Modèle RMI

Application répartie en Java

Définition préliminaire

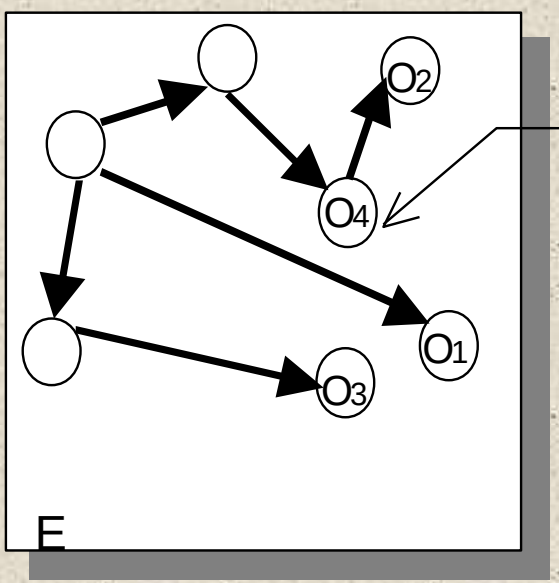
Ce modèle à pour but de faciliter la mise en œuvre d'applications Java réparties, c-a-d composées de plusieurs machines d'exécution supportant les objets constituant celles-ci.

- Ce modèle permet de faire fonctionner une application Java, entre-autres l'envoi de message en s'abstraire des problème de communication entre objets se situant sur des machines d'exécution différentes.

- Cette présentation repart du modèle d'exécution localisée et montre comment l'étendre au modèle d'exécution répartie.

Modèle localisé

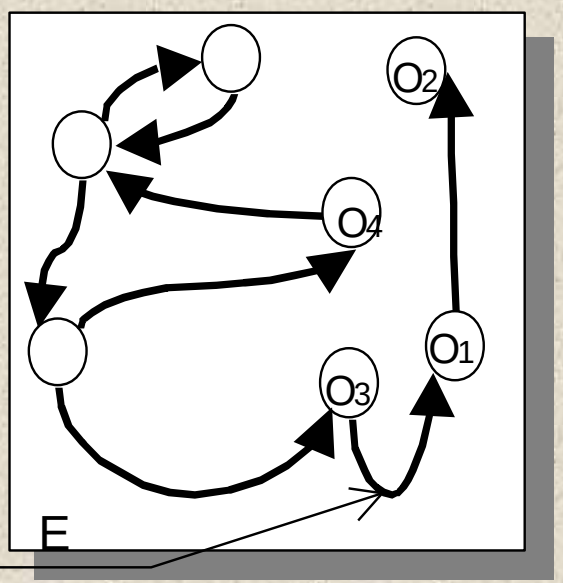
- **Modèle d'exécution localisée**
  - La création d'instance se fait
    - Ex-nihilo (new)
    - Par restauration (read)
    - Par duplication (clone)



Arbre d'intanciation

L'objet O_4 est à l'origine de la création de l'objet O_2

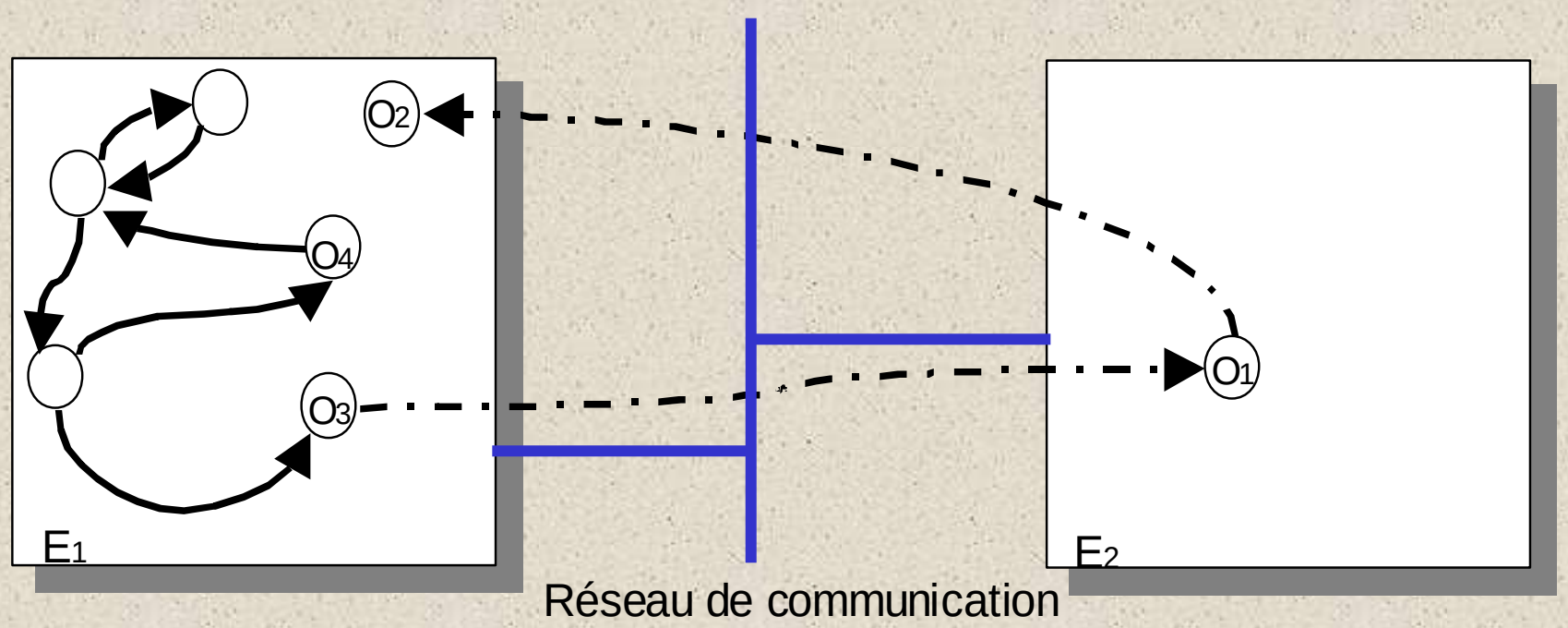
L'objet O_3 possède un attribut du type de O_1 qui est une référence vers l'objet O_1



Réseau de communication

Modèle distribué

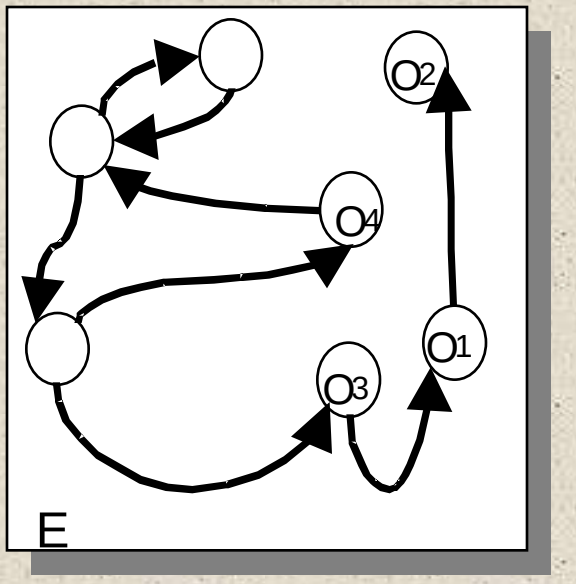
- **Modèle distribué**
  - Le principe consiste à distribuer l'ensemble des objets de l'application sur plusieurs JVM.
  - Ces JVM peuvent être situées sur des machines physiques distinctes.



Envoi de message en mode localisé

O_3 envoie un message à O_1 : O_1.m(…);

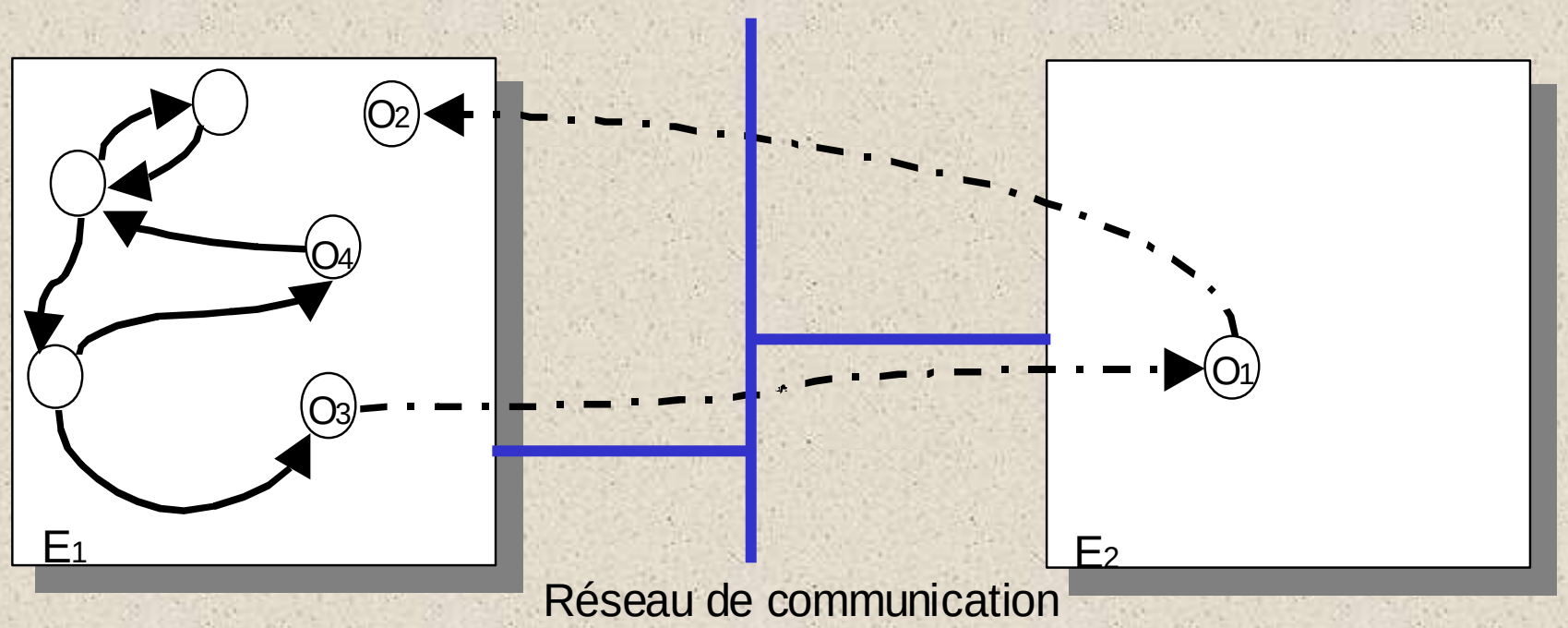
O_1 est une référence dans l'espace d'adressage.



Réseau de communication

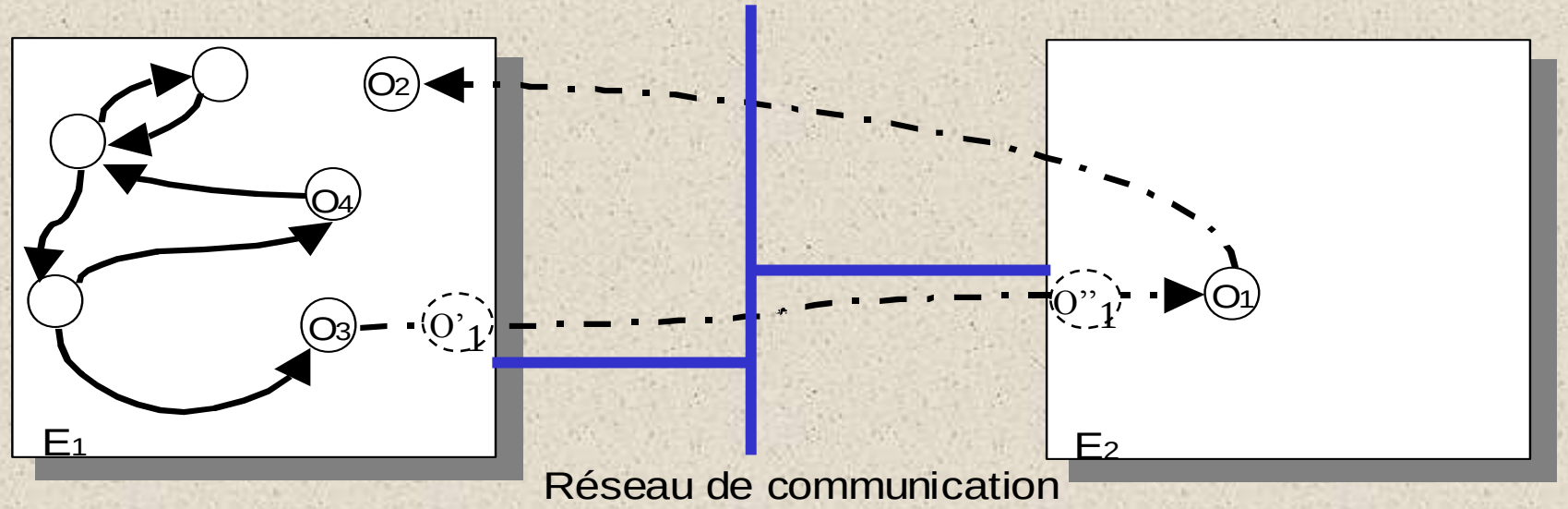
Envoi de message en mode distribué

- **Règle de distribution**
  - O_3 ne doit pas être tributaire de la localisation de O_1. Que celui-ci soit localisé ou distribué le comportement doit être le même.
    - Attention cette règle n'est pas si facile à garantir.

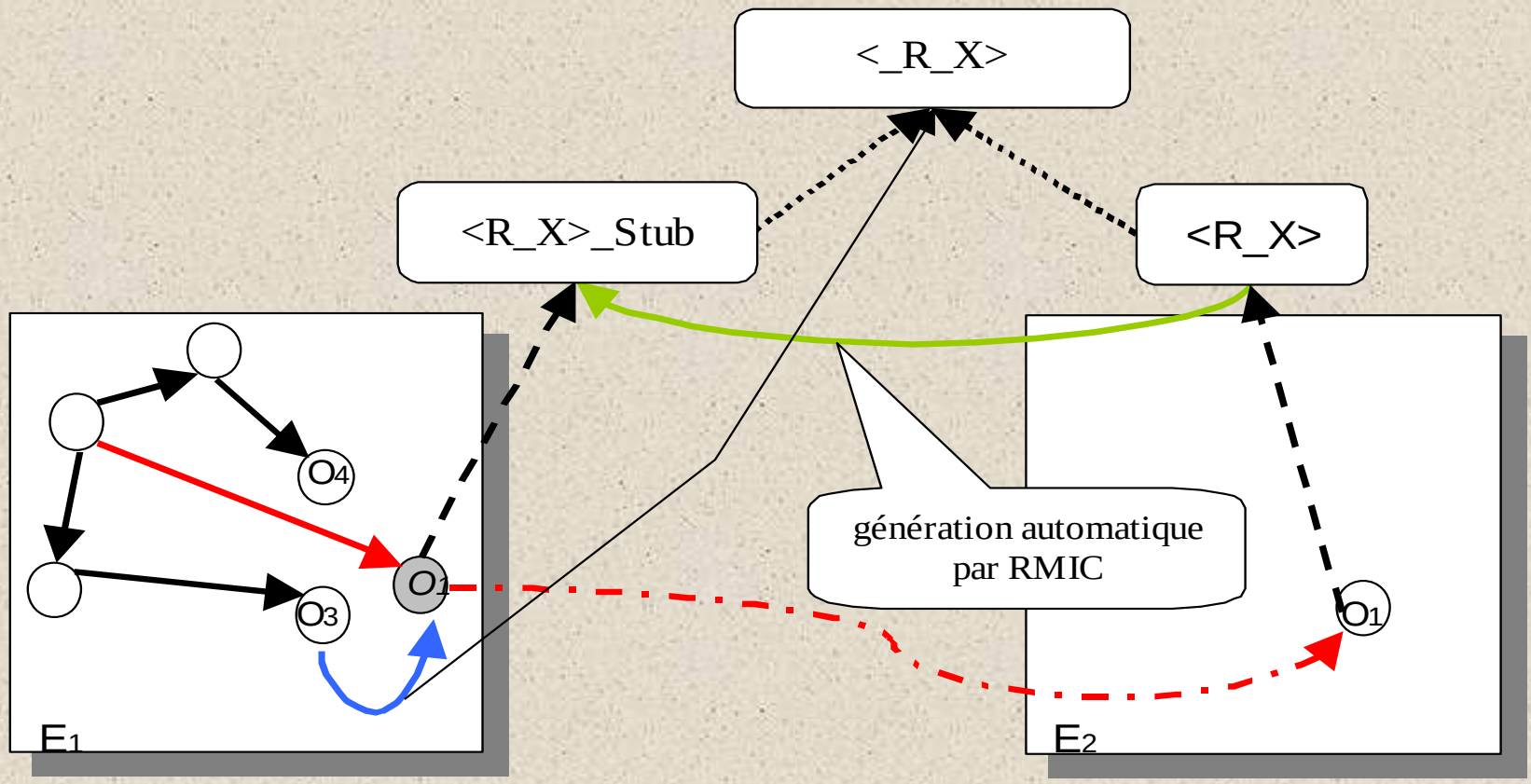


Envoi de message en mode distribué

- **Règle de distribution**
  - O_3 ne doit pas être tributaire de la localisation de O_1. Que celui-ci soit localisé ou distribué le comportement doit être le même.
    - Attention cette règle n'est pas si facile à garantir.
- **Principe de réalisation**
  - On crée un plénipotentiaire de O_1 dans l'espace de E_1.
  - Gérer la communication via le réseau.



Modèle distribué



Profil

- **Profil d'une méthode invocable à distance**
  - Une méthode invocable à distance peut ne pas remplir son contrat
    - Pour les mêmes raisons qu'un appel localisé
    - A cause d'un problème lié au réseau
      - RemoteException

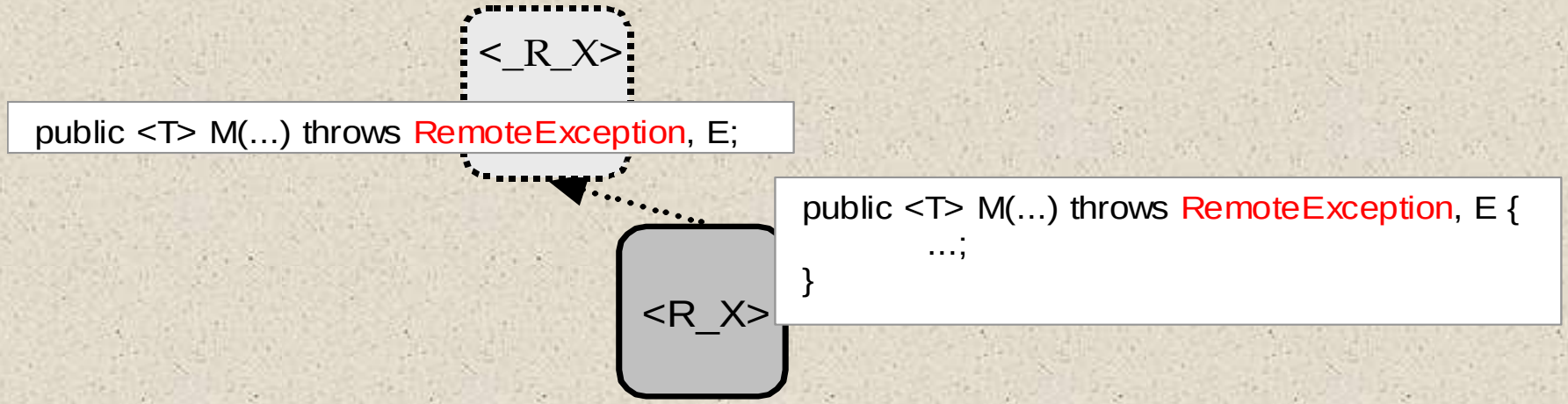
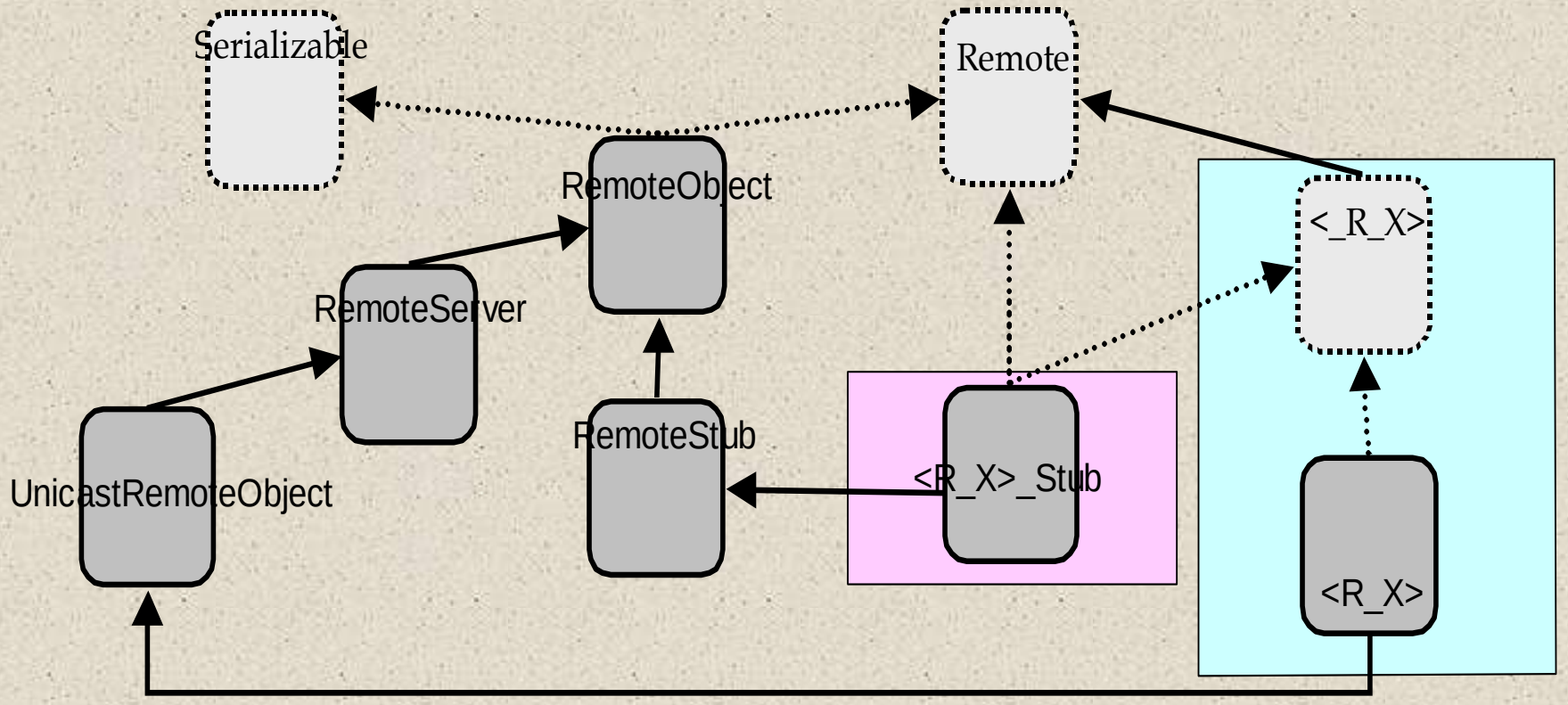


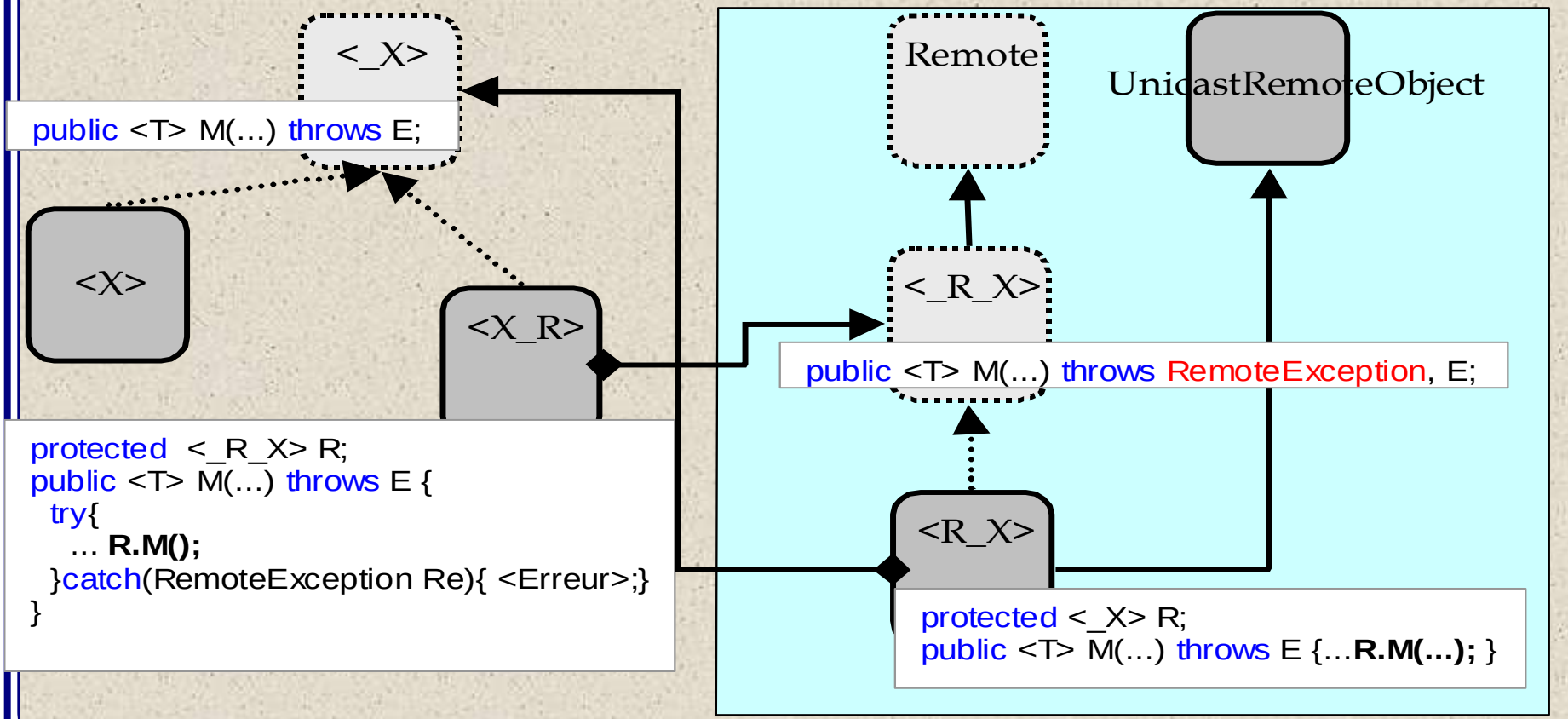
Schéma général des classes



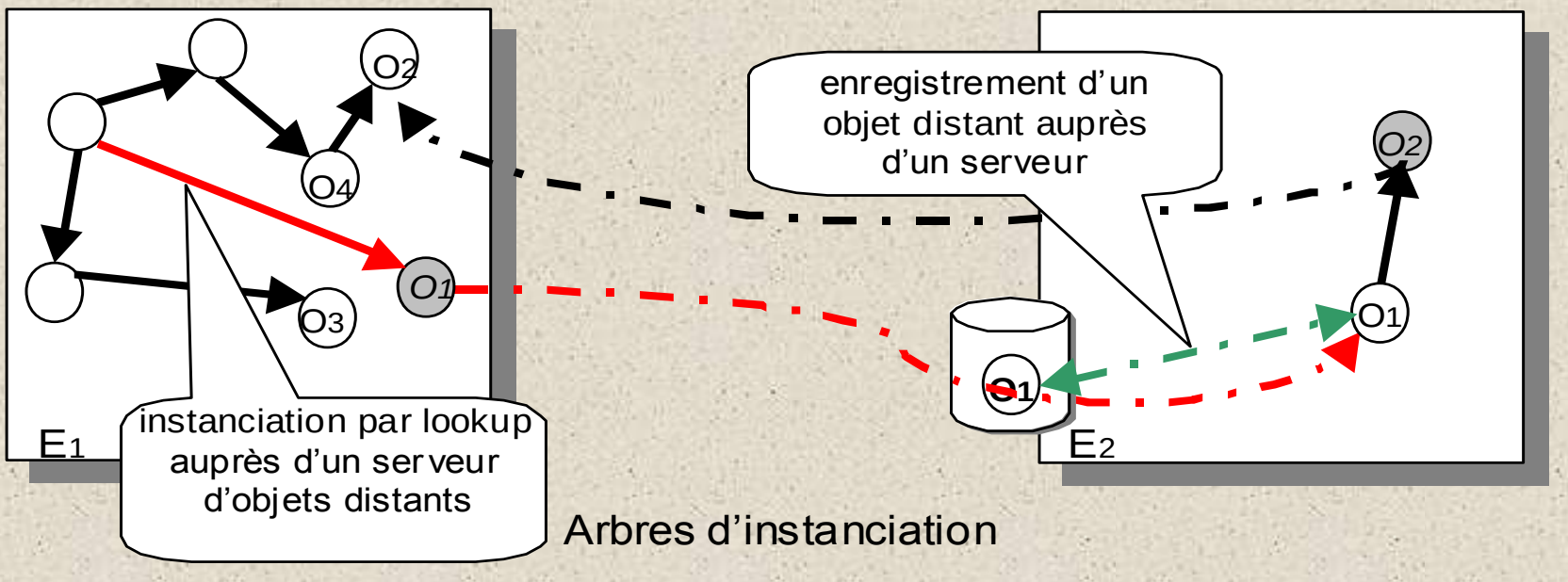
Structure générale d'un Stub

```
import java.rmi.*;import java.rmi.server.*;
public final class <R_X>_Stub extends RemoteStub implements <_R_X>, Remote {
   public <R_X>_Stub(RemoteRef ref) { super(ref); }
   private static final long serialVersionUID = 2;
   private static Method $method_<Method>_0;
   ...
   static {
   try {
      $method_name_0 = <_R _X>.class.getMethod("<Method>", new Class[] {...});
      ...
   } catch (NoSuchMethodException e) {
      throw new NoSuchMethodError("stub class initialization failed");
   }
}
// methods from remote interfaces
public <Type> <Method>(...) throws RemoteException {
      try {
         Object $result = ref.invoke(this, $method_name_0, new object[]{...}, -1972199368556355414L);
         return ((<Type>) $result);
      } catch (RuntimeException e) {throw e;
      } catch (RemoteException e) { throw e;
      } catch (Exception e) {throw new UnexpectedException("undeclared checked exception", e);
      }
      }
      ....
   }
```


Patron généralisé de distribution



Serveur d'objets distribués



Exportation d'objet distribuable

```
public static void main( String[] args) {
   try {
        if (System.getSecurityManager() == null) { System.setSecurityManager(new RMISecurityManager()); }
        <R_X> Remote = new <R_X>(…);
        Naming.rebind(<Name>, Remote);
        System.out.println("le <Name> "+" a été enregistré");
   }catch(Exception x) {
        System.out.println(x);System.exit(-1);
   }
}
```

Création d'un objet exportable

Exportation de l'objet Remote sous le nom <Name> dans l'environnement standard

Importation d'objet distribué

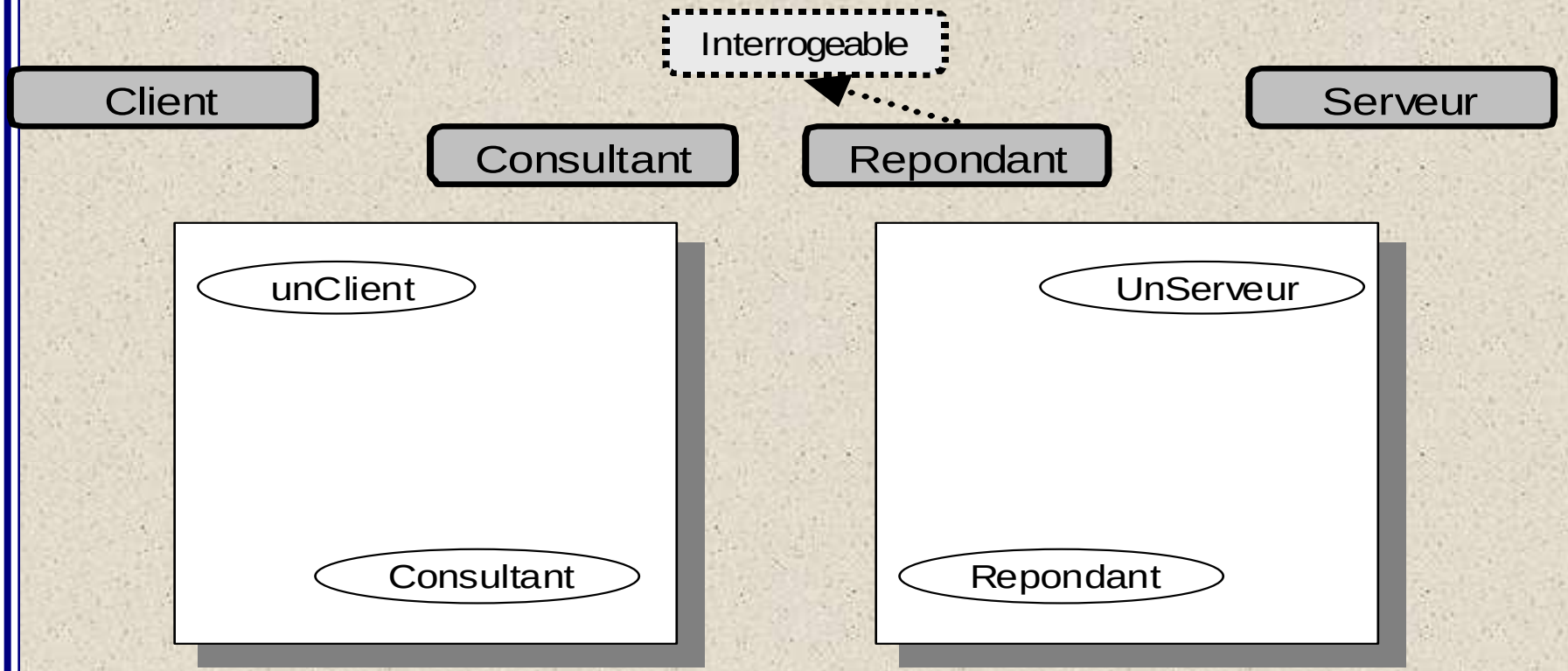
```
<Interface_X> Remote;
try {
   // recherche de l'objet exporté à partir de sa pseudo URL
   Remote = (<_X>)Naming.lookup("rmi://<Host>:<port>/<Name>");
}catch(java.net.MalformedURLException  x) {…;
}catch(java.rmi.NotBoundException  x) {…;
}catch(java.rmi.UnknownHostException  x) {…;
}catch(java.rmi.RemoteException  x) {…;}
```

Le nom de l'objet distant

L'accès au serveur d'objet distant

importation de l'objet distant

Exemple


Interrogeable
Client
Consultant
Repondant
Serveur
unClient
Consultant
UnServeur
Repondant

Interface Interrogeable

```
import java.rmi.*;
/**définit l'interface de l'objet "remotable" capable de répondre au message*/
public interface Interrogeable extends Remote {
   public String question() throws RemoteException;
   public String name() throws RemoteException;
}
```


Classe Server

```
public class Server {
  public static void main(final String args[]) {
    String nom="";
    int nombre=1, port = 1099;
    Registry registry=null;
    // récupération des arguments
    if (args.length!=3){
        System.out.println("Server <nom générique des objets distants> <nombre de noms> <port registry>");
        System.exit(1);
    }
    try  {
        nom = args[0]; nombre = Integer.parseInt(args[1]); port = Interger.parseInt(args[2]);
    }catch(Exception e) {
        System.out.println("Server <nom générique des objets distants> <nombre de noms> <port registry>");
        System.exit(1);
    }
    // installation d'un securityManager
    if (System.getSecurityManager() == null) { System.setSecurityManager(new RMISecurityManager()); }
    if((registry=locateRegistry.getRegistry(port))==null) registry=LocateRegistry.createRegistry(port);
    try {
        for(int i=1;i<=nombre;i++)  {registry.rebind(nom+i, new Repondant(i));}
        System.out.println("Tous les objets sont enregistrés dans le serveur d'objets distants");
    } catch (Exception e) { System.out.println("Server err: " + e); }
  }
}
```


Classe Repondant

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class Repondant extends UnicastRemoteObject implements Interrogeable {
   private String num;
   public Repondant() throws RemoteException  { super();}
   public String question() throws RemoteException { return "reponse de "+this;}
   public String name() throws RemoteException { return toString();}
   public String toString() throws RemoteException { return "Repondant"+num;}
}
```


Classe Client

```
import java.rmi.RMISecurityManager;
public class Client {
   public static void main(String args[]) {
    String nom="", host="localhost";
    int nombre=1;
    // récupération des arguments
    if (args.length!=3){
          System.out.println("Client <hostname> <nom générique des objets distants> <nombre de noms>");
          System.exit(1);
    }
    try  {
          host = args[0];
          nom = args[1];
          nombre = ((int)(Math.random()*Integer.parseInt(args[2])));
    }catch(Exception e) {
          System.out.println("Client <hostname> <nom générique des objets distants> <nombre de noms>");
          System.exit(1);
    }
    // installation d'un securityManager
    if (System.getSecurityManager() == null) {  System.setSecurityManager(new RMISecurityManager()); }
    Consultant consultant = new Consultant();
    consultant.consulter(host,nom,nombre);
   }
}
```


Classe Consultant

```
import java.rmi.Naming;
public class Consultant {
  public void consulter(String ou, String qui, int num) {
  try {
       Interrogeable obj = (Interrogeable) Naming.lookup("rmi://"+ou+"/"+qui+num);
       System.out.println("L'objet distant "+obj.name()+" est lié");
       System.out.println("obj.question() = "+obj.question());
  } catch (Throwable e) {
       System.out.println("Consultant erreur: " + e);
  }
  }
}
```


Suite & fin

```
// police générale, ouvrant tous les droits
grant {
  permission java.security.AllPermission;
};
```

```
/* compilation et génération du Stub du serveur  sur la machine  <X> */
javac  Interrogeable.java Repondant.java Server.java
rmic  -d<localisation codebase> -v1.2 Repondant
/* démarrage du serveur RMI sur la machine  <X> : attention il faut un  classpath reduit */
rmiregistry&
/* lancement du serveur sur la machine  <X> */
java -Djava.rmi.server.codebase=<localisation codebase> -Djava.security.policy=<policy>
Server …
/* compilation du client sur la machine  <Y> */
javac  Interrogeable.java  Consultant.java Client.java
/* lancement du client sur la machine  <Y> */
java  Client …
```


Références

Quelques sites :

http://www.jmdoudoux.fr/java/dej/chap023.htm

http://marine.edu.ups-tlse.fr/~torguet/cours/csti/RMI_Cours.pdf