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 11

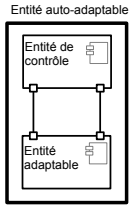
De l'entité autonomique

Objectifs pédagogiques du cours 11

- Comprendre les problématiques liés à la création d'une entité autonomique par composition entre une entité adaptable et une entité de contrôle.
- Introduire la contractualisation comme moyen d'exprimer et de vérifier les engagements réciproques entre entités adaptables et entités de contrôle.
- Étudier la validation et la vérification des entités autonomiques par l'intégration de la simulation dans les architectures logicielles comme moyen pour supporter les tests, l'identification au déploiement des modèles comportementaux, la détermination de la loi de contrôle et la vérification dynamique des entités autonomiques.

Plan

Entité auto-adaptable = entité de contrôle ∘ entité adaptable

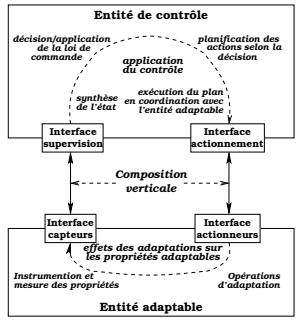


- En s'inspirant de l'architecture de référence d'IBM, l'*entité auto-adaptable* est décomposée en une *entité adaptable*, avec ses capteurs et actionneurs, et une *entité de contrôle* conférant l'autonomie de décision et d'adaptation.
- Ceci assure une meilleure *modularité* et une meilleure *séparation des préoccupations*.
- Pour IBM, l'entité auto-adaptable est dénommée *élément autonomique*.
- Du point de vue logiciel, l'entité auto-adaptable résulte de la *composition verticale* entre l'entité adaptable et l'entité de contrôle.
- Un système cyber-physique et/ou autonomique résulte alors de la *composition horizontale* entre entités auto-adaptables.

Auto-adaptabilité = contrôle ∘ adaptabilité

- En termes fonctionnels, l'auto-adaptabilité résulte aussi de la *composition verticale* entre la fonction de contrôle et un système de base observable et adaptable, d'où l'« équation » conceptuelle :

  Auto-adaptabilité = contrôle ∘ adaptabilité



- Si, informatiquement parlant, la composition verticale n'est que de la connexion entre composants, sur le fond, nous l'avons vu, la difficulté *majeure* est de s'assurer ou d'obtenir un *bon comportement conjoint* des deux entités.
- Précédemment, nous avons vu comment s'assurer que les entités adaptables et de contrôle implantent correctement leurs fonctionnalités propres.
- Pour l'entité auto-adaptable, c'est leur composition fonctionnellement correcte qui assure la correction.
- Et cette correction résulte de la compatibilité et de la garantie des engagements réciproques entre les deux entités, ce qui peut être capturé par *contractualisation*.

Note : la composition, *verticale ou horizontale*, entre contrôles, réalisée parfois par *fusion*, mais plus généralement par *coopération* et *coordination*, est sujet du cours 12.

Qu'est-ce que la programmation contractuelle ?

- Principe : organiser la relation client/serveur autour d'engagements réciproques et de raisonnements hypothèses/garanties.
- Trois principaux types de contraintes :
  - pré-conditions : conditions devant être remplies par le requérant lors d'un appel à l'offrant ;
  - post-conditions : garanties apportées par l'offrant si le requérant a respecté ses pré-conditions ;
  - invariants : relations maintenues pendant son exécution, sauf peut-être pendant ses changements d'états.
- Raisonnement hypothèses/garanties :
  Sous l'hypothèse que ses pré-conditions et l'invariant sont observés, le code appelé garantit de rendre un résultat et un état de système respectant les post-conditions et l'invariant.
- Introduite principalement par B. Meyer en Eiffel, où ces contraintes font partie intégrante du langage, à partir d'une théorie appelée *sémantique axiomatique*.

Contrats dans les architectures logicielles

- La programmation contractuelle peut être étendue au-delà de la relation client/serveur à toute composition entre entités logicielles.
- Notion déclinée dans plusieurs contextes, dont par exemple :
  - modèles à composants, par extension de leur interprétation dans le monde des objets aux interfaces offertes et requises
    ⇒ plutôt orienté comportement fonctionnel ;
  - architectures fondées sur les services : « *service level agreements, service level objectives* » (SLA/SLO)
    ⇒ plus orienté QdS.
- Contrats sur les interfaces :
  - interfaces requises ⇔ contrats *requis R*
  - interfaces offertes ⇔ contrats *offerts O*
  - connexion : *conformité* entre contrats

$$O \Longrightarrow R$$

Ex. : durée offerte < 5 $\Longrightarrow$ durée requise < 10

Contrats dans les architectures autonomiques

Engagements réciproques entre entités adaptable et de contrôle pour couvrir :

1. des aspects fonctionnels, comme les signatures et les pré- et post-conditions sur les interfaces capteurs et actionneurs ;
   Ex. : $0 \leq N_{vm} \leq 10$, où N_{vm} est le nombre de VM ajoutées ou retranchées.
2. des aspects de configuration, comme les modes de transmission et les fréquences d'appels des capteurs ;
   Ex. : transmission_mode = push (capteur actif)
   call_frequency $\leq$ 0.1 Hz (au plus 1 appel/10 secondes)
3. des aspects non-fonctionnels (ou de QdS), comme la précision des valeurs de capteurs et la durée d'exécution des actionneurs ;
   Ex. : latency $\leq$ 10 seconds.
4. des aspects de qualité de contrôle, comme le domaine de stabilité du contrôle, le délai de réglage, ou le taux de variation maximal des valeurs de capteurs que le contrôleur peut traiter.
   Ex. : $a_r \leq 10^6/sec \wedge |\frac{d}{dt} a_r| \leq 1.2$ où a_r est le taux d'arrivée des requêtes.

Expression des contrats fonctionnels et de qualité de service

- Besoin pour les contrats de QdS : langage permettant
  1. d'exprimer les contraintes requises et offertes sous des formes nouvelles par rapport aux assertions classiques,
  2. de vérifier la conformité d'un contrat offert avec un requis, et
  3. d'assurer le respect des contrats à l'exécution.
- Un exemple en QdS :
  QML (*Quality of service Modeling Language*),
  dont les concepts fondamentaux sont :
  - *dimension* : propriété nommée, avec son type de valeurs et autres informations ;
  - *type de contrats* : un ensemble de dimensions caractérisant une catégorie de QdS comme la performance, la disponibilité, la sécurité ou le temps ;
  - *contrat* : instantiation d'un type de contrats avec des contraintes particulières pour une certaine catégorie de QdS ;
  - *profil* : association d'un contrat à une interface.

Exemple QML : serveur de taux de change

```
interface RateServiceI {           // Functional interface
  Rate latest(Currency from, Currency to) ;
  Forecast analysis(Currency c) ;
}

type Reliability = contract {
  numberOfFailures: decreasing numeric no / year ; // small is beautiful
  TimeToRepair: decreasing numeric sec ;
  availability: increasing numeric ;               // big is beautiful
}
type Performance = contract {
  delay: decreasing numeric msec ;
  throughput: increasing numeric no / sec ;
}
systemReliability = Reliability contract {
  numberOfFailures < 10 no / year ;
                                        // statistical constraints
  TimeToRepair { percentile 100 < 20000 ; mean < 500 ; variance < 0.3 }
  availability > 0.8 ;
}
rateServerProfile for RateServiceI = profile {
  require systemReliability ;                  // apply to all operations
  from latest require Performance contract {   // refinements by addition
    delay { percentile 50 < 10 msec ;          // of constraints
            percentile 80 < 20 msec ;
            percentile 100 < 40 msec ; }
  } ;
  from analysis require Performance contract { delay < 4000 msec ; } ;
}
```


De la qualité de service à la qualité du contrôle

- Rappel, propriétés SASO caractérisant la qualité du contrôle :
  1. **s**tabilité (***S**tability*)
  2. précision (***A**ccuracy*)
  3. réactivité (*short **S**ettling time*)
  4. économie des moyens (*small/no **O**vershoot*)

  Il est essentiel de pouvoir analyser et contraindre ces propriétés.
- Le faire *a priori* demande d'identifier les effets des opérations d'adaptation sur ses propriétés contrôlables et d'en fixer les limites acceptables pour garantir les propriétés SASO.
- Dans le meilleur des cas, il est souhaitable qu'il existe une relation déterministe *simple* entre l'ampleur d'une opération d'adaptation et l'ampleur de son impact.
- En pratique, la relation est souvent complexe et requiert l'*expression complète* des modèles (hybrides) de comportement et de contrôle dans le contrat de contrôle.
- À défaut, une *vérification dynamique* reste envisageable.

Exemples de propriétés de contrôle à exprimer

- Communication de données par réseau sans fil :
  - l'application de l'adaptation « compression des données » résulte dans une diminution correspondante de la consommation de bande passante (selon le taux de compression) ;
  - la période entre les adaptations ne doit pas être inférieure à la durée de l'adaptation (si elle n'est pas interruptible).
- Allocation de machines virtuelles aux applications :
  - à taux d'arrivée des requêtes constant, l'ajout de machines virtuelles résulte dans une diminution quantifiable de la durée de traitement des requêtes (attente plus service) ;
  - la plage de variation de la dérivée du taux d'arrivée des requêtes contrôlable fixée à partir des taux maximaux d'allocation ou de désallocation des machines virtuelles.
- Mais, rappelons que l'effet des opérations d'adaptation peut aussi être aléatoire, voire non-déterministe, au prix de modèles de comportement et de contrôle, et donc d'une expression des contrats, plus complexes.

Contrats internes à l'entité adaptable

- Contrats *entre* actionneurs et capteurs.
- Objectif : capturer les relations entre actionneurs et capteurs permettant de caractériser le modèle de l'entité adaptable et de construire un modèle de contrôle correct.
- Difficile $\Rightarrow$ possibilité de précision croissante :

  Contrat logique : détermine quels actionneurs impactent quels capteurs.
  Contrat qualitatif : étend le contrat logique en donnant des relations qualitatives entre les actions d'adaptation des actionneurs et l'évolution des mesures observables sur les capteurs (directions, ordres de grandeurs, raisonnement dans l'imprécision).
  Contrat quantitatif : étend le contrat qualitatif en donnant des relations mathématiques entre les actions d'adaptation des actionneurs et les mesures qui seront obtenues sur les capteurs.

- La notion de « relation » déborde de l'expressivité de QML, et sont beaucoup plus difficile à manier (vérification de conformité, ...).

Contrat de contrôle, côté entité adaptable

- Objectif : exprimer les propriétés des capteurs et des actionneurs nécessaires et suffisantes pour construire un contrôleur correct.
- Les interfaces capteurs sont contractualisées sur leurs méta-propriétés, incluant celles des mesures sous-jacentes.
  - Pré-conditions : période d'appel.
  - Post-conditions : délai de mise en route, latence, fraîcheur des données, cohérence temporelle, coût.
  - Invariants : mesures utilisées (qualitatif), portée des mesures, réactivité, précision, biais, erreur, corrélation entre capteurs.
- Les interfaces actionneurs peuvent être contractualisées sur leurs méta-propriétés, incluant celles des opérations d'adaptation.
  - Pré-conditions : fonctionnelles (paramètres, état de l'entité) + période minimale entre appels.
  - Post-conditions : fonctionnelles (résultats, état de l'entité) + durée, coûts en ressources.
  - Invariants : fonctionnels (état de l'entité).

Contrat de contrôle, côté entité de contrôle

- Côté entité de contrôle, le contrat de contrôle exprime
  1. les méta-propriétés des capteurs et des actionneurs attendues selon les hypothèses du modèle de contrôle, et
  2. les méta-propriétés de la loi de contrôle ou de commande réalisée (amplitude des variations contrôlables en entrée, amplitude des actions d'adaptation, et les relations entre les deux).
- C'est d'abord le *mirroir*, en exigences, des propriétés exprimées côté entité adaptable.
- Ce sont ensuite des propriétés du contrôleur comme les durées d'exécution du code du contrôleur, exprimable en durée minimale de la période de contrôle ou encore en délai minimal entre deux événements sporadiques.
- Enfin, le contrat peut inclure des contraintes assurant les propriétés SASO du contrôleur, comme les plages de valeurs pour la période du contrôleur assurant la stabilité du contrôle.

Contrat de contrôle, vision entité autonomique I

- Du côté de l'entité adaptable, le contrat de contrôle exprime *la partie offerte du contrat*.
- Du côté de l'entité de contrôle, il exprime *la partie requise*.
- À la composition des deux entités, la *conformité* du contrat offert au contrat requis permet d'en assurer la bonne connexion pour former l'entité autonomique.
- Il s'agit ensuite d'exprimer des contraintes sur des valeurs du modèle de l'environnement, mais aussi sur les objectifs et les méta-propriétés du contrôle lui-même :
  1. des contraintes sur les propriétés de l'environnement (ex.: taux d'arrivée des requêtes, ...),
  2. des engagements sur le maintien des propriétés adaptables sur des cibles précise (la référence),
  3. des méta-propriétés associées à la manière de réaliser ce maintien (ex.: délais de réaction aux variations des entrées ; rapidité, précision et coût des adaptations).

Contrat de contrôle, vision entité autonomique II

- Exemple : performance d'une application web où $\bar{d}_s$ est la durée moyenne de service des requêtes et δ le délai entre détection d'une valeur d_s hors cible et retour à la cible,
  - Objectif de contrôle (cible) : $0.9 \leq \bar{d}_s \leq 1.1$
  - Délai maximal : $\delta \leq 120$s
    Attention, ce n'est pas une politique à seuil ! C'est une propriété que la politique doit observer...
  - Pénalité au contrôleur :
    si $\bar{d}_s > 1.1$ alors $\tau_- \times \bar{\delta}^2 \times (1 + (\bar{d}_s - 1.1))^3$ euros
    si $\bar{d}_s < 0.9$ alors $\tau_+ \times \bar{\delta}^2 \times (1 + 0.9 - \bar{d}_s)^2$ euros
    où τ_-, τ_+ : taux de pénalité déterminés entre parties contractantes.

Note : ces informations peuvent permettre de déterminer la loi de commande à utiliser ; nous y reviendrons au cours 13.

Vérification du contrat de contrôle

- Comme nous l'avons déjà indiqué en introduisant QML, la vérification formelle de conformité entre offert et requis dépend beaucoup des expressions (mathématiques) autorisées dans l'expression du contrat.
  - Ex.: Comment vérifier la conformité entre deux fonctions, par exemple que $(\forall x)\ f_1(x) < f_2(x)$?
    $\Rightarrow$ *Preuves mathématiques...*
- Par ailleurs plusieurs des propriétés à contractualiser sont des variables aléatoires dont la densité de probabilité sera imposée par contrat, or la vérification de ce type de méta-propriétés demande de faire des tests statistiques sur les valeurs *observées*.
  - Comment vérifier que la densité de probabilité d'un échantillon est conforme à la densité de la variable ?
    $\Rightarrow$ *Tests statistiques...*
- Enfin, modèles, contrats et identification ne sont jamais parfaits.

$\Rightarrow$ *Nécessité de la vérification et validation dynamiques*.

Tests d'intégration en phase de développement

- Rappel sur les tests unitaires :
  - en général, tester individuellement chaque fonctionnalité, avec des « bouchons » pour remplacer le logiciel pas encore disponible ;
  - ici, compléter les « bouchons » fonctionnels déjà bien connus par des « bouchons » de comportement (simulation SIL/HIL) ;
  - *i.e.*, les modèles de comportement fournissent des « bouchons » pour générer des valeurs de capteurs réalistes.
- Tests d'intégration :
  - les « bouchons » sont progressivement retirés au profit des entités réelles, logicielles ou matérielles ;
  - en phase de développement, le contexte de déploiement n'est pas encore précisé ;
  - il faut donc préparer des plans de tests où
    - des cas d'utilisation génériques sont identifiées,
    - puis ils sont testés selon plusieurs hypothèses en modifiant les paramètres des modèles.

Plan

Déploiement réparti et localisation

- Dans les architectures à base de composants, la composition d'un composant adaptable avec un composant de contrôle produit une entité autonomique composite dont les éléments ne sont pas nécessairement co-localisés.
- Dans un système autonomique réparti, on peut être amené à contrôler à distance le composant adaptable si le nœud sur lequel il doit être déployé est à faible ressources.
- La nature du contrôle peut cependant exiger des propriété SASO incompatibles avec les délais de communication.
- La disponibilité ou non d'un réseau à garanties de service est également un critère important dans le choix de localisation des composants de contrôle versus les composants adaptables.
- Lorsque les ressources disponibles viennent en contradiction avec les propriétés SASO requises, le contrôleur peut devoir être structuré de manière plus complexe par exemple, selon une architecture délibérative/réactive.

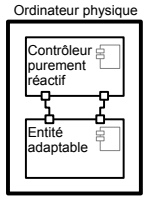
Déploiement réparti de contrôleurs plus complexes

- Considérons des contrôleurs suivant des architectures délibératives, réactives et hybrides délibératives/réactives.
- Choix de localisation possibles :
  - purement délibératif ou partie délibérative d'un contrôleur hybride peuvent être distants de l'entité adaptable s'ils ne nécessitent pas de garanties fortes sur leurs délais de réactions, donc les délais de communication aléatoires sont absorbés dans la durée déjà aléatoire et non bornée des calculs du contrôleur ;
  - purement réactif ou encore partie réactive d'un contrôleur hybride doivent généralement être co-localisés avec l'entité adaptable pour éviter des délais aléatoires de communication.
- La délocalisation des contrôleurs réactifs par rapport à l'entité adaptable est le sujet d'étude des *systèmes de contrôle en réseau* déjà évoqués.

Déploiements de contrôleurs purement réactifs

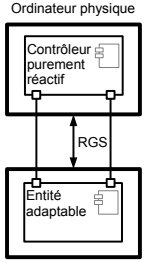
Centralisés :

- exige un peu de ressources supplémentaires de l'ordinateur de l'entité adaptable ;
- permet de garantir les délais d'échanges de données ;
- le choix le plus courant pour un contrôleur purement réactif.



Répartis :

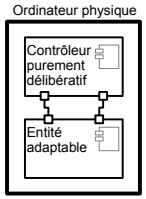
- exige un minimum de ressources supplémentaires de l'ordinateur de l'entité adaptable ;
- requiert un réseau à garanties de service (RGS) si les délais de communication doivent être maîtrisés.



Déploiements de contrôleurs purement délibératifs

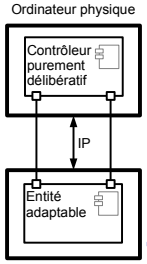
Centralisés :

- peut exiger beaucoup de ressources supplémentaires de l'ordinateur de l'entité adaptable ;
- permet de garantir les délais d'échanges de données ;
- un choix qui peut s'imposer si les délais de communications sont contraints par une fréquence de contrôle relativement élevée.



Répartis :

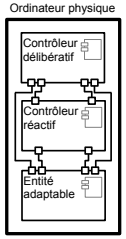
- exige un minimum de ressources supplémentaires de l'ordinateur de l'entité adaptable ;
- un réseau IP peut suffire si les délais de communication ne sont pas trop contraints.
- un choix assez courant pour un contrôleur purement délibératif.



Déploiements de contrôleurs hybrides délibératifs/réactifs

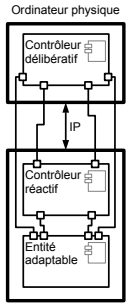
Centralisés :

- peut exiger beaucoup de ressources supplémentaires de l'ordinateur de l'entité adaptable ;
- un choix qui peut s'imposer si les délais de communications sont contraints y compris pour le niveau délibératif.



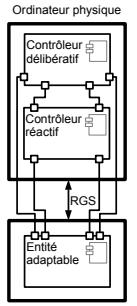
Répartis v1 :

- exige un peu de ressources supplémentaires de l'ordinateur de l'entité adaptable ;
- un réseau IP peut suffire.
- un choix assez courant pour un contrôleur hybride.



Répartis v2 :

- exige un minimum de ressources supplémentaires de l'ordinateur de l'entité adaptable ;
- peut requérir un réseau à garanties de service ;
- un choix peu courant pour un contrôleur hybride.



En résumé, critères de choix du déploiement

- Le choix du bon déploiement dépend du contrôle employé, mais aussi de considérations d'architecture physique :
  - si la fréquence de contrôle est élevée et les ressources disponibles limitées, les solutions purement réactives ou hybrides à partie délibérative délocalisée s'imposent ;
  - si la fréquence de contrôle est élevée et les ressources suffisantes, une solution co-localisée pourra être privilégiée ;
  - si la fréquence du contrôle est faible, une solution purement délibérative peut être employée, co-localisée ou non selon les ressources disponibles.
- Dans le cas d'un contrôleur hybride, les interactions entre les parties délibérative et réactive seront aussi impactés :
  - un contrôleur hybride en *supervision hiérarchique* suggère la co-localisation des parties délibératives et réactives ou alors une délocalisation de la partie délibérative mais avec un réseau à garanties de service ;
  - symétriquement, des parties délibératives et réactives distantes poussent plutôt vers une implantation en *contrôle adaptatif*.

Plan

Contrôle de qualité des logiciels

Contrôle de qualité du logiciel

Processus consistant à analyser dans quelle mesure un logiciel répond à ses exigences et qu'il remplit bien la fonction pour laquelle il a été développé.

Validation du logiciel

Processus évaluant dans quelle mesure un logiciel *satisfait ses spécifications*, c'est-à-dire le logiciel développé est-il le bon ?

Vérification du logiciel

Processus évaluant dans quelle mesure un logiciel *réalise correctement* la fonction pour laquelle il a été développé, c'est-à-dire le logiciel développé est-il correct ?

- Valider et vérifier se font par des méthodes formelles (preuve de programme, examen exhaustif, ...) ou systématiques (relecture de code, plan de tests, ...).
- Malgré une recherche soutenue, les approches formelles demeurent limitées dans leur automatisation et coûteuses lorsqu'appliquées manuellement.
- Le test systématique demeure donc le moyen le plus courant pour vérifier et valider les logiciels.

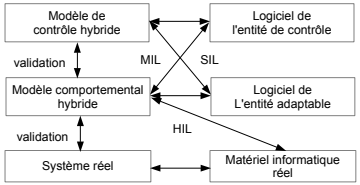
Validation et vérification des systèmes autonomiques

- La principale difficulté du test logiciel est de produire un jeu de tests, c'est-à-dire un ensemble de tuples de données d'entrée au logiciel, minimal couvrant l'ensemble des besoins :
  - en validation, l'ensemble des fonctionnalités attendues,
  - en vérification, l'ensemble du code pour y trouver les éventuelles erreurs.
- Il s'agit donc de produire ou générer un ensemble de tuples de données d'entrée couvrant toutes les fonctions et tout le code.
- Difficultés particulières aux systèmes autonomiques (et cyber-physiques) :
  - Dans les logiciels standards, comme les logiciels web, il arrive que les données d'entrée comportent des séquences de données et d'actions (sur une interface graphique) qui doit posséder une cohérence temporelle pour être valide.
  - La problématique est encore plus complexe dans les systèmes autonomiques et cyber-physiques où les données d'entrée à des capteurs, par exemple, doivent non seulement être en elles-mêmes cohérentes mais en plus être cohérentes avec les actions du système sur son environnement.
- Idée (déjà exprimée) : utiliser les modèles hybrides et la simulation pour fournir ses données cohérentes lors des tests, ce qui est fait par exemple dans le tests des systèmes en général et des robots en particulier.

Formes de simulation

La simulation est utilisée dans la validation progressive des systèmes cyber-physiques depuis l'utilisation de modèles purs en conception jusqu'à l'intégration de tout ou partie du système réel en développement :

- *Model-in-the-loop (MIL)* : simulation des modèles pour valider leur adéquation au système réel et leur exactitude les uns par rapport aux autres et le bon comportement du contrôle dans différentes « conditions » logicielles, matérielles et d'environnement.
- *Software-in-the-loop (SIL)* : intégration des logiciels avec les modèles de comportement des matériels et de l'environnement pour les valider dans différentes conditions matérielles et d'environnement.



- *Hardware-in-the-loop (HIL)* : intégration des logiciels et des matériels avec les modèles de l'environnement pour les valider dans différentes conditions d'environnement.

Dans le contexte d'un système de taille plus importante, des simulations SIL et HIL *partielles* sont utiles, c'est-à-dire où une partie seulement du logiciel ou du matériel réel est intégré, le reste étant encore simulé.

Utilisations statiques de la simulation

Tests unitaires et d'intégration : les composants non encore développés sont représentés par leur simulateur uniquement, le composant client (de la simulation) joue le rôle de cas de test.

Identification et détermination des lois de commande : avec un type de simulation développé à cet effet sur l'entité adaptable, l'entité de contrôle jouant le rôle de client cherchant à recevoir les paramètres du modèle de comportement de l'entité adaptable et, le cas échéant, caler ses paramètres de contrôle.

Plan

Validation et vérification dynamique

- L'impossibilité de valider et vérifier statiquement de manière complète un système pousse à introduire dans son exécutable des mécanismes permettant de détecter ses éventuelles fautes.
- Par rapport à ce qui se fait classiquement en validation et vérification dynamique, les systèmes autonomiques cyber-physiques présentent deux principales spécificités :
  - la validation et la vérification des modèles ainsi que
  - la vérification des contrats de contrôle.

Vérification dynamique des modèles

- La qualité des contrôles exercés (propriétés SASO) dépend totalement de leur adéquation avec le système *réel*.
- Mais, les contrôleurs sont généralement mis au point à l'aide de modèles définis *à la conception*.
- Il est donc essentiel de s'assurer de la pertinence et de la bonne identification des modèles ainsi que de la qualité des lois de commande qui en sont déduites.
- Pour ce faire, il y a deux approches principales :
  - pour les modèles déterministes, les simuler en parallèle avec l'exécution normale de manière à comparer les résultats de simulation avec la réalité ;
  - pour les modèles stochastiques, collecter des données et procéder à des tests statistiques pour s'assurer de la bonne adéquation des lois de probabilités utilisées par rapport à cet échantillon.

Vérification dynamique des contrats de contrôle

- Supervision et vérification dynamique des contrats supposent une instrumentation supplémentaire pas nécessairement prévue pour le contrôle.
- Idée : collecter des données dynamiquement pour vérifier
  - le respect des contrats offerts et leur conformité avec les contrats requis,
  - le traitement adéquat des situations où les contrats ne sont plus respectés par des mécanismes ad hoc (ex. : taux d'arrivée des requêtes variant trop vite impliquant le rejet immédiat d'une partie des requêtes).
- *Mais* cette instrumentation et ces vérifications imposent un surcoût à l'exécution.
- Il faut trouver un équilibre entre l'importance de vérifier que les entités respectent leurs engagements et le surcoût engendré par cette vérification dynamique.

Cycle de vie de l'entité auto-adaptable

Principales activités à conduire :

- En phase de développement :
  1. Vérification statique de la conformité des interfaces capteurs et actionneurs requises et offertes.
  2. Test d'intégration SIL et vérification statique des contrats.
  3. Identification du domaine de fonctionnement correct.
- Puis en phase de déploiement :
  1. Détermination de la répartition sur l'architecture physique.
  2. Connexion explicite entre les deux entités.
  3. Identification des paramètres des modèles, de la loi de commande et des contrats.
  4. Configuration des paramètres de fonctionnement des capteurs, des actionneurs et du contrôleur.
  5. Vérification et validation de l'entité auto-adaptable.
- Et enfin en phase d'exécution :
  1. Supervision et vérification dynamique de l'entité et des contrats.
  2. Adaptation dynamique du contrôle (méta-contrôle, cours 13).

Récapitulons...

1. L'entité autonomique est la **composition** entre une entité adaptable et une entité de contrôle pour former une entité **auto-adaptable**.
2. Parmi les activités à conduire lors de la composition et le déploiement des entités autonomiques, il y a l'**identification** de l'entité adaptable, la **détermination** précise de la loi de commande, les **tests** et la **validation** où la **simulation** complète les bouchons fonctionnels classiques par des **bouchons comportementaux**.
3. En plus de la technique de contrôle utilisée, le choix de la bonne architecture d'entité de contrôle dépend également des **facteurs de déploiement**, dont la nature répartie et les ressources disponibles sur les nœuds pour le contrôle.
4. La **composition** entre entité de contrôle et entité adaptable peut se formaliser par un **contrat de contrôle** spécifiant les objectifs et les méta-propriétés du contrôle.
5. La répartition sur plusieurs nœuds d'un système de contrôle soulève les questions de la **localité du temps** et du caractère **aléatoire** des délais de communication, incompatibles avec le contrôle en temps-réel.
6. Pour faciliter les tests et la validation SIL puis HIL, les entités autonomiques doivent avoir une capacité d'**auto-simulation** progressivement désactivable.

Pour aller plus loin : sélection de lectures recommandées

- *QML : A Language for Quality of Service Specification*, S. Frølund and J. Koistinen, Rapport de recherche HPL-98-10, Software Technology Laboratory, Hewlett-Packard.
- *Contracts for Systems Design*, A. Benveniste et al., rapport de recherche INRIA n° 8147, novembre 2012, 64 p.