L’application annuaire

L’application annuaire

La réalisation d’un gestionnaire de répertoires d’adresses en JAVA RMI

L’application annuaire

La réalisation d’un gestionnaire de répertoires d’adresses en JAVA RMI
la création des serveurs et des clients

L’application annuaire

La réalisation d’un gestionnaire de répertoires d’adresses en JAVA RMI
la création des serveurs et des clients

L’application annuaire

La réalisation d’un gestionnaire de répertoires d’adresses en JAVA RMI
la création des serveurs et des clients

La mise en œuvre avancée

L’application annuaire

La réalisation d’un gestionnaire de répertoires d’adresses en JAVA RMI
la création des serveurs et des clients

La mise en œuvre avancée
l’utilisation du service Nommage

L’application annuaire

La réalisation d’un gestionnaire de répertoires d’adresses en JAVA RMI
la création des serveurs et des clients

La mise en œuvre avancée
l’utilisation du service Nommage
la création à distance d’objets RMI

L’application annuaire

La réalisation d’un gestionnaire de répertoires d’adresses en JAVA RMI
la création des serveurs et des clients

La mise en œuvre avancée
l’utilisation du service Nommage
la création à distance d’objets RMI
la notification des clients

L’application annuaire

La réalisation d’un gestionnaire de répertoires d’adresses en JAVA RMI
la création des serveurs et des clients

La mise en œuvre avancée
l’utilisation du service Nommage
la création à distance d’objets RMI
la notification des clients

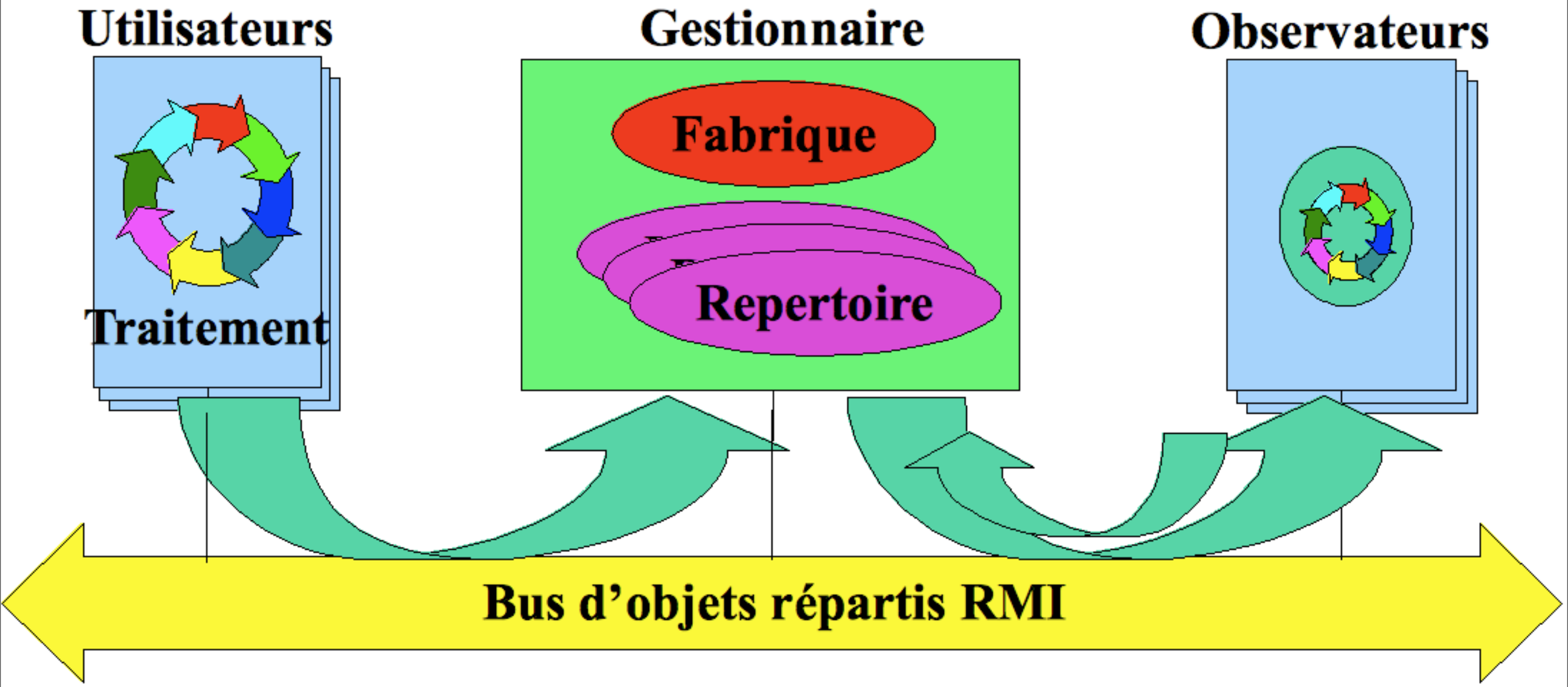
Les types d'objets de l'application annuaire

- Répertoire d'adresses
  - stockant une liste de personnes
  - fournissant des opérations pour ajouter, supprimer, modifier, obtenir et lister les personnes
  - partageable par plusieurs utilisateurs

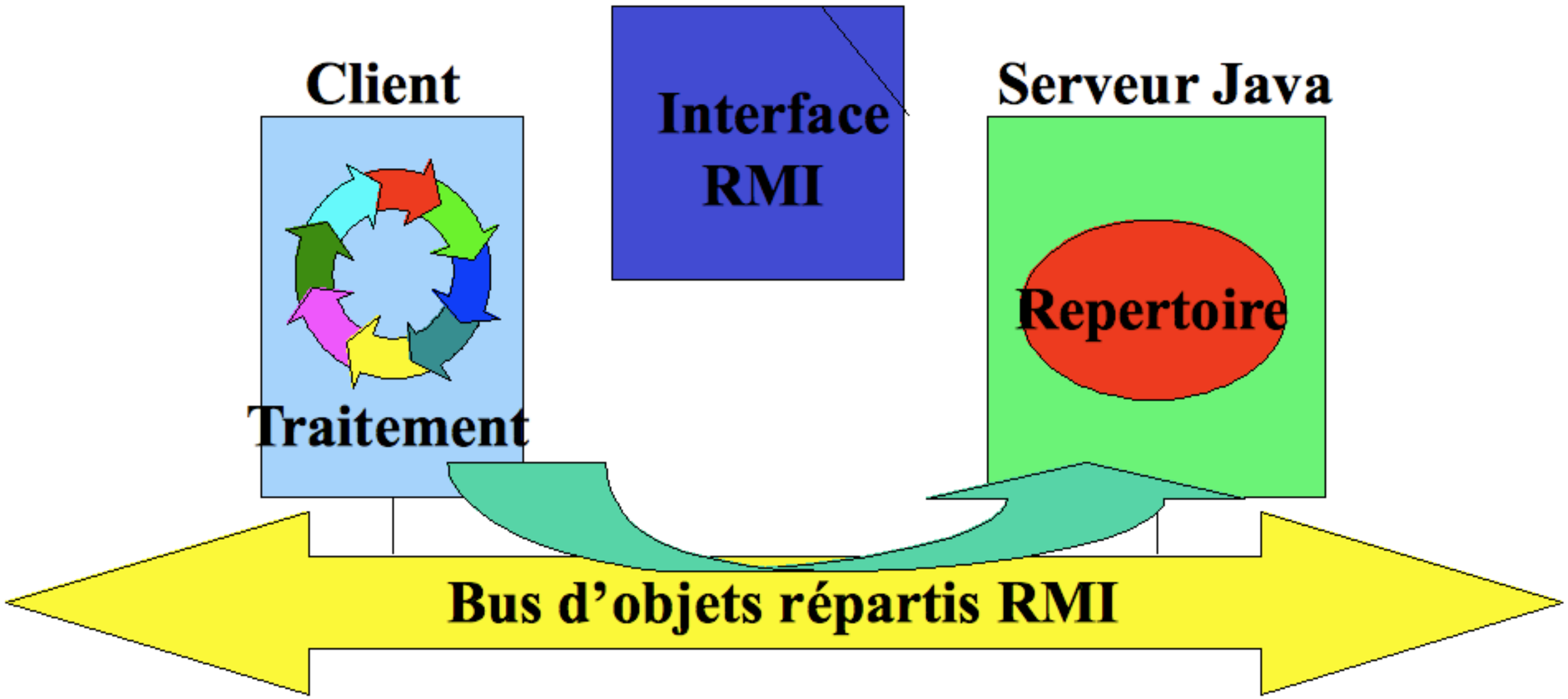
- Fabrique de répertoires
  - permettant de créer et détruire des répertoires

- Observateur
  - observant les modifications des répertoires

L’application annuaire



La première étape : une application client / serveur



Les données échangées

```
public class Personne implements java.io.Serializable
{
  protected String nom; protected String email;
  protected String url; protected String info;

  public Personne (String nom, String email, String url, String info)
  {
    this.nom = nom; this.email = email; this.url = url; this.info = info;
  }


  public void modifier (Personne p)
  {
    this.nom=p.nom; this.email=p.email; this.url=p.url; this.info=p.info;
  }

  public String getNom (){ return this.nom;} ....

  public String toString (){
   return "[nom="+nom+",email="+email+",url="+url+",info="+info+"]";
  }
}
```

L’interface Répertoire

```
public interface Repertoire extends java.rmi.Remote
{

  public boolean ajouterPersonne (Personne personne)
                  throws java.rmi.RemoteException;

  public boolean modifierPersonne (Personne personne)
                  throws java.rmi.RemoteException;

  public boolean retirerPersonne (String nom)
                  throws java.rmi.RemoteException;

  public Personne chercherPersonne (String nom)
                  throws java.rmi.RemoteException;

  public String[] listerPersonnes ()
                  throws java.rmi.RemoteException;

}
```

Implantation de l'interface Répertoire

```
public class RepertoireImpl extends javax.rmi.PortableRemoteObject
                            implements Repertoire, java.io.Serializable
//Serializable pour pouvoir utiliser le rmiregistry avec iiop
//il vaudrait mieux utiliser jndi (cours de l'année prochaine)
{
  protected java.util.Hashtable<String,Personne> personnes;

  public RepertoireImpl ()
                      throws java.rmi.RemoteException
  {
    super();
    this.personnes = new java.util.Hashtable<String,Personne>();
  }

  public boolean ajouterPersonne(Personne pers)
                    throws java.rmi.RemoteException
  {
    if (personnes.containsKey (pers.getNom()))return false;
    this.personnes.put(pers.getNom(),pers);
    return true;
  }
```

Implantation de l'interface Répertoire

```
public boolean modifierPersonne (Personne personne)
                    throws java.rmi.RemoteException
{
  Personne p = personnes.get(personne.getNom());
  if ( p == null )return false;
  p.modifier(personne);
  return true;
}
```

```
public boolean retirerPersonne (String nom)
                    throws java.rmi.RemoteException
{
  return this.personnes.remove (nom) != null;
}

public Personne chercherPersonne (String nom)
                    throws java.rmi.RemoteException
{
  return this.personnes.get(nom);
}
```

Implantation de l'interface Répertoire

```
  public String[] listerPersonnes ()
                    throws java.rmi.RemoteException
  {
    return personnes.keySet().toArray ( new String[0] );
  }
}
```

Un serveur RMI

```
public class Serveur
{
 public static void main(String args[]) throws Exception
 {
   // Créer l'objet accessible par Java RMI.
   RepertoireImpl obj = new RepertoireImpl();

   // Enregistrer cet objet dans l'annuaire RMI.
   java.rmi.Naming.rebind("Repertoire", obj);

   // Ici le programme ne se termine pas !
   // Car attente des invocations distantes sur l'OD.
  }
}
```

Un client RMI

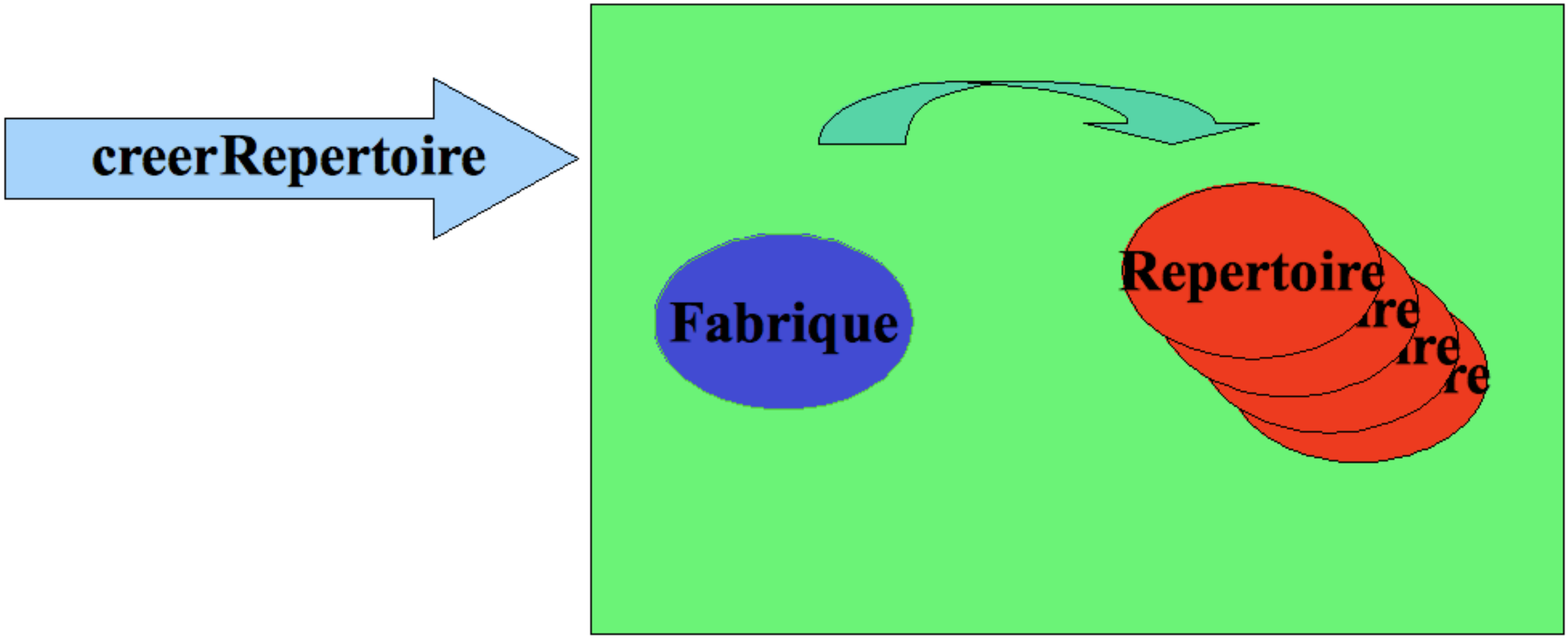
```
public class ClientSimple {

 public static void main(String args[]) throws Exception
 {
   // Obtenir la souche sur l'OD via l'annuaire RMI.
   Repertoire rep = (Repertoire)
           java.rmi.Naming.lookup("//localhost/Repertoire");

   // Invoquer des méthodes comme si l'objet était local.
    traitement (rep);
    System.exit(0);
 }

static void traitement(Repertoire repertoire) throws Exception
 {
 Personne pers=new Personne("Roos","roos@lifl.fr","www.lifl.fr/~roos","");
  . . .
  if(!repertoire.ajouterPersonne(pers))
    System.out.println("ATTENTION : Roos a déjà été ajouté");
 }
}
```

La fabrique de répertoire



La fabrique de répertoire

```
public interface Fabrique extends java.rmi.Remote
{

  public Repertoire creerRepertoire () throws java.rmi.RemoteException;
}


import java.rmi.RemoteException;

public class FabriqueImpl extends javax.rmi.PortableRemoteObject
                          implements Fabrique, java.io.Serializable
{

public FabriqueImpl() throws RemoteException { super(); }

public Repertoire creerRepertoire () throws RemoteException
  {
    return new RepertoireImpl();
  }

}
```

Le serveur de la fabrique

```
public class ServeurFabrique
{
 public static void main(String args[]) throws Exception
 {
   // Créer l'objet accessible par Java RMI.
   FabriqueImpl obj = new FabriqueImpl();

   // Enregistrer cet objet dans l'annuaire RMI.
   java.rmi.Naming.rebind("Fabrique", obj);

   // Ici le programme ne se termine pas !
   // Car attente des invocations distantes sur l'OD.
  }
}
```

Une application d’administration de la fabrique

- Création d’un nouveau répertoire et mise à disposition par le service Nommage

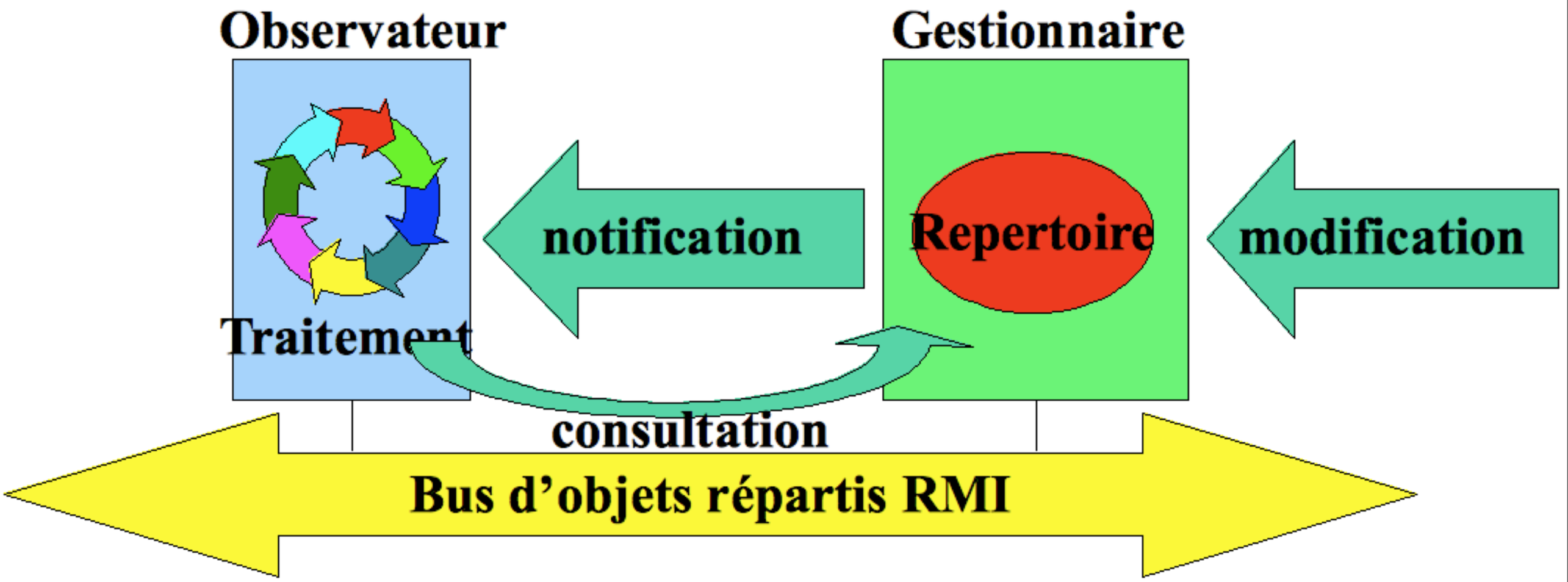
```
public class ClientFabrique {

 public static void main(String args[]) throws Exception
 {
   // Obtenir la souche sur l’OD via l'annuaire RMI.
   Fabrique fab = (Fabrique)
           java.rmi.Naming.lookup("//localhost/Fabrique");

   Repertoire rep = fab.creerRepertoire();

   java.rmi.Naming.rebind("NouveauRepertoire", rep);
 }
```

La notification des clients



Un canevas de conception pour la notification des clients

- Design Pattern « Observé - Observateur »

- Côté application notifiée (Observateur)
  - 1 interface à implanter : ObservateurRepertoire
  - 1 opération par type d’événement

- Côté serveur notifiant (Observé)
  - 1 interface pour enregistrer les observateurs
  - héritage d’interfaces

L’interface Observateur

```
public interface ObservateurRepertoire extends java.rmi.Remote
{

   public void annoncerCreationPersonne (String nom)
                  throws java.rmi.RemoteException;

   public void annoncerDestructionPersonne (String nom )
                  throws java.rmi.RemoteException;

   public void annoncerModificationPersonne (String nom)
                  throws java.rmi.RemoteException;
}
```

L’interface Répertoire Observé

```
public interface RepertoireObserve extends Repertoire
{

  public void ajouterObservateur(ObservateurRepertoire obs)
                    throws java.rmi.RemoteException;

  public void retirerObservateur(ObservateurRepertoire obs)
                    throws java.rmi.RemoteException;
}
```

L’implantation du répertoire observé

```
public class RepertoireObserveImpl extends RepertoireImpl
                                   implements RepertoireObserve
{
  protected java.util.ArrayList<ObservateurRepertoire> observateurs;

  public RepertoireObserveImpl ()
         throws java.rmi.RemoteException
  {
    super();
    this.observateurs = new java.util.ArrayList<ObservateurRepertoire>();
  }

  public void ajouterObservateur(ObservateurRepertoire obs)
                  throws java.rmi.RemoteException
  {
    observateurs.add(obs);
  }

  public void retirerObservateur(ObservateurRepertoire obs)
                  throws java.rmi.RemoteException
  { observateurs.add(obs); }
```

L’implantation du répertoire observé

```
public boolean ajouterPersonne(Personne pers)
                    throws java.rmi.RemoteException
{
 boolean result = super.ajouterPersonne (pers);

  for(ObservateurRepertoire o : observateurs)
    try {
      o.annoncerCreationPersonne(pers.getNom());
    } catch (java.rmi.RemoteException re) {
      retirerObservateur(o);
    }
}
```

de même pour retirerPersonne et modifierPersonne

Une application cliente notifiée

- Une application cliente peut implanter des objets
  - souvent objets client = objets « callback »
  - RMI n’est pas strictement client/serveur

- Le scénario de l’observation
  - obtenir l’objet observable
  - créer l’objet observateur
  - abonner l’observateur auprès de l'observé
  - … traiter les notifications ...
  - mettre fin à l’abonnement