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 1

Introduction aux systèmes de contrôle cyber-physiques auto-adaptables

Description courte de l'UE

Étude des architectures logicielles appliquées aux systèmes de contrôle cyber-physiques autonomiques, de leurs concepts, théories, techniques et méthodologies. Applications aux architectures à base de composants et fondées sur les services, aux architectures pour la robotique autonome, pour l'informatique autonomique, pour les systèmes cyber-physiques et plus généralement pour les systèmes embarqués et répartis à grande échelle.

Plan

Généralités

- UE professionnalisante et recherche STL :
  - exige un équilibre entre les aspects conceptuels et pratiques.
- 14 semaines (du 18/09–23/10, 6/11, 27/11–18/12, 8/01–22/01) :

  Mercredi, 13h45 – 15h45 : 2h cours magistraux.

  Mercredi, 16h00 – 18h00 : 2h de travaux pratiques/dirigés
  - présentations techniques (outils)
  - travail sur les projets
  - audits de projets
  - travail sur les sujets de recherche
- Évaluations :
  - Audit 1 (projets) : 23/10/2019
  - Soutenances mi-semestre : 20/11/2019 (rendu 17/11, minuit)
  - Audit 2 (projets) : 8/01/2020
  - Soutenances finales (projets) : ? ?/02/2020 (rendu 2/02, minuit)
  - Rapports (sujets de recherche) : ? ?/02/2020, minuit

Modalités d'évaluation

À choisir parmi deux modalités :

1. Une modalité parcours professionnel ou recherche empirique :
   - un projet à faire en équipe de 2 à soutenir à la fin de l'UE :
     1. premier audit intermédiaire : 10%
     2. soutenance à mi-semestre (programme) : 30%
     3. second audit intermédiaire : 10%
     4. recette de fin de projet (programme, doc., résultats) : 50%
   - sujets :
     - parcours pro : auto-élasticité dans les centre de calculs.
     - parcours recherche : modèles de composants et outils pour les systèmes autonomiques, extensions au projet centre de calcul.
2. Une modalité parcours recherche conceptuelle et théorique :
   - un travail individuel de recherche bibliographique avec une présentation et un rapport en fin d'UE :
     1. pré-rapport, plan du reste à faire à mi-semestre : 25%
     2. présentation finale : 25%
     3. rapport : 50%

Contenu semaine par semaine I

1. Introduction aux systèmes de contrôle cyber-physiques auto-adaptables
2. Architectures logicielles dynamiquement adaptables
3. Modélisation du comportement des systèmes de contrôle cyber-physiques
4. Contrôle et autonomicité
5. Simulation des systèmes cyber-physiques et autonomiques
6. Simulation modulaire et DEVS
7. De l'entité adaptable — principes et conception
8. De l'entité adaptable — méthodologie et développement

Contenu semaine par semaine II

9. De l'entité de contrôle — principes et conception
10. De l'entité de contrôle — méthodologie et développement
11. De l'entité autonomique
12. Composition, coopération et coordination des entités autonomiques
13. Décision et méta-contrôle
14. Informatique systémique et émergence

Objectifs pédagogiques du cours 1

- Comprendre les enjeux de l'informatique contemporaine dont les systèmes s'exécutent de plus en plus en permanence et en lien direct avec le monde réel sur lequel ils agissent de manière autonome (systèmes de contrôle *cyber-physiques*) et capables de s'auto-adapter (autonomiques).
- Comprendre ce qu'est l'*auto-adaptabilité dynamique* des logiciels, et les besoins auxquels elle répond.
- Comprendre les notions fondamentales des *architectures autonomiques*, leurs caractéristiques ainsi que les connaissances et les savoir-faire mis en jeu dans ce domaine.
- Introduire l'outil principal qui servira à la partie pratique de l'UE :
  - un modèle de programmation par composants concurrents et répartis minimal en Java.

Plan

L'informatique aujourd'hui

- Tout devient réparti et embarqué :
  - Omniprésence de l'informatique répartie, ouverte et à ressources partagées (*service-oriented computing*, *cloud computing*), s'exécutant en permanence (24/24, 7/7) sous de fortes variations de charge et de conditions d'exécution.
- Internet des objets et mobilité comme partie intégrante dans une part croissante des applications (réseaux sans fil), et donc une plus grande variation encore des ressources disponibles.
- Systèmes « cyber-physiques » (*cyber-physical systems*), en connexion directe avec des phénomènes du monde physique, en temps réel.
  - **Définition** : systèmes intégrant de manière forte des calculs numériques avec le monde physique réel parce que ce dernier a un impact sur les calculs effectués (par exemple, la localisation) ou sur sa performance (par exemple, disponibilité de l'énergie).

Des systèmes adaptables aux systèmes autonomiques I

1. Un besoin d'adapter les systèmes à leur contexte d'exécution pour assurer la meilleure performance et la meilleure utilisation des ressources possibles.
   - **Systèmes adaptables** : systèmes qui peuvent être adaptés à leur contexte d'exécution soit par choix des algorithmes ou par des choix de paramètres de déploiement.
2. Si les conditions d'exécution évoluent dynamiquement de manière importante, un besoin de s'adapter en permanence, pendant toute la durée de l'exécution.
   - **Systèmes dynamiquement adaptables** : systèmes qui peuvent être adaptés pendant leur exécution.
3. Parce que les évolutions sont rapides et que les interventions manuelles ne sont pas toujours possibles, un besoin de systèmes sachant s'adapter eux-mêmes de manière autonome.

Des systèmes adaptables aux systèmes autonomiques II

- **Systèmes auto-adaptables** : systèmes qui peuvent s'adapter eux-mêmes pendant leur exécution.

4. Et, concommittamment, la nécessité de réagir aux évolutions de son environnement physique en fonction de son état par des actions sur celui-ci, c'est à dire d'exercer sur le monde physique une forme de contrôle.
   - **Systèmes de contrôle cyber-physiques** : systèmes cyber-physiques agissant sur le monde physique grâce à des capteurs lui permettant d'en connaître l'état et des actionneurs lui permettant d'agir sur ce dernier (ex.: robotique autonome, pilote automatique, . . .).

- *Focalisation de l'UE :*
  *les **systèmes de contrôle cyber-physiques autonomiques**.*

Adaptabilité des systèmes informatiques

- Plusieurs moyens :
  - modification de paramètres de configuration (taille de tampons de données, nombre de *threads*, etc.),
  - modification de l'allocation des ressources à l'exécution du programme (capacité système, bande passante réseau, etc.),
  - modification du code (programme de l'application, bibliothèques appelées chargées dynamiquement, ...) ou de la façon d'exécuter le code (comment il se compile — au sens large, signification donnée aux instructions, compilation à la volée),
  - modification de la configuration de l'application (nombre de composants, déploiement sur serveurs virtualisés, ...).
- Donc, quatre grands types d'adaptations :
  1. Paramétriques (paramètres d'exécution, tailles, ...),
  2. Ressources (QdS versus ressources),
  3. Fonctionnelles (modfications des fonctionnalités, du code),
  4. Architecturales (modifications de l'assemblage de composants).

Exigences de l'auto-adaptabilité dynamique

- une **connaissance de lui-même**, de sa forme, de son contenu,de sa configuration, de son état d'exécution courant, etc. ;
- une **connaissance de son contexte d'exécution**, à tous les niveaux, y compris les données de l'exécution en cours ;
- des **mécanismes lui permettant de se modifier lui-même**, c'est-à-dire de changer les ressources mises à sa disposition, de même que modifier sa propre architecture et son propre code ;
- une **capacité à décider quand et comment** il doit s'adapter par rapport à son état courant ;
- et enfin, en poussant à la limite, une **capacité à modifier sa propre manière de s'auto-adapter**.

Problématiques induites en développement logiciel

- Représentation du système lui-même qu'il puisse manipuler et modifier à l'exécution.
- Conception indépendante, ouverte ; composabilité (interconnexion) sans avoir à connaître l'implantation.
- Configuration au déploiement et reconfiguration dynamique.
- Adaptation pendant l'exécution, par exemple lors de la connexion avec d'autres systèmes.
- Allocation et réallocation régulières des ressources disponibles à l'exécution.
- Déploiement continu, à chaud, des évolutions logicielles.

- Dans un contexte où les coûts humains (développement, maintenance, exploitation) croissent continuement.

  $\Rightarrow$ *automatisation.*

Plan

Typologie des causes d'adaptation à prendre en compte

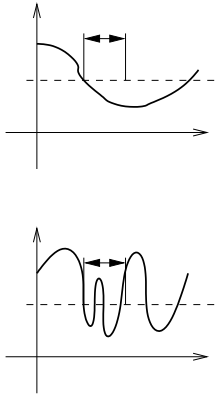
- Les adaptations sont réalisées pour répondre à des besoins de nouvelles fonctionnalités, de meilleure qualité de service, de pannes à circonvenir ou de réactions imposées par l'évolution de l'environnement.
- Comment ces besoins apparaissent-ils et à quel rythme ?

Continuum entre

- des changements rares, prédictibles et déterministes
  Exemples : changements planifiés par l'utilisateur dans l'évolution logicielle ou le mode d'opération, niveau de charge de la batterie (+ ou -).
- des changements fréquents, non-prédictibles et stochastiques
  Exemples : disponibilité de nouveaux services (ambiants *i.e.*, localisation), évolution de la bande passante du réseau, etc.

Problème : *le temps !*

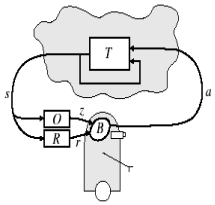
- Exemple.: adaptation à la bande passante du réseau sans fil.
- S'adapter consomme du **temps** et des **ressources**.
- L'état (interne et contexte) évolue dans le temps (ici, la bande passante).
- *Que se passe-t-il si l'état courant n'est plus compatible avec une adaptation qui vient de se terminer ? Que se passe-t-il si les ressources disponibles sont insuffisantes pour adapter complètement le système à un instant donné ?*
- Le nouvel état va requérir une nouvelle adaptation.



- Ce processus peut potentiellement mener à une forme d'emballement (« *thrashing* ») : le système s'adapte continuellement, consommant toutes les ressources, et ne faisant plus rien d'autre.

Problème bien connu

- Problème d'automatique (théorie du contrôle) connu depuis les premières applications, comme le pilotage automatique ou les canons anti-aériens pendant la deuxième guerre mondiale.
- Modèle d'automate de contrôle
  - T : fonction de transition (évolution) de l'environnement selon la décision et l'état
  - O : observation faite par l'agent
  - R : récompense (reward)
  - B : comportement (behavior) c'est-à-dire la décision, le contrôle



- Il faut faire entrer dans l'adaptation des politiques de décision (contrôle) qui vont équilibrer le gain potentiel étant donné le temps nécessaire pour faire l'adaptation et l'évolution probable du système et de son état d'exécution pendant cette période.

Auto-adaptation et contrôle = représentation + décision

- À partir du moment où on cherche à s'adapter à la variation des conditions d'exécution et des ressources disponibles en agissant sur un système ou son contexte, il faut savoir décider *quand* et *comment* s'adapter.
- Cela devient un problème :
  - de décision, au sens de la *théorie de la décision* c'est-à-dire quelle adaptation choisir,
  - et de contrôle au sens de la *théorie du contrôle*, c'est-à-dire comment gérer son impact sur la dynamique du système étant donné son inertie intrinsèque et les contraintes posées sur les adaptations possibles,

  qui peut être abordé par le *contrôle (optimal)*, par la *recherche opérationnelle*, ou par des techniques de *décision en IA*.

Du besoin au logiciel

- Réaliser un tel système auto-adaptable est une tâche complexe.
  - Ils sont reconnus comme difficiles à spécifier, mettre au point, tester, vérifier et valider.
- L'objectif principal de l'UE ALASCA est de répondre à la question :

  *Que doivent proposer un **modèle à composants** et une **architecture logicielle** pour faciliter ces activités et soutenir la mise en œuvre d'une méthodologie de développement propre aux systèmes de contrôle cyber-physiques autonomiques et comment les construire **efficacement** tout en s'assurant qu'ils soient **corrects** et **sûrs** ?*
- Pour répondre à cette question, nous allons d'abord regarder ce qui se fait sous le nom d'*autonomic computing*, que nous traduisons par *informatique autonomique* pour plus de simplicité.

Informatique autonomique : objectifs et concepts

Définition : ensemble des concepts et techniques utilisés pour permettre à un système informatique de gérer lui-même les éléments qui le composent et d'entreprendre automatiquement les actions nécessaires à son fonctionnement optimal, sans intervention humaine.

Principaux instigateurs : IBM et ses chercheurs au début des années 2000 qui visaient à

- Mettre l'informatique au service de la gestion des applications.
- Élever considérablement le niveau de fiabilité des services informatiques.
- Réduire le coût humain de la fonction de gestion des services et applications.

Caractéristiques d'un système autonomique selon IBM

- Auto-configurabilité, en fonction des conditions de déploiement et de leur évolution en cours d'exécution ;
- Auto-optimisabilité, en fonction des usages et des ressources disponibles ;
- Auto-réparabilité, lors de pannes ou d'événements imprévus ;
- Auto-protection, face aux attaques externes ;
- Ouverture, ne s'enferme pas dans des contextes hermétiques mais on s'ouvre plutôt à l'extérieur en adhérant à des standards reconnus ;

Lien entre autonomicité et contrôle cyber-physique

- L'informatique autonomique s'intéresse pour l'essentiel à l'auto-adaptabilité informatique fermée sur elle-même.
- Pour cela, le contrôleur autonomique utilise des notions de la théorie du contrôle (capteurs, actionneurs, politique, etc.).
- Ce sont exactement les mêmes approches qui sont utilisées par les systèmes de contrôle cyber-physiques pour suivre les évolutions du monde physique et agir sur lui.
- Un système de contrôle cyber-physique autonomique utilise donc les mêmes modèles et techniques à deux niveaux :
  1. Au niveau de l'élément logiciel de base (élément géré en informatique autonomique) parce qu'il sera aussi un logiciel de contrôle cyber-physique.
  2. Au niveau de l'élément de contrôle (élément autonomique) parce qu'il agit sur l'élément de contrôle cyber-physique de manière similaire à ce que ce dernier fait avec le monde physique.

Disciplines d'appui : un domaine au spectre très large

- Systèmes répartis en réseau
- Systèmes embarqués temps-réel
- **Génie des logiciels à base de composants**
- Modélisation comportementale
- Modélisation stochastique
- Simulation
- Décision
- Théorie du contrôle
- Coordination
- Systèmes complexes

Et bien d'autres encore...

Plan

Caractère auto-adaptable des architectures logicielles

- Deux problèmes fondamentaux :
  1. Comment programmer des entités pour les rendre auto-adaptables ?
     - code, modèles, connaissance de son propre état et de celui du monde physique, capacité d'auto-adaptation, ...
  2. Comment les composer pour former des systèmes auto-adaptables ?
     - compatibilité, composition « parallèle » du code et des modèles, coordination des comportements, ...
- Tryptique central :

  comportements (code, modèles) / interfaces / composition
- L'entité de contrôle cyber-physique et auto-adaptable doit être conçue autour :
  1. de modèles fonctionnels, de comportement et d'adaptation ;
  2. d'interfaces explicites exposant et contractualisant fonctions, adaptations et QdS ;
  3. de mécanismes de composition couvrant fonctions, modèles, adaptation et QdS.

Vers des architectures CCP autonomiques

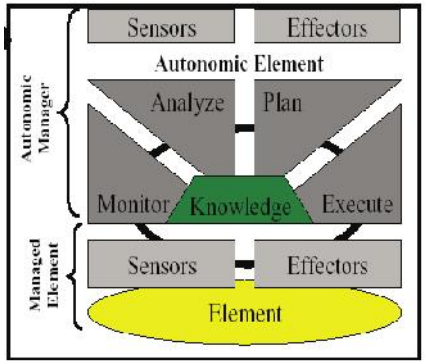
- Composants de contrôle cyber-physiques autonomiques inspirés :
  - des modèles de composants répartis, temps réel et embarqués ;
  - de l'automatique par l'utilisation de capteurs, d'actionneurs et de contrôleurs en boucle fermée ;
  - de la mesure, l'instrumentation et la réflexion pour réaliser la « *connaissance de lui-même* » ;
  - des automates et systèmes hybrides pour modéliser le comportement ;
  - des interfaces riches et de la coordination pour la composition,
  - de la simulation modulaire pour le test, la validation et la vérification.
- Intégration de notions de décision : ces composants s'inspirent de l'automatique, de la RO, de l'IA et de la robotique autonome.
- Le contrôle vise deux objectifs :
  1. homéostasie : maintien des équilibres à court terme avec une décision souvent purement réactive.
  2. évolution : inférence de nouvelles capacités et de nouvelles politiques sur le plus long terme.

Architecture des éléments autonomiques (inspirés IBM)

- Composants autonomiques : un continuum du matériel aux composants logiciels métiers en passant par le système d'exploitation, le réseau, les intergiciels et les programmes.
- Se découpent en deux parties : élément géré et élément contrôleur.
- Éléments gérés (*managed elements*) : composants adaptables implantant des interfaces capteurs et actionneurs.
- Éléments contrôleurs (*autonomic manager*) : boucle de contrôle apportant l'auto-adaptabilité par :
  1. la collecte des informations sur l'exécution et le contexte,
  2. leur traitement et le diagnostic pour arriver à une décision,
  3. l'élaboration des actions d'adaptation nécessaires, et enfin
  4. l'intervention sur l'élément géré pour l'adapter.

Éléments autonomiques

- La composition d'un élément géré et de son contrôleur autonomique forme un *élément autonomique*.



- Surveillance : réception des données via les senseurs.
- Analyse : obtention d'un diagnostique.
- Planification : détermination des actions à prendre.
- Exécution : mise en œuvre du plan via les actionneurs.
- On reconnaît d'emblée l'automate de contrôle générique déjà présenté dans cette boucle...

De l'élément géré : capteurs et actionneurs

- Les architectures autonomiques s'intéressent aux éléments gérés que dans la mesure où ils permettent :
  - d'observer leur structure interne, leur état et leur QdS courants ;
  - de leur appliquer des adaptations.
- Empruntant au vocabulaire de l'automatique, on utilise :
  - le terme *capteur* pour désigner les dispositifs mis en place par l'élément géré pour observer son état courant, et
  - le terme *actionneur* pour désigner l'ensemble des opérations mises en place par l'élément géré permettant de l'adapter dynamiquement.
- En termes d'architectures logicielles, les informations fournies par les capteurs sont rendues visibles et accessibles via des *interfaces capteurs* et les opérations d'adaptation le sont par les *interfaces actionneurs*.

Du contrôleur autonomique : contrôle en boucle fermée I

Quatre grandes étapes :

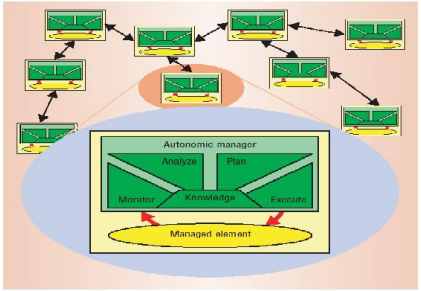
1. Supervision :
   - opérations permettant de reconstruire une image fidèle de l'état de l'élément géré, de son contexte et de son environnement ;
   - interfaces capteurs de l'élément géré, de même que des sondes sur le contexte et l'environnement ;
   - techniques fondées sur l'historique pour filtrer, débruiter, lisser, simplifier et abstraire les données obtenues pour construire un état utilisable par le modèle de décision et contrôle.
2. Analyse : applique des techniques de corrélation, décision et diagnostic pour
   - déterminer l'état courant à partir des données de supervision,
   - identifier les écarts entre cet état courant et l'état souhaitable,
   - décider s'il est nécessaire d'adapter ou non, et
   - identifier l'*action* d'adaptation et les éventuels paramètres de cette adaptation.

Du contrôleur autonomique : contrôle en boucle fermée II

3. Planification :
   - détermine l'ensemble des *opérations* d'adaptation nécessaires pour réaliser les *actions* d'adaptations choisies, et
   - les organise dans un *plan*, incluant des opérations composites, exécutables sur différentes entités et avec des dépendances temporelles explicites entre elles.
4. Exécution :
   - lance et supervise l'exécution du plan obtenu de la phase de planification ;
   - moteur d'exécution adapté aux types d'opérations élémentaires et composites apparaissant dans le plan, aux actions préparatoires nécessaires (comme l'arrêt momentané d'exécution des éléments gérés) et la synchronisation des actions élémentaires.
   - phase d'exécution est la mise en œuvre du plan par l'ordonnancement, le lancement, le contrôle de réalisation et l'enchaînement des opérations d'adaptation.

Composition des éléments autonomiques

- La plupart des systèmes étant trop complexes et physiquement répartis, ils ne peuvent être contrôlés de manière *monolithique*.
- Il faut donc les voir comme un *assemblage* de composants autonomiques, chacun exerçant son propre contrôle local.



Nouveaux problèmes :

- Connexion entre les éléments autonomiques.
- Coordination des objectifs de contrôle.
- Planification et exécution conjointes des adaptations.

Architectures logicielles des domaines connexes I

- Plusieurs architectures autonomiques ont été développées, soit en suivant le plan architectural d'IBM, soit en suivant des variantes de ce dernier.
- Mais d'autres domaines de l'informatique se posent des problèmes similaires et fournissent des sources d'inspiration et de coopération intéressantes.
- Dans les systèmes temps réels embarqués répartis où des actions doivent être prises en réaction à des événements ou à l'évolution de l'état d'un système :
  - architectures synchrones,
  - architectures réparties de type GALS (*globalement asynchrones, localement synchrones*),
  - architectures asservies par le temps (*time-triggered*).
- Architecture de contrôle pour la robotique autonome, où le robot se contrôle lui-même en boucle fermée et peut se coordonner avec d'autres robots :

Architectures logicielles des domaines connexes II

- architectures réactives,
- architectures délibératives,
- architectures hybrides, délibératives/réactives.

- Dans le domaine des systèmes d'information en réseau, les systèmes de traitement d'événements complexes (*complex event processing, CEP*) cherchent à capturer, identifier et corréler des événements spatialement et temporellement répartis pour déclencher des actions.
- Dans le domaine des systèmes multi-agents où s'exercent beaucoup de coordination, de décision collective et d'émergence de comportements globaux, mais généralement sans exigence de temps réel.

Plan

Un outil d'expérimentation

- ALASCA : expérimentation avec les architectures autonomiques à base de composants.
- Quelle technologie de composants adopter ?
  - Un modèle académique développé dans le cadre de la recherche : *Basic Component Model* (BCM).
  - Applications à base de composants répartis en Java, utilisant des technologies sous-jacentes classiques, comme le RMI et la programmation concurrente (*threads*, `java.util.concurrent`, ...) en Java.
  - Malgré son caractère académique, BCM a été utilisé dans un projet ANR et une thèse pour implanter des applications comportant des milliers de composants déployés sur des dizaines de JVM en réseau.
  - Il a également subi le test de quatre promotions d'étudiants d'ALASCA qui l'ont utilisé dans leurs projets.

Principes de base de BCM I

- Les composants peuvent agir en tant que **clients** lorsqu'ils appellent d'autres composants, ou comme **fournisseurs** de service, lorsqu'ils sont appelés, ou **les deux**.
- Les composants fournisseurs déclarent leurs services offerts à d'autres composants par leurs **interfaces offertes**.
- Les composants clients déclarent leurs besoins des services d'autres composants par leurs **interfaces requises**.
- Des composants jouant l'un pour l'autre un rôle pair-à-pair, offrant et requérant des services, peuvent le déclarer par des **interfaces à double sens** (ou *two-way interfaces*).
- Un composant est représenté par un objet, instance de sa classe de définition ; ses **services** fournis et ses **tâches** clientes sont implantés par des méthodes. Il peut être passif ou actif, et alors il possède ses propres *threads*.

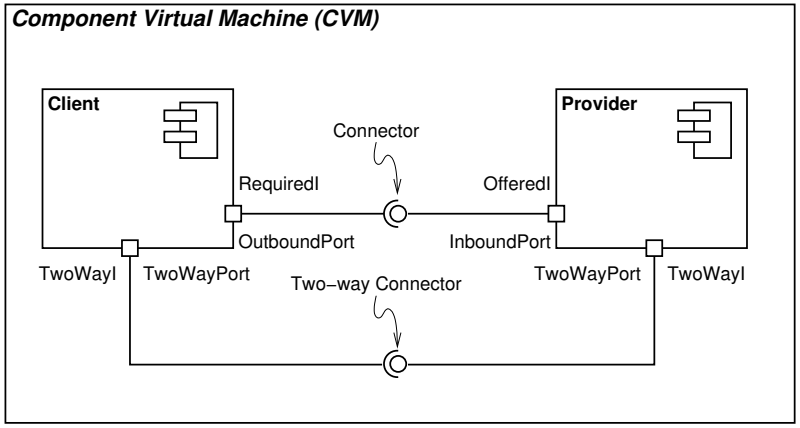
Principes de base de BCM II

- Les composants exposent leurs services offerts ou requis par des **ports explicites**, les points d'entrée (*inbound*) ou de sortie (*outbound*) obligatoires de tous les appels inter-composants.
- Les connexions entre composants sont faites explicitement entre ports sortants et entrants via des **connecteurs** susceptibles de faire la médiation entre une interface requise et une interface offerte qui ne sont pas identiques ou directement compatibles.
- Un composant peut inclure en son sein des **sous-composants** (récursivement), mais les sous-composants sont *invisibles* de l'extérieur à moins que leur super-composant n'expose explicitement leurs services via ses propres ports et interfaces.
- Les **assemblages de composants** peuvent être réalisés statiquement ou avec un noyau statique par la **machine virtuelle des composants** (CVM) puis dynamiquement d'abord par le noyau statique.

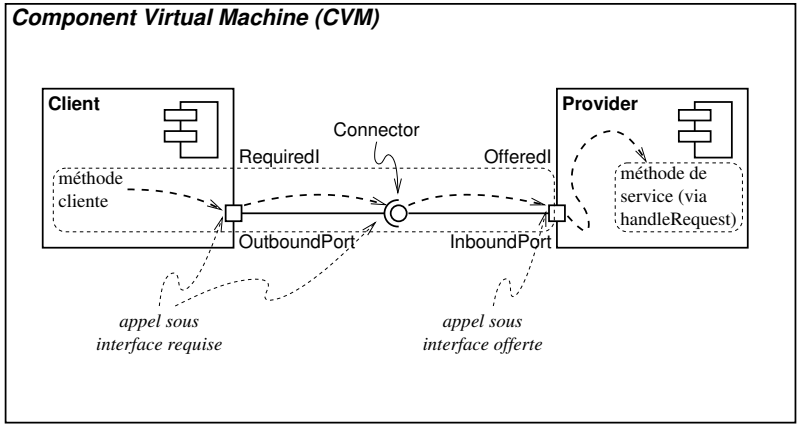
Principes de base de BCM III

- Des **registres** (spécifiques et RMI) sont utilisés pour publier les ports localement ou globalement sous l'identification de leur URI, puis pour récupérer les informations nécessaires aux connexions.
- La CVM peut être exécutée **localement**, à l'intérieur d'une seule JVM, ou encore de manière **répartie** sur plusieurs JVM possiblement sur plusieurs ordinateurs hôtes.
- Dans un déploiement réparti, un **fichier de configuration** en XML donne les informations nécessaires pour lier les sites d'exécution de la CVM entre eux. Chaque site possède une **URI** et correspond à un processus exécutant une JVM. URI et nom du fichier de configuration sont passés au lancement des JVM exécutant les sites de la CVM.

Illustration des cas de base



Suivi d'un appel client-serveur



Trois familles de déploiements

1. Mono JVM : une seule JVM contient tous les composants qui sont connectés via leurs ports et connecteurs par des références Java.
2. Multi JVM[1], mono hôte : plusieurs JVM s'exécutent sur le même hôte, mais les composants déployés sur des JVM différentes sont connectés par des références RMI.
3. Multi JVM[1], multi hôte : plusieurs JVM s'exécutent sur le même hôte et d'autres sur différents hôtes, et les composants situés sur des JVM différentes, sur le même hôte oun non, sont toujours connectés par des références RMI.

[1] Dans les cas multi-JVM, les composants déployés au sein d'une même JVM sont toujours connectés par des références Java.

Illustration : mono JVM

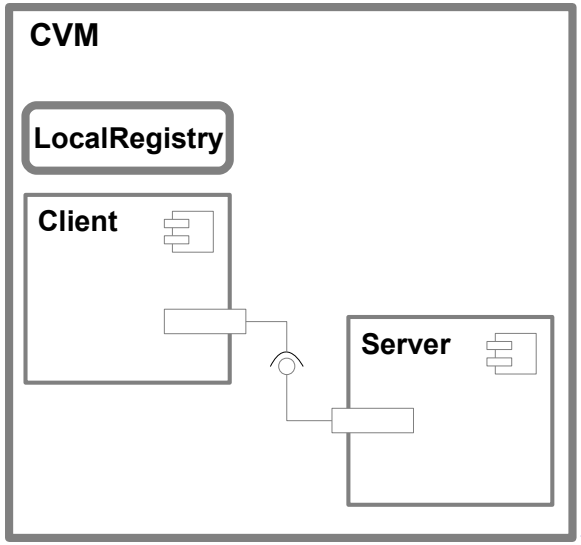


Illustration : multi JVM, mono hôte

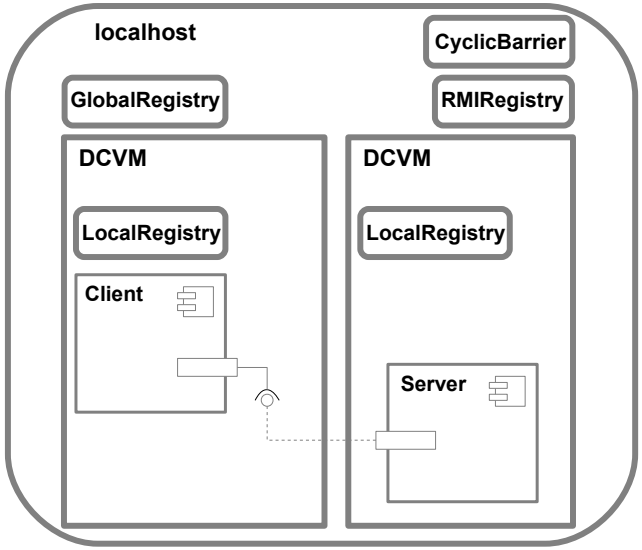
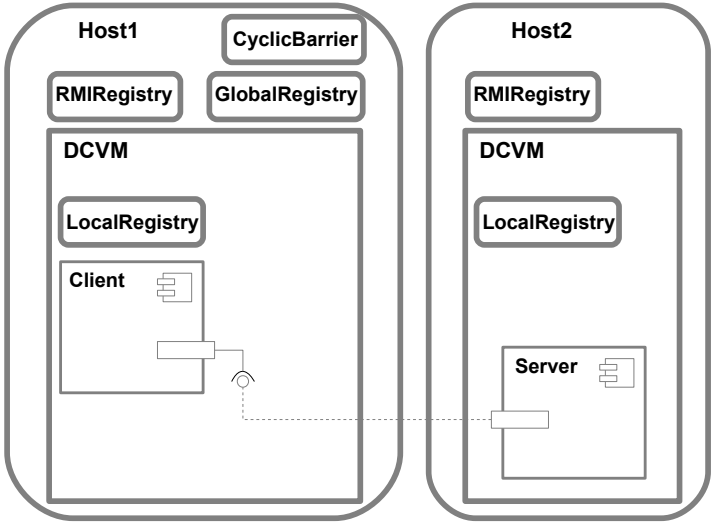


Illustration : multi JVM, multi hôte



Utilisation pratique

- Une bibliothèque Java implantant les concepts abstraits de BCM (composants, interfaces, ports, connecteurs et machines virtuelles) vous sera fournie.
- Il s'agit essentiellement d'un cadriciel (*framework*) qui permettra de programmer en étendant des classes abstraites, en étendant et implantant des interfaces, et en exécutant les machines virtuelles de composants.
- La programmation est fondée sur des conventions qui doivent être respectées, et qui sont illustrées par des exemples fournis avec la bibliothèque.
- Certaines techniques J2SE, comme l'utilisation des « security managers » doivent être utilisées pour faire fonctionner les applications.

Récapitulons...

1. L'informatique aujourd'hui repose de plus en plus sur des logiciels s'exécutant en permanence, avec une grande variabilité de leur contexte d'exécution, ce qui impose de les adapter dynamiquement.
2. L'***auto-adaptabilité dynamique*** vise à utiliser la puissance de l'informatique au service de l'adaptabilité dynamique en construisant des logiciels capables de s'adapter eux-mêmes.
3. L'objectif de l'UE est de ***comprendre les problèmes*** liés à la conception et l'implantation de tels logiciels, et de former des ingénieur informatique ***capables de coopérer avec des spécialistes*** des autres disciplines nécessaires pour mener à bien des projets de développement de logiciels auto-adaptables.

Pour aller plus loin : sélection de lectures recommandées

- *The Vision of Autonomic Computing*, J.O. Kephart et D.M. Chess, Computer, 36, pp. 41–50.
- *Autonomic Computing*, P. Lalanda, J. McCann et A. Diaconescu, Springer-Verlag, 2013, chapitres 1 à 4.
- *An Architectural Blueprint for Autonomic Computing*, IBM, 2006.
- *Autonomic Computing — Concepts, Infrastructure and Applications*, M. Parashar et S. Hariri, CRC Press, 2007, chapitres 1, 3 et 5.
- La documentation de la bibliothèque `BasicComponentModel`.