visibilité des actions d'une réaction aux suivantes a limité leurs capacités.
- Au-delà de fonctions de base, les robots réactifs ne peuvent exhiber un comportement suffisamment intelligent pour la plupart des applications envisagées.

Un compromis : les architectures hybrides

- Constatant les limites des architectures précédentes, pourquoi ne pas rechercher le meilleur des deux mondes, réactif et hiérarchique : *les architectures hybrides délibératives/réactives*.
- L'idée de base est assez simple : coupler une architecture réactive, suivant selon un cycle court les contrôles en temps réel des fonctions réflexes, avec une architecture délibérative, chargée des fonctions requérant des connaissances et de la planification ?
- Deux couches, l'une (réactive) avec les fonctions perception/action, et l'autre (délibérative) reprenant la fonction planification.



- Ces couches *coopérent* en donnant au contrôleur délibératif la capacité de modifier les règles du contrôleur réactif pour les adapter de plans en plans.

Entrelacements entre délibératif et réactif

- Principale difficulté à surmonter, l'entrelacement doit être géré conceptuellement, entre les différentes fonctions, et opération-nellement par l'ordonnancement des tâches, la prise en compte des communications inter-couches, etc.
- Les interactions, et la meilleure façon de les résoudre, dépend souvent de la fonction considérée.
- Solution adoptée en navigation par plusieurs architectures :
  - le planificateur découpe les chemins en segments faciles à suivre délimités par des points de passage et il fournit au contrôleur réactif les segments les uns après les autres ;
  - si des informations venant des capteurs invalident le plan (obstacles, ...), le contrôleur réactif peut opter pour une solution purement réactive (évitement d'obstacles), ou remonter au planificateur pour reviser le chemin.
- *Il faut donc des communications et se passer la main dans les deux sens entre les niveaux réactif et délibératif.*

Continuité du contrôle réactif

- Une autre difficulté à surmonter en pratique est la continuité du contrôle réactif, y compris pendant qu'il est modifié, soumis à des contraintes de temps desquelles dépendent le robot.
- La couche réactive elle-même combine plusieurs compétences à tout instant, mais susceptibles d'être modifiées.
- Lorsque le contrôleur délibératif met à jour les compétences du contrôleur réactif, ou encore tarde à le faire à la fin de l'exécution d'un plan, le robot doit maintenir une réactivité à l'environnement.
  - *Que doit faire un drone autonome pendant qu'on met à jour ses compétences pour suivre le prochain segment de sa trajectoire ou le prochain plan de sa mission ?*
- Clairement, il faut maintenir *en permanence* un ensemble de compétences ou de comportements de base capables d'assurer une situation saine et sûre pour le robot par rapport à son environnement.

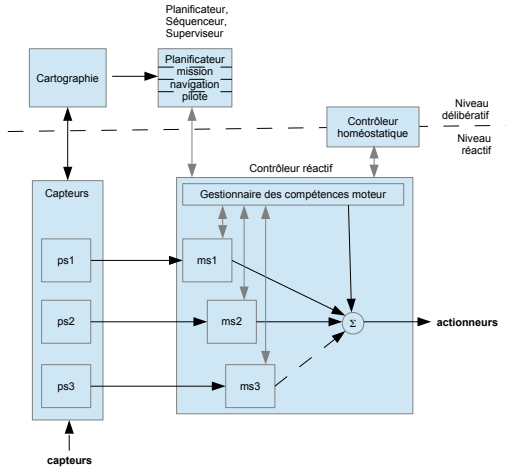
Un autre lien avec les architectures autonomiques

- Notons que depuis les architectures hiérarchiques jusqu'aux couches délibératives des architectures hybrides, les robots autonomes incluent généralement une fonction de supervision de leurs performances.
- L'objectif de cette fonction peut être l'apprentissage, pour améliorer ses plans, ses actions et ses décisions ou encore, dans une architecture hybride, à détecter quand le jeu de compétences actuellement utilisé par la couche réactive n'est plus adapté à la tâche courante de manière à le modifier.
- On se retrouve alors avec des architectures robotiques autonomiques, en fait cyber-physiques *et* autonomiques, c'est-à-dire capables d'auto-adaptation dynamique...

Un exemple d'architecture hybride : AuRA

Autonomous Robot Architecture : *la plus ancienne !*

- Planificateur mission : interface utilisateur.
- Navigateur : génère les chemins, et les subdivise en segments.
- Pilote : génère les « compétences » à partir des segments, et interagit avec le contrôleur réactif pour les mettre en place.
- Le gestionnaire de « compétences » ordonnance les modules de compétences.
- Ces modules lisent les capteurs et génèrent des commandes pour les moteurs, la composition de ces derniers se faisant ici par simple sommation.



Leçons I

- Les architectures hybrides, certes, couplent délibération et réaction
  - pour maîtriser les différentes échelles de temps dans un contrôleur complexe,
  - mais aussi pour maîtriser des problèmes spatiaux de déploiement, comme le fait de déployer un contrôleur réactif au plus près de l'entité adaptable alors que le contrôleur délibératif peut être déployé à distance par rapport à l'entité.
- L'organisation de la coopération entre le contrôleur délibératif et le contrôleur réactif dépend du contrôle à réaliser et peut nécessiter des échanges non seulement du délibératif vers le réactif mais aussi du réactif vers le délibératif.

Leçons II

- Si l'architecture hybride est fondée sur le principe d'une couche délibérative modifiant régulièrement la couche réactive, la programmation de la couche réactive doit permettre des modifications régulières efficaces de son code.
  $\Rightarrow$ *modules échangeables dynamiquement ? réflexion ?*
- L'adaptabilité du contrôleur réactif ne doit pas faire oublier les contraintes de temps du contrôle, et donc l'architecture doit garantir l'observation de ces contraintes en tout temps, y compris *pendant* les phases de modifications du contrôleur réactif.

Plan

Objectif général

- Comme pour l'élément géré, on peut distinguer deux approches de conception des entités de contrôle :
  1. Conception ouverte : la conception de l'entité de contrôle est indépendante de l'entité adaptable, et le contrôleur obtenu est alors susceptible de s'appliquer à plusieurs entités adaptables.
  2. Conception fermée : la conception de l'entité de contrôle se fait avec la connaissance de l'entité adaptable, et alors il est possible de se contenter d'un contrôle spécifique à celle-ci.
- Comme noté pour les éléments gérés, il est difficile de réaliser des entités de contrôle génériques capables de contrôler des entités adaptables très différentes ; la réutilisabilité des entités de contrôle est limitée par le type de contrôle qu'elles mettent en œuvre qui est rarement universellement applicable.
- Une paramétrisation adéquate du contrôleur peut cependant améliorer sa réutilisabilité.
  Ex. : paramètres de la loi de commande, période, mode de fonctionnement des capteurs (« push » versus « pull »), ampleur des adaptations, ...

Rappel des principales architectures génériques

- La conception de l'entité de contrôle est donc principalement guidée par les types de contrôleur et de contrôle réalisés.
- Trois grandes architectures génériques sont proposées de manière plus concrète :
  1. Contrôleur *purement réactif*, inspiré de l'automatique et du paradigme réactif en robotique et dans les systèmes multi-agents ;
  2. Contrôleur *purement délibératif*, inspiré des systèmes multi-agents cognitifs et du paradigme hiérarchique en robotique ;
  3. Contrôleur *hybride délibératif/réactif*, inspiré du paradigme de même nom en robotique, également appliqué dans certains systèmes multi-agents.
- Le choix entre ces dernières est guidé par la présence ou non de fonctions venant se greffer à la stricte application de la loi de commande : supervision, planification, adaptation, coordination, historiques, construction de modèles, etc.

Cycle de conception et de vie de l'entité de contrôle I

1. Élaboration des exigences par l'élicitation des contrôles à mettre en œuvre (cas d'utilisation, qualités souhaitées, ...).
2. Élaboration du *contrat de contrôle*, c'est-à-dire les contraintes ainsi que les engagements réciproques et conjoints entre l'entité adaptable et l'entité de contrôle sur les objectifs et les moyens de ce contrôle.
3. Modélisation du contrôle :
   - Élicitation des propriétés effectives des contrôles (contraintes temporelles, ...), triant entre contrôles réactifs et délibératifs.
   - Élaboration des modèles pour les différents contrôles identifiés.
4. Conception et implantation des contrôles (algorithmique, programmation, ...).

Cycle de conception et de vie de l'entité de contrôle II

5. Intégration de la fonction d'auto-configuration du contrôleur, incluant celle de l'entité adaptable dont les informations doivent être propagées vers les paramètres du contrôle.
6. Test unitaires des fonctions du contrôleur (sous simulateur).
7. Test d'intégration avec des entités adaptables prévues à cet effet.
8. Déploiement et connexion (composition) avec chaque entité adaptable avec leur auto-configuration :
   - réglage des paramètres côté entité adaptable,
   - propagation et réglage des paramètres du contrôle.
9. Connexion avec d'autres entités de contrôle pour la coopération et la coordination.
10. Exécution : déconnexions/reconnexions entre entités de contrôle, reconfigurations, évolutions fonctionnelles.

Préoccupation transversale : la coordination I

Définition

ensemble des actions, accompagnées d'appels de méthodes ainsi que d'échanges de données et de signaux, permettant de réguler les décisions et les actions d'un groupe d'entités.

- Comme déjà vu, la coordination est une préoccupation transversale pour l'entité de contrôle, dans la mesure où il s'agit d'une fonction de support de l'ensemble des autres fonctions :
  - supervision : pour la synchronisation des lectures, l'échanges et la mise en cohérence temporelle des données recueillies, la synthèse d'un état commun et la synchronisation du passage à la fonction décision ;
  - décision : elle est cruciale pour obtenir une décision globalement optimale, et ce sera le sujet du cours 13 ;

Préoccupation transversale : la coordination II

  - planification : les conditions de synchronisation et éventuels échanges de données *pendant les adaptations* doivent faire l'objet d'une planification commune, donc répartie et coordonnée, entre les planificateurs des différentes entités de contrôle ;
  - exécution : si une adaptation se répartit sur plusieurs entités adaptables, les moteurs d'exécution peuvent avoir à se coordonner pour exécuter conjointement le plan réparti.
- L'introduction de la fonction coordination suppose :
  - d'introduire une algorithmique de coordination, et
  - des interfaces de communication des signaux et données utilisées par cette algorithmique,

  mais ces éléments sont directement liés aux fonctions à coordonner, donc il faut les appréhender au cas par cas.

Récapitulons...

1. Il existe trois grands **types de contrôleurs**, réactifs, délibératifs et hybrides, exerçant deux grands **types de contrôle**, à fréquence fixe et par événements.
2. La mise en œuvre des contrôleurs réactifs demande d'**ordonnancer** les calculs, ce qui nécessite de connaître l'ensemble des **contraintes temporisées** sur la durée et les délais de réactions de différentes tâches, données qui ne peuvent être obtenues que dans le contexte de déploiement.
3. La combinaison de plusieurs contrôles au sein d'un même contrôleur (MIMO) induit à la fois des problèmes d'ordonnancement mais aussi d'**accès multiples** aux capteurs, d'**accès concurrents** aux variables d'état et de *consignes multiples* aux actionneurs qu'il faut résoudre.
4. En robotique autonome, des limites des architectures hiérarchiques et purement réactives a émergé la proposition d'achitectures **hybrides délibératives/réactives**, susceptible d'apporter les avantages des deux premières approches, mais soulevant le problème de la bonne **coopération** entre les niveaux délibératifs et réactifs.
5. Tout comme les entités adaptables, les entités de contrôle sont **difficiles** à concevoir de manière **ouverte** et **réutilisable**.
6. Le **coordination** est une préoccupation **transversale** aux différentes fonctionnalités d'une entité de contrôle qui va nécessiter l'introduction d'une algorithmique et d'interfaces spécifiques.

Pour aller plus loin : sélection de lectures recommandées

- *From Control Loops to Real-Time Programs*, P. Caspi et O. Maler, dans *Handbook of Networked and Embedded Control Systems*, Birkhäuser, 2005, pp. 395–418.
- *Introduction to AI Robotics*, R.R. Murphy, MIT Press, 2000.
- *Autonomous Mobile Robots*, R. Siegwart, I.R. Nourbakhsh et D. Scaramuzza, MIT Press, 2è édition, 2011.