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 12

Composition, coopération et coordination des entités autonomiques

Objectifs pédagogiques du cours 12

- Rappeler les interactions potentielles entre les entités auto-adaptables induisant des besoins de coopération et de coordination entre leurs entités de contrôle.
  - *Nota : nous avons vu précédemment ces besoins entre les parties capteurs ou actionneurs des entités adaptables.*
- Étudier plus profondément la coopération intégrée comme premier moyen d'assurer l'atteinte d'objectifs communs par un ensemble d'entités de contrôle.
- Explorer différentes approches de coordination permettant d'atteindre les mêmes objectifs mais d'une manière qui assure une meilleure modularité et un meilleur découplage entre les entités de contrôle.

Plan

De la nécessité de la coordination

- Dans tout CPCS réaliste, plusieurs entités adaptables échangent et interfèrent, ce qui se traduit par :
  - un besoin d'*échanges entre leurs entités de contrôle* et
  - une *composition des contrôles* (automatique) qu'elles exercent.
  (Nota : composition des contrôles $\neq$ composition des entités de contrôle.)
- Deux types de compositions des contrôles (pas binaire !) :
  1. *Verticale* : plusieurs contrôles différents s'appliquent à la (aux) même(s) entité(s) adaptable(s).
  2. *Horizontale* : des contrôles différents d'entités adaptables différentes doivent arriver *conjointement* à des décisions et des actions *compatibles* les unes avec les autres.
- Tirant des enseignements de la théorie du contrôle mais aussi de la décision et des architectures logicielles réparties, nous allons
  - dans un premier temps, explorer des approches applicables à la coordination d'un *nombre limité* d'entités de contrôle,
  - puis, dans un second temps (cours 14), explorer des approches susceptibles de *passer à l'échelle*.

Coopération entre entités autonomiques

- Dans les cas les plus simples, les entités de contrôle peuvent être conçues et implantées *conjointement* de manière à *coopérer* entre elles pour assurer les fonctions de l'auto-adaptation (supervision, contrôle/décision, planification, adaptation, ...).

Conception fondée sur la *coopération*

Par coopération, nous entendons une algorithmique des différentes entités conçue *conjointement* pour intégrer dans l'ensemble de leurs opérations les échanges de signaux et de données nécessaires pour atteindre leur but commun.

- Cette approche de coopération exige une mainmise sur la conception et le développement des différentes entités et produit des implantations enchevêtrées difficiles à maintenir.
- De plus, il est difficile de lui faire prendre en compte la dynamicité du système et son passage à l'échelle.

Vers la coordination

- L'approche de coordination consiste à appliquer des modèles d'échange *explicites* et *génériques* permettant de *découpler* autant que possible les algorithmes des différentes entités entre elles ainsi que de l'implantation de ces échanges.

Conception fondée sur la *coordination*

Par coordination, nous entendons des approches génériques d'échange de données et de signaux permettant de concevoir les entités à coordonner de manière indépendante les unes des autres, puis d'intégrer tardivement (c'est-à-dire, *late-binding*) une implantation de la coordination lors de l'assemblage des entités.

- L'approche de coordination a été étudiée depuis une trentaine d'années comme mécanisme général de programmation répartie et dans quelques travaux comme mécanisme spécifique au contrôle réparti et à l'informatique autonomique.

Mécanismes de coordination

- Pour concrétiser un peu plus ce qu'on entend par coordination, il y a deux aspects à la coordination :
  1. L'échange d'informations.
  2. La synchronisation ou l'ordonnancement entre les actions.
- Les mécanismes de coordination vont chercher à fournir des moyens d'échange et de synchronisation qui :
  - évite aux entités impliquées de se connaître explicitement et
  - autorise la modification de l'implantation de la coordination sans avoir à toucher le code des entités impliquées.
- Exemple : échange de tâches générées dynamiquement.
  - Version coopération : les entités sont connectées les unes aux autres pour échanger des tâches à exécuter.
  - Version coordination : les entités partagent un agenda de tâches à exécuter sur lequel elles peuvent ajouter des tâches à traiter par d'autres et en prendre pour les traiter elles-mêmes.

Coordination et ingénierie du contrôle

- La coordination de contrôles apparaît dès que plusieurs contrôles ont des interactions potentielles, de quelque nature que ce soit.
- En ingénierie du contrôle, s'il y a interaction, une première solution consiste à fusionner tous les contrôles en un seul modèle.
  - Fusion : permet de définir une loi de contrôle *commune* tenant compte de la *globalité des besoins et des interactions* entre les contrôles, autant entre les décisions qu'entre les effets potentiellement contradictoires des actions correspondantes.
- Cependant, cette approche trouve rapidement ses limites (complexité du modèle, passage à l'échelle) et ne respecte pas la modularité souhaitable dans les architectures logicielles.
- La solution alternative consiste à maintenir des contrôles séparés, en les implantant indépendamment, mais en les faisant coopérer ou en les coordonnant pour éviter les conflits entre eux.

Coordination dans la décision

- Lorsque plusieurs entités adaptables sont *interdépendantes*, leurs décisions d'adaptation doivent être *coordonnées*.
- Une vision naïve consiste à laisser les entités de contrôle trouver leurs décisions indépendamment puis d'exercer une coordination *a posteriori* pour négocier une décision partagée.
- Malheureusement, cette approche naïve ne fonctionne que dans des cas bien particuliers ; dans la plupart des cas, arriver à une bonne décision commune nécessite de se coordonner dès la sélection des décisions individuelles par les contrôleurs car ces dernières ont souvent des contraintes croisées entre elles.
- Nous verrons dans les prochains cours des techniques de décision individuelles puis collectives fondées sur la coopération, puis nous aborderons des techniques de décision collective plus susceptibles de passer à l'échelle.

Coordination et architectures logicielles auto-adaptables I

- La coordination des entités de contrôle se traduit par des échanges de signaux et de données pour :
  - synchroniser leurs différents traitements ;
  - partager des informations (bien que ce simple partage n'empêche généralement pas les actions conflictuelles) ;
  - passer des consignes de l'une à l'autre ; etc.
- Cycle de vie de la mise en œuvre de la coordination :
  1. À partir des contrôles identifiés, modéliser leurs interactions :
     - expliciter les sous-ensembles de contrôles qui ont des interactions à coordonner et la nature et les éléments de ces interactions ;
     - déterminer pour chacun des sous-ensembles, les informations à échanger et les points de synchronisation nécessaires.
  2. Définir la stratégie de coordination.
  3. Choisir les outils, techniques et solutions architecturales.
  4. Mettre en œuvre.
  5. Tests unitaires et d'intégration.
  6. Déploiement.

Coordination et architectures logicielles auto-adaptables II

- Dans le reste du cours, nous allons étudier les principaux patrons de conception permettant d'assurer l'exécution de plusieurs contrôles se synchronisant et échangeant des données pour résoudre les conflits résultant de leurs interactions.
- En particulier nous allons couvrir :
  - la coordination centrée données,
  - la coordination hiérarchique des contrôles,
  - le contrôle hiérarchique et
  - la coordination pair-à-pair des contrôles.

Plan

Première approche pour « défusionner » les contrôles

- Considérons comme point de départ l'approche par fusion ; tous les contrôles sont intégrés dans un modèle unique où :
  - l'ensemble de données capteurs permettent de synthétiser un état global de la ou des entités à contrôler,
  - à partir duquel les décisions sont prises simultanément.
- Pour rendre une certaine modularité entre contrôles fusionnés, une première idée consiste à produire des copies *locales* du modèle global permettant de prendre indépendamment les décisions relatives aux différents contrôles.
  - Autrement dit, chaque modèle local contient *tout* le modèle global mais les décisions et les actions sont prises localement.
  - Idée sous-jacente : si tous les contrôleurs locaux utilisent un *même* modèle global sur les *mêmes* données et applique la *même* loi de contrôle, ils arriveront *localement* aux mêmes décisions qui les concernent que celles qui auraient été prises par le contrôleur global fusionné.

Mise en œuvre et relaxation des contraintes

- Problème : comment partager les données du modèle global ?
  - Pour les données capteurs, chaque interface capteurs des entités de contrôle peut être étendue pour collecter les données des capteurs autres que ceux de son entité adaptable.
  - Pour les données d'état des contrôles, il faut définir une interface permettant à chaque entité de contrôle de rendre accessibles ses propres données d'état aux autres contrôleurs.
- On peut croire que la mise en œuvre de cette approche de défusion suppose la synchronisation des contrôleurs pour prendre les décisions *au même moment*, mais cette synchronisation n'est pas toujours nécessaire.
- De plus pour certains systèmes, une analyse plus approfondie peut conclure que les modèles locaux peuvent se contenter de sous-ensembles des données et du comportement du contrôleur global fusionné réduisant le partage et leur code.

Exemple : contrôleur IBM I

- Dans un des premiers articles publiés sur le sujet, Kephart et al. ont proposé une coordination centrée données entre un contrôleur de consommation d'énergie agissant sur la fréquence des processeurs et un contrôleur de performance.
  - Le contrôleur de consommation d'énergie reçoit une consigne de consommation maximale et module, par un contrôle de la famille proportionnelle linéaire, la fréquence du processeur, *donc sa capacité de traitement*, de manière à ne pas dépasser cette limite.
  - Le contrôleur de performance utilise un modèle prédictif et une fonction d'utilité fondés sur la *capacité de traitement* des processeurs pour allouer ou désallouer des processeurs de manière à assurer les objectifs de performance.
  - Demande de capacité de traitement et besoin de réduction de consommation d'énergie peuvent se trouver en contradiction.
  - Les deux contrôleurs existaient préalablement, et le travail consistait à les composer correctement.

Exemple : contrôleur IBM II

- L'interaction entre les deux contrôleurs se produit du fait que la capacité de traitement des processeurs dépend de leur fréquence ; le contrôleur de performance doit donc connaître la fréquence courante des processeurs pour avoir des prédictions et prendre des décisions justes :
  - solution coopération : échanges directs entre contrôleurs des données prévus lors de leur programmation ;
  - solution coordination : publication des données par les contrôleurs de consommation et abonnement des contrôleurs de performance.
- Les expérimentations ont montré dans ce cas qu'une fois fixées les périodes des deux contrôleurs selon leurs contraintes temporelles respectives, il n'a pas été nécessaire de les synchroniser pour obtenir un contrôle conjoint correct et efficace.
  En fait, une certaine imprécision dans la connaissance des fréquences n'entache pas la prédiction d'une erreur trop importante.

Coordination centrée données par *tuple space*

- C'est peut-être le mécanisme le plus connu dans le domaine.
- Un espace commun (mémoire partagée, logiquement si non physiquement) sert à communiquer par des *tuples de données*.
- Tous les processus répartis voient l'espace de tuples et généralement tous les tuples sont visibles de tous les processus.
- La coordination est réalisée par échange de tuples avec deux opérations de base : la lecture bloquante et l'écriture asynchrone.
  $\Rightarrow$ *C'est de la publication/souscription, en plus libre...*
- Avantages pour le découplage des entités :
  - Les entités à coordonner écrivent et lisent des tuples depuis l'espace commun, sans savoir qui les produit ou les lit.
  - Producteurs et consommateurs sont donc conçus et implantés sans connaissance explicite les uns des autres.
  - L'assemblage consiste simplement à connecter les différentes entités au *même* espace de tuples.

Plan

Introduction

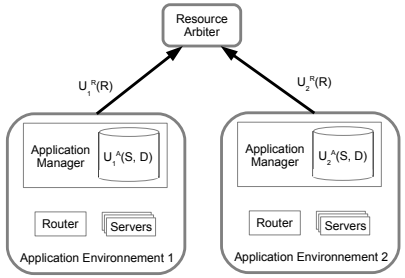
- La coordination hiérarchique consiste à imposer une (ou des) entité(s) maître(s) à un ensemble de contrôleurs composés horizontalement pour organiser leurs interactions et régler leurs éventuels conflits.
- Le rôle de cette couche d'arbitrage peut inclure :
  - l'orchestration lectures via les capteurs des contrôleurs,
  - l'équilibrage de leurs décisions,
  - la résolution des conflits de planification et
  - l'orchestration des appels des actionneurs.
- Parce que cette couche d'arbitrage a un impact sur les lois de commandes des contrôleurs, son action peut être vue et analysée (stabilité, ...) comme un contrôle de plein droit (voir plus loin).
- En automatique, l'idée de construire des contrôleurs hiérarchiques où les contrôleurs de plus haut niveau interviennent sur les décisions et les actions des contrôleurs de plus bas niveau a été étudiée et expérimentée depuis plusieurs décennies.

Coordination par arbitrage centralisé et hiérarchique

- Une première version de coordination hiérarchique consiste à utiliser un *arbitre* qui
  - reçoit les propositions de décisions ou d'actions des entités,
  - résoud leurs conflits puis
  - impose un ensemble de décisions et d'actions *cohérentes*.
- La coordination par arbitrage centralisé est une solution simple, bien adaptée à la coopération entre un relativement petit nombre de contrôleurs mais elle peut nécessiter de l'arbitre une connaissance assez poussée de l'état des différents contrôleurs.
- Ainsi, l'une des limites de cette approche est le *passage à l'échelle* et les *délais* de l'arbitrage qui doivent respecter les contraintes temporelles des contrôles.
- Il est possible d'augmenter l'échelle en adoptant un *arbitrage lui-même hiérarchique*, similaire à une hiérarchie militaire, sans pour autant pouvoir atteindre de très grandes échelles.

Exemple d'arbitrage centralisé (Tesauro et al., IBM)

- But : allocation des ressources aux applications.
- Chaque application $1 \leq i \leq n$ possède une fonction U_i^A de son utilité selon le niveau de service S et la demande D.



- À chaque cycle d'allocation, les gestionnaires d'application calculent, à partir de $U_i^A(S, D)$, une fonction de l'utilité instantanée $U_i^R(R)$ de la quantité de ressources R pouvant leur être affectées au moment de la décision.
- Après avoir collecté ces fonctions, l'arbitre effectue une allocation optimale des ressources, c'est-à-dire une répartition $R_i, 1 \leq i \leq n$ qui optimise $\sum_i U_i^R(R_i)$.

Vers des solutions heuristiques

- Parce que les approches du type fonctions d'utilité sont souvent trop complexes à mettre en œuvre (p.e., élicitation des fonctions), on se tourne alors vers des approches de coordination fondées sur des *heuristiques*.
  - Par heuristiques, on entend des méthodes fondées sur des idées ou des règles (souvent inspirées du monde réel) qui fonctionnent bien en général mais qui n'offrent aucune garantie de toujours bien fonctionner, que ce soit dans leurs résultats (impossible parfois de même borner l'erreur) ou dans leurs performances (idem, impossible souvent d'obtenir des bornes).
- Ces méthodes sont généralement développées et testées de manière totalement *empirique*.
- Et elles utilisent souvent de nombreux paramètres de réglage qui ont un impact significatif sur leur qualité et leur performance et qu'on n'arrive à fixer que par *essai et erreur* à partir de données réelles, et qui devront être adaptés à chaque utilisation.

Exemple : allocation de ressources par enchères I

- Si on voit les ressources disponibles comme des biens que cherchent à acquérir des processus, une des approches souvent utilisée dans les systèmes multi-agents est celle des *enchères*.
- Tout le monde connaît le principe des enchères : un bien est mis à prix, puis les acquéreurs potentiels vont tenter d'acquérir le bien en offrant plus que leurs concurrents.
- Dans un système où les entités reçoivent des budgets d'acquisition de ressources en fonction de leur importance relative, les enchères permettent d'allouer les ressources non pas de manière égalitaire mais en fonction de l'importance.
- Il existe plusieurs systèmes d'enchères :
  - À l'anglaise : un prix initial est fixé puis chaque enchérisseur doit annoncer un prix plus élevé que le précédent pour tenter d'acquérir le bien, jusqu'à ce que plus personne ne renchérisse.

Exemple : allocation de ressources par enchères II

- À la japonaise : un prix initial est fixé, les enchérisseurs font connaître leur intention d'acquérir le bien, puis le vendeur annonce des prix graduellement plus élevés auxquels répondent les enchérisseurs en maintenant ou retirant leur participation, l'enchère s'arrêtant quand il ne reste qu'un seul enchérisseur.
- À l'hollandaise : un prix initial élevé est fixé, puis le vendeur appelle successivement des prix moins élevés, le premier enchérisseur qui répond emporte le bien au prix alors appelé.
- Par offres scellées : un bien est annoncé, et tous les enchérisseurs transmettent secrètement leur offre ; l'offre la plus élevée emporte le bien.
  Une variante peut être appliquée à la répartition d'une certaine quantité de biens : les offres incluent une quantité souhaitée, et le vendeur écoule les biens en satisfaisant les demandes dans l'ordre décroissant des offres jusqu'à épuisement du stock.

et bien d'autres variantes encore !

Implantation des enchères et limitations

- Pour implanter les approches à base d'enchères, il est possible
  - soit de définir une(des) entité(s) qui va(vont) réaliser toutes les enchères (commissaires-priseurs),
  - soit de développer un rôle de vendeur et un rôle d'acheteur pour toutes les entités, toute entité vendeuse opérant comme un commissaire-priseur et les autres comme acheteurs pour le temps de chaque vente.
- Des protocoles sont mis en œuvre pour l'annonce des ventes et la communication des offres, et des paramètres (comme des délais de réaction) adoptés, ces paramètres ayant un impact important sur le fonctionnement du système.
- Les deux limitations principales de ces approches sont ;
  1. le délais non-bornés pour arriver à un résultat (allocation des ressources dans notre cas), et
  2. le maintien d'une entité centralisée pouvant devenir un goulot d'étranglement, et donc ne passant pas à l'échelle.

Une variante : les marchés

- Une variante de l'approche des enchères est celle des *marchés*, inspirée des marchés de biens ou financiers, où les prix s'ajustent dynamiquement selon l'offre et la demande.
- Ces approches ont l'avantage de traiter le cas où il y a *plusieurs* vendeurs (ou fournisseurs) d'un même bien et de prendre en compte les échanges au *fil du temps* sans synchronisation des achats/ventes.
- Par contre, elles maintiennent une entité centralisée (le marché) comme goulot d'étranglement potentiel.
- On peut essayer de décentraliser le marché, ce qui est plus facile que pour les enchères, mais au prix d'une complexification de l'implantation et surtout d'une plus grande difficulté à bien maîtriser les plus nombreux paramètres de réglage et l'utilisation du système pour obtenir un fonctionnement correct et efficace.

Plan

Notion de hiérarchie en automatique

- En ingénierie du contrôle, le *contrôle hiérarchique* est apparu pour traiter le cas où le(s) même(s) processus font l'objet de plusieurs contrôles en même temps.
  Ex.: dans un centre de calcul, les contrôles consistant à adapter les fréquences des cœurs, ceux consistant à modifier l'allocation des cœurs aux machines virtuelles, et ceux consistant à allouer les machines virtuelles aux applications.
- Observation faite en automatique : lorsque plusieurs contrôles s'appliquent aux mêmes entités, une hiérarchie « naturelle » émerge le plus souvent entre ces contrôles, « autorisant » ceux de plus haut niveau à *préempter* ceux de plus bas niveau.
  - Rappelons une réalisation de cette notion dans les architectures réactives à subsomption de Brooks, de même que dans les architectures hiérarchiques et hybrides délibératives/réactives pour la robotique autonome.
- Nous avons constaté plus tôt que la coordination hiérarchique par arbitrage centralisé est déjà une forme de contrôle hiérarchique.

Caractérisation des contrôles hiérarchiques

Appliquer le principe du contrôle hiérarchique à un ensemble de contrôles identifiés va demander dans un premier temps d'analyser ces contrôles et le système pour répondre aux questions suivantes :

- portée spatiale : à quelles entités adaptables s'appliquent-ils ?
- portée temporelle : à quelles fréquences, à quelles échelles de temps s'exercent-ils ?
- données d'entrée : quelles données capteurs utilisent-ils ?
- actions exercées : quelles actions utilisent-ils pour l'adaptation, avec quelles durées d'exécution ?
- techniques de contrôle : quelles techniques de contrôle appliquent-ils ? proportionnel ? prédictif ? optimal ?

Composition hiérarchique des contrôleurs

Définition

Composition entre contrôleurs agissant sur les mêmes entités adaptables telle que chaque contrôleur se voit attribuer un niveau hiérarchique qui l'autorise à agir sur les contrôleurs de niveau hiérarchique inférieur pour modifier ou préempter leurs actions et leurs décisions et symétriquement de céder sa propre décision ou sa propre action aux contrôleurs de niveaux supérieurs.

- Rappel des avantages et inconvénients :
  - fusion : plus facile de résoudre les problèmes d'interaction entre contrôles, mais contrôles complètement intégrés dans un *contrôleur monolithique* ensuite difficile à modifier ;
  - composition hiérarchique : plus difficile d'assurer l'absence d'interactions négatives, mais maintien de contrôleurs séparés plus faciles à mettre en œuvre puis à modifier *individuellement*.

Conditions d'utilisation : fusion versus hiérarchie

- Toujours dans les cas où plusieurs contrôles s'intéressent aux mêmes (groupes d') entités adaptables et sachant qu'en termes de génie logiciel des entités séparées sont préférables, quand est-il plus indiqué de fusionner ces contrôles et quand est-il plus indiqué d'appliquer le contrôle hiérarchique ?
- L'expérience en ingénierie du contrôle nous apprend que :
  - Si les contrôles à composer s'appliquent à des fréquences similaires, utilisent les mêmes techniques de contrôle et ont des actions similaires, alors il vaut mieux les fusionner en un seul contrôleur pour réguler les interactions négatives.
  - Par contre, si les contrôles s'appliquent sur des groupes différents (mais bien sûr avec une intersection non-nulle) d'entités ou à des fréquences différentes, utilisent des techniques de contrôle différentes ou ont des actions de natures différentes (ampleur, délai de mise en œuvre, ...), alors une composition hiérarchique est envisageable.

Nature de l'interaction hiérarchique

- Un contrôleur de niveau supérieur peut exercer son influence sur des contrôleurs inférieurs de deux principales façons déjà évoquées en parlant des architectures hybrides délibératives/réactives :
  1. Préemption des décision : le contrôleur de plus haut niveau bloque la décision des contrôleurs de niveaux inférieurs et produit une décision propre.
  2. Pilotage des contrôles : le contrôleur de plus haut niveau fixe les objectifs et les contraintes aux contrôleurs de niveaux inférieurs.
- Symétriquement, un contrôleur de niveau inférieur peut céder *volontairement* sa décision à un contrôleur de niveau supérieur lorsqu'il se rend compte que les actions qu'il peut exercer ne seront pas appropriées.

  Ex.: lorsqu'un contrôleur adaptant les fréquences des cœurs alloués aux machines virtuelles d'une application se rend compte que ces modifications ne vont pas suffire, il peut explicitement notifier son contrôleur supérieur qui tentera d'allouer de nouveaux cœurs aux machines virtuelles, lequel pourra notifier son propre contrôleur supérieur qui tentera d'allouer de nouvelles machines virtuelles à l'application.

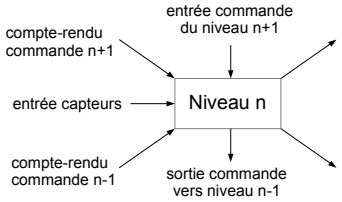
Frontière entre contrôle hiérarchique et méta-contrôle I

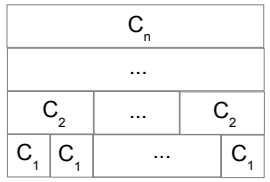
- Le pilotage des contrôles par le contrôleur de niveau supérieur peut s'apparenter au *méta-contrôle*, qui sera traité au cours 13.
  - La nature du méta-contrôle s'entend dans le sens d'un *contrôle du contrôle*, fait sur la base des *propriétés* du contrôle, comme son adéquation et sa stabilité ;
  - le contrôle hiérarchique consiste plutôt à avoir plusieurs contrôleurs sur la même entité adaptable, mais agissant sur des aspects différents et mis dans une relation de priorité pour résoudre leurs éventuels conflits.
- Dans le cas du pilotage des contrôles, on considérera :
  - qu'il s'agit de *contrôle hiérarchique* quand le contrôleur de plus haut niveau fixe par exemple de nouveaux objectifs au contrôleur inférieur pour accompagner un passage dans un nouveau mode de fonctionnnement, comme dans un système hybride le passage d'un état discret à un autre ;
    ex.: un robot auquel on fixe successivement le prochain point de passage à atteindre pour accomplir un certain trajet ;

Frontière entre contrôle hiérarchique et méta-contrôle II

  - mais qu'il s'agit de *méta-contrôle* quand le contrôleur de haut niveau cherche à améliorer les *qualités* du contrôle de niveau inférieur (stabilité, précision, ...) en changeant sa loi de commande et son implantation.
- Notez que l'architecture autonomique de référence d'IBM favorise une vision hiérarchique de la composition des gestionnaires autonomiques, et plutôt dans sa forme méta-contrôle, en liant les capteurs et actionneurs du gestionnaire supérieur aux interfaces du gestionnaire inférieur (et non à ceux de l'élément géré).

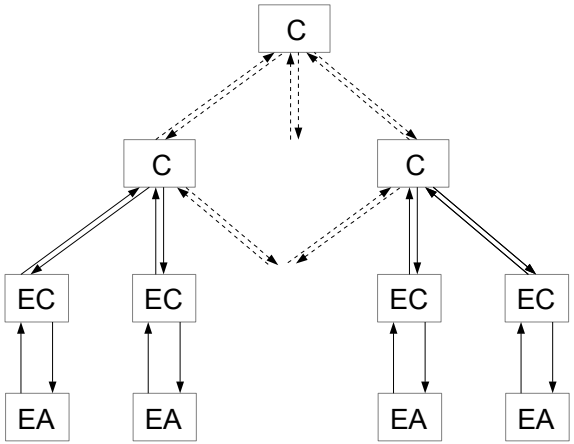
Composition hiérarchique à différentes portées temporelles





- Chaque contrôleur C_i applique une loi de contrôle propre, à des échelles de temps différentes.
- Les contrôleurs de chaque niveau connaît les activités des niveaux inférieurs et se rapportent au niveau supérieur.
- Lorsque les décisions sont conflictuelles, une communication et possiblement une synchronisation doit se faire pour résoudre les conflits dans les contraintes temporelles des différents contrôleurs impliqués.

Composition hiérarchique à différentes portées spatiales



- EA : entité adaptable
- EC : entité de contrôle
- C : contrôleur hiérarchique

Exemple de contrôle hiérarchique

- Litoiu et al. propose un *framework* de contrôle hiérarchique à la fois spatial et temporel pour optimiser la performance d'applications dans un centre de calcul, où chaque niveau de contrôleurs applique de plus des techniques de contrôle différentes : *ils explorent donc plusieurs types de variabilité*.
- Les entités adaptables sont des composants et des applications, alors que les trois niveaux de contrôle sont :
  1. Contrôleur de composant : contrôle des paramètres d'exécution comme la taille des *pools* de *threads*.
  2. Contrôleur d'application : ajuste les objectifs de QdS de chaque composant dans une application de manière à optimiser leur fonctionnement commun.
  3. Contrôleur de ressources : au niveau global, arbitre l'allocation dynamique des ressources aux différentes applications.
- Les composants de contrôle implantent différents types de modèles pour le contrôle, depuis de simples modèles linéaires prédictifs jusqu'à des modèles de files d'attente en couches.

Mise en œuvre du contrôle hiérarchique

- Le contrôle hiérarchique est une forme de contrôle à périodes multiples, mais l'implantation des contrôleurs à périodes multiples vue au cours 9 procède d'une approche par fusion.
- On peut aussi considérer que les langages synchrones visent l'implantation d'un contrôle hiérarchique par événements, mais par fusion.
- Comme déjà indiqué, un parallèle peut être fait avec les architectures réactives de Brooks en robotique, mais ces dernières sont également plutôt utilisée dans un contexte de fusion des contrôles ; sauf le respect des contraintes temporelles, rien n'empêche cependant d'implanter les architectures à la Brooks de manière modulaire, voire répartie.
- Ici les entités de contrôle demeurent indépendantes mais hiérarchisées, alors la résolution des conflits entre ces entités doit se faire par des techniques algorithmiques de la coopération ou de la coordination.

Plan

Principes

- Rappel : la coordination est l'échange de données et signaux permettant à un ensemble d'entités d'atteindre un but commun.
  - Dans les approches *hiérarchiques*, données et signaux passent par des entités centralisées ayant l'autorité.
  - Les approches *pair-à-pair* rejettent toute centralisation et considèrent les entités comme égales.
- La coordination pair-à-pair propose une algorithmique où l'atteinte des objectifs communs repose sur l'application commune d'échanges, de synchronisation et de calculs similaires dans toutes les entités, sans recours à des entités externes de natures différentes ou supérieures.
- Ces approches ont été étudiées depuis longtemps en programmation répartie et dans les systèmes multi-agents ; leurs algorithmes les plus standards ont été formalisés et caractérisés dans leurs principales propriétés (complexité, robustesse, etc.) en algorithmique répartie.

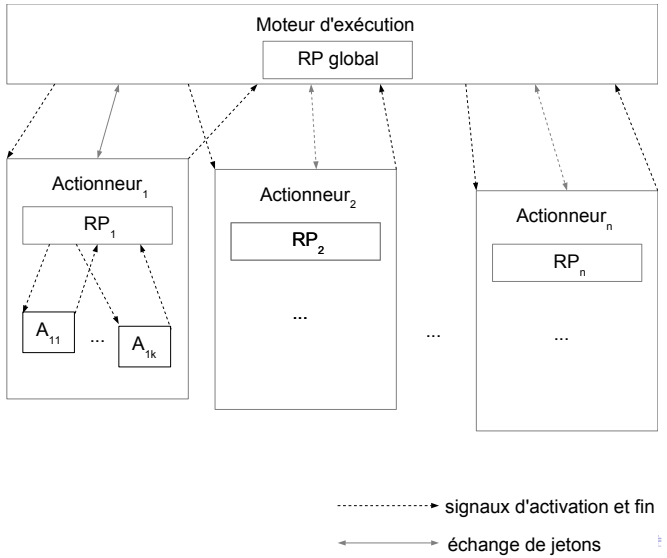
De la coopération à la coordination pair-à-pair

- La notion d'algorithmique pair-à-pair s'applique déjà dans l'approche dite de coopération introduite au début du cours, au moins au sens où les différentes entités coopérantes peuvent être conçues sans notion de hiérarchie, si ce n'est en agissant de manière totalement similaire les unes par rapport aux autres.
  - En *coopération pair-à-pair*, les entités de contrôle seront programmées de manière à coopérer explicitement l'une avec l'autre par des synchronisations explicites (cours 8).
  - En *coordination pair-à-pair*, ces instructions de synchronisation sont éliminées du code des entités au profit de moyens d'échange plus génériques n'exigent pas entre les entités une connaissance explicite les unes des autres.
- Par exemple, les besoins de synchronisation des actionneurs de l'exemple WiFi du cours 8 peuvent être exprimés par des réseaux de Petri et la coordination assurée par la composition de ces derniers.

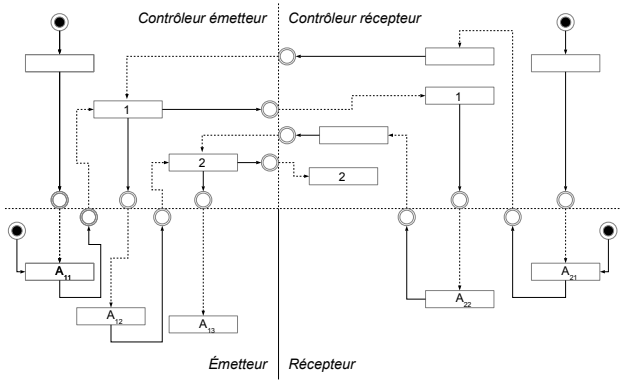
Coordination pair-à-pair par réseaux de Petri

- Nous avons présenté au cours 8 une solution de l'exemple de la communication WiFi fondée sur la coopération directe entre les actuateurs des entités adaptables.
- Nous présentons maintenant une solution fondée sur la coordination des moteurs d'exécution des entités de contrôle.
- Cette solution est fondée sur l'utilisation de la représentation des plans comme des réseaux de Petri.
  - approche simple pour découpler entité de contrôle et entité adaptable : le moteur d'exécution exécute le réseau de Petri et les actionneurs sont des tâches élémentaires qu'il ordonnance ;
  - approche plus sophistiquée extensible à la coordination pair-à-pair des entités de contrôle : chaque moteur d'exécution et actionneur organise ses opérations élémentaires grâce à un réseau de Petri, et ils coopèrent entre eux par une composition des réseaux réalisée par l'échange de jetons via des places communes dîtes *places de communication*.

Exécution répartie sur entité autonomique par RP



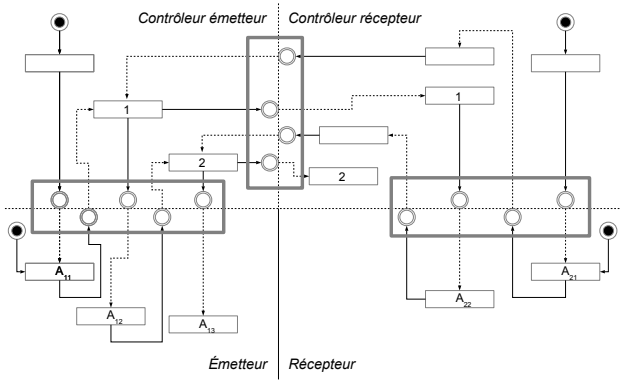
Coordination pair-à-pair par RP : exemple WiFi



- A_{11} : création du composant de compression et arrêt de l'émission.
- A_{12} : connexion du composant de compression.
- A_{13} : reprise de l'émission.
- A_{21} : création du composant de décompression.
- A_{22} : connexion du composant de décompression.
- Les transitions 1 et 2 correspondent aux deux barrières de synchronisation de la solution du cours 8.

Principes Coordination centrée données Coordination hiérarchique Contrôle hiérarchique Coordination pair-à-pair

Coordination pair-à-pair par RP : exemple WiFi



- A_{11} : création du composant de compression et arrêt de l'émission.
- A_{12} : connexion du composant de compression.
- A_{13} : reprise de l'émission.
- A_{21} : création du composant de décompression.
- A_{22} : connexion du composant de décompression.
- Les transitions 1 et 2 correspondent aux deux barrières de synchronisation de la solution du cours 8.

Principes Coordination centrée données Coordination hiérarchique Contrôle hiérarchique Coordination pair-à-pair

Mise en œuvre : coopération versus coordination

Implanter les échanges et la synchronisation par réseau de Petri peut se faire selon deux approches :

1. coopération : l'algorithmique des entités est définie de telle façon à prévoir les interconnexions nécessaires pour, par exemple, gérer les débuts et fin d'exécution par des appels/retours de méthodes et gérer les places communes, par exemple en se partageant des références à des objets places ;
2. coordination : échanger les signaux et les jetons via une implantation de la coordination et alors il y a deux solutions :
   - hiérarchique : la coordination est assurée par une entité tierce centralisé, par exemple un espace de tuples ;
   - pair-à-pair : l'entité de coordination centralisée est remplacée par un moyen de communication pair-à-pair, par exemple un réseau logique reliant toutes les entités et via lequel les signaux et jetons sont échangés comme des messages.

Principes Coordination centrée données Coordination hiérarchique Contrôle hiérarchique Coordination pair-à-pair

Coordination pair-à-pair versus centrée données

- La coordination centrée données n'est pas toujours pair-à-pair puisqu'elle peut utiliser des entités tierces (espace de tuples).
- La *coordination pair-à-pair centrée données* rejette l'utilisation d'entités tierces au profit d'échanges *anonymes* entre les entités.
- Pour mieux comprendre les différentes formes d'échanges, reprenons l'exemple de la coordination des contrôleurs de consommation d'énergie et de performance d'IBM :
  - *Coopération centrée données* : insérer dans les contrôleurs du code pour envoyer les données entre les deux ensembles de contrôleurs en les interconnectant deux à deux.
  - *Coordination centrée données* : utiliser un espace de tuples pour faire cet échange.
  - *Coordination pair-à-pair centrée données* : échanger entre contrôleurs pairs par *diffusion* des données sur un *réseau logique* sans routage explicite (cf. cours 14).

Coordination par algorithmes répartis classiques

- L'algorithmique répartie classique propose de nombreux algorithmes pair-à-pair pouvant servir de base à la coopération et la coordination d'entités de contrôle.
  - Exemple : élection d'un coordonnateur (chef) parmi les nœuds.
- Autres algorithmes : propagation d'informations par vagues, détection de terminaison, détection de pannes, synchronisation d'horloges locales, etc.
- Ces algorithmes sont généralement fondés sur l'existence de liens de communication entre les nœuds (le réseau logique) et sur des protocoles anonymes d'échanges de messages.
- Ils s'appuient cependant sur des hypothèses plus ou moins fortes comme : absence de pannes de liens et/ou de nœuds, topologie statique pendant leur exécution, durée de transmission point-à-point bornée, attribution d'identifiants uniques aux nœuds, capacité à découvrir (forger) l'identité des nœuds, etc.

Exemple : algorithme de Chang et Roberts (1979) I

- Source : Wikipedia, *Chang and Roberts algorithm*, 12/01/2015.
- Objectif : élire un maître dans un *anneau* (réseau logique).
- Hypothèses de départ :
  1. tous les nœuds $i, 1 \leq i \leq N$ disposent d'un identifiant unique $u_i \in UID$ muni d'une relation d'ordre ($\prec$), ce qui permet d'élire le nœud ayant le plus grand identifiant parmi les présents, et
  2. les nœuds sont en mesure de s'organiser en anneau de communication unidirectionnel allant de chaque nœud à son voisin dans le sens de l'horloge.
- Algorithme en deux étapes, la première étape est :
  1. Initialement, chaque nœud est marqué *non-participant*.
  2. Notant l'absence de coordinateur, un/des nœud/s lance/nt l'élection en créant (chacun) un message d'élection contenant son UID et en l'envoyant à son voisin.
  3. À chaque fois qu'un nœud envoie ou retransmet un message d'élection, il se marque comme *participant*.

Exemple : algorithme de Chang et Roberts (1979) II

4. Lorsqu'un nœud i reçoit un message d'élection m, il compare son UID u_i à l'UID du message u_m et :
   - si $u_i \prec u_m$, i retransmets inconditionnellement m à son voisin ;
   - si $u_m \prec u_i$ et i marqué *non-participant*, i remplace u_m par u_i et envoie ce message modifié à son voisin ;
   - si $u_m \prec u_i$ et i marqué *participant*, i détruit simplement le message ;
   - si $u_m = u_i$, i se déclare *élu* et débute la seconde étape.

- Deuxième étape :
  1. L'élu se marque comme non-participant et envoie le message *élu* avec son UID à son voisin.
  2. Lorsqu'un nœud reçoit un message *élu*, il se marque comme *non-participant*, mémorise l'UID de l'élu et retransmet le message *élu* à son voisin.
  3. Lorsque le message *élu* atteint l'élu, l'élu détruit le message et l'élection est terminée.
- L'algorithme fonctionne pour tout N et se termine s'il n'y a pas de fautes ; il est sûr, vivace et prend $\mathcal{O}(3N-1)$ messages quand il y a un seul initiateur.

Avantages et inconvénients

- Le principal avantage des algorithmes répartis classiques réside dans leurs propriétés (complexité, vivacité, tolérance aux fautes, auto-organisation, auto-stabilisation, etc.) étudiées et démontrées formellement.
- Ils ont souvent des implantations immédiatement utilisables.
- Leur principal inconvénient réside dans leur durée d'exécution souvent longue et non-bornable, contradictoire avec les contraintes temporelles du contrôle.
- Cet inconvénient est exacerbé lorsqu'on augmente le nombre d'entités à coordonner et en passant à l'échelle (voir cours 14).
- Assez souvent, les hypothèses de départ des algorithmes sont difficiles à assurer en pratique.

  Pour l'algorithme de Chang et Roberts, l'absence de fautes est une hypothèse bien restrictive.
  De manière générale, les démonstrations de propriétés pour les systèmes ou les nœuds et les liens peuvent tomber en panne sont pratiquement inexistantes.

Vers des solutions heuristiques

1. Comme pour la coordination hiérarchique, parce que les algorithmes répartis classiques sont souvent trop lents et se fondent sur des hypothèses trop restrictives, les chercheurs se sont tournés vers des approches de coordination plus *heuristiques*.
2. Ces approches visent également à lever certaines des limitations des algorithmes comme le délai non-bornable pour arriver à un résultat permettant d'exercer le contrôle localement, d'assurer la tolérance aux pannes, etc.
3. Elles utilisent des idées et des règles aussi souvent inspirées du monde réel, qui fonctionnent bien empiriquement mais qui n'offrent aucune garantie sur les résultats et la performance, ou que des garanties *statistiques*.
4. Exemple : algorithmes à jetons.

Exemple : k-exclusion mutuelle par passage de jetons

- Objectif : contrôle d'accès à une ressource autorisant au plus k utilisateurs simultanés par sections critiques.
- Idée générale :
  - les nœuds sont liés par un réseau logique connexe ;
  - chacun exécute un programme où les accès à la ressource sont protégés par des sections critiques dont l'entrée exige la possession d'un jeton ;
  - un jeton représente donc un droit d'accès k-exclusif à la ressource ;
  - k jetons circulent sur le réseau selon k *marches aléatoires* ;
  - chaque nœud n'entre dans une de ses sections critiques que s'il a capturé un jeton qui lui est parvenu et s'il parvient à une section critique sans avoir de jeton, il est bloqué dans l'attente d'un jeton ;
  - lorsqu'un nœud sort d'une section critique, il relâche le jeton qui lui a permis d'y entrer et lui fait poursuivre sa marche aléatoire en l'expédiant à l'un de ses voisins choisi aléatoirement.

Avantages et inconvénients

1. Les algorithmes à base de jetons évitent un contrôle d'accès centralisé et sont robustes au passage à l'échelle.
2. Ils se fondent sur une marche aléatoire pour assurer que la probabilité pour tout nœud d'accéder à la ressource dans un délai inférieur à $d > 0$ est non-nulle, et donc l'absence de famine dépend totalement de la qualité de la marche aléatoire.
3. La connexité du réseau, un écart aussi petit que possible entre les degrés des nœuds et la probabilité uniforme dans la marche aléatoire permettent d'obtenir la meilleure équité possible, mais sans pouvoir apporter de garanties formelles (l'obtention de garanties statistiques est un sujet de recherche).
4. En termes de contrôle, si une action nécessite l'accès à une ressource, moyennant une durée maximale de capture du jeton avant de l'utiliser, un contrôleur peut activer ou inhiber cette action à chaque instant de décision selon qu'il a pu capturer un jeton avant ou non, ce qui autorise une décision sans délai.

Récapitulons...

1. La coordination des entités de contrôle est rendue nécessaire par les **interactions entre leurs contrôles** sur des entités adapotables qui par ailleurs interagissent entre elles.
2. La réponse de l'automatique à l'utilisation de plusieurs contrôles est soit la **fusion en un seul contrôle monolithique** ou le **contrôle hiérarchique**.
3. Pour défusionner des contrôles, la **coordination centrée données** supposent que les contrôles peuvent s'exécuter indépendamment à partir du moment où ils ont accès à un **modèle global des différents contrôles**.
4. La **coordination hiérarchique** se fonde sur le contrôle hiérarchique pour proposer des architectures multi-contrôles **modulaires**.
5. Parce que le contrôle hiérarchique suppose une centralisation qui peut devenir impraticable, la **coordination pair-à-pair** cherchent à utiliser des techniques de coordination permettant aux entités de se coordonner entre elles sans passer par une entité tierce centralisée.
6. Si la coordination pair-à-pair peut utiliser des **algorithmes répartis** classiques, ces derniers nécessitent souvent des **durées d'exécution non-bornables** incompatibles avec les **contraintes temporelles** du contrôle ; des **algorithmes heuristiques** doivent alors être employés.

Pour aller plus loin : sélection de lectures recommandées

- *Hierarchical Model-Based Autonomic Control of Software Systems*, M. Litoiu et al., DEAS 2005.
- *Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs*, J.O. Kephart et al., ICAC 2007.
- *A Multi-Agent Systems Approach to Autonomic Computing*, G. Tesauro et al., AAMAS 2004.
- *A multiple random walks based self-stabilizing k-exclusion algorithm in ad hoc networks*, T. Bernard et al., 2010.