MAD – *M*iddleware et *A*pplications a*D*apatables

Introduction aux composants

Sara Bouchenak, Sara.Bouchenak@imag.fr

http://sardes.inrialpes.fr/~bouchena/teaching/MAD/

Des objets …

- Limitations des objets (comme unité de construction)
  - Pas de possibilité d'expression de besoins "non fonctionnels" d'une application (ex. persistance, sécurité, etc.)
  - Pas d'expression explicite des ressources requises par un objet (autres objets, services fournis par l'environnement)
  - Pas de vue globale de l'architecture d'une application
  - Peu d'outils pour le déploiement et l'administration

… aux composants

- Notion de composant logiciel (unité de construction)
  - Nécessité perçue très tôt [McIlroy 1968] …
  - … mais conception et réalisation n'ont pas suivi
  - Les objets sont une première approche pour la décomposition
    - Encapsulation (séparation interface-réalisation)
    - Mécanismes de réutilisation (héritage, délégation)
  - Ce que les composants apportent de plus
    - Architecture (structure) de l'application, déploiement
    - Intégration de ressources et de besoins non fonctionnels

Plan

- Modèles et infrastructures à composants
- Composition
  - Paradigmes
  - Exemples
  - ADL
- Organisation des composants
  - Contrôleur
  - Conteneur
- Gestion des composants
  - Cycle de vie des composants
  - Désignation, liaison et appel de composant

Mise en œuvre des composants

- Pour mettre en œuvre des composants, il faut
  - Un modèle à composants
  - Une infrastructure à composants

Modèle à composants

- Un modèle de composants définit
  - les entités
  - leur mode d'interaction
  - leur mode de composition
- Plusieurs niveaux de modèles
  - Abstrait (définition des entités, de leurs relations)
  - Concret (représentation particulière du modèle abstrait)

Infrastructure à composant

- Une infrastructure à composants
  - met en œuvre le modèle
  - permet de
    - construire
    - déployer
    - administrer
    - exécuter des applications conformes au modèle

Modèles et infrastructures : Situation actuelle

- Dans le domaine de la recherche
  - Des modèles à composants
    - Fractal
    - Piccola
  - Des infrastructures prototypes
    - Julia/Fractal
- Des infrastructures industrielles
  - EJB,
  - CCM,
  - .NET,
  - OSGi

Que doit fournir un modèle à composants ? (1)

- Encapsulation
  - Séparation entre *interface* et *réalisation*
  - Interfaces = seule voie d'accès à un composant
  - Possibilité de définir des interfaces multiples (plusieurs vues d'un même composant)
- Composition
  - Dépendances explicites (expression des ressources fournies *et* requises)
  - Composition hiérarchique (un assemblage de composants est un composant)

Que doit fournir un modèle à composants ? (2)

- Description globale
  - Si possible exprimée formellement (langage de description)
- Réutilisation et évolution
  - Modèles génériques de composants
  - Adaptation (interface de contrôle)
  - Reconfiguration

Que doit fournir une infrastructure à composants ? (1)

- Couverture du cycle de vie
  - Non limitée aux phases de développement et d'exécution
  - Administration
    - Déploiement : installation et activation des composants
    - Surveillance : collecte et intégration de données, tenue à jour de la configuration
    - Contrôle : réaction aux événements critiques (alarme, surcharge, détection d'erreur)
  - Maintenance
    - Maintien de la disponibilité de l'application
    - Évolution (redéploiement, reconfiguration) pour réagir à l'évolution des besoins et de l'environnement

Que doit fournir une infrastructure à composants ? (2)

- Services communs
  - Propriétés non fonctionnelles
  - Ex.
    - Persistance
    - Transactions
    - Sécurité
    - QoS (*Quality of Service* : qualité de service)

Plan

- Modèles et infrastructures à composants
- **Composition**
  - **Paradigmes**
  - Exemples
  - ADL
- Organisation des composants
  - Contrôleur
  - Conteneur
- Gestion des composants
  - Cycle de vie des composants
  - Désignation, liaison et appel de composant

Paradigmes de la composition (1)

- Composant
  - Unité de composition
  - Unité de déploiement
  - Remplit une fonction spécifique
  - Peut être assemblé avec d'autres composants
  - Donc, porte une description des interfaces requises et fournies
- Connecteur
  - Élément permettant d'assembler des composants en utilisant leurs interfaces fournies et requises
  - Remplit 2 fonctions : liaison et communication

Paradigmes de la composition (2)

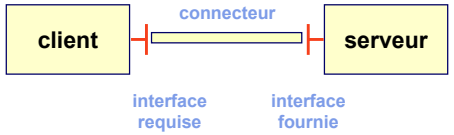
- Configuration
  - Un assemblage de composants
  - Peut, par exemple, être elle-même un composant (selon le modèle)
- N.B.
  - La différence entre un composant et un connecteur est une différence de fonction, non de nature (un connecteur est lui-même un composant)

Plan

- Modèles et infrastructures à composants
- **Composition**
  - Paradigmes
  - **Exemples**
  - ADL
- Organisation des composants
  - Contrôleur
  - Conteneur
- Gestion des composants
  - Cycle de vie des composants
  - Désignation, liaison et appel de composant

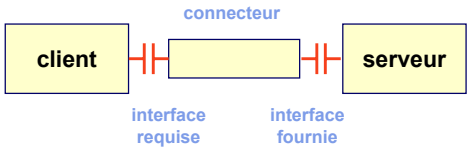
Exemples de schémas de composition (1)

Application client-serveur locale



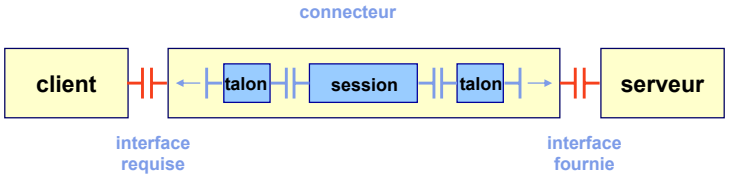
Exemples de schémas de composition (2)

Une autre vue de l'application client-serveur locale :
Le connecteur comme composant
Ex. le connecteur effectue une fonction plus évoluée



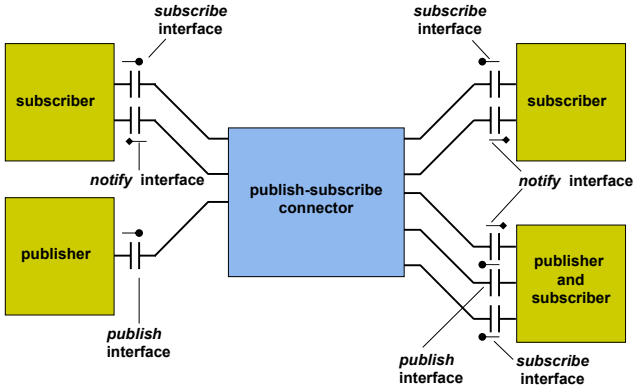
Exemples de schémas de composition (3)

Application client-serveur répartie :
Le connecteur comme composant composite
Ex. le connecteur gère la communication entre les deux composants répartis



Exemples de schémas de composition (4)

Connecteur plus évolué : *publish-subscribe*



Plan

- Modèles et infrastructures à composants
- **Composition**
  - Paradigmes
  - Exemples
  - **ADL**
- Organisation des composants
  - Contrôleur
  - Conteneur
- Gestion des composants
  - Cycle de vie des composants
  - Désignation, liaison et appel de composant

Décrire la composition

- Situation actuelle
  - Notion d'ADL (*Architecture Description Language*)
  - Pas de standard reconnu (des tentatives)
    - ACME, ADML, Xarch, UML-2
  - Vers l'utilisation de XML comme support commun (invisible) ?

Éléments d'un ADL (1)

- Description des composants
  - Définition des interfaces
    - fournies
    - requises
  - Types des interfaces
    - synchrone
    - asynchrone
  - Signatures, contrats, annotations

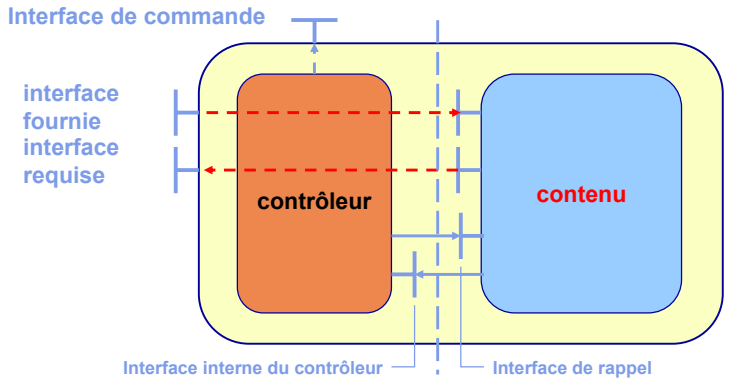
Éléments d'un ADL (2)

- Description des configurations
  - Association interfaces fournies - interface requises
    - Directe (réalisée par édition de liens)
    - Via un connecteur : description, "rôle" = interface
  - Composition hiérarchique (si le modèle l'autorise)
    - Interfaces exportées/importées
  - Interface graphique (visualisation, action)
  - Aspects dynamiques

Plan

- Modèles et infrastructures à composants
- Composition
  - Paradigmes
  - Exemples
  - ADL
- **Organisation des composants**
  - **Contrôleur**
  - **Conteneur**
- Gestion des composants
  - Cycle de vie des composants
  - Désignation, liaison et appel de composant

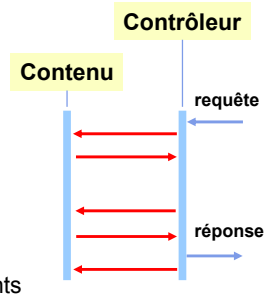
Organisation logique d'un composant



La séparation entre contrôleur et contenu est motivée par la séparation des préoccupations : isoler la partie purement fonctionnelle (propre à l'application)

Fonctions d'un contrôleur

- Gestion du cycle de vie
  - Création, destruction
  - Activation, passivation
- Gestion des composants inclus
  - … si le modèle comporte des composants composites
- Liaison
  - Connexion avec d'autres composants
- Médiation
  - Gestion des interactions avec d'autres composants
  - Médiation pour la fourniture de services externes
- Autres opérations réflexives
  - … si le modèle le permet



Conteneur : Un canevas pour composants (1)

- Infrastructures à conteneurs
  - Canevas de base pour le support des composants
  - Séparation des préoccupations
    - La partie "contenu" réalise les fonctions de l'application
    - La partie "conteneur" fournit les fonctions de l'infrastructure
  - Fonctions du conteneur
    - Mise en œuvre des fonctions du contrôleur (gestion, médiation)
    - Allocation des ressources aux composants
    - Réalisation des aspects "non fonctionnels" (services communs)

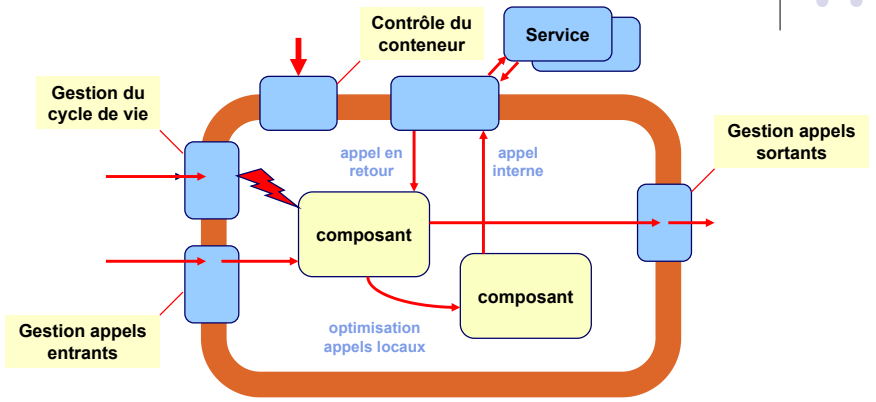
Conteneur : Un canevas pour composants (2)

- Exemples (sous des formes diverses)
  - *Enterprise Java Beans* (EJB)
  - *CORBA Component Model* (CCM)
  - Julia (réalisation du modèle Fractal)
  - OSGi

Conteneur

- Conteneur : Mise en œuvre du contrôleur de composants
  - Infrastructure de gestion pour un ou plusieurs composants
  - Généralement, un conteneur par “type” de composants
- Rappel des fonctions : cycle de vie, médiation, gestion de services, composition (si composants composites)
- Génération automatique des contrôleurs à partir
  - des descriptions d’interfaces des composants (cf. talons)
  - de paramètres de configuration fournis
- “Contrats” entre le conteneur et les composants inclus
  - Interface interne du contrôleur
  - Interfaces de rappel des composants

Schéma générique de conteneur

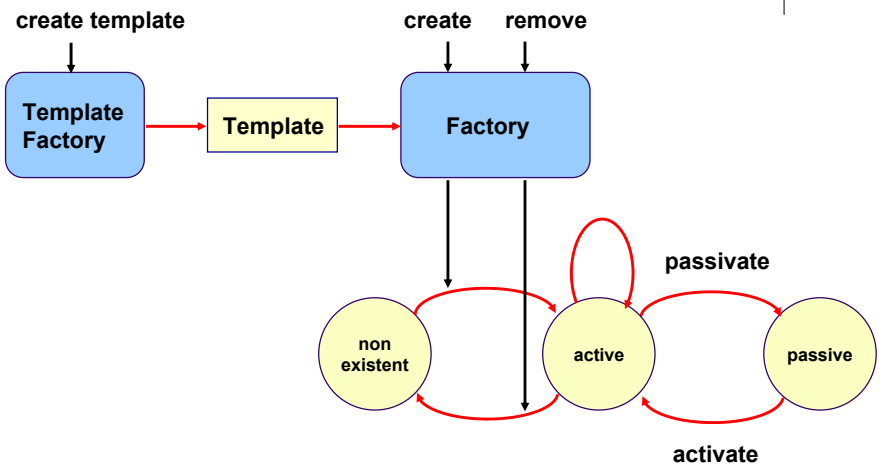


Illustrations spécifiques dans les autres cours : EJB, CCM, Julia, OSGi, …

Plan

- Modèles et infrastructures à composants
- Composition
  - Paradigmes
  - Exemples
  - ADL
- Organisation des composants
  - Contrôleur
  - Conteneur
- **Gestion des composants**
  - **Cycle de vie des composants**
  - Désignation, liaison et appel de composant

Cycle de vie des composants (1)



Cycle de vie des composants (2)

- Maison (*home)*
  - Partie du conteneur (visible de l'extérieur) qui gère le cycle de vie des composants qu'il inclut (composants du même type)
    - Engendré automatiquement (à partir d'un IDL)
  - Fonctions
    - Créer / détruire les composants : implémente *Factory*
    - Gérer les composants pendant leur existence (en particulier nommer, localiser)
  - Interface fournie
    - *create*, *find*, *remove*
    - Utilise une interface de rappel (à implémenter par le programmeur)

Cycle de vie des composants (3)

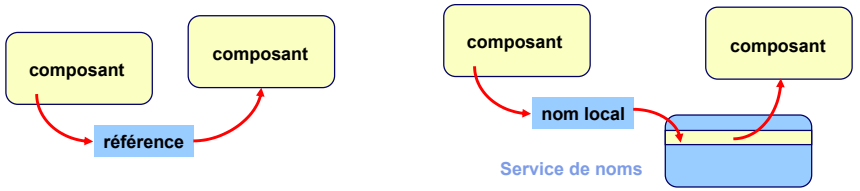
- Mécanismes pour économiser les ressources
  - Activation-passivation
    - *passivate* : Élimine le composant de la mémoire si non utilisé - appelé par le conteneur
    - *activate* : Restitue l'état du composant, peut alors être réutilisé
  - *Pool* d'instances
    - Le conteneur maintient un *pool* fixe d'instances de composants qui peuvent donner vie à des composants "virtuels"
    - Un composant virtuel incarné par une instance du *pool* devient utilisable, à condition de charger son état
    - Évidemment plus simple pour les composants sans état
  - Différences entre *passivation* et *pool*
    - Les deux mécanismes s'appliquent à des types différents de composants (c.f. plus loin)

Plan

- Modèles et infrastructures à composants
- Composition
  - Paradigmes
  - Exemples
  - ADL
- Organisation des composants
  - Contrôleur
  - Conteneur
- **Gestion des composants**
  - Cycle de vie des composants
  - **Désignation, liaison et appel de composant**

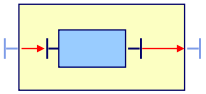
Désignation des composants

- Principe : désignation contextuelle
  - Désignation dans un contexte global
    - Service de noms (CORBA, J2EE/JNDI, …)
  - Désignation locale pour éviter noms globaux “en dur”
    - Symbolique
    - Référence



Liaison des composants

- Plusieurs types de liaison selon localité
  - Entre composants de même niveau dans un même espace: référence dans cet espace (exemple : référence Java)
  - Entre composants dans des espaces différents : objet de liaison (avec éventuellement optimisations locales)
    - Exemple EJB :
      - Dans un même conteneur : LocalObject
      - Entre deux conteneurs : RemoteObject
      - Entre client et serveur répartis : RMI (stub + skeleton)
  - Entre composants inclus (dans les modèles qui l’autorisent)
    - Exportation - Importation



Appel d’un composant (1)

- Principe
  - Le conteneur réalise une médiation pour l’appel d’une méthode d’un composant, via un objet intermédiaire (intercepteur)
  - Comme le reste du conteneur, l’intercepteur est engendré automatiquement (fait partie de la chaîne de liaison)
  - Le conteneur fournit en général une optimisation locale pour les appels entre composants qu’il contient

Appel d’un composant (2)

- Fonctions réalisées
  - Lien avec gestion de ressources (activation d’un composant passivé)
  - Sécurité
  - Fonctions supplémentaires “à la carte” (intercepteurs programmables)

Vers des infrastructures à composants adaptables

- Motivations
  - Les systèmes commerciaux à composants actuels sont peu adaptables (structure fermée de conteneurs)
- Quelques pistes …
  - Conteneurs ouverts
    - Rendre visible la structure interne des conteneurs (séparation des fonctions)
    - Définir des interfaces de contrôle internes au conteneur
    - Piste suivie dans OpenCCM
  - Intercepteurs programmables
    - Permettre l'insertion d'intercepteurs dans la chaîne d'appel
      - Julia, JBoss (avec AOP)

MAD – *M*iddleware et *A*pplications a*D*apatables

EJB : Exemple d'infrastructure à composants

Infrastructures à composants (1)

- 2 infrastructures commerciales pour grandes applications
  - J2EE
    - utilise modèle de composants EJB
    - standard de Sun
    - http://java.sun.com/j2ee/
  - .NET
    - utilise modèle de composants COM+
    - Microsoft
    - http://www.microsoft.com/net/

Infrastructures à composants (2)

- Autres infrastructures
  - CCM : CORBA Component Model – standard (OMG) – voisin d'EJB
    - http://www.omg.org/technology/documents/formal/components.htm
  - OSGi : applications embarquées, domotique – standard (consortium)
    - http://www.osgi.org
  - Fractal/Julia – ObjectWeb – Logiciel libre
    - http://fractal.objectweb.org
  - Avalon – Apache – Logiciel libre
    - http://avalon.apache.org/

Plan

- Rappels
- Exemples de patrons de conception
- Fonctions du conteneur
  - Fonction d'isolation
  - Cycle de vie des composants
  - Fonction d'interposition
- Désignation des composants
- Adaptation de composants
- Déploiement de composants

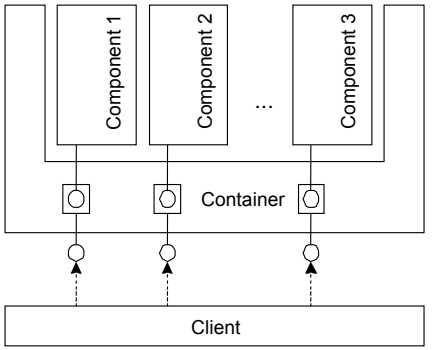
Rappel sur les conteneurs (1)

- Infrastructures à conteneurs
  - Canevas de base pour le support des composants
  - Séparation des préoccupations
    - La partie "contenu" réalise les fonctions de l'application
    - La partie "conteneur" fournit les fonctions de l'infrastructure
  - Fonctions du conteneur
    - Mise en œuvre des fonctions du contrôleur (gestion, médiation)
    - Allocation des ressources aux composants
    - Réalisation des aspects "non fonctionnels" (services communs)

Rappel sur les conteneurs (2)

- Conséquences pour le développeur
  - Format prédéfini
    - pour la réalisation des fonctions de l'application
    - pour la spécification des services
    - pour la fourniture de fonctions de rappel *(callback)*
  - Choix d'un *type* de composant (donc de conteneur) en fonction des besoins spécifiques

Conteneur : vue du client



- **Le conteneur fournit un mode d'accès standard aux composants et à la gestion de leur cycle de vie**
- **Le conteneur isole le client des aspects qui ne relèvent pas directement de l'application**
  - **caractéristiques de la plate-forme sous-jacente**
  - **aspects non-fonctionnels (sécurité, transactions, etc.)**

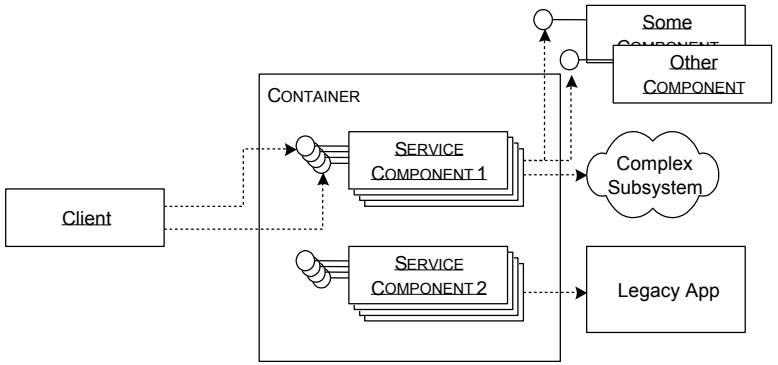
Plan

- Rappels
- **Exemples de patrons de conception**
- Fonctions du conteneur
  - Fonction d'isolation
  - Cycle de vie des composants
  - Fonction d'interposition
- Désignation des composants
- Adaptation de composants
- Déploiement de composants

Exemple 1 de composants EJB

- Patron de conception
  - Fourniture de service
  - Le service utilise d'autres applications existantes
- Solution
  - "Composants de service"
  - Sert de façade aux applications existantes utilisées
- Solution EJB
  - *Stateless Session Beans*

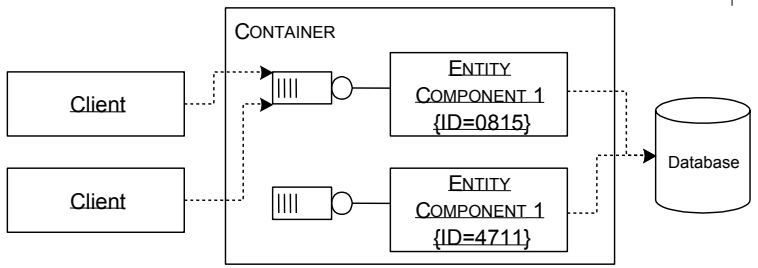
EJB – *Stateless Session Bean*



Exemple 2 de composants EJB

- Patron de conception
  - Application utilisant des données persistantes (SGBD)
- Solution
  - Composants contenant la représentation d'un état persistant
- Solution EJB
  - *Entity Beans*

EJB – *Entity Bean*

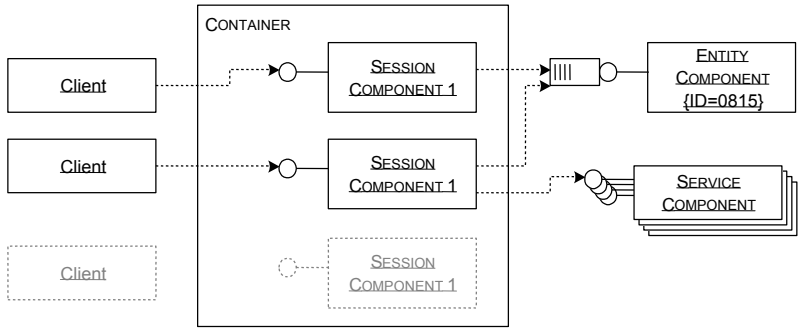


- Contraintes
  - Support de la persistance : par le composant, ou déléguée au conteneur
  - Lien entre composant et SGBD : clé primaire

Exemple 3 de composants EJB

- Patron de conception
  - Session de client (suite d'opérations)
  - Préservation de l'état de la session (ex. *cookie* pour une session HTTP)
  - Utilisation de données persistantes
- Solution
  - Composants de session avec état + Composants contenant la représentation d'un état persistant
- Solution EJB
  - *StatefulSessionBeans* + *Entity Beans*

EJB – *Stateful Session Bean + Entity Bean*

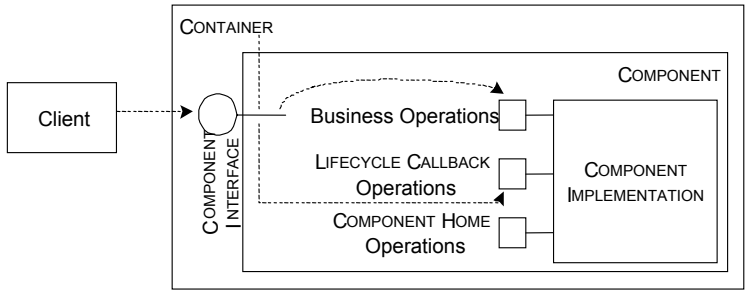


Plan

- Rappels
- Exemples de patrons de conception
- **Fonctions du conteneur**
  - **Fonction d'isolation**
  - Cycle de vie des composants
  - Fonction d'interposition
- Désignation des composants
- Adaptation de composants
- Déploiement de composants

Fonction d'isolation du conteneur (1)

- Le conteneur fournit une séparation entre
  - l'interface et l'implémentation d'un composant
  - l'interface fonctionnelle et l'interface réalisant le cycle de vie (*home*)
    - La gestion du cycle de vie impose au composant de fournir une interface de rappel (*callback*)



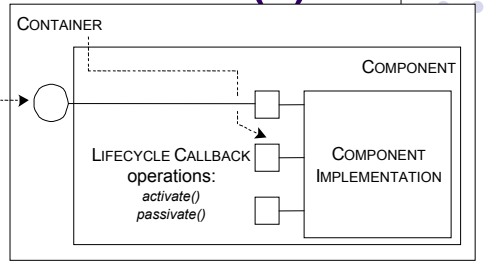
Fonction d'isolation du conteneur (2)

- Isolation mutuelle des composants
  - Un conteneur peut gérer plusieurs composants dans un même espace d'adressage
  - Il doit garantir leur isolation
  - Cela est réalisé en imposant des règles pour la programmation des composants. Exemple en EJB
    - Pas de *socket* serveur
    - Pas de code natif
    - …
  - Certaines de ces restrictions sont vérifiées lors du déploiement

Plan

- Rappels
- Exemples de patrons de conception
- **Fonctions du conteneur**
  - Fonction d'isolation
  - **Cycle de vie des composants**
  - Fonction d'interposition
- Désignation des composants
- Adaptation de composants
- Déploiement de composants

Cycle de vie : activation et passivation (1)

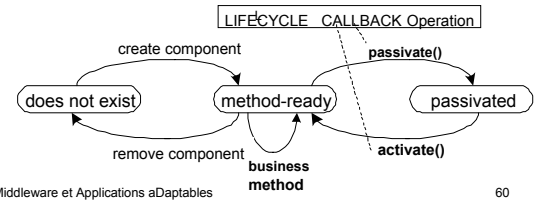


Objectif : économiser les ressources

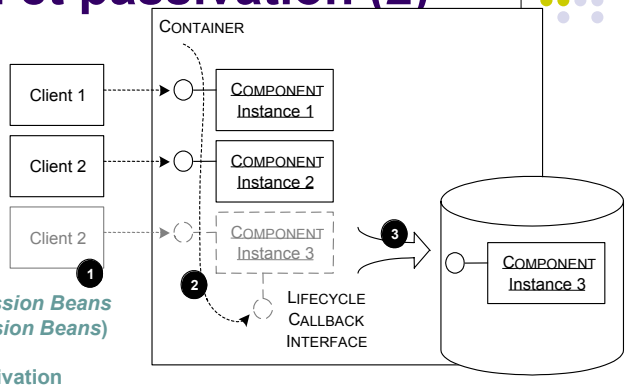
Passivation : éliminer un composant de la mémoire

Activation : recharger un composant en mémoire, en restaurant son état

Le composant doit fournir des opérations de rappel pour ces fonctions : *passivate*, *activate*



Cycle de vie : activation et passivation (2)



En EJB :
S'applique aux *Stateful Session Beans*
(inutile pour *Stateless Session Beans*)

***ejbPassivate*() avant passivation**
***ejbActivate*() après activation**

1: A client keeps a reference to an instance, but does not invoke any operations on it
2: The CONTAINER calls specific LIFECYCLE CALLBACK operations to prepare the instance for PASSIVATION
3: The instance is passivated by storing its state in a database
Later: Client can invoke operations again, instance is reactivated from database

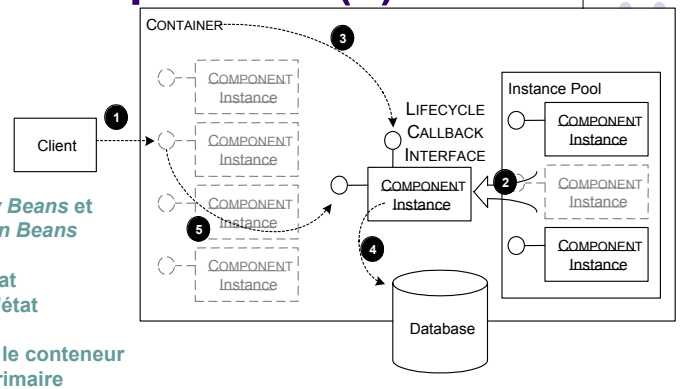
Cycle de vie : *pool* de composants (1)

- Séparation entre composant “logique” et composant “physique”
  - Un composant logique est tout composant de l’application
  - Pour être utilisé, un composant logique doit être incarné par un composant physique
  - Le *pool* est un ensemble de composants physiques
  - Sa taille est limitée
  - Le *pool* donne l’illusion que tout composant est disponible en permanence, alors qu’il n’y a de place que pour un nombre limité

Cycle de vie : *pool* de composants (2)

- Intérêt d’utilisation du *pool*
  - Limiter l’occupation des ressources
  - Réduire le nombre de créations – destructions (coûteuses)
    - Un composant “créé” est un composant pris du *pool*
    - Un composant “détruit” est un composant rendu au *pool*

Cycle de vie : *pool* de composants (3)



En EJB :
S'applique aux *Entity Beans* et aux *Stateless Session Beans*

***ejbStore*() range l'état**
***ejbLoad*() restaure l'état**

Pour un *Entity Bean*, le conteneur doit modifier la clé primaire

1: A client invokes an operation on a VIRTUAL INSTANCE
2: CONTAINER takes an instance from the instance pool
3: CONTAINER invokes suitable LIFECYCLE CALLBACK operations to prepare the instance
4: Instance load its data from the database
5: Invocation is forwarded to the instance
Later: COMPONENT Instance will be put back to the pool

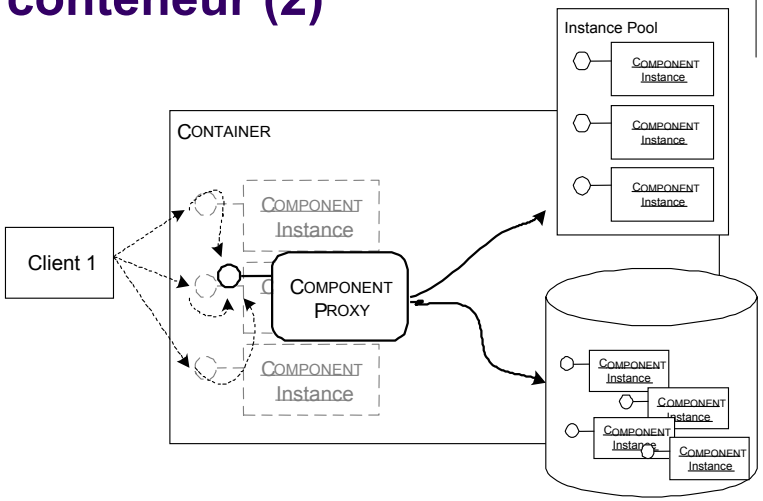
Plan

- Rappels
- Exemples de patrons de conception
- **Fonctions du conteneur**
  - Fonction d'isolation
  - Cycle de vie des composants
  - **Fonction d'interposition**
- Désignation des composants
- Adaptation de composants
- Déploiement de composants

Fonction d'interposition du conteneur (1)

- Le client ne doit pas avoir connaissance des opérations
  - d'activation/passivation des composants
  - de gestion du pool :
  - Ce sont des opérations "techniques" internes au conteneur
- Le client n'a qu'une référence indirecte aux composants (via un *proxy*)
- Le *proxy* est engendré automatiquement

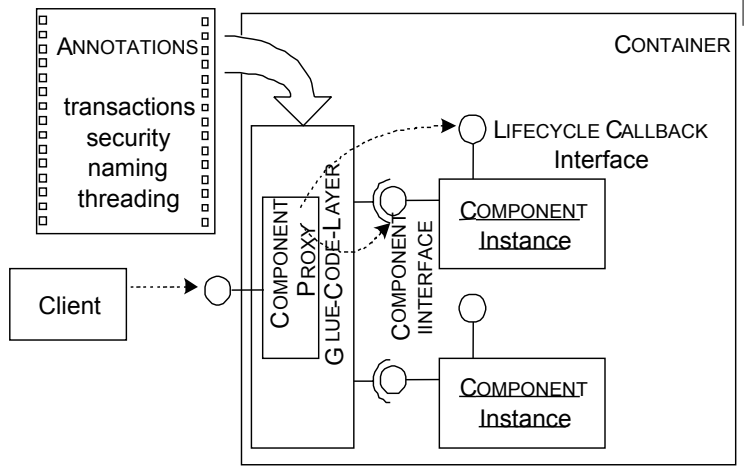
Fonction d'interposition du conteneur (2)



Fonction d'interposition du conteneur (3)

- Le *proxy* est généré automatiquement
- Le *proxy* prend en charge également l'aiguillage vers diverses fonctions techniques à la charge du conteneur
  - transactions,
  - sécurité,
  - nommage,
  - etc.

Fonction d'interposition du conteneur (4)



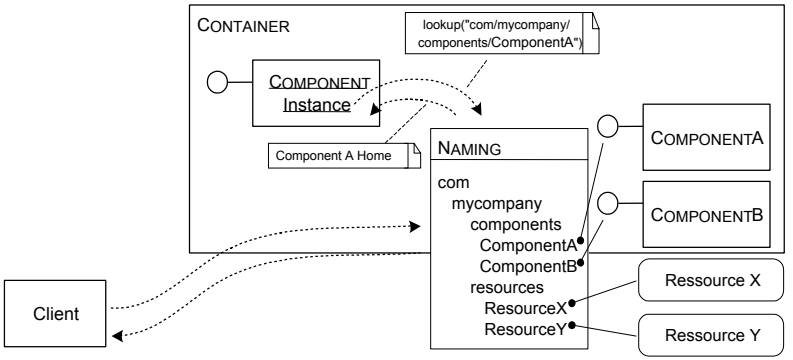
Plan

- Rappels
- Exemples de patrons de conception
- Fonctions du conteneur
  - Fonction d'isolation
  - Cycle de vie des composants
  - Fonction d'interposition
- **Désignation des composants**
- Adaptation de composants
- Déploiement de composants

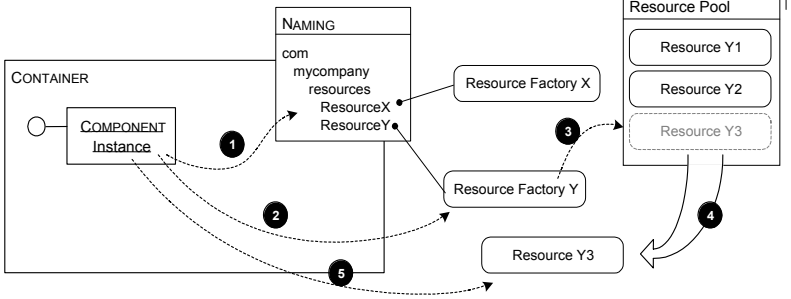
Désignation (1)

- Fonction du service de désignation : fournir une correspondance entre nom et référence d'objet
- Utilisé pour désigner toute entité (composant, ressource)
- Accessible initialement par un service connu
- En EJB : accessible à travers JNDI (interface standard)

Désignation (2)

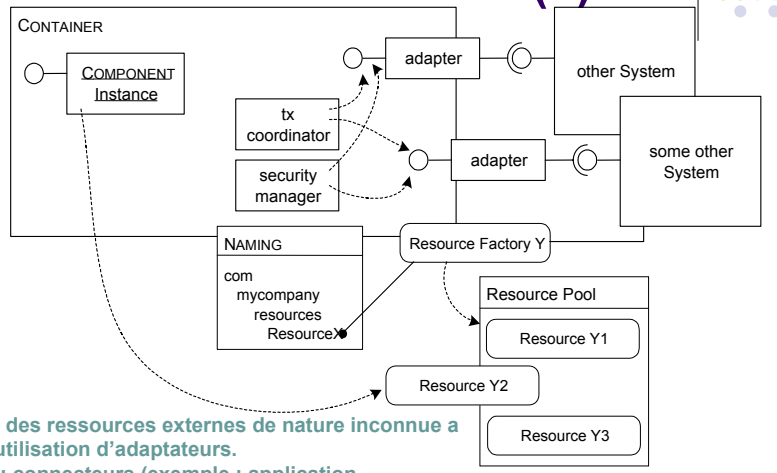


Gestion des ressources (1)



Before: CONTAINER sets up a resource pool
1: Client looks up a resource factory in the NAMING service
2: Client accesses factory to request a resource
3: Resource factory accesses the ressource pool, and
4: a resource is taken from the pool, passed to the client
5: Client accesses the resource, integrated with the technical concerns such as security and transactions

Gestion des ressources (2)



Accès à des ressources externes de nature inconnue a priori : utilisation d'adaptateurs.
En EJB : connecteurs (exemple : application patrimoniale (*legacy*) telle qu'une BD

Plan

- Rappels
- Exemples de patrons de conception
- Fonctions du conteneur
  - Fonction d'isolation
  - Cycle de vie des composants
  - Fonction d'interposition
- Désignation des composants
- **Adaptation de composants**
- Déploiement de composants

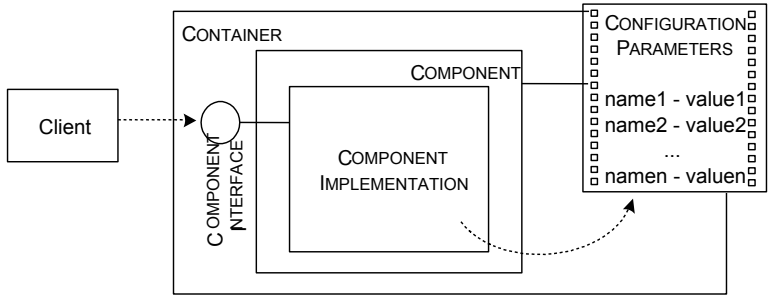
Adaptation des composants (1)

- Motivation : la réutilisation
  - Pour réutiliser un composant dans différents environnements, une adaptation doit être possible
  - On doit pouvoir changer :
    - Des paramètres de configuration
    - Les implémentations d'autres composants dont le composant considéré utilise les interfaces

Adaptation des composants (1)

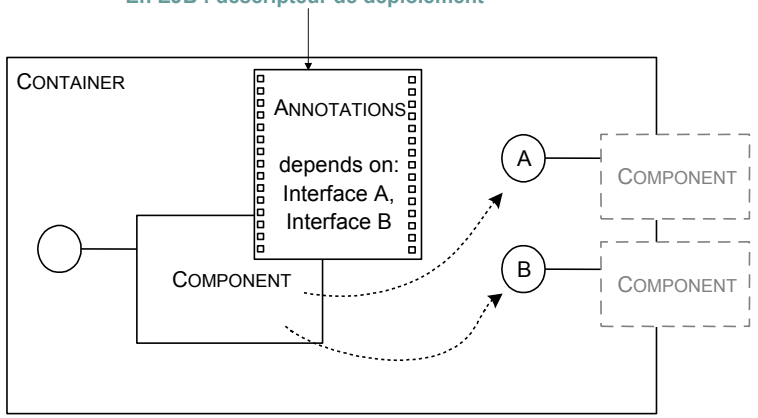
- Motivation
  - La réutilisation
  - Pour réutiliser un composant dans différents environnements, une adaptation doit être possible
- On doit pouvoir changer
  - Les paramètres de configuration
  - Les implémentations d'autres composants dont le composant considéré utilise les interfaces

Adaptation des composants (2)



Adaptation des composants (3)

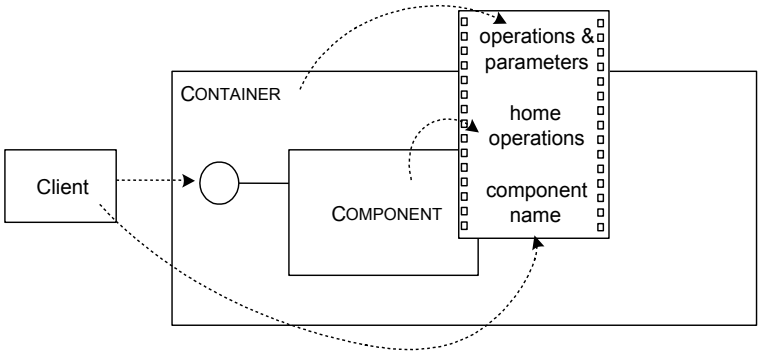
En EJB : descripteur de déploiement



Adaptation des composants (4)

Introspection :
possibilité pour le composant de connaître des informations sur lui-même

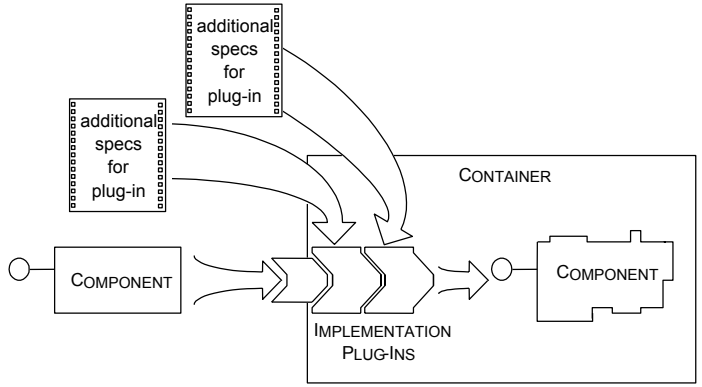
Utilisation possible de Java Reflection, *EJBMetaData*



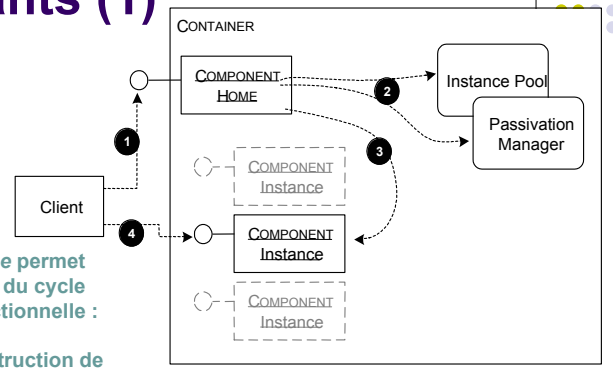
Adaptation des composants (5)

Possibilité d'ajouter du code (automatiquement généré) dans l'implémentation d'un composant

Exemple : *Container Managed Persistence* (CMP) : *ejbLoad*(), *ejbStore*() engendrés lors du déploiement



Gestion du cycle de vie des composants (1)



En EJB : l'interface *Home* permet de séparer les fonctions du cycle de vie de l'interface fonctionnelle :

création, recherche, destruction de composants

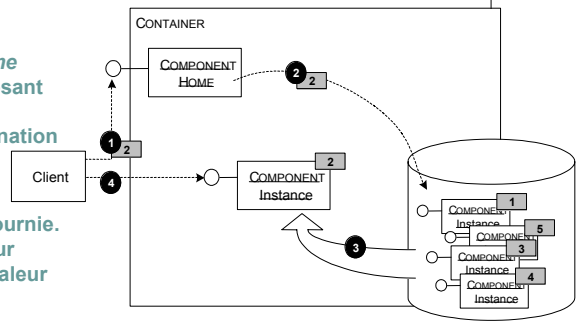
Before: Client looks up the COMPONENT HOME in NAMING
1: Client accesses COMPONENT HOME requesting a (new) COMPONENT instance
2: Depending on the COMPONENT kind, the COMPONENT HOME accesses the instance pool or the passivation manager
3: The COMPONENT HOME provides an instance to the client
4: The client accesses the instance

Gestion du cycle de vie des composants (2)

Les détails de l'interface *Home* dépendent du type de composant

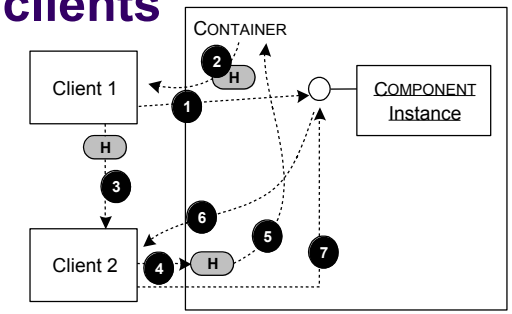
Cas des *Entity Beans* : désignation par clé primaire

Une classe *PrimaryKey* est fournie. Elle doit être sérialisable, pour pouvoir passer les clés par valeur



Before: Client looks up a COMPONENT HOME in NAMING
1: Client accesses the COMPONENT HOME, providing the PRIMARY KEY as a parameter
2: COMPONENT HOME accesses the database to locate the data for the instance for the PRIMARY KEY
3: instance is created/taken from the pool/activated. Instance is associated with the PRIMARY KEY
4: Client accesses the instance

Transfert de références entre clients



En EJB : utilisation de poignées (*Handle*)

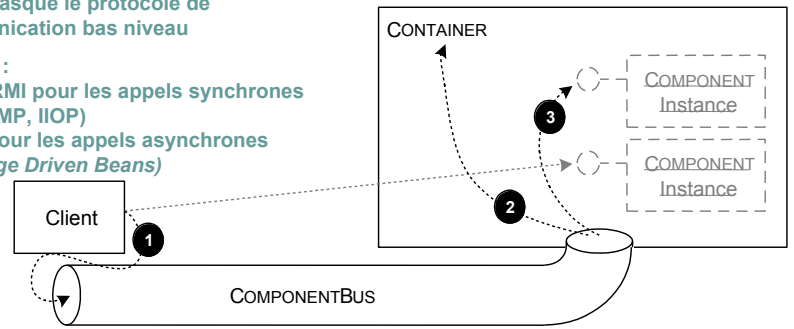
Javax.ejb.Handle getHandle()

1: Client invokes a specific operation on an instance to request a HANDLE
2: HANDLE is created by the CONTAINER and returned to the client
3: HANDLE is stored, or passed to another client
4: Other client accesses HANDLE and requests the original instance reference
5: HANDLE accesses the CONTAINER, which
6: returns the instance from which the HANDLE was created
7: Client accesses original instance

Appel de composants à distance (1)

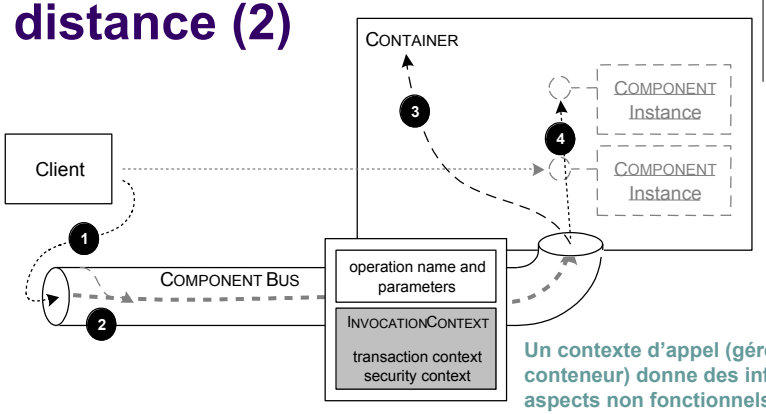
Le système de communication logique (bus) masque le protocole de communication bas niveau

En EJB :
- Java RMI pour les appels synchrones (sur JRMP, IIOP)
- JMS pour les appels asynchrones *(Message Driven Beans)*



1: Client invokes an operation on the remote VIRTUAL INSTANCE
2: COMPONENTBUS accesses the CONTAINER to manage the technical concerns
3: Invocation is forwarded to the VIRTUAL INSTANCE

Appel de composants à distance (2)

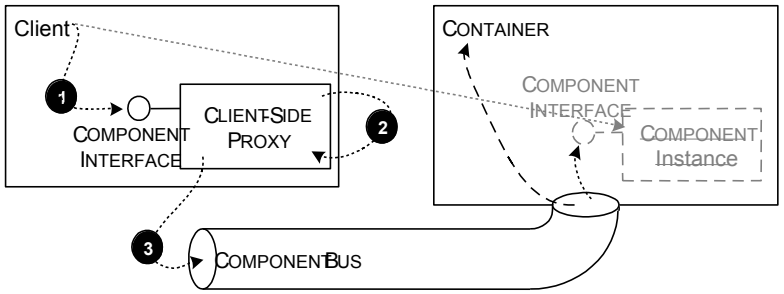


Un contexte d'appel (géré par le conteneur) donne des info. sur les aspects non fonctionnels (ex. transactions, sécurité, etc.)

1: Client invokes an operation on a remote VIRTUAL INSTANCE
2: COMPONENT BUS adds the data for the INVOCATION CONTEXT
3: COMPONENT BUS forwards the data from the INVOCATION CONTEXT to the CONTAINER
4: It forwards the invocation to the VIRTUAL INSTANCE

Appel de composants à distance (3)

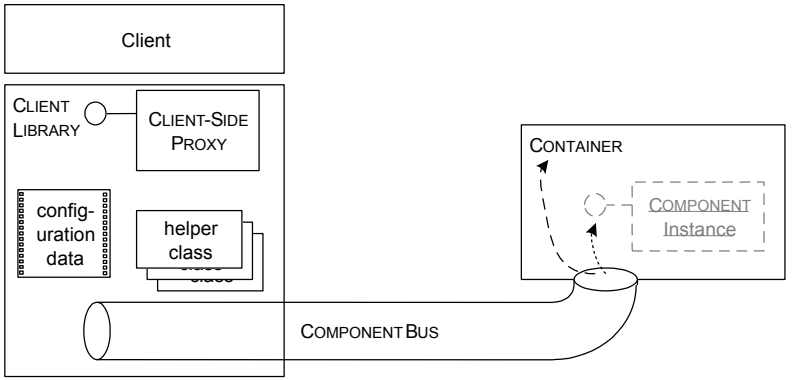
Côté client, schéma classique : *proxy* client généré automatiquement
Proxy coopère avec le conteneur, par exemple pour passer le contexte



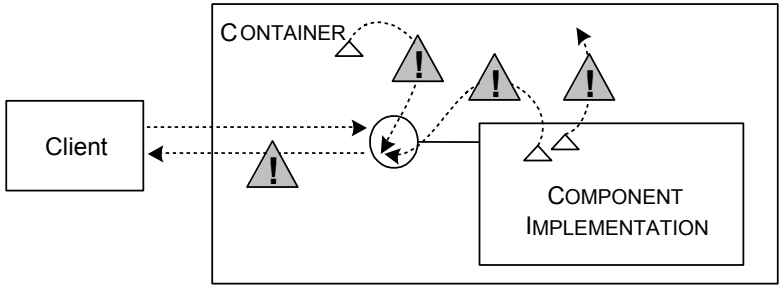
1: Client invokes an operation on VIRTUALINSTANCE; in reality it reaches the CLIENT-SIDE PROXY
2: Proxy adds the data for the INVOCATIONCONTEXT
3: Proxy hands over the invocation to the COMPONENTBUS

Appel de composants à distance (4)

Côté client, diverses classes auxiliaires (*helpers*, etc) sont engendrées automatiquement, outre le *proxy*



Gestion des exceptions



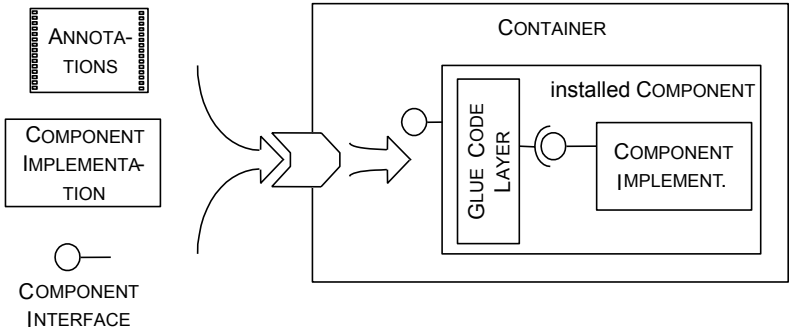
- Différentes sortes d'exception peuvent survenir
  - liées à l'application ("normales")
  - provenant du traitement d'aspects non fonctionnels (dans le conteneur)
    Ces dernières doivent pouvoir être retournées au client
- En EJB : le conteneur récupère les *EJBException*s et lance des *RemoteException*s vers le client

Plan

- Rappels
- Exemples de patrons de conception
- Fonctions du conteneur
  - Fonction d'isolation
  - Cycle de vie des composants
  - Fonction d'interposition
- Désignation des composants
- Adaptation de composants
- **Déploiement de composants**

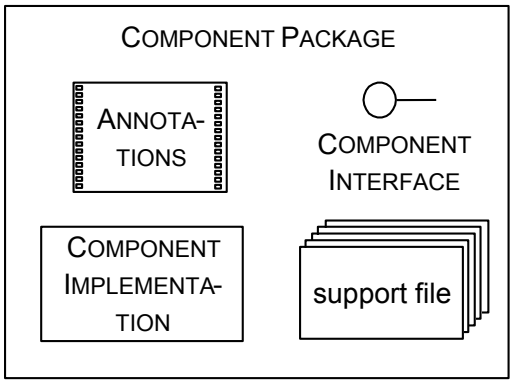
Déploiement (1)

- Déploiement = mise en place des composants avant exécution
- Le processus de déploiement peut être automatisé
- À partir de descriptions déclaratives, le conteneur crée le code nécessaire au déploiement



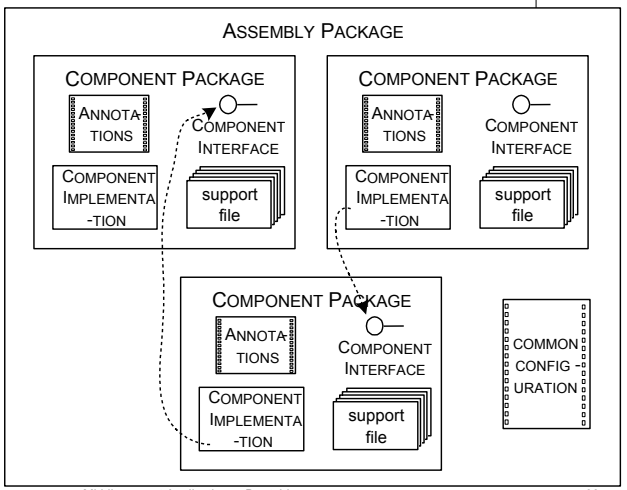
Déploiement (2)

- En EJB, le descripteur de déploiement et les différentes parties du composant (interfaces *Home* et *Remote*, implémentation) sont empaquetés dans un format standard (fichier jar)



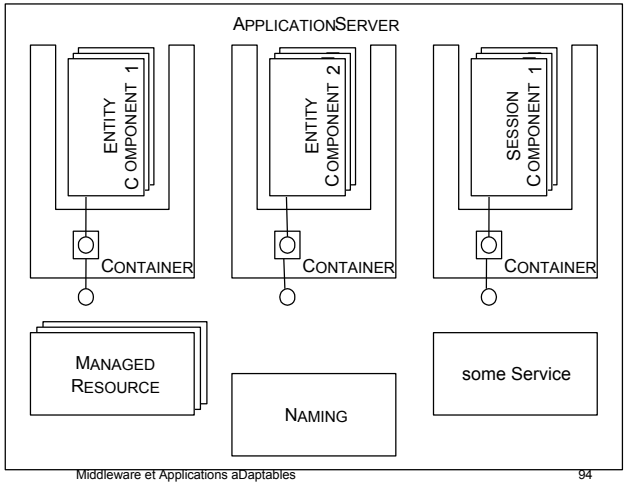
Déploiement (3)

En EJB, l'ensemble des composants d'une application sont rassemblés dans un format standard : EAR (*Enterprise ARchives*)



Intégration

L'ensemble des conteneurs EJB, des autres conteneurs (Servlets), des services et ressources sont regroupés dans un serveur d'application J2EE

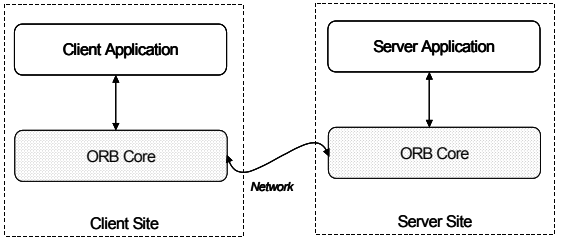


MAD – *M*iddleware et *A*pplications a*D*apatables

CORBA/CCM : Exemple d'infrastructure à composants

Qu'est-ce que CORBA ?

- CORBA (Common Object Request Broker Architecture)
- Un middleware (une plate-forme) pour l'exécution d'applications réparties



Motivations de CORBA

- **Répartition**. Abstractions nécessaires à une gestion transparente de la répartition
- **Hétérogénéité**. Cacher l'hétérogénéité des systèmes sous-jacents :
  - Réseau : Ethernet, ATM, Token ring, etc.
  - Architecture : x86, Sparc, etc.
  - Système d'exploitation : Unix, Windows, Solaris, etc.
  - Langage de programmation des applications : C, COBOL, Ada, Smalltalk, C++, Java, C#

Terminologie CORBA

- ***Objet CORBA***. Une entité qui peut être localisée par un ORB et sur laquelle des requêtes peuvent être invoquées.
- ***Requête***. Une opération qui peut être invoquée sur un objet CORBA.
- ***Client***. Une entité qui invoque une requête sur un objet CORBA, localisé ou non sur le même site que le client.
- ***Serveur***. Une application dans laquelle des objets CORBA existent.
- ***Servant***. Implantation / représentation concrète d'un objet CORBA dans un serveur. On dit qu'un servant *incarne* un objet CORBA.
- ***Référence d'objet***. Information qui permet d'identifier, de localiser et d'accéder à un objet CORBA (IOR: Interoperable Object Reference).

Plan

- **IDL CORBA**
- Etapes de développement d'une application CORBA
- Adaptation

IDL (Interface Definition Language)

- Client invoque une requête sur un objet CORBA réparti
  ↳ Connaître l'interface fournie par l'objet
- IDL :
  - Définition de l'interface d'objets CORBA
  - Indépendamment de tout langage de programmation
  - Spécification de l'IDL : http://www.omg.org

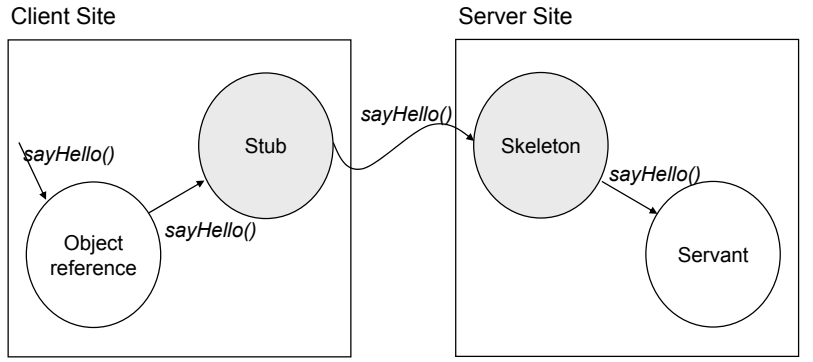
Exemple d'IDL pour *Hello*

```
module HelloApp
{
  interface Hello
  {
    string sayHello(in string
   name);
    oneway void shutdown();
  };
};
```

Compilation IDL

- Code du talon client
- Code du squelette serveur

Talon (*stub*) / Squelette (*skeleton*): Gestion de la répartition



Plan

- IDL CORBA
- **Etapes de développement d'une application CORBA**
- Adaptation

Etapes de développement d'une application CORBA

1. Déterminer les objets répartis (objets CORBA) de l'application et définir leurs interfaces en IDL
2. Compiler l'IDL vers un langage de programmation pour produire des talons clients / squelettes serveurs
3. Ecrire les classes des servants qui incarnent les objets CORBA définis
4. Ecrire et compiler le programme principal du serveur
5. Ecrire et compiler le programme client

Etapes de développement de l'application Hello (1 / 5)

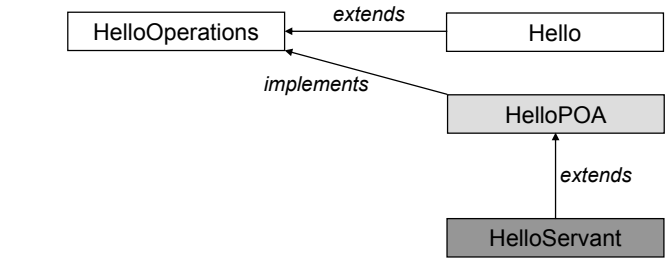
1. Déterminer les objets répartis : objets CORBA *Hello*
2. Définir leurs interfaces en IDL : fichier *Hello.idl*

```
module HelloApp
{
  interface Hello
  {
    string sayHello(in string name);
    oneway void shutdown();
  };
};
```


Etapes de développement de l'application Hello (2 / 5)

1. Compiler l'IDL vers un langage de programmation
   - Utilisation du compilateur IDL/Java fourni par le Java CORBA du J2SDK 1.4 : commande *idlj Hello.idl*
2. Classes produites par le compilateur IDL
   - **Hello** : interface Java correspondant à l'interface IDL *Hello*
   - **HelloOperations** : interface d'un objet distribué CORBA ayant pour interface IDL *Hello*
   - **HelloPOA** : classe abstraite définissant le squelette d'accès à un objet distribué CORBA d'interface *Hello*
   - **HelloHelper** : classe gérant la création de talons pour les objets distribués CORBA d'interface *Hello*
   - **_HelloStub** : classe définissant le talon associé à un objet distribué d'interface *Hello*

Classes produites par le compilateur IDL



Etapes de développement de l'application Hello (3 / 5)

- Ecrire la classe du servant qui incarne l'objet CORBA

```
public class HelloServant
   extends HelloPOA {

   private ORB orb;
   private String language;

   // constructor
   public HelloServant(String language,
                       ORB orb){
      this.language = language;
      this.orb      = orb;
   }

   public void shutdown() {
      if (orb != null) {
         orb.shutdown(false);
      }
   }
```

```
   public String sayHello(String name) {
      if (language.equals("fr")) {
         return "\nSalut " + name + " !!\n";
      } else if (language.equals("it")){
         return "\nCiao " + name + " !!\n";
      } else if (language.equals("es")){
         return "\nHola " + name + " !!\n";
      }else if(language.equals("en-us")){
         return "\nHi " + name + " !!\n";
      } else if (language.equals("de")){
         return "\nHallo " + name + " !!\n";
      } else {
         return "\nHello " + name + " !!\n";
      }
   }
}
```

Etapes de développement de l'application Hello (4 / 5)

- Ecrire et compiler le programme principal du serveur

```
public class HelloServer {
public static void main(String args[]) {
   // Create and initialize the ORB
   ORB orb = ORB.init(args, null);
   // Reference to rootpoa,activate
   // POAManager
   CORBA.Object poa_ref = orb.resolve_
         initial_references("RootPOA");
   POA rootpoa = POAHelper.narrow(poa_ref);
   rootpoa.the_POAManager().activate();
   // Create servant and register it in ORB
   HelloServant helloServant =
      new HelloServant("en-us", orb);
   // Get object reference from the servant
   CORBA.Object ref = rootpoa.servant_
         to_reference(helloServant);
   Hello href = HelloHelper.narrow(ref);
```

```
   String bindingname = args[1];
   // Get the root naming context
   CORBA.Object nsRef = orb.resolve_
         initial_references("NameService"
);
   // Use NamingContextExt (INS spec.)
   NamingContextExt ncRef =

NamingContextExtHelper.narrow(nsRef);
   // Bind the Object Reference in Naming
   NameComponent path[] = ncRef.
         to_name(bindingname);
   ncRef.rebind(path, href);
   // Wait for invocations from clients
   System.out.println("HelloServer ready
         and waiting ...");
   orb.run();
}
}
```


Etapes de développement de l'application Hello (5 / 5)

- Ecrire et compiler le programme principal du client

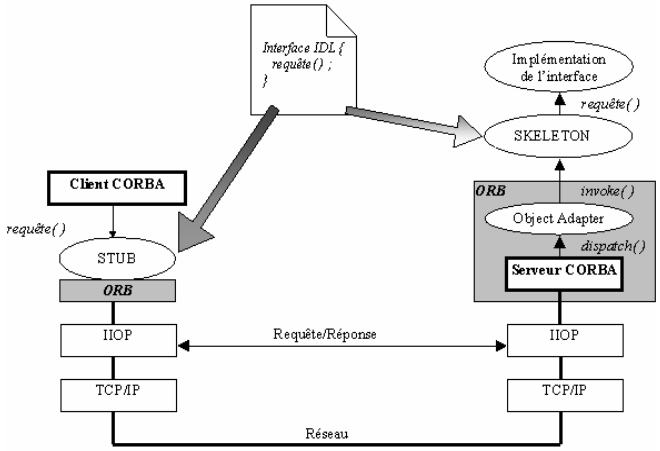
```
public class HelloClient {
public static void main(String args[]) {
   // Create and initialize the ORB
   ORB orb = ORB.init(args, null);
   // Get the root naming context
   CORBA.Object ncoRef = orb.resolve_
         initial_references("NameService");
   // Use NamingContextExt (INS spec.)
   NamingContextExt ncRef =
      NamingContextExtHelper.narrow(ncoRef);
```

```
   // Resolve the Object Reference in
   // Naming
   String bindingname = args[1];
   CORBA.Object objRef = ncRef.
         resolve_str(bindingname);
   Hello hello =
HelloHelper.narrow(objRef);
   // Request methods on the CORBA object
   System.out.println("Obtained a handle
on
         server object: " + hello);
   System.out.println(
         hello.sayHello("World"));
   hello.shutdown();
}
}
```


Plan

- IDL CORBA
- Etapes de développement d'une application CORBA
- **Adaptation**

POA (Portable Object Adapter)



⇨ Rôle du POA :

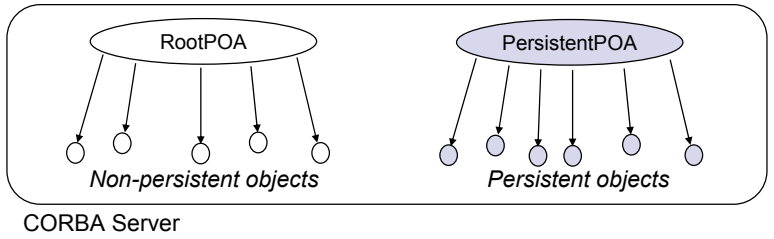
-Créer les références d'objets

-Activer les objets

-Transmettre les requêtes aux servants

Adaptation : Plusieurs POA

- Un POA par défaut
- Plusieurs POA peuvent exister dans un même serveur CORBA
- Un POA gère un ensemble d'objets CORBA partageant les mêmes caractéristiques



Pour finir …

Suite du cours : étude de 2 modèles à composants avec expérimentation

- OSGi
  - Standard industriel (consortium)
  - Domaine dédié : équipements professionnels et grand public
    - Exemple : services mobiles, domotique
  - Implémentation en logiciel libre sur ObjectWeb
- Fractal
  - Modèle issu de la recherche
  - Grande généralité, peu de restrictions
  - Base conceptuelle et formelle
  - Implémentation de référence en logiciel libre sur ObjectWeb

Références

- **Programming .NET Components.** J. Lowy, O'Reilly, 2003.
- **Enterprise Java Beans**. R. Monson-Haefel, B. Burke, S. Labourey, O'Reilly - 4ème édition, 2004.
- **Enterprise Application Integration with CORBA Component and Web-Based Solutions**. R. Zahavi, John Wiley & Sons, 1999.
- **Ce cours a été conçu à partir des supports :**
  - Sacha Krakowiak, http://sardes.inrialpes.fr/people/krakowia/