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 8

De l'entité adaptable — méthodologie et développement

Objectifs pédagogiques du cours 8

- Comprendre les problématiques liées à la conception des interfaces capteurs et actionneurs d'une entité adaptable en général et d'un élément géré (entité logicielle adaptable).
- Comprendre les problématiques liées à la fonctionnalité d'adaptation des éléments gérés.
- Introduire des techniques d'implantation des fonctions capteurs et actionneurs.
- Proposer des éléments méthodologiques et des outils nouveaux à utiliser dans le cadre du processus de développement des éléments gérés, notamment pour le test, la validation et l'identification au déploiement des modèles comportementaux.

Plan

De la mesure aux capteurs

- Rappels :
  - Dans un CPS classique, un capteur est un dispositif mécatronique avec :
    - un dispositif matériel assurant la mesure et
    - une interface permettant la lecture des données par des dispositifs de calcul numérique.
  - Par analogie, dans un système autonomique, un capteur a :
    - une partie logicielle intégrée à l'élément géré pour assurer la mesure et
    - une interface (signatures + code de mise en forme) permettant la lecture des données.
- Organisation de la partie interface :
  - Bien que les dispositifs mécatroniques soient moins malléables que les dispositifs logiciels, il demeure possible de concevoir une couche logicielle au-dessus de l'interface de base pour offrir des accès plus souples aux données.
- Pour les éléments gérés, les mesures doivent souvent être programmées, ce qui soulève des problématiques que nous allons aborder.

Conception des capteurs

- Objectif : conception des dispositifs organisant les mesures et donnant accès aux valeurs.
- Problématique : proposer un ensemble de capteurs dont les interfaces soient flexibles, ouverts et robustes, permettant à des entités tierces d'observer et d'adapter l'entité à bon escient.
- Éléments de mise en œuvre à couvrir :
  - déclinaison des exigences pour la fonction capteurs ;
  - définition d'un modèle pour les données de capteurs ;
  - mise en œuvre des mesures *ad hoc* et injection du code correspondant au sein des éléments gérés ;
  - définition d'interfaces flexibles et ouvertes pour l'émission et la transmission des données (issues des mesures) ;
  - intégration d'une capacité de configuration et d'adaptation dynamique de la fonction capteurs ;
  - explicitation dans ses interfaces des informations nécessaires pour utiliser correctement des capteurs.

Qualités souhaitées d'un ensemble de capteurs

Simplicité : produire l'ensemble des données utiles uniquement par les moyens *nécessaires et suffisants*.

Modularité : *découpler* les différents capteurs autant que possible pour faciliter la gestion de leurs cycles de vie respectifs (conception, implantation, tests, mises à jour, etc.).

Redondance : introduire suffisamment de *redondance* entre capteurs pour induire *tolérance aux pannes et robustesse* (surtout capteurs liés aux phénomènes physiques).

Méta-propriétés : déduire, exprimer et déclarer *explicitement* les propriétés qualitatives et quantitatives des capteurs.

Auto-évaluation : offrir une capacité réflexive pour *superviser la qualité de service* des capteurs eux-mêmes (capteurs de capteurs) et apprécier à quel point les valeurs obtenues ont les qualités attendues ou non.

Modèle générique de données mesurées

- Concevoir une instrumentation correcte et efficace exige la définition du modèle des données de mesure elles-mêmes.
- Une mesure se caractérise d'abord par :
  - sa *valeur*, avec toutes les méta-propriétés identifiées (domaine, précision, erreur moyenne, biais, ...) ;
  - l'*unité de mesure*, sans laquelle la valeur ne peut pas être interprétée (standards, JSR-275, JSR-363) ;
  - la *date* (heure) de la mesure, qui permettra à la fois de raisonner sur elle en tenant compte de son âge et de vérifier la cohérence temporelle des données utilisées conjointement ;
  - l'*identité* du dateur, par exemple l'adresse IP de l'hôte possédant l'horloge ayant servi à la datation, ce qui permettra d'interpréter correctement la date.
- Toute mesure produite, transmise et consommée doit être définie par ces quatre éléments d'information.

Mise en œuvre des instruments de mesure

- La mise en œuvre des mesure est l'implantation de procédés précis où on distingue en pratique deux cas :
  1. l'extraction des mesures utiles parmi les mesures offertes par des instruments intégrés à des outils tiers,
  2. l'introduction manuelle de nouveaux instruments de mesure au sein de l'élément géré ou de son environnement logiciel.
- Parmi les outils tiers, les systèmes d'exploitation et les réseaux offrent par exemple de nombreux outils de mesure de performance et de consommation de ressources.
- Pour l'introduction manuelle, la variété des possibilités rend difficile un traitement générique, mais quelques principes de base :
  - bien identifier ce sur quoi porte les mesures (portée temporelle et spatiale *i.e.*, portion de code concernée) ;
  - faire apparaître explicitement le compromis précision et fraîcheur versus perturbation du système ;
  - rendre accessibles et configurables les paramètres de mesure comme la fréquence, etc., permettant de choisir le compromis.

Mesures fournies par des outils existants

- Instrumentation déjà implantée par des outils existants :
  - Outils mesurant la consommation des ressources du système et de l'ordinateur physique (ex.: `ps` ou `top` sous Unix).
  - Services de métrologie offerts par le réseau, pour tout ce qui concerne les échanges de données (ex.: commande `airport`).
  - Services spécifiques offerts par différentes applications (bases de données, serveurs Web comme Apache, etc.).
- Leur utilisation pour la mesure d'une propriété demande :
  1. identifier parmi les mesures offertes celles qui sont pertinentes,
  2. avoir une portée adéquate permettant d'associer une mesure générale à la propriété de l'entité visée,
  3. exhiber les bonnes méta-propriétés (précision, biais, ...), et
  4. être capable d'extraire ces données *efficacement*.
- Si ces mesures peuvent paraître *gratuites* (*cf.* P. Lalanda et al.), ne jamais oublier le coût à l'exécution, souvent élevé pour ces outils généraux (5-6% du CPU pour `top`, par exemple, selon les paramètres donnés au lancement).

Instruments de mesure ad hoc

- Mettre en place des instruments de mesure ad hoc peut se faire en suivant les étapes générales suivantes :
  1. Concevoir les procédés et algorithmes de mesure.
  2. Identifier les données brutes nécessaires pour calculer la(les) mesure(s).
  3. Injecter le code nécessaire pour récolter les données brutes aux endroits et aux moments où elles sont disponibles.
  4. Injecter le code nécessaire pour le calcul des mesures à partir des données brutes et en adéquation avec leur récolte (*i.e.*, à quels moments ?).
  5. Tester, estimer les méta-propriétés et valider les instruments de mesure sur des cas réalistes.
  6. Identifier les sources éventuelles de biais et introduire des compensations dans le code de calcul des mesures.
  7. Étudier l'évolution des mesures dans le temps et, le cas échéant, ajouter du filtrage pour produire des séquences de valeurs bien conditionnées.

Injection du code de mesure

- Ayant identifié les mesures à réaliser et conçu les algorithmes de mesure appropriés, deux possibilités s'offrent au développeur :
  - une approche manuelle, par laquelle le développeur introduit lui-même le code nécessaire en chacun des points du code source où les algorithmes exigent d'intervenir ou
  - une approche plus systématique et mieux organisée, par laquelle le développeur va pouvoir
    - définir en un seul endroit le code nécessaire pour la mesure et
    - utiliser un moyen automatique pour injecter les différentes parties de ce code aux différents endroits nécessaires dans le source.
- Méthodes d'injection automatique
  - Approche réflexive : interception (*proxy*, méta-objets) ou insertion réflexive de code à la volée.
  - Approche par aspects : définition modulaire de code à injecter et de points de jointures dynamiques.
    $\Rightarrow$ programmation par aspects, AspectJ.

Organisation du processus de prise des mesures

- Question : comment ordonnancer la prise (calcul) des mesures ?
- Plusieurs considérations :
  - disponibilités des données brutes : où et quand ?
  - capteurs mono/multi-mesures *i.e.*, cohérence temporelle,
  - besoin des clients *i.e.*, fréquence des lectures, jointures temporelles, filtrage, fraîcheur des données, etc.
- Produire les mesures à chaque lecture via l'interface capteur n'est pas toujours possible :
  - prise des mesures peut devoir se faire à un autre moment/endroit,
  - latence de la prise de mesure peut trop retarder les lectures,
  - lectures des capteurs peuvent devenir trop fréquentes,
  - besoin d'utiliser une même mesure pour plusieurs capteurs, etc.

  $\Rightarrow$ planification des prises de mesure indépendante des lectures.
- Fréquence dépend des besoins des clients car :
  - crée deux niveaux d'échantillonnage (mesure et lecture), source supplémentaire d'erreurs ;
  - doit éviter le sur-échantillonnage ou le sous-échantillonnage.

  $\Rightarrow$ moyens de configurer en fonction des besoins des clients.

De la mesure au capteur et à son interface

- Une fois les algorithmes de mesures déterminés et réalisés, il faut ensuite déterminer les méta-propriétés de ces mesures et les documenter sur leurs interfaces dites riches.

  Ex.: documentation Javadoc, mais aussi utilisation des annotations Java pour capturer les méta-propriétés.
- Ensuite, il faut passer à l'exposition des résultats de ces mesures à travers les interfaces capteurs.
- Pour chaque capteur rendu visible, il faut déterminer deux mécanismes centraux :
  1. le mode d'émission des données, et
  2. le mode de transmission des nouvelles données.
- Ces mécanismes peuvent mettre en jeu des inter-relations entre capteurs (mesures communes, par exemple) nécessitant la possibilité de les définir conjointement.

Mode d'émission des mesures via les capteurs

- Deux modes temporels principaux :
  1. émission passive (*pull*) : le consommateur de la mesure la demande en appelant l'entité sur l'interface capteur.
  2. émission active (*push*) : le consommateur s'abonne au capteur et ce dernier lui envoie les mesures selon un certain programme de publication, le plus souvent à une certaine fréquence fixe.
- Principales propriétés de configuration déterminant le fonctionnemement capteur/mesure et capteur/lecture :
  - Dans le mode passif, il faut déterminer quand un appel doit donner lieu à une nouvelle mesure, si la mesure se fait au besoin.
  - Dans le mode actif, la fréquence des mesures peut être fonction des fréquences d'émission et de la fraîcheur requises par les clients.
- Attention : un capteur peut être utilisé par *plusieurs* clients avec des besoins différents (mode, paramètres de fonctionnement).

  Principe : produire une fois, utiliser plusieurs fois !

Mode de transmission des nouvelles mesures

- Quoi transmettre à chaque émission :
  1. La nouvelle mesure telle que produite ?
  2. Seulement les différences par rapport à la dernière valeur ?
- Deux modes de transmission essentiellement.
  1. Par rafraichissement complet :
     - transmission complète de la nouvelle mesure ;
     - s'applique bien aux données de petite taille.
  2. Par mise-à-jour différentielle :
     - transmission des modifications ou du delta par rapport à l'ancienne mesure (genre `diff/patch` sous Unix) ;
     - s'applique bien aux données de grande taille mais pour lesquelles les évolutions sont localisées,
     - mais moins robuste aux pertes !

       $\Rightarrow$ *ajouter un historique des dernières modifications à chaque transmission ?*

       $\Rightarrow$ *adopter un mode mixte ?*
- Compromis nécessaire entre efficacité et robustesse.

Configuration des émissions et des mesures I

- Prenons un scénario concret pour comprendre :
  - Deux mesures : CPU et MEM,
  - deux capteurs en mode *push*,
  - utilisées par trois contrôleurs :
    1. Le contrôleur 1 veut CPU toutes les 60 secondes avec une fraîcheur d'au plus 10 secondes.
    2. Le contrôle 2 veut MEM toutes les 30 secondes aussi avec une fraîcheur d'au plus 10 secondes.
    3. Le contrôle 3 veut MEM et CPU toutes les 40 secondes en cohérence temporelle de moins de 20 secondes avec une fraîcheur d'au plus 30 secondes pour les deux données.
- Problème : fixer les instants de mesure et d'émission des deux propriétés permettant de satisfaire ces besoins.
- Hypothèse : démarrage synchronisé de la production

Configuration des émissions et des mesures II

- PPCM des fréquences de production est 120 secondes, donc un programme sur cette durée pourra être répété indéfiniment.
- Déroulement possible :
  1. 0s : mesures CPU et MEM, puis envoi à 1, 2 et 3.
  2. 30s : nouvelle mesure MEM puis envoi à 2.
  3. 40s : nouvelle mesure CPU (fraîcheur = 40s > 30s et cohérence = 40s > 20s) et lecture de MEM puis envoi de CPU+MEM à 3.
  4. 60s : nouvelles mesures CPU et MEM (fraîcheur = 20s et 30s > 10s) puis envois à 1 et 2.
  5. 80s : lecture et envoi CPU+MEM à 3.
  6. 90s : nouvelle mesure MEM (car fraîcheur = 30s > 10s) puis envoi à 2.
  7. 120s : nouvelles mesures CPU et MEM (fraîcheur CPU = 40s > 30s et alors cohérence = 30s > 20s), puis envois à 1, 2 et 3.

  et on boucle...

Interface de configuration

- Comment généraliser cet exemple et définir de manière la plus générique possible une interface de configuration ?
  - interface opérationnelle : fixe pour chacune des propriétés le programme précis de ses mesures ;
  - interface déclarative : exprime les besoins sous forme de contraintes et calcule un programme pour toutes les mesures.
    - Pas besoin de connaître les détails internes de mise en œuvre ;
    - permet l'auto-configuration et facilite l'adaptation dynamique.
- Vers les systèmes *Complex Event Processing* (CEP, §Cours 14) :
  - Ensemble de concepts, de techniques et d'outils permettant de comprendre, analyser et contrôler les systèmes d'information dirigés par événements.
  - Systèmes répartis de détection, de propagation causale, de corrélation et de traitement à grande échelle des évenements.

Documenter une interface capteurs I

1. Propriétés adaptables mesurées :
   - capteurs simples ⇒ *une* propriété, *un* instrument de mesure,
   - capteurs logiques ⇒ plusieurs mesures, plusieurs propriétés.
2. Portée : chaque mesure produite porte sur l'évolution d'une propriété sur une certaine durée, et un certain domaine.
3. Délai de (re)mise en route : délai entre activation du capteur et la production des premières mesures.
4. Latence : délais entre demande d'une mesure et l'émission.
5. Période minimale : délai minimal entre deux lectures successives du même capteur nécessitant une nouvelle mesure.
6. Fraîcheur des données : délai entre le moment de lecture et le moment de prise de la mesure retournée.
7. Réactivité : délai entre l'apparition d'un phénomène dans l'évolution de la propriété et sa visibilité via le capteur.

Documenter une interface capteurs II

8. Précision, biais, erreur : qualité des valeurs produites, bruit systématique et aléatoire entachant les valeurs produites.
9. Interférences : interactions entre capteurs pouvant induire des contraintes croisées dans leurs utilisations respectives.
   Exemple : utilisation de mesures communes.
10. Corrélation et redondance : ensemble des relations fonctionnelles entre capteurs décrivant comment leurs valeurs évoluent les unes en relation avec les autres.
11. Cohérence temporelle : plus grande différence entre les dates des différentes mesures utilisées dans un même calcul.
12. Coût d'utilisation : impact de l'utilisation du capteur en fonction du temps, part fixe (du seul fait de sa présence) et part variable (selon la quantité d'émission).

Plan

Problématiques

- Dans l'éléments gérés, la bonne conception des opérations d'adaptation est aussi souvent vue d'une manière trop simpliste.
- Les hypothèses implicites sont souvent que :
  - les opérations d'adaptation sont *unitaires* et *indépendantes*,
  - elles se réalisent par modification d'*un seul* élément géré et
  - *n'interfèrent pas* ou très peu avec le fonctionnement de l'élément lui-même, des autres éléments et du système.

  hypothèses *fausses* pour tout système *réaliste*.
- Distinguons (à nouveau) opérations d'adaptation de leur visibilité externe par des interfaces, les questions se posant :
  - Quelles opérations offrir ?
  - Quelles propriétés adaptables affectent-elles et comment ?
  - Comment les mettre en œuvre et les rendre robustes ?
  - Comment en tirer des interfaces cohérentes et flexibles ?
  - Comment gérer les adaptations multiples, dépendantes et réparties entre plusieurs entités ?

Types d'opérations d'adaptation (rappel)

- Rappel, quatre types d'opérations d'adaptation :
  1. l'*adaptation paramétrique*, où l'opération change des paramètres, généralement scalaires, de l'exécution de l'entité ;
  2. l'*adaptation de ressources*, où l'opération modifie la quantité de ressources allouées à l'entité, ce qui peut inclure la création de ressources nouvelles comme des machines virtuelles ;
  3. l'*adaptation fonctionnelle*, où l'opération modifie les fonctionnalités de l'entité en changeant son code par chargement dynamique, compilation dynamique, réflexion, etc. ;
     et
  4. l'*adaptation architecturale*, où l'opération modifiant l'architecture de l'application en déconnectant, détruisant, ajoutant et reconnectant des entités (composants, services, etc.).
- Les frontières entre ces types d'adaptation ne sont pas *étanches* et des opérations *mixtes* sont monnaie courante.

Opérations d'adaptation élémentaires

- Concevoir un élément géré suppose d'abord d'identifier des opérations d'adaptation de base, ou *élémentaires*.
- Elles se caractérisent généralement par leur :
  - *exclusion mutuelle*, ne pouvant s'exécuter en parallèle ni avec le fonctionnement normal de l'élément ni avec toute autre opération d'adaptation sur celui-ci ;
  - *complétude*, formant un passage direct d'un état de fonctionnement cohérent à un autre état de fonctionnement cohérent, sans état intermédiaire cohérent (en général) ;
  - *non interruptibilité*, leur interruption laissant nécessairement l'élément dans un état de fonctionnement incohérent.
- Une notion qui dépend de l'application, car ces propriétés supposent que l'exécution de l'élément adaptée soit suspendue pendant toute la durée de leur exécution, ce qui dépend de leur définition et de leur utilisation.
- Exemple : *opérations d'adaptation paramétrique*.

Opérations d'adaptation composites

- Les autres types d'opérations d'adaptation (de ressources, fonctionnelle, architecturale) sont généralement trop complexes, trop longues et impactant d'autres éléments pour être considérées élémentaires.
- Elles peuvent s'appuyer sur des opérations élémentaires, en les assemblant pour former une adaptation *composite* cohérente.
- Elles se caractérisent par leur :
  - *durée*, trop longue pour simplement suspendre l'exécution de l'élément géré pendant toute leur exécution, une meilleure gestion de l'exclusion mutuelle devenant nécessaire ;
  - *parallélisabilité*, se décomposant en plusieurs (sous-)adaptations ou calculs qui peuvent se faire (partiellement) en parallèle ;
  - *non indépendance*, pouvant devoir être précisément entrelacées avec des opérations d'adaptation sur d'autres entités pour respecter des invariants locaux ou globaux du système.

Ces caractéristiques sont déjà visibles dans l'exemple d'adaptation à la bande passante du réseau sans fil.

Atomicité et opérations atomiques

- Si une adaptation *ne peut être* implantée comme une opération élémentaire de courte durée, comment traiter les évolutions et les événements se produisant *pendant* son exécution ?
- Il peut devenir nécessaire de l'interrompre si ses conditions de déclenchement ne sont plus réunies.
  Exemple : dans le cas de l'échange de données par réseau sans fil, que faire si la bande passante change d'état pendant une adaptation ?
- Si une opération d'adaptation est interrompue, dans quel état de fonctionnement seront laissées les éléments ? Distinguons :
  - un état fatalement incohérent imposant de modifier l'opération pour isoler des parties élémentaires
  - d'un état partiellement incohérent ne rendant pas les éléments inopérants mais dans un état d'adaptation incomplet réparable.
- *Opération atomique* : opération d'adaptation élémentaire ou composite qui sera entièrement réalisée ou pas du tout, grâce un mécanisme de réparation ou de retour arrière (*rollback*).

Supervision et coordination des opérations d'adaptation

- Les opérations d'adaptation réparties sur plusieurs éléments gérés nécessitent une implication de touts les éléments (adaptables ou de contrôle) pour les exécuter, les superviser et les coordonner.
- Deux options sont possibles :
  1. l'imposition d'un élément unique, l'un des éléments gérés, un contrôleur ou une entité tierce unique pour ce faire ou
  2. une supervision et une coordination pair-à-pair, soit directement entre éléments gérés ou entre leurs contrôleurs, voire les deux (voir cours 10, 11 et 12).
- Dans les deux cas, cela va nécessiter l'insertion d'actions intermédiaires dans les opérations d'adaptation pour assurer leur supervision, leur synchronisation et leur coordination.

Opérations d'adaptation supervisables et coordonnables

- Deux grandes approches de conception possibles :
  - ne rendre visibles que des opérations de haut niveau fonctionnellement cohérentes exécutant des notifications, des attentes et des synchronisations extérieures par lesquelles vont s'intégrer une supervision ou une coordination externes ;
  - rendre visibles des opérations plus élémentaires assemblables en opérations d'adaptation réparties sous la supervision des entités de contrôle ou par coordination entre entités adaptables.
- Le choix entre ces deux approches ne peut se faire dans l'absolu, cela dépend de l'application et du contexte :
  - l'approche des opérations de haut niveau simplifie la conception de l'entité de contrôle (exécution des adaptations), alors que
  - l'approche des opérations élémentaires est plus ouverte et flexible, et elle facilite la mise en œuvre de multiples opérations composites sur plusieurs entités adaptables par réutilisation d'opérations plus simples.

Méta-propriétés des actionneurs

1. Type et atomicité : élémentaire versus composite, exécutabilité comme tout ou partie d'une transaction d'adaptation.
2. Pré-conditions : conditions sur les paramètres, sur l'état de l'élément et les autres opérations à appeler avant.
3. Post-conditions : conditions sur les résultats, sur l'état de l'élément après l'exécution de l'opération.
4. Invariants : conditions maintenues sur l'état de l'élément.
5. Propriétés temporelles : durée, période minimale entre les appels, y compris de manière croisée (entre actionneurs).
6. Coûts en ressources : de toutes natures.
7. Supervisabilité : moyens offerts pour suivre le progrès et intervenir pour les synchroniser et les coordonner.
8. Interruptibilité : possibilité ou non d'interrompre l'exécution de l'opération d'adaptation.
9. Effets sur les propriétés adaptables : propriétés affectées, inertie (délai entre l'action et son effet, selon son ampleur).

Plan

Mise en œuvre des actionneurs

- La mis en œuvre des actionneurs passe d'abord par une réalisation, c'est-à-dire des algorithmes et leur implantation sous la forme d'*opérations d'adaptation*.
- Comme pour les capteurs, la grande variété des cas rend difficile toute tentative d'élaboration d'une méthodologie générale.
- Nous nous concentrons sur les concepts génériques :
  - relation entre opérations d'adaptation et fonctionnement de l'élément géré,
  - coopération et coordination entre actionneurs,
  - interruptibilité, traitement des erreurs et exécution atomique.

Adaptation et exécution des éléments gérés I

- Sauf pour des cas particuliers, adapter un élément doit se faire au moins partiellement en exclusion mutuelle avec l'exécution de ses services fonctionnels.
- La conception d'une entité adaptable doit donc prévoir la possibilité d'interrompre son exécution pendant son adaptation, ce qui n'est pas toujours évident car il faut limiter l'impact des adaptations sur l'exécution « normale ».
  - L'accès exclusif, par l'utilisation de clauses comme `synchronize` sur toutes les méthodes en Java, peut être utile mais pas nécessairement la meilleure solution, car elle implique potentiellement la sérialisation totale de l'exécution de tous les services de l'élément.

Adaptation et exécution des éléments gérés II

  - Il est plus judicieux de prévoir l'exclusion mutuelle entre les deux groupes : opérations d'adaptation d'une part et autres opérations (services) d'autre part.
    Nota : en BCM, où les composants peuvent avoir plusieurs *threads*, une discipline rigoureuse doit être adoptée pour réaliser cela.
- Par ailleurs, pour toute opération d'adaptation, bien identifier sa section critique pendant laquelle l'exécution de l'élément devra est mise en pause d'une manière ou d'une autre.
- Une fois la mise en pause des services rendue possible et l'identification claire des sections critiques des opérations d'adaptation obtenue, il devient possible d'obtenir l'exclusion mutuelle strictement *nécessaire*.

Coopération et coordination entre adaptations multiples I

- Prenons à nouveau le scénario concret de l'adaptation à la bande passante WiFi pour bien comprendre la difficulté soulevée par l'exécution de plusieurs adaptations plus élémentaires pour obtenir une adaptation complète d'un système.
- Clairement, se posent deux problèmes dans l'interaction entre le fonctionnement des deux entités et l'adaptation à réaliser :
  1. Comment entrelacer la mise en place de l'adaptation et les échanges de données des deux entités ?
     Principe : *Pour minimiser l'impact de l'opération d'adaptation sur les deux entités, il vaut mieux en interrompre le fonctionnement le moins longtemps possible.*

Coopération et coordination entre adaptations multiples II

  2. Comment faire en sorte que l'émetteur ne commencent pas à compresser les données envoyées avant que le récepteur ne soit prêt à les décompresser, et vice versa, que le récepteur ne commence pas à décompresser tant que l'émetteur ne commence à compresser ?
     Chaque entité réalise ses propres adaptations, mais les deux doivent coopérer pour que les échanges de données se déroulent correctement.
- Ces opérations sont *élémentaires* et *atomiques* au sens où elles peuvent et doivent être réalisées sans interruption et complètement ou pas du tout.

Coopération entre opérations d'adaptation

- Les deux entités vont devoir effectuer les opérations suivantes :
  - Client :
    1. Créer et initialiser le composant de compression
    2. Interrompre l'émission des données
    3. Connecter le composant de compression dans la chaîne d'émission
    4. Reprendre l'émission des données
  - Serveur :
    1. Créer et initialiser le composant de décompression
    2. S'assurer de l'interruption de la réception des données
    3. Connecter le composant de décompression dans la chaîne de transmission
    4. Reprendre la réception des données
- Adaptation en deux phases, calcul puis *commit*, ou validation.
- Synchronisations avant d'interrompre les émissions/réceptions, et à nouveau avant de reprendre.
- *Comment prendre en compte ces aspects ?*

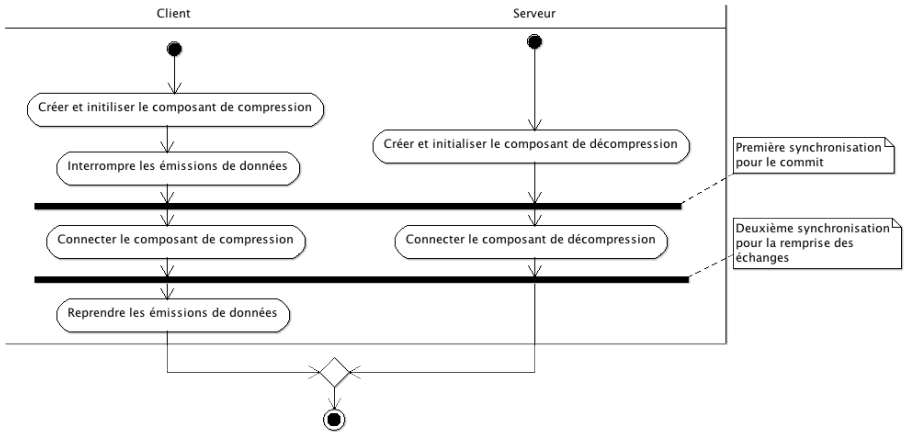
Protocoles de coopération entre actionneurs

Protocole

Ensemble des conventions nécessaires pour faire coopérer des entités indépendantes, en particulier pour établir et entretenir des échanges d'informations entre ces entités.

1. Un protocole peut s'exprimer de différentes manières avec différents outils, par exemple :
   - automates, les états marquant la progression dans le protocole et les transitions indiquant les prochaines événements et opérations possibles ;
     C'est l'approche classique dans les domaines des réseaux et de la programmation répartie.
   - en UML, les diagrammes d'activité permettent d'exprimer le déroulement des comportements et des échanges entre différentes entités coopérantes.
     C'est une approche plus couramment utilisée en modélisation et développement logiciel.

Activités d'adaptation pour la compression



Mise en œuvre I

- Les deux *jointures* du diagramme d'activités précédent peuvent se réaliser de plusieurs manières :
  - Mise en place d'une entité commune supervisant les deux adaptations en recevant les notifications de progression des deux entités et leur envoyant des signaux pour leur indiquer de continuer leur traitement.
    Ceci pourrait être géré par l'une ou l'autre des deux entités de contrôle, nous y reviendrons au cours 10...
  - Mettre les deux processus d'adaptation en liaison directe pair-à-pair pour s'échanger leurs signaux de progrès et savoir quand continuer.
    Ceci revient à mettre en place un protocole entre les deux.
  - Utiliser des outils de synchronisation.

Mise en œuvre II

- Ici, la solution la plus simple et probablement la plus élégante dans ce cas est l'utilisation d'une *barrière de synchronisation* :
  - est créée avec *n* processus devant se synchroniser ;
  - chaque processus appelle une méthode `wait` sur la barrière ;
  - tant que tous les processus n'ont pas appelé `wait`, ceux qui l'ont déjà appelé sont en attente, et dès que le dernier arrive, tous les processus sont libérés.
- Par rapport à des opérations d'adaptation indépendantes, les deux opérations côté client et côté serveur doivent partager la même barrière de synchronisation.
- Il faut donc faire apparaître l'utilisation de cette barrière d'une manière ou d'une autre entre les deux entités adaptables, par exemple en leur fournissant la référence à une même barrière lors de leur création et leur initialisation.

Interruptibilité, traitement des erreurs et transactions

- Interruption, traitement d'erreurs et transactions ajoutent de la robustesse aux opérations d'adaptation.
- En complément de l'identification des sections critiques des opérations d'adaptation, une opération robuste devrait :
  - limiter la période de non-interruptibilité au franchissement coordonné de toutes les sections critiques de toutes les opérations d'adaptation composées ;
  - assurer que toute interruption mène à un abandon complet de l'opération et ramène toutes les entités dans un état prêt à fonctionner ;
  - traiter toute exception levée de manière à ramener le(s) entité(s) touchée(s) dans leur état initial, prête(s) à fonctionner ;
  - introduire à chaque fois que nécessaire une exécution transactionnelle des opérations d'adaptation en mettant en place une mécanique permettant le retour à l'état initial, avant le lancement de l'adaptation.

Plan

Comment valider les systèmes cyber-physiques ?

- Comme pour tout système, la validation peut se faire par des tests, unitaires et d'intégration, permettant de valider leur bon fonctionnement et le respect de leurs méta-propriétés.
- Idéalement, on voudrait valider le système réel, mais cela devrait se faire dans le contexte de déploiement, en soumettant le système à un large spectre de conditions de fonctionnement, mais cela engendre des coûts souvent élevés.
- La principale difficulté pour les éléments gérés est la prise en compte des mesures de propriétés non-fonctionnelles requérant des mesures qui ne sont souvent disponibles que sur des systèmes réels de déploiement.
- D'autre part, pour assurer une bonne couverture des scénarios possibles, il faut des expérimentations contrôlées faisant varier systématiquement les valeurs des variables non-fonctionnelles.

Co-simulabilité

- Question : comment produire des scénarios de test réalistes du point de vue des valeurs des variables non-fonctionnelles ?
  - Si dans les tests fonctionnels, il peut arriver de soumettre au système des entrées de tests incohérentes, cette possibilité est beaucoup plus forte pour les variables non-fonctionnelles.
  - Les relations mathématiques exprimées par le modèle hybride du comportement de l'entité nous renseignent sur les ensembles et séquences de valeurs cohérentes qui devraient être observées.
  - Idée : calculer à l'aide de modèles hybrides des ensembles de valeurs cohérentes en fonction des données d'entrées qu'il est possible de contrôle pour produire les valeurs.
- *Cette idée peut être mise en œuvre par la co-simulation.*
  - La co-simulation consiste à simuler la plate-forme matérielle et l'environnement avec le logiciel (SIL), puis d'intégrer la plate-forme matérielle (HIL) et ne simuler que l'environnement.
  - En robotique, tout projet sérieux commence ses tests par une co-simulation avec un *moteur physique* où le robot et l'environnement sont simulés (outils : Gazebo, Morse, ...).

Principe de simulabilité des éléments gérés

Principe de simulabilité des éléments gérés

Tout élément géré doit être conçu et mis en œuvre de manière à pouvoir être simulé, d'abord pour contrôler ses propriétés non-fonctionnelles, mais possiblement aussi certaines de ses propriétés fonctionnelles.

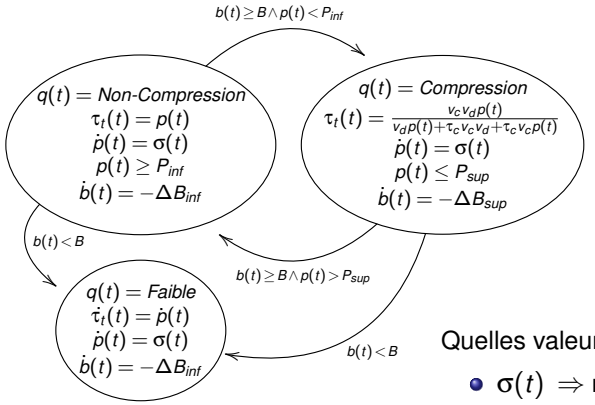
- L'objectif principal de cette simulabilité est de produire lors d'une exécution des valeurs de propriétés non-fonctionnelles correctes, cohérentes entre elles et cohérentes dans le temps.
- Un objectif qui peut aussi être très important dans certains cas est la simulation de certains aspects fonctionnels pour simplifier le déploiement des tests unitaires.
  Ex.: simuler sur un ordinateur unique une application qui devrait être déployée en réparti.
- La simulation peut être à événements discrets ou encore mixte, discrète et continue, en fonction du modèle hybride de l'entité.

Composants simulables

- Dans les systèmes embarqués, la question du test a longtemps été considérée sous l'angle du test logiciel classique.
- La robotique autonome a changé les perspectives en voyant le test puis la validation et la vérification de plus en plus sous l'angle de la simulation du robot, physiquement et informatiquement, dans un environnement simulé également.
- Pour les systèmes cyber-physiques, il paraît nécessaire d'aller un cran plus loin en considérant la simulation comme un outil tellement essentiel à un développement correct qu'elle doive être totalement intégrée à la fois :
  - comme outil permettant d'attacher aux entités logicielles (éléments gérés et contrôleurs) des modèles de comportement *exécutables* (ici, de simulation donc) et
  - comme approche de test, de validation et de vérification *fondée sur les modèles* (*model-based testing, V&V*).
- D'où la proposition du concept de composant cyber-physique en BCM vue au cours 6 allant dans ce sens.

Plan

Problématique — retour sur notre exemple concret



Quelles valeurs pour :

- $\sigma(t)$ $\Rightarrow$ mesures de terrain
- P_{sup}, P_{inf}, B $\Rightarrow$ choix
- $v_c, v_d, \tau_c, \Delta B_{inf}, \Delta B_{sup}$? ? ?
- durée des adaptations ? ? ?

Importance des paramètres du modèle

- Un bonne estimation des paramètres du modèle de comportement de l'entité adaptable est un aspect crucial pour obtenir un contrôle *correct* et *efficace*.
- En ingénierie du contrôle, l'identification du système est une phase préalable au choix de la loi de commande.
- Par exemple, dans une application d'allocation dynamique de machines virtuelles à une application sur un centre de données, nous avons expérimenté l'effet très négatif (instabilité) d'une sous-estimation de la durée des adaptations (mise en route des MV et stabilisation de la performance).
- Dans l'exemple de la compression, la durée des adaptations (dans un sens comme dans l'autre) fournit une indication du temps minimal qu'il faudra passer dans l'état compression pour amortir l'effort d'adaptation, et ainsi éviter de revenir trop rapidement dans l'état sans compression et provoquer éventuellement des oscillations entre les deux états.

Contrôle et identification du système

- En informatique autonomique, la plupart des paramètres définissant le modèle de comportement des éléments gérés dépendent du contexte de déploiement : performance du matériel, outils logiciels (ex.: quels outils de compression ?), etc.
- Pour les estimer de manière suffisamment précise pour établir un bon modèle de contrôle, il faut connaître ce(s) contexte(s). Deux options sont alors possibles :
  1. identifier au développement les contextes possibles et estimer ces paramètres à l'avance sur ces contextes pour les spécifier comme paramètres de configuration au préalable, ou
  2. attendre au déploiement et les mesurer *in situ* avant de commencer l'exécution ou pendant l'exécution.
- Dans tous les cas, pour estimer ces paramètres, il faudra instrumenter l'entité adaptable pour ce faire.

Limites de la modélisation *amont* pour l'adaptabilité

- Le génie logiciel classique prévoit de modéliser en amont du déploiement pour des développements produisant du logiciel déployable sur des plates-formes très variées.
- Ainsi à cette étape, on ne peut établir qu'un modèle comportemental *générique*, puisque ses paramètres précis dépendent du contexte de déploiement logiciel et matériel (p.e., performance).
  Ex. : $p(k+1) = ap(k) + bu(k)$, *Cours 4.*
- Ce modèle générique doit ensuite être *identifié* pour donner un modèle *concret*, c'est-à-dire que des valeurs précises pour tous les paramètres, coefficients, etc. sont déterminées à partir de mesures faites *dans le contexte de déploiement*.
  Ex. : $p(k+1) = 0,43p(k) + 0,47u(k)$, *Cours 4.*
- Enfin, pour procéder aux tests unitaires et d'intégration, puis valider le système, il faudra *simuler* tout ou partie des modèles fonctionnels et comportementaux.

Auto-caractérisation et auto-configuration

- Automatiser ces fonctionnalités :
  - Proposer des procédures prédéfinies permettant d'estimer les caractéristiques de l'entité, de ses mesures, de ses opérations d'adaptation et de l'impact des opérations d'adaptation sur les propriétés adaptables.
  - Proposer une interface pour récupérer ces données, qui pourra être utilisées par l'entité de contrôle de manière à configurer son propre modèle.
- Nous verrons lorsque nous étudierons l'entité de contrôle que certaines approches de contrôle adaptatif ou d'apprentissage automatique poursuivent le même but : estimer les paramètres du modèle de comportement de l'entité à contrôler pour obtenir un modèle de contrôle correct et efficace.
- Un lien intéressant peut aussi être fait avec l'approche *BITE* pour *built-in test equipment* de plus en plus utilisée dans l'industrie pour superviser les appareils et détecter les anomalies en cours de fonctionnement.

Récapitulons...

1. La mise en œuvre des capteurs nécessitent la réutilisation ou la mise en place de **mesures**, le choix de modes d'**émission** et de **transmission** des données, et la possibilité de **configurer** les interfaces pour servir plusieurs clients.
2. Du besoin d'utiliser ***conjointement*** de nombreuses mesures capteurs découle la nécessité de les ***synchroniser***, soit par des opérations de ***mesures jointes*** ou par de la ***datation systématique***.
3. La capacité d'adaptation exigée dans les application réelles dépasse la simple juxtaposition d'opérations d'adaptation : il faut en ***superviser l'exécution***, les organiser comme ***transactions***, et les ***coordonner*** avec d'autres adaptations.
4. Pour être réellement contrôlable, une entité adaptable doit savoir ***caractériser de la manière la plus précise possible les effets*** des adaptations sur les mesures des capteurs pour appliquer la bonne loi de contrôle et les exprimer comme méta-propriétés.
5. La validation par le test pouvant rarement se faire systématiquement dans tous les contextes de déploiement, la **simulation** est un outil essentiel à la mise au point des entités adaptables.
6. La plupart des propriétés non-fonctionnelles des entités adaptables ne peuvent être **quantifiées** que dans le contexte de déploiement ; il est donc très utile de disposer de moyens **automatiques** pour **identifier** le modèle de comportement de l'entité au moment de son **déploiement**.

Pour aller plus loin : sélection de lectures recommandées

- *Autonomic Computing*, P. Lalanda, J. McCann et A. Diaconescu, Springer-Verlag, 2013, chapitres 5 et 6.
- X. Dutreilh *et al.*, *From Data Center Resource Allocation to Control Theory and Back*, CLOUD 2010, juillet 2010.
- *Theory of Modeling and Simulation*, B.P. Zeigler et al., 2e édition, Academic Press, 2000.