CPS — Programmation par composants

© 2019- Jacques Malenfant

Master informatique, spécialité STL – UFR 919 Ingénierie
Sorbonne Université

Jacques.Malenfant@lip6.fr

Cours 1

Introduction aux systèmes à base de composants

Plan

Généralités

- 10 séances de cours et 10 de TD/TME :
  Mardi, 16h00 – 18h00 : 2h cours magistraux.
  Lundi, 13h45 – 18h00 : 4h de travaux dirigés/encadrés sur machine.
- Évaluations (barême sur 100) :
  - Audit 1 (5) : TD/TME 4 du 21/02/2022.
  - Soutenance mi-semestre (35) : semaine du 14 au 18/03/2022 avec rendu préalable le 13/03/2022 à minuit au plus tard.
  - Audit 2 (5) : TD/TME 9 du 11/04/2022.
  - Soutenance finale (55) : semaine du 16 au 20/05/2022 avec rendu préalable le 15/05/2022 à minuit au plus tard.

Déroulement des évaluations

- Projet « Gestion d'événements complexes » en BCM4Java.
- Projet en quatre étapes.
  1. Audit 1 (10 à 15 minutes) : étape 1 ; discussions/questions autour du code, démonstration.
  2. Soutenance à mi-parcours (20 minutes) : étapes 1 et 2 ; discussions/questions autour du code, démonstration, avec rendu de code préalable.
  3. Audit 2 (10 à 15 minutes) : étape 3 ; discussions/questions autour du code, démonstration.
  4. Soutenance finale (30 minutes) : étapes 1 à 4, avec rendu de code préalable.
     - Présentation sur transparents
     - Discussions/questions autour du code
     - Démonstration
- Critères d'évaluation (cf. cahier des charges pour les détails) :
  - projet complet et exécutable ;
  - qualité des solutions logicielles adoptées ;
  - qualité du code et de la documentation.

Contenu semaine par semaine

1. Introduction aux systèmes à base de composants
2. Programmation par composants séquentialisée
3. Assemblage et exécution des programmes à composants
4. Composants parallèles en BCM
5. Composants concurrents en BCM
6. Composants répartis en BCM
7. Architectures logicielles dynamiques
8. Conception et programmation par contrats
9. Systèmes répartis à grande échelle
10. Composants temps réel et leur programmation

Plan

Objectifs de la séquence

1. Objectifs pédagogiques
   - Comprendre les objectifs de l'approche par composants.
   - Comprendre les racines de cette approche.
   - Comprendre les principaux concepts de cette approche.
2. Compétences à acquérir
   - Savoir articuler les besoins industriels pour la conception, l'implantation et l'utilisation de composants avec les principaux concepts des modèles à composants.
   - Savoir décrire les rôles des différentes parties prenantes du développement à base de composants.
   - Savoir faire le lien entre un rôle dans le développement à base de composants (concepteur, développeur, utilisateur, . . .) et les concepts et les artéfacts apparaissant dans les modèles à composants.

Vision globale (*the big picture*)

- Modèle : industrie comptemporaine.
  - Exemple de l'industrie automobile :
    1. équipementiers : conçoivent et produisent des pièces « standardisées », les *composants* ;
    2. concepteurs : conçoivent des produits intégrant des *composants standards* à partir de leurs *fiches techniques* ;
    3. fabricants : assemblent les produits à partir des *composants achetés* chez les équipementiers.
- Idée : répéter en informatique ce qui a si bien réussi dans d'autres industries.
  - Définir des « standards » permettant de concevoir et fournir des *composants sur étagères*.
  - Ouvrir un marché compétitif pour des composants de mêmes fonctionnalités.
  - Transformer les fournisseurs de logiciels en assembleurs de composants offerts sur étagère.
  - Augmenter significativement la réutilisation logicielle pour rendre l'industrie informatique plus sûre et plus efficace.

Qu'est-ce qu'un composant logiciel ?

Une ***entité logicielle*** se caractérisant comme suit :

- morceau de logiciel encapsulé directement déployable, définissant explicitement tous ses points d'interconnexion qui exposent des interfaces explicites ;
- assemblable avec d'autres composants par des tiers, sans avoir à en examiner le contenu mais seulement en connaissant les interfaces exposées par les points d'interconnexion ;
- dont l'assemblage se fait par connexion des points de sortie (appelant) avec des points d'entrée (appelé) réalisable depuis l'extérieur des composants ;
- dont tout le cycle de vie, depuis le chargement et l'initialisation jusqu'à la destruction en passant par l'activation (et désactivation), est contrôlable de l'extérieur, également sans avoir à connaître son contenu.

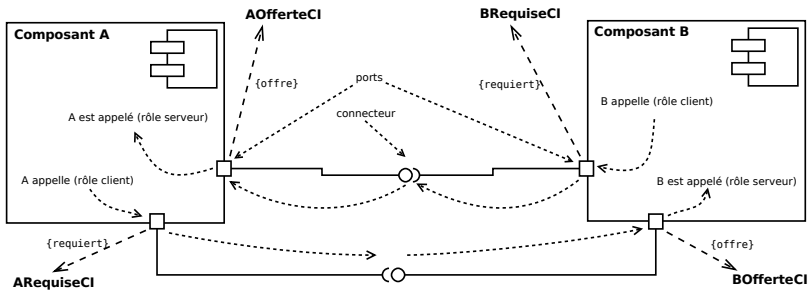
Contraintes de conception I

- Déployable ?
  - Une entité concrète, du *code exécutable*, chargeable et intégrable au sein d'une application (comparable à un objet ou à une bibliothèque chargeable dynamiquement mais pas à une classe).
  - Mais, besoin d'une notion de *description* des composants (équivalente à la classe dans les langages à objets) qui ne doit pas être confondue avec les composants eux-mêmes.
    - *Attention*, le vocabulaire courant utilise généralement le même terme de *composant* dans les deux cas ; comparez :
      - *Tout* ordinateur possède un *composant* de stockage de masse.
      - Le *composant* "disque dur" de *mon* ordinateur est en panne.
- Utilisable par des tiers ? (tiers $\neq$ ses développeurs)
  - Ses services offerts et requis sont décrits par des *interfaces* fournissant *toutes* les informations nécessaires pour les connecter et les utiliser.
  - Il possède un *cycle de vie* décrit par des *opérations standards*.

Contraintes de conception II

  - Il forme donc une *unité d'encapsulation*, une *boîte noire* dont le contenu est *masqué* aux autres composants.
- Assemblable ou composable ?
  - Créer une application à base de composants se dit *assembler* des composants.
  - Lier deux composants se dit *connecter*.
  - Tous les points de connexion des composants sont explicitement visibles (référençables) de l'extérieur.
    - Implique que les composants définissent aussi leurs points de connexion *en sortie* (contrairement aux objets où ils sont masqués), c'est-à-dire les *services qu'ils requièrent et qu'ils appellent*.
    - Implique aussi que chaque point possède un identifiant unique à l'échelle de l'architecture.
  - L'opération de connexion est réalisée explicitement *sur* (de l'extérieur) ou *par* chaque composant (de l'intérieur).
  - La connexion utilise (généralement) un *connecteur* explicite.

Première vision générale



Comment définir un composant logiciel ? I

1. Services, activités et rôles
   - Les composants implantent des **services** dont l'exécution est déclenchée par les **appels** (requêtes) d'autres composants.
   - Les composants adoptent un **rôle**, client, serveur ou les deux, selon qu'ils sont appelés par ou appellent d'autres composants.
   - Les composants peuvent aussi avoir des **activités** propres (tâches) exécutées à leur initiative, indépendamment de l'exécution de leurs services (répondant eux à des appels extérieurs).
2. Interfaces
   - 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**.
3. Ports
   - Les composants exposent leurs services offerts ou requis par des **ports explicites**, les points d'entrée (*inbound*) ou de sortie (*outbound*) obligatoires pour tous les appels inter-composants.

Comment définir un composant logiciel ? II

4. Connexion et connecteurs
   - Les connexions entre composants sont faites explicitement entre ports sortants et entrants via des **connecteurs**.
   - Lorsque les interfaces requises et offertes ne sont pas identiques, les connecteurs peuvent faire la « médiation » entre les appels aux services requis et les appels aux services offerts (comme des adaptateurs).
5. Assemblage
   - Les **assemblages de composants** peuvent être réalisés *statiquement* lors du démarrage de l'application, *dynamiquement* pendant l'exécution de l'application après le démarrage, ou les deux (partiellement au démarrage, partiellement après).
   - Un assemblage **statique** est entièrement réalisé avant de démarrer les composants :
     - les composants peuvent être *créés et connectés* par la **machine virtuelle des composants** ;

Comment définir un composant logiciel ? III

     - les composants peuvent aussi être créés par la machine virtuelle des composants mais ensuite *se connecter eux-mêmes*, des clients vers les serveurs, au moment de leur démarrage.
   - Des composants peuvent aussi être créés **dynamiquement** par d'autres composants puis se connecter, pendant l'exécution.
6. Sous-composants
   - Un composant peut inclure en son sein des **sous-composants** (récursivement), mais les sous-composants sont *invisibles* de l'extérieur à moins que leur composant composite n'expose explicitement leurs services via ses propres ports et interfaces.
7. Déploiement
   - Les composants assemblés peuvent être déployés dans un **unique processus** (au sens du système d'exploitation).
   - Ils peuvent également être déployés sur **plusieurs processus**, sur **un seul ou plusieurs ordinateurs** (donc répartis).

Plan

Objectifs de la séquence

1. Objectifs pédagogiques
   - Décliner les concepts de la programmation par composants en Java avec BCM.
   - Introduire les premiers éléments de programmation par composants en BCM.
2. Compétences à acquérir
   - Prendre en main BCM et savoir retrouver dans cette bibliothèque et sa documentation la réalisation des différents concepts de l'approche à base de composants.

Qu'est-ce que BCM ?

- Basic Component Model : modèle de composants.
  BCM4Java = BCM for Java : implantation de BCM en Java.
- Une bibliothèque définissant une machine virtuelle pour composants s'exécutant en Java.
- Permet d'implanter des applications réparties :
  - a servi à implanter une application avec 10.000 composants déployés sur 50 JVM s'exécutant sur 5 ordinateurs ;
  - est utilisé dans des cours à Sorbonne Université depuis 2014 (une dizaine de projets de M1, 100 étudiants de M1 et plus de 150 étudiants de M2 à ce jour).
- Technologies Java utilisées (volontairement minimales et stables) :
  - J2SE (robuste aux passages de versions).
  - package `java.util.concurrent`
  - appel de méthodes à distance avec RMI.

D'une exécution mono-processus au réparti

- BCM vise principalement à produire des applications s'exécutant sur plusieurs processus (JVM exécutant BCM) répartis sur plusieurs hôtes.
- Le grand principe général qui guide sa conception et la programmation en BCM est :

  Toute application BCM programmée selon ses préceptes (à expliciter plus loin) pourra passer d'une exécution mono-processus à une exécution multi-processus voire en réparti par simple redéploiement des composants sur plusieurs JVM et répartition de ces dernières sur plusieurs hôtes, sans changer le code des composants.

- Pour y arriver, il faut se contraindre à appliquer les préceptes de programmation que nous allons présenter car BCM4Java n'a pas toujours la possibilité de les imposer par la contrainte.

Comment BCM4Java réalise les principaux concepts ?

- C'est un *framework* (cadriciel) écrit en Java.
- Un composant est un objet Java décrit par une classe marquée comme telle, dont les méthodes définissent ses services.
- Les interfaces de composants sont des interfaces Java marquées comme telles qui déclarent les signatures appelées/appelables.
- Les ports et les connecteurs sont des objets Java créés et détenus par les composants.
- Les connexions entre ports via les connecteurs se font
  - par référence Java s'ils sont dans la même JVM ;
  - par référence RMI s'ils sont dans des JVM différentes.

  Mais le programmeur n'a pas à s'en soucier autrement qu'en faisant en sorte que ses appels aient la même sémantique en appel local qu'en appel RMI (on y reviendra).
- Un sous-composant est représenté par un objet dont la référence est détenue par son composite.

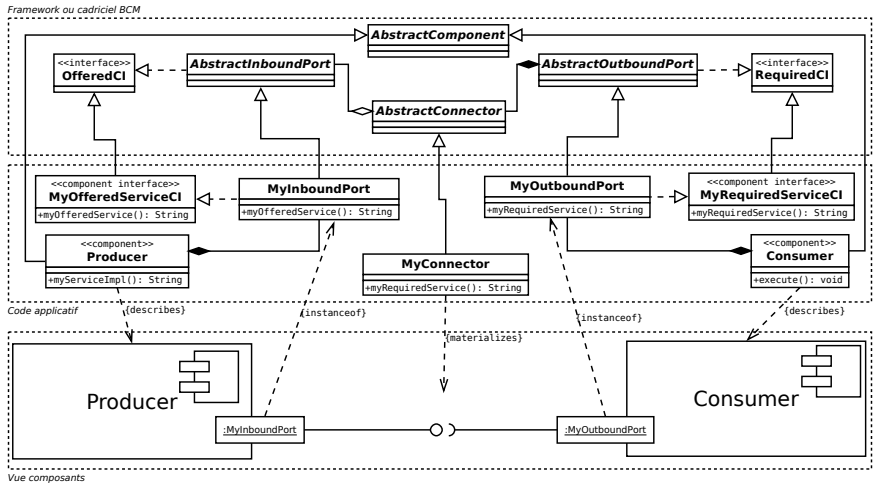
Comment réaliser les assemblages et le déploiement ?

- Les assemblages de composants *statiques* et leurs déploiements sont décrits dans des classes particulières définissant le « main » de la machine virtuelle pour composants.
- Le déploiement peut se faire au sein d'un processus, c'est-à-dire dans une seule JVM, auquel cas l'application s'exécute comme un programme Java classique.
- Le déploiement peut aussi se faire dans plusieurs processus, c'est-à-dire plusieurs JVM, auquel cas il faudra lancer autant de processus que de JVM plus des processus externes pour gérer les registres et la synchronisation des déploiements statiques.
  - Les détails seront abordés plus tard ;
  - une restriction : le nombre de processus participant à l'exécution d'une application est fixé définitivement au départ.

Le *framework* BCM

- La bibliothèque BCM4Java est en fait un *framework* ou cadriciel.
  - *framework* : ensemble d'interfaces, de classes abstraites et de classes concrètes définissant un noyau applicatif et que les utilisateurs doivent implanter (interfaces) et étendre (classes abstraites) pour développer leur application.
    Ex.: *framework* d'interfaces graphiques ou d'applications web.
- BCM fournit (entre autres) :
  - `AbstractComponent` que toute classe décrivant un composant doit étendre ; définit les opérations internes aux composants.
  - Des interfaces de composants `OfferedCI` et `RequiredCI` que toutes les interfaces de composants doivent étendre.
  - Différentes classes comme `AbstractInboundPort` et `AbstractOutboundPort` que tous les ports doivent étendre.
  - `AbstractConnector` que tous les connecteurs doivent étendre.
  - Une classe `AbstractCVM` qui doit être étendue pour définir les assemblages et le déploiement des composants pour exécution ; elle définit les opérations de la CVM sur les composants.

Illustration partielle du *framework* BCM



Programmation en BCM4Java

- Développer en BCM4Java consiste à programmer puis assembler des composants en suivant les étapes suivantes :
  1. Définir les interfaces de composants offertes et requises.
  2. Définir en Java les classes et les signatures des services implantés pour chaque type de composants.
  3. Implanter les classes de ports entrants et sortants.
  4. Programmer les méthodes des services de chaque composant.
  5. Implanter les classes de connecteurs.
  6. Implanter la classe CVM pour assembler les composants puis exécuter l'application obtenue.
- L'une des principales difficultés pour bien programmer en BCM4Java est de bien comprendre la distinction entre :
  - le code de l'application à base de composants (services implantés dans les classes définissant les composants) et
  - le code d'implantation de BCM4Java dans le *framework* qui est aussi directement appelable

  car les deux sont en Java !

Langage d'implantation versus langage implanté

- Dans toute implantation de langage de programmation, on se retrouve avec une hiérarchie de langages : un langage, par exemple Java, est implanté dans un langage, par exemple C, dans lequel on programme la machine virtuelle Java.
  - Dans ce contexte, pas d'ambiguïté car il ne nous n'est pas possible de mélanger dans un programme du code en Java avec du code en C, le compilateur assurant l'étanchéité entre les deux.
- Pourtant, certains langages, surtout interprétés, sont partiellement ou totalement implantés en eux-mêmes (par exemple, Scheme en Scheme, Lisp en Lisp ou Prolog en Prolog).
- En BCM4Java, les classes comme `AbstractComponent` fournissent des méthodes (en Java) qui définissent des opérations de la machine virtuelle BCM qu'on ne doit utiliser que dans certains contextes :
  - dans la classe du composant sur lequel l'opération est effectuée ;
  - dans certains cas, dans le code de déploiement statique (*i.e.*, classes héritant de `CVM`) des composants.

Plan

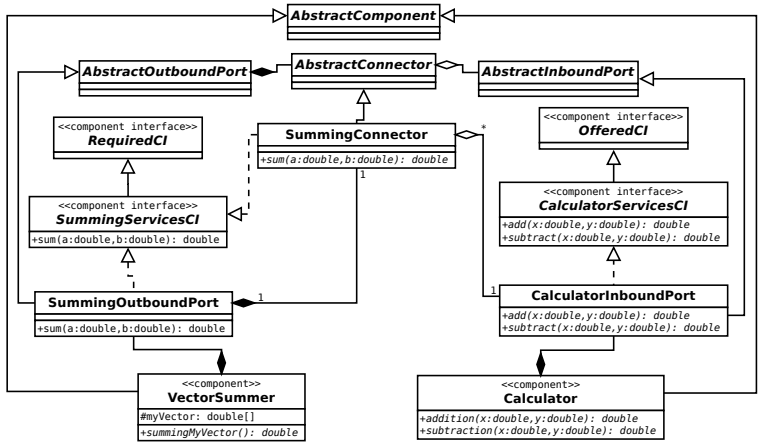
Objectifs de la séquence

1. Objectifs pédagogiques
   - Apprendre les rudiments de programmation en BCM grâce à un premier exemple complet.
2. Compétences à acquérir
   - Être en mesure de décrire et de suivre les grandes étapes de la mise en œuvre d'une application simple avec BCM du départ jusqu'à une exécution sur une (unique) machine virtuelle Java.
   - Savoir programmer les différents éléments d'une application BCM dans une première version simple : interfaces de composants, composants, ports, connecteurs, interconnexion et déploiement sur une (unique) machine virtuelle Java.

Premier exemple complet : somme d'un vecteur

- Pour comprendre comment définir une application en BCM, nous allons développer sous Eclipse un premier exemple complet.
- Cet exemple est volontairement simple pour mettre l'accent sur les mécanismes des composants de BCM.
- Cahier des charges :
  - Un composant offre un ensemble de services de calcul (addition, soustraction, multiplication et division).
  - Un composant client requiert ces services de calcul pour réaliser la somme des valeurs contenues dans un vecteur.

Diagramme de l'exemple



Activités à réaliser avant le prochain TME

1. Récupérer la bibliothèque BCM4Java dans la rubrique Projet du site de l'UE.
   - Le jar de BCM4Java à utiliser comme tel dans vos projets sous Eclipse pour séparer nettement votre code du code de BCM4Java.
   - L'archive de sources de BCM4Java à installer comme projet séparé dans votre workspace pour consulter facilement le code source de BCM4Java, sa documentation et les exemples fournis.
2. Examiner le code de l'exemple `basic_cs` et sa documentaiton pour comprendre comment développer une première application en BCM4Java.
3. Récupérer le cahier des charges du projet dans la rubrique Projet du site de l'UE, le lire et préparer vos questions.
4. Réfléchir à former vos équipes et envoyer à `Jacques.Malenfant@lip6.fr` un message comportant un nom d'équipe (voir le cahier des charges) et les noms de ses deux membres.