I Présentation générale

L'infrastructure développée dans ce projet est un exemple d'ORB facilement supervisable par JMX.

L'ORB permet la définition/déploiement d'objets servants et leur invocation transparente par des clients distants. Il s'apparente à java RMI mais s'en différencie sur plusieurs points :

- On ne déclare pas les interfaces invocables à distance par extension de java.rmi.Remote mais par une annotation archi.orb.infra@JAPREmote.
- Il n'est pas nécessaire d'imposer la présence d'une exception java.rmi.RemoteException dans la signature des méthodes de l'interface. Il n'est pas non plus nécessaire de déclarer une exception dans le constructeur de la classe de l'objet servant. Il existe une exception liée à l'infrastructure : archi.orb.infra.JAPRemoteException mais elle étend l'exception RuntimeException : elle peut ainsi être récupérée par le code client sans être déclarée dans les méthodes des interfaces.

- L'interface du serveur de nom est identique à celle du rmiregistry : bind, unbind, list, lookup.
- Il n'y a aucune contrainte de localisation de serveur de noms : il n'est pas nécessaire par exemple, de localiser le serveur de noms sur la machine des objets servants ni d'imposer que les interfaces ou certaines classes soient dans le classpath du serveur de noms. Il est possible de ne disposer que d'un seul serveur de noms pour des objets servants répartis sur plusieurs machines.
- Il n'y a pas de contrainte sur la forme des noms associés aux objets servants : les noms sont traités comme de simples chaines de caractères sans structure.
- Il est possible de regrouper sous un même nom plusieurs interfaces : la méthode bind a 2 arguments, respectivement la référence de l'objet servant et un nom. La méthode lookup construit un stub qui donne l'accès aux méthodes de l'ensemble des interfaces invocables à distance de l'objet servant.

- Il est uniquement nécessaire de disposer de la définition des interfaces dans le classpath coté client : les classes de stubs sont générées à la volée au moment du bind.

La supervision non intrusive de la plateforme est réalisée par composition événementielle : l'infrastructure permet de glisser de façon transparente entre les objets des intercepteurs capables de :

- enregistrer des objets listeners, auxquels seront envoyées des notifications correspondant aux invocations et retour de méthodes. Chaque intercepteur peut être paramétré individuellement de façon à ne notifier que certaines méthodes à certains instants (à l'invocation, à la réponse).

- La mise en oeuvre repose sur JMX : il est possible d'associer à chaque intercepteur un MBean qui peut donc être enregistré dans un MBeanServer et avec lequel il est possible d'interagir via la jconsole par exemple. Par ailleurs chaque intercepteur implémente l'interface NotificationBroadCaster et émet des notifications conformes au standardJMX.

II L'ORB

II.i La programmation des clients et des serveurs

La création puis l'enregistrement d'un objet servant dans le service de noms est réalisé de la façon suivante :

```
//obtention du service de nommage :
InetAddress svn = …;
JAPNameService sce = new JAPNameService(svn, 7777);

//création des objets servants puis enregistrement via la méthode bind. Ici chaque nom est associé //à une seule interface.
for (int i = 0; i < nbS; i++) {
        IGestBaguettes s = new GestBagActif(nbPhil,"servant numero " + i);
        sce.bind(s, "synchro_baguettes_" + i);
}
```

Remarque : l'ORB dispose d'un skeleton générique utilisable pour un nombre quelconque d'interfaces. La version actuelle par contre impose un skeleton par objet servant : cette contrainte est invisible du programmeur car la création du skeleton est encapsulée dans la méthode bind du service de noms.

La programmation d'un client dans l'ORB est réalisée de la façon suivante :

```
//obtention du service de noms
InetAddress svn = …;
JAPNameService sce = new JAPNameService(svn, 7777);
//création du stub via la méthode lookup et un cast vers une des interfaces de l'objet servant
IGestBaguettes s = (IGestBaguettes) sce.lookup("synchro_baguettes_0");
```

II.ii Le service de nommage

Le service de nommage implémente l'interface archi.orb.naming.IJAPService. Cette dernière est très proche de l'interface du rmiregistry. Elle ne s'en distingue que par la méthode listInterfacesOfService qui permet de connaître l'ensemble des interfaces associées à un nom i.e. associées à un objet servant.

```
public interface IJAPNameService {
        public Object lookup(String sName);
        public void bind(Object obj, String sName);
        public String[] listServices();
        public String[] listInterfacesOfService(String srv);
}
```

La classe archi.orb.naming.JAPService est une implémentation simple (non persistante) de l'interface IJAPService. Ce serveur n'impose aucune contrainte de classpath et peut se trouver sur un site différent de celui des objets servants. Il peut centraliser le nommage de l'ensemble des objets servants.

La classe archi.orb.naming.LaunchJAPNameServer lanceun serveur de nommage. Le port par défaut est le 7777, mais il est possible de le spécifier explicitement comme un argument de la commande de lancement.

II.iii Divers

Une interface d'objet servant doit pour pouvoir être invoquée à distance, être annotée par l'annotation archi.orb.infra.@JAPRemote

L'exception archi.orb.infra.JAPRemoteException est une exception qui indique les problèmes liées à l'ORB. Elle étend RuntimeException et encapsule toutes les exceptions générées par les couches d'IO, de communication réseau, de traitement réflexif, etc....

La classe archi.orb.naming.JAPName associe à un ensemble d'interfaces l'adresse de la socket d'un skeleton :

```
public class JAPName implements Serializable {
   public JAPName(String[] iN, InetAddress  sA, int p) { … }
   public String[] getInterfaceNames() { ….. }
   public InetAddress getInetAddress() { ….. }
   public int getPort() { ….  }
}
```

La classe archi.orb.util.StubGenerator est une classe utilisée par l'ORB pour générer les stubs fournis aux client lors d'une invocation d'un lookup sur le service de nommage. La classe archi.orb.util.ORBHelper est une classe regroupant des méthodes statiques facilitant l'analyse des interfaces.

La classe archi.common.GenericRequest comporte l'ensemble des informations décrivant une requête (nom de la méthode, signature, type de retour, valeurs des arguments, ….). La classe archi.common.GenericResponse étend la classe précédente en indiquant la valeur de retour ou l'exception générée.

La classe archi.orb.infra.GenericAnnotSkel est un skeleton invoquant de façon réflexive un objet servant. Les informations nécessaires à l'appel sont récupérées à partir d'une socket via un objet sérialisé archi.common.GenericRequest (Il existe une forme plus simple correspondant à la classe archi.infra.GenericSkel.

La classe archi.orb.infra.JAPGenericStub est la classe parente de tous les stubs générés par l'ORB : elle

encapsule la communication et la synchronisation entre les threads des clients et le thread de réception des réponses envoyées par le skeleton.

III la supervision par JMX

III.i La programmation

La programmation de la supervision dans l'infrastructure repose sur l'utilisation d'une factory capable de générer à la volée des classes d'intercepteur implémentant un ensemble d'interfaces. Un exemple simple est le suivant :

```
//création de la factory générant les intercepteurs
JAPInterceptorFactory fact = new JAPInterceptorFactory();

//création de l'intercepteur qui implémente l'interface IGestBaquettes et qui pointe sur l'objet s
JAPGenericInterceptor inter = (JAPGenericInterceptor)
        fact.generateInterceptor(new String[] { "philosophe.api.IGestBaguettes"} , s);
//contrôle des notifications émises : notification à l'invocation et au retour des methodes acqBaguettes
inter.configureNotificationMask("acqBaguettes", true, true);

//création d'un listener ….. et enregistrement via le MBeanServer
AcqBaguettesStatistics stat1 = new AcqBaguettesStatistics();
mbs.addNotificationListener(name, stat1, null, null);

//création d'un MBean capable de contrôler les notifications émises par
//enregistrement des listeners auprès de l'intercepteur
MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();
ObjectName name = new ObjectName("jap.philosophes:type=time-statistics");
mbs.registerMBean(mbean, name);

JAPGenericInterceptorCtl mbean = new JAPGenericInterceptorCtl(inter);

//démarrage des clients
for (int i = 0; i < nbPhil; i++) {
new PhiloActif(i, serveurs, (IGestBaguettes) inter, i, (i + 1) % nbPhil).start();
```

III.ii Divers

La classe archi.jmx.infra.JAPGenericInterceptor encapsule l'ensemble des mécanismes communs à tous les intercepteurs de l'infrastructure. Elle étend la classe NotificationBroadcasterSupport et implémente via la délégation l'interface JAPGenericInterceptorCtlMBean. Il faut noter que les classes d'intercepteur générées à la volée par la factory archi.jmx.util.JAPInterceptorFactory étendent cette classe et ne sont pas des Mbeans. Les méthodes desactivate et activate mettent en oeuvre un mécanisme générique de suspend/resume.

La classe archi.jmx.infra.JAPGenericInterceptorCtl implémente l'interface JAPGenericInterceptorCtlMBean. Les instances de cette classe sont des MBean qui sont associées à leur création à un intercepteur (via l'argument de leur ctor). Ils permettent de contrôler le comportement des intercepteurs par l'intermédiaire des services JMX (le MBeanServer, la jconsole, …).

```
public interface JAPGenericInterceptorCtlMBean {
  //applique le masque à une surcharge précise de la méthode.
  public void configureNotificationMask(String methNm, String sign, boolean pre, boolean post);
  //applique le masque à toutes les formes surchargées de la méthode.
  public void configureNotificationMask(String methNm, boolean pre, boolean post);
  //applique le masque à toutes les méthodes de l'intercepteur.
  public void configureNotificationMask(boolean pre, boolean post);

  public void setNotificationMessage(String id);
  public String getNotificationMessage();

  public void activate();
  public void desactivate();
```

}

la classe archi.jmx.util.JAPInterceptorFactory est la classe qui génère à la volée les intercepteurs JMX. La classe JMXHelper regroupe quelques méthodes statiques d'aide utilisées essentiellement par la factory

Les notifications émises par les intercepteurs sont des notifications JMX standard. Elles décrivent les invocations et retours de méthodes par des DataObject de la classe JAPGenericRequest et JAPGenericResponse.