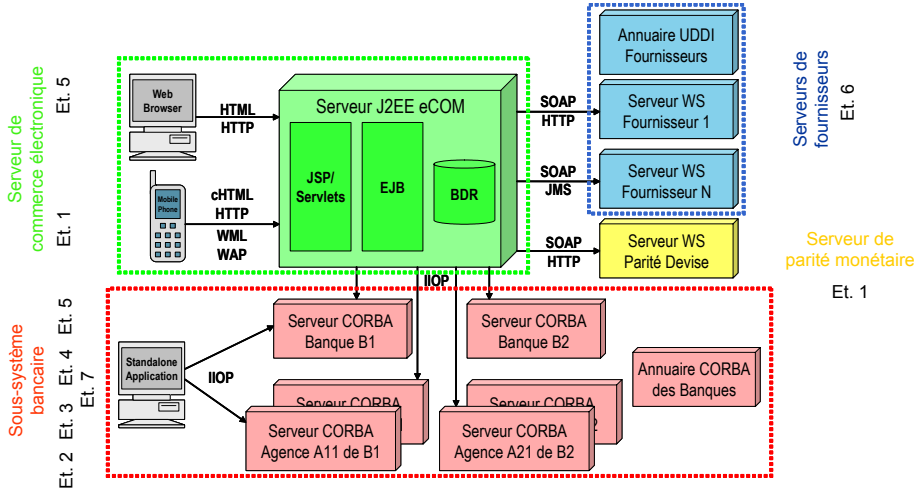
GICOM – M2GI SRR
RICOM – M2RICM

Etape 3 :
Sous-système bancaire persistant

Sara Bouchenak, Johann Bourcier, Didier Donsez

Pierre-Yves Gibello

Architecture globale de GICOM



Etapes de GICOM 1

- *Etape 1 : Sous-système bancaire*
- Etape 2 : Sous-système bancaire persistant
- Etape 3 : Sous-système bancaire persistant et fiable

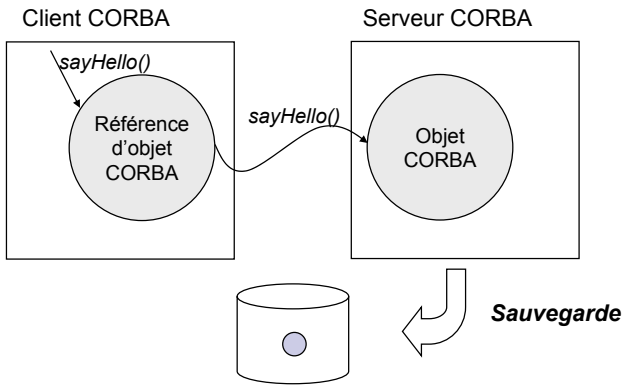
Plan

1. Introduction à la persistance
2. Persistance dans CORBA
3. Persistance dans GICOM

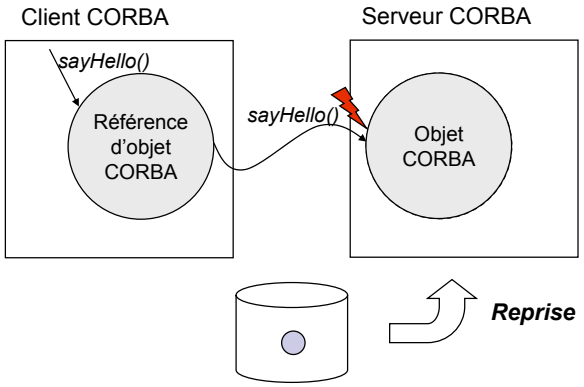
Qu'est-ce que la persistance ?

- Par défaut : Objets CORBA non persistants
  - Objet CORBA valide durant la vie de son serveur CORBA
  - Objet perdu après l'arrêt ou la panne du serveur
  - Référence d'objet CORBA distribué non valide après panne/arrêt du serveur
- Objets CORBA rendus persistants
  - Objet persistant : survit à la panne/arrêt de son serveur
  - Objet valide après le redémarrage du serveur
  - Référence d'objet CORBA valide après redémarrage du serveur
  - Facilite l'écriture des programmes clients

Principe de la persistance : Sauvegarde avant panne/arrêt



Principe de la persistance : Reprise après redémarrage



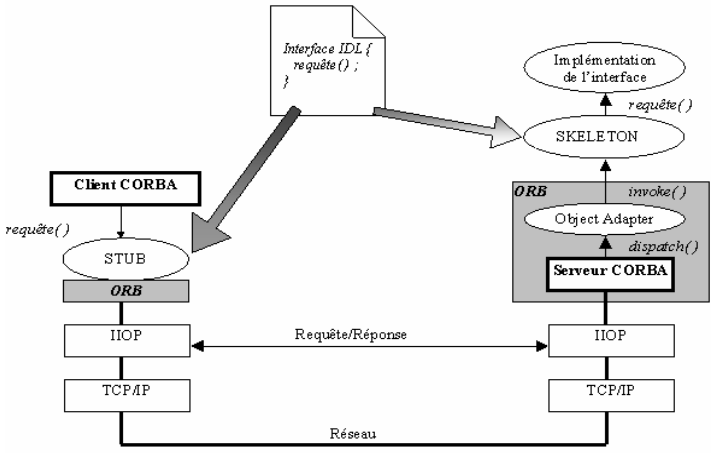
Principes de mise en œuvre de la persistance

- Etat stable
  - Tout objet persistant possède un état stable sauvegardé sur disque
- Opérations de sauvegarde
  - Opérations de sauvegarde des objets persistants effectuées régulièrement par l'application
- Sauvegarde atomique
  - Si un problème survient lors d'une sauvegarde, possibilité de retrouver un état sauvegardé précédemment
- Reprise
  - Après le redémarrage d'un serveur CORBA (suite à panne/arrêt), l'accès à un objet persistant provoque sa réinitialisation à partir de son dernier état stable
  - Reprise effectuée par le service de persistance du serveur CORBA

Plan

1. Introduction à la persistance
   - Qu'est-ce que la persistance ?
   - Sauvegarde/reprise
   - Principes de mise en œuvre de la persistance
2. *Persistance dans CORBA*
3. Persistance dans GICOM

POA (Portable Object Adapter)



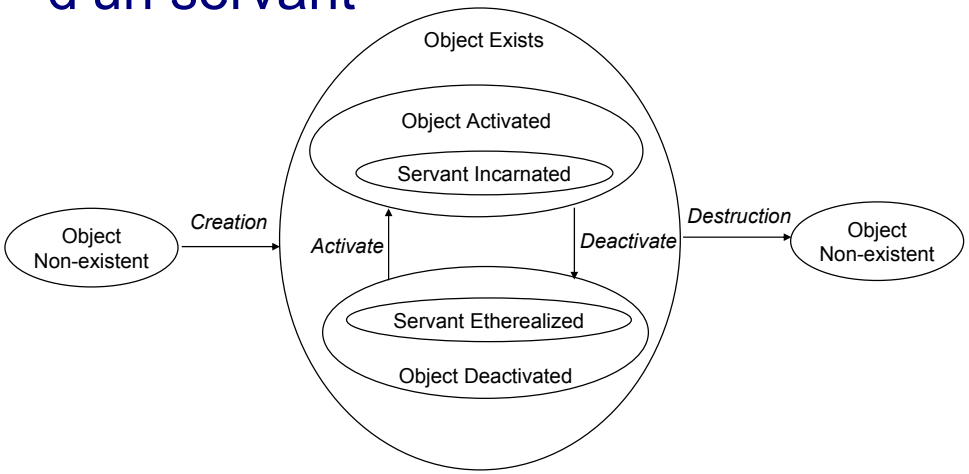
⇨ Rôle du POA :

-Créer les références d'objets

-Activer les objets

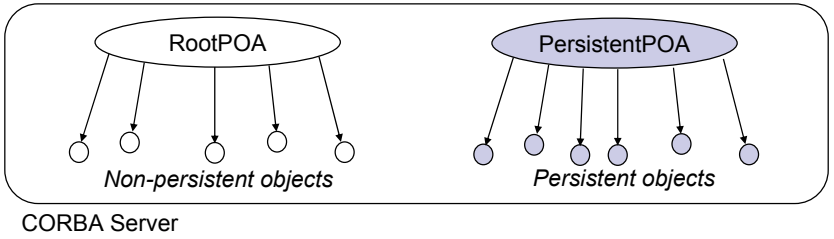
-Transmettre les requêtes aux servants

Cycles de vie d'un objet CORBA et d'un servant



Plusieurs POA

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



Spécialisation du POA

- Caractéristiques contrôlées via des règles (politiques) associées au POA
  - **LifespanPolicy** : Persistance/non persistance des objets
  - **IdAssignmentPolicy** : Identification des objets
  - **ServantRetentionPolicy** : Association entre objectId et servants
  - **RequestProcessingPolicy** : Transfert des requêtes aux servants
  - **ThreadPolicy** : Mono- ou multi-threading du POA
  - **ImplicitActivationPolicy** : Création/activation implicites ou explicites des objets
  - **IdUniquenessPolicy** : Un seul servant par objet ou même servant incarnant plusieurs objets

LifespanPolicy : Persistance / non persistance des objets

- TRANSIENT
  - Après désactivation du POA, les objets TRANSIENT ne peuvent plus être réactivés
  - Une référence d'objet CORBA distribué est alors invalide
  - Les objets TRANSIENT représentent des informations non persistantes
- PERSISTENT
  - Une référence d'objet CORBA distribué reste valide
- Valeur dans le RootPOA : TRANSIENT
- Valeur par défaut dans un nouveau POA : TRANSIENT

IdAssignmentPolicy : Identification des objets

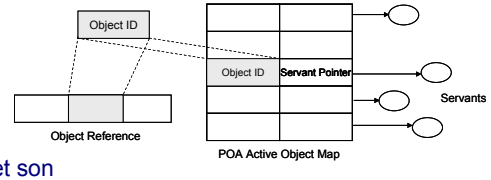
- SYSTEM_ID
  - Le POA crée les objectId et les affecte automatiquement aux objets CORBA distribués
- USER_ID
  - L'application gère elle-même les objectId

IP address	Port number	POA name	Object ID

IOR Object Reference

- Valeur dans le RootPOA : SYSTEM_ID
- Valeur par défaut dans un nouveau POA : SYSTEM_ID

ServantRetentionPolicy : Association entre objectId / servants



- RETAIN
  - L'association entre un objectId et son servant est maintenue dans la table AOM du POA
- NO_RETAIN
  - L'association entre un objectId et son servant est gérée par l'application (utilisation d'un *ServantManager* ou d'un *servant par défaut*)
- Valeur dans le RootPOA : RETAIN
- Valeur par défaut dans un nouveau POA : RETAIN

RequestProcessingPolicy: Transfert des requêtes aux servants

- USE_ACTIVE_OBJECT_MAP_ONLY
  - Exception levée, si objectId ∉ AOM
- USE_DEFAULT_SERVANT
  - Requête transmise à un servant par défaut
- USE_SERVANT_MANAGER
  - Utilisation d'un *ServantManager* qui crée/trouve le servant approprié
- Valeur dans le RootPOA : USE_ACTIVE_OBJECT_MAP_ONLY
- Valeur par défaut dans un nouveau POA : USE_ACTIVE_ OBJECT_ MAP_ONLY

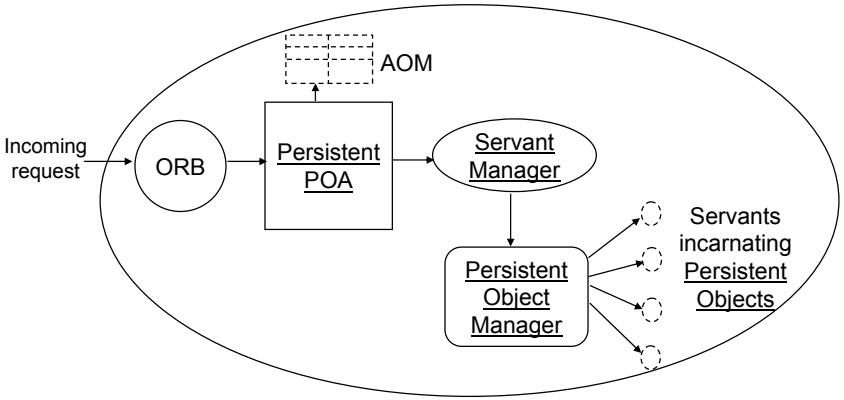
ServentRetentionPolicy vs. RequestProcessingPolicy

		ServentRetentionPolicy	
		RETAIN	***NO_RETAIN***
RequestProcessingPolicy	***USE_SERVANT_MANAGER***	ServantActivator 1. Requête : POA consulte l'AOM 2. Si servant ∉ AOM, POA appelle l'opération *incarnate* du *ServantActivator* 3. L'opération *incarnate* crée et retourne une instance de servant ou lève une exception 4. L'opération *etherealize* sert à se défaire d'un servant. 5. Le POA appelle *etherealize* lors de la désactivation de l'objet ou du POA	ServantLocator 1. A chaque requête : POA appelle *ServantLocator* 2. Le POA appelle l'opération *preinvoke* du *ServantLocator* pour obtenir un servant 3. A la fin de l'exécution de la requête, le POA appelle l'opération *postinvoke* du *ServantLocator* pour se défaire du servant 4. Un servant retourné par un *preinvoke* n'est utilisé que pour une seule requête

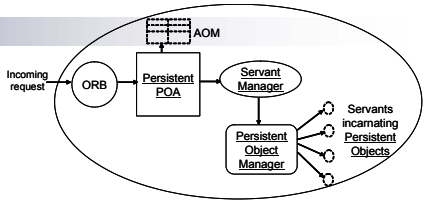
Plan

1. Introduction à la persistance
2. Persistance dans CORBA
   - Rôle du POA
   - Règles/politiques du POA
   - ServantManager
3. *Persistance dans GICOM*

Mise en œuvre de la persistance dans GICOM

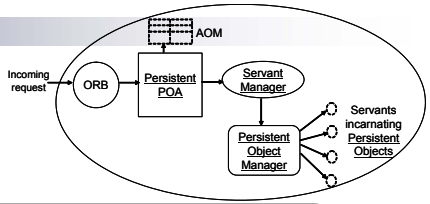


Mise en œuvre du Persistent POA



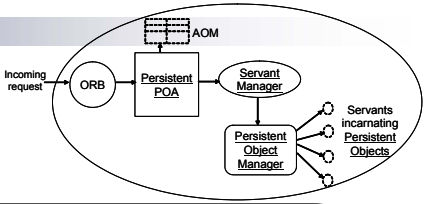

```
CORBA.Policy[] policies = new CORBA.Policy[4];
policies[0] = rootPOA.create_lifespan_policy(
   PortableServer.LifeSpanPolicyValue.PERSISTENT);
policies[1] = rootPOA.create_id_issignment_policy(
   PortableServer.IdAssignmentPolicyValue.USER_ID);
policies[2] = rootPOA.create_servant_retention_policy(
   PortableServer.ServantRetentionPolicyValue.RETAIN);
policies[3] = rootPOA.create_request_processing_policy(
   PortableServer.RequestProcessingPolicyValue.USE_SERVANT_MANAGER);
persistentPOA = rootPOA.create_POA("PersistentPOA",
   rootPOA.the_POAManager(), policies);
```


Mise en œuvre Servant Manager (nommé PersistentActivator)



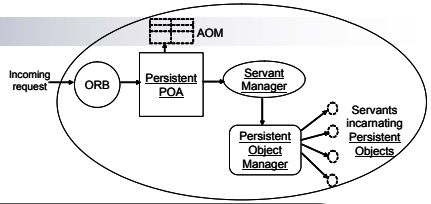

```
public class PersistentActivator extends PortableServer.ServantActivatorPOA {
   public PortableServer.Servant incarnate(byte[] id,
         PortableServer.POA adapter)
         throws PortableServer.ForwardRequest {
         ...
   }
   public void etherealize(byte[] id, PortableServer.POA adapter,
         boolean clean_up_in_progress, boolean remaining_activations)
   {
         ...
   }
}
```


Association du Servant Manager au Persistent POA



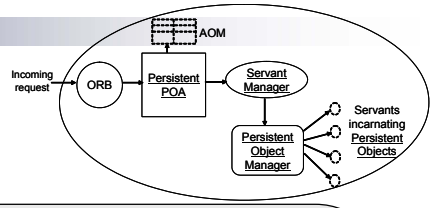

```
// Create a new servant manager object
PortableServer.ServantActivator myActivator =
   new PersistentActivator()._this(orb);

// Set the servant manager for the persistent POA
persistentPOA.set_servant_manager(myActivator);
```


Mise en œuvre du Persistent Object Manager




```
package Persistence;
public interface PersistentObjectManager {
   // Store a persistent object using an atomic operation
   public void store(java.lang.Object obj, byte[] id);
   // Load an object from its persistent store and initialize it
   public PersistentObject load(byte[] id);
   // Assign an objectId to the persistent object
   public byte[] register(PersistentObj obj);
}
public class PersistentObjectManagerImpl
   implements PersistentObjectManager;
```

Mise en œuvre de PersistentObject




```
package Persistence;
public interface PersistentObject extends java.io.Serializable {
    // Call an atomic storage operation on the current object
    public void save();
    // Create a servant that incarnates the current persistent object
    public PortableServer.Servant createServant();
    // Initialize the persistent object (transient attributes, etc.)
    // at load time
    public void init();
    // Activate this object and return its CORBA object reference
    public CORBA.Object getRef();
    // Deactivate this object and remove it from the persistent store
    public void destroy();
}
public abstract class PersistentObjectImpl implements PersistentObject;
```


Modèle de programmation

- Les implantations des objets persistants dans GICOM doivent :
  - Étendre la classe *PersistentObjectImpl*
  - Surcharger si nécessaire la méthode *init*
- Donc, modèle de programmation par délégation
  - Objets CORBA persistants héritent de *PersistentObjectImpl*
  - Ne peuvent hériter de la classe du squelette

Gestion des références persistantes distantes

- Référence distante : adresse d'un talon, locale, non sérialisable, donc non persistante
- Définir une référence distante comme une donnée *transient*, non sauvegardée lors de la sérialisation
- Transformer la référence distante en chaîne de caractères, la sauvegarder avec l'objet persistant
  - String *object_to_string*(CORBA.Object);
  - CORBA.Object *string_to_object*(String);

Gestion des références persistantes locales

- Problème
  - Référence locale : référence Java
  - Sauvegarde de O sur disque : sauvegarde d'une *copie* de O' sur disque
  - Référence Java non persistante : restauration de O' du disque, adresse de O' contenue dans O non valide
- Solution
  - Transformer les références persistantes locales en références persistantes distantes

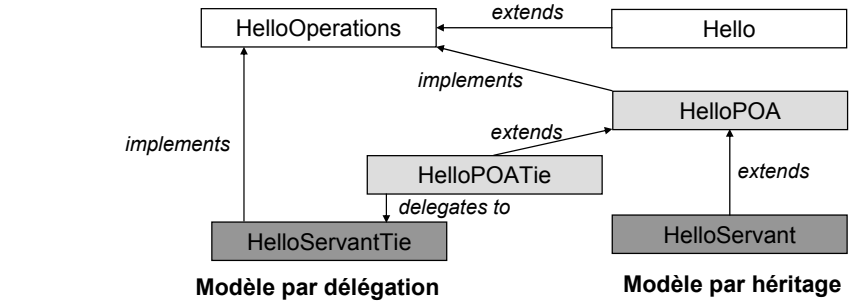
Plan

1. Introduction à la persistance
2. Persistance dans CORBA
3. Persistance dans GICOM
   - Persistent POA
   - Servant Manager / Persistent Activator
   - PersistentObjectManager / PersistentObject
   - Modèle de programmation par délégation
   - Références CORBA persistantes

Implications sur l'application CORBA

1. ✓ Déterminer les objets répartis (objets CORBA) de l'application et définir leurs interfaces en IDL
2. ✓ Compiler l'IDL, selon le modèle par délégation, pour produire des talons clients / squelettes serveurs
3. Ecrire les classes des servants qui incarnent les objets CORBA définis
   - Classes des objets persistants étendent *PersistentObjectImpl*
   - Classes des objets persistants surchargent *init*
   - Transformer les références locales vers des objets persistants en références distantes
4. ✓ Ecrire et compiler le programme principal du serveur
   - Serveur générique étendu pour ajout de POA
5. ✓ Ecrire et compiler le programme client

Modèle par héritage vs. modèle par délégation



Questions ?

- Etape 2 : Sous- système bancaire
  - Mise en œuvre des classes des servants
  - Extension/configuration du serveur générique
  - User-Agents
- Etape 3 : Sous- système bancaire persistant
  - Mise en œuvre de la classe *PersistentActivator*
  - Mise en œuvre des classes *PersistentObjectImpl* et *PersistentObjectManagerImpl*
  - Mise en œuvre des classes servants selon le modèle par délégation, héritage de *PersistentObjectImpl*, surcharge de la méthode *init*, transformation des références locales vers des objets persistants