
# TI 1 : Méthodes de conception de systèmes d'information distribués

Master 2 Traitement de l'Information

Lionel Médini

Septembre 2010

# Plan du cours

- Outils de programmation avancés
- Systèmes d'information distribués
  - Appel de méthodes distantes (RPC)
  - Objets hétérogènes distribués (CORBA)
  - Objets Java distribués (RMI, RMI/IIOP)
  - Synthèse
  - Frameworks Java (Struts, Spring, JEE 5)
- Objets transactionnels distribués
  - Exemples d'EJB 2 et de descripteurs de déploiement
  - EJB 3 : POJO et annotations

# Objets distribués



Approche orientée objet
Architectures distribuées
} Objets distribués

- Exemples d'application : agence de voyage en ligne
- Exemples d'objets distribués
  - Logique applicative client (connexion, recherche commandes)
  - Gestion sécurisée des paiements
  - Gestion des réservations
  - Interrogation des fournisseurs de voyages
  - …

# Infrastructures

## *middleware*



- But : gestion des communications entre les objets hétérogènes *via* le réseau dans les architectures distribuées
- Exemples
  - RPC (Sun, Microsoft...)
  - CORBA (OMG)
  - RMI (Java)
  - RMI/IIOP (Java)
  - Java EE (Sun)
  - .Net (Microsoft)

# RPC

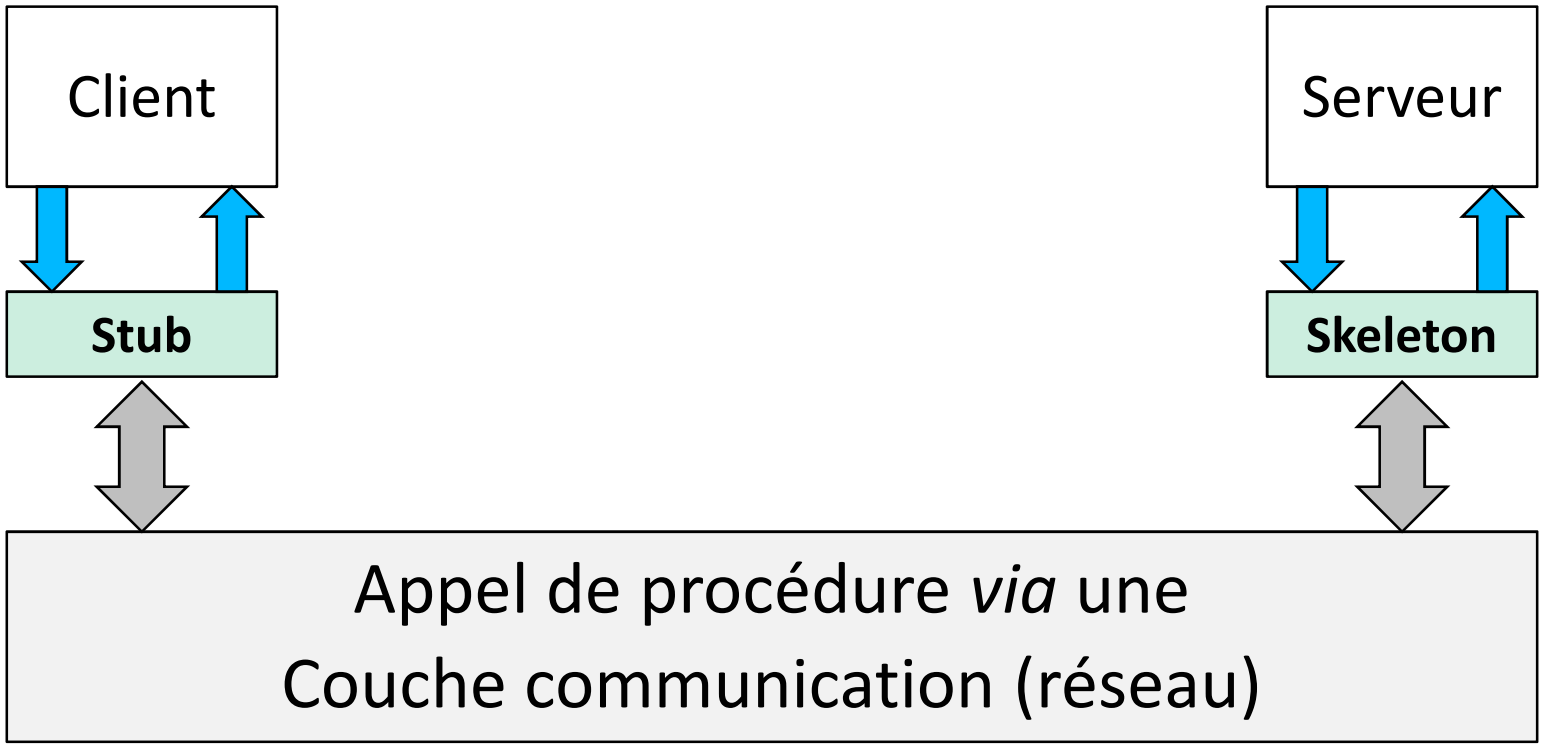


- Origine
  - Créé pour le système de fichiers NFS
  - Version « originale » : Sun RPC (libre)
  - Dernière RFC : 1057, juin 1988
- Principe
  - Appel de fonctions distantes
  - Fonctionne sur un mode client-serveur
  - Langages de programmation hétérogènes
  - Transparence des appels distants pour les composants locaux

# RPC



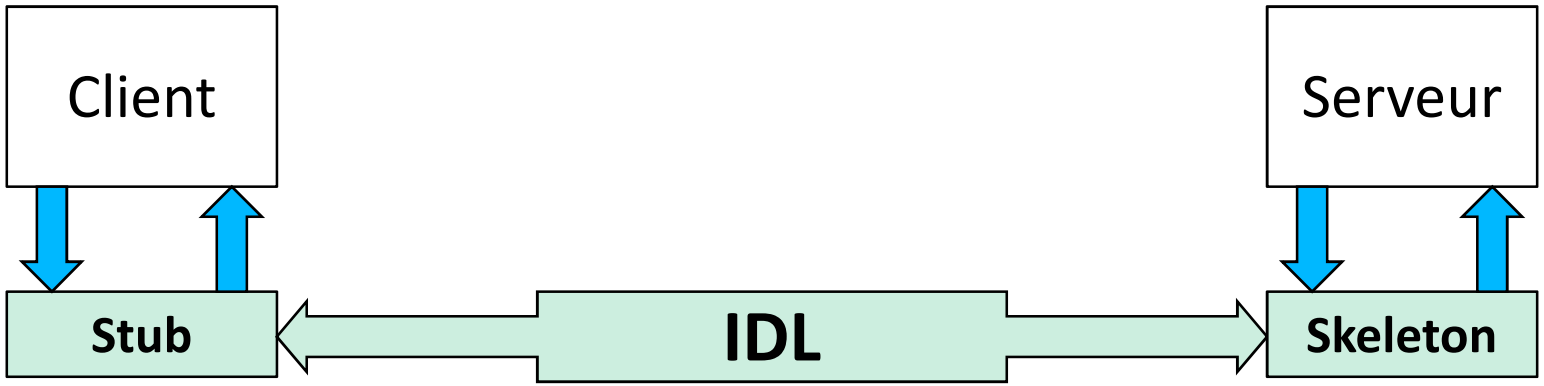
- Fonctionnement
  - Des composants locaux (stub et skeleton) masquent la couche réseau au client et au serveur



# RPC



- Fonctionnement
  - Au niveau logique, les prototypes des fonctions et types de données exposés sont décrits dans un langage commun (IDL)



# RPC



- Langages/notions de base
  - Stub/skeleton
    - Proxys  de gestion des communications entre objets
    - Stub : proxy côté client
      - reçoit et achemine les requêtes du client
      - récupère les réponses et les transmet au client
    - Skeleton : proxy côté serveur
      - Récupère les requêtes et les transmet au serveur
      - reçoit et achemine les réponses du serveur
    - Spécifiques aux langages, couches de communication, types d'objets

# RPC



- Langages/notions de base
  - IDL : langage neutre de spécification d'interfaces
    - Représentation des interfaces (informations échangées) dans le même langage
    - Typage des données en XDR (eXternal Data Representation : dernière RFC : 4506)
    - Traduction (projection) des interfaces dans différents langages (C, C++, SmallTalk, Ada95 et Cobol OO, Java, Eiffel et Common Lisp)

# RPC



- Conclusion
  - Limité à l'appel de méthodes (pas d'objet)
  - Introduit les principes fondamentaux
    - d'indirection dans l'accès aux ressources distantes
    - de **serveur d'application**
- Remarques
  - Plusieurs versions, incompatibles entre elles
  - Implémentation sur HTTP : XML-RPC
- Références
  - http://www.ietf.org/rfc/rfc1057.txt
  - http://www.crevola.org/francois/?content=articles&show=1

# CORBA



- But
  - Communication entre objets hétérogènes et distants
  - Invocation de « services » entre objets d'applications distribuées
- Principes généraux
  - Séparation stricte Implémentation/Interface
  - Localisation transparente des objets
- Caractéristiques techniques
  - Architecture client/serveur
  - Objets distribués ou non
  - Multi-plateforme

# CORBA



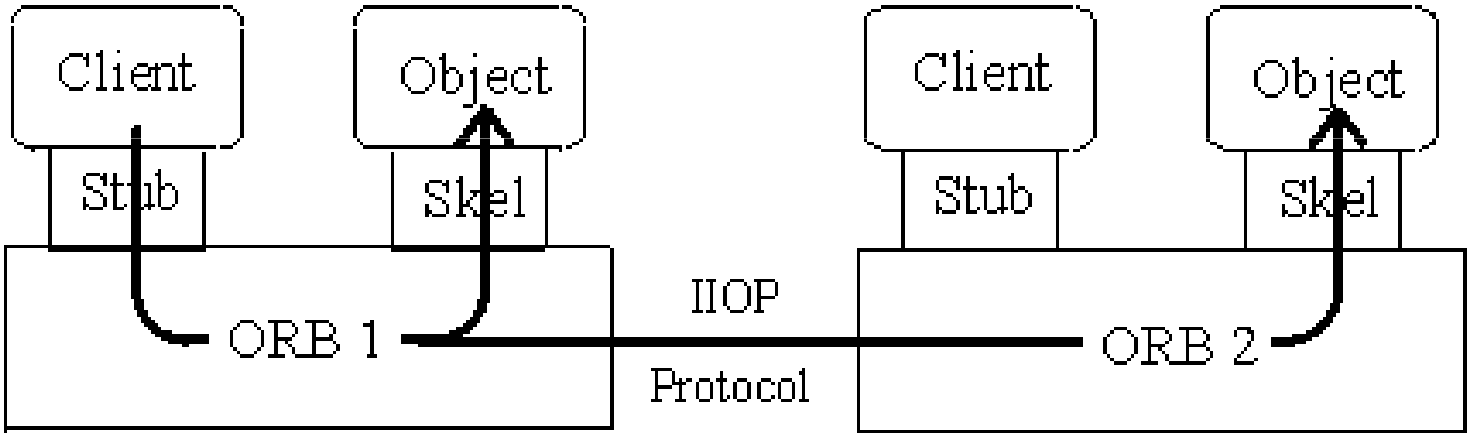
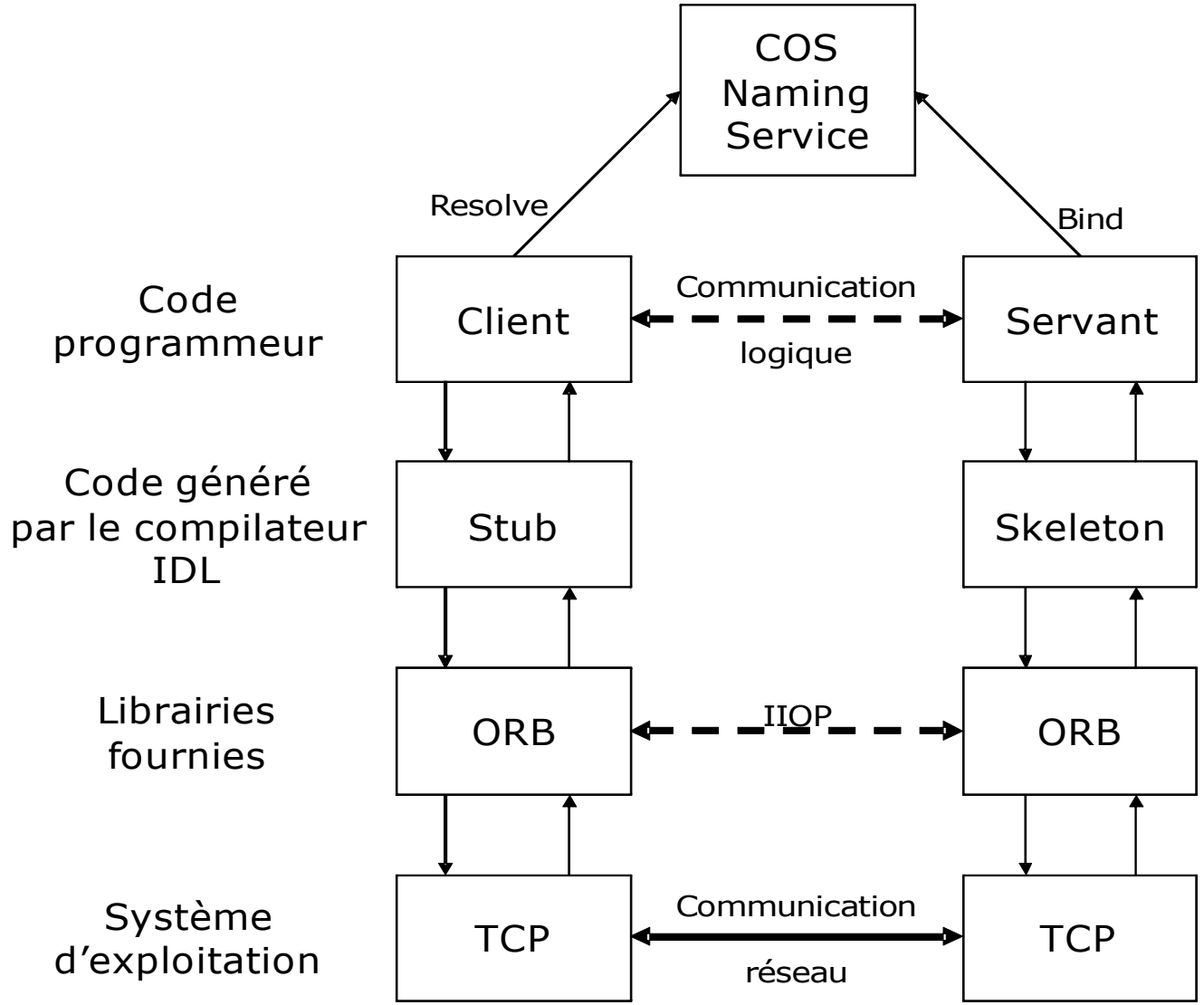



Figure 2: Interoperability uses ORB-to-ORB communication

Copyright © 2000 Object Management Group

# CORBA





# CORBA



- Objets/librairies
  - ORB : Couche « communication » intégrée aux objets
    - Responsable des mécanismes nécessaires pour
      - Trouver l'implémentation de l'objet pour la requête
      - Préparer cette implémentation à recevoir la requête
      - Communiquer les données constituant la requête

# CORBA



- Objets/librairies
  - ORB : Couche « communication » intégrée aux objets
    - L'interface que voit le client est indépendante
      - De l'endroit où l'objet est situé
      - Du langage dans lequel l'objet est implémenté
    - Un ORB contient
      - Une interface IDL
      - Un support au service de nommage COS
      - Un support IIOP

# CORBA



- Objets/librairies
  - Stub : proxy client
  - Skeleton : proxy serveur

  } s'interfacent avec un ORB
- CORBA COS : services objets communs
  - Naming Service : service de nommage permettant de retrouver les objets servants pour les clients
  - Life Cycle Service
  - Object Transaction Service
  - Security Service
  - …

# CORBA



- Langages/protocoles
  - IDL : langage neutre de spécification d'interfaces
    - Traduction effectuée à la compilation (statique)
    - Utilise un compilateur IDL spécifique au langage utilisé pour produire
      - Un stub/skeleton lié à un ORB
      - Une description des interfaces des services

```
Interface Chat {
  void SetMessage (in string auteur, in string
  texte);
}
```

# CORBA



- Langages/protocoles
  - IIOP : protocole de communication entre ORB
    - Transmission des messages
    - Implémentation de GIOP sur TCP
    - Échange de messages entre « ORB »

# CORBA



- Langages/protocoles
  - DII : accès dynamique aux serveurs
    - Permet de « court-circuiter » un ORB
    - Découverte dynamique de nouveaux objets
    - Construction et distribution d'invocation
    - Réception de réponses

# CORBA



- Exemple d'utilisation
  - Fichier Chat.idl

```
interface Chat {
 void setMessage (in string auteur, in string texte);
}
```

  - Compilation

```
idlj –fallTIE Chat.idl
```

  - Interface que doit implémenter le servant

```
public interface ChatOperations {
  void setMessage (String auteur, String texte);
}
```

# CORBA



- Exemple d'utilisation
  - Interface distante que doit implémenter le servant
    ```
    public interface Chat extends ChatOperations,
    org.omg.CORBA.Object, org.omg.CORBA.portable.IDLEntity
    {}
    ```
  - Classe skeleton
    `Chat_Tie`
  - Classe stub
    `ChatStub`
  - Classe contenant des méthodes auxiliaires :
    `ChatHelper`

# CORBA



- Exemple d'utilisation
  - À Programmer

```
class ChatServant implements ChatOperations {
  void setMessages (String auteur, String texte){…}
}

class ChatServer {
  public static void main (String args[]){…}
}

class ChatClient {
  public static void main (String args[]){…}
}
```

# CORBA



- Exemple d'utilisation
  - Lancement
    - Serveur de noms (machine m1)

      `tnameserv` (depuis JDK 1.3)
    - Serveur (machine m2)

      `java ChatServer –ORBInitialHost m1`
    - Client(machine m3)

      `java ChatClient –ORBInitialHost m1`

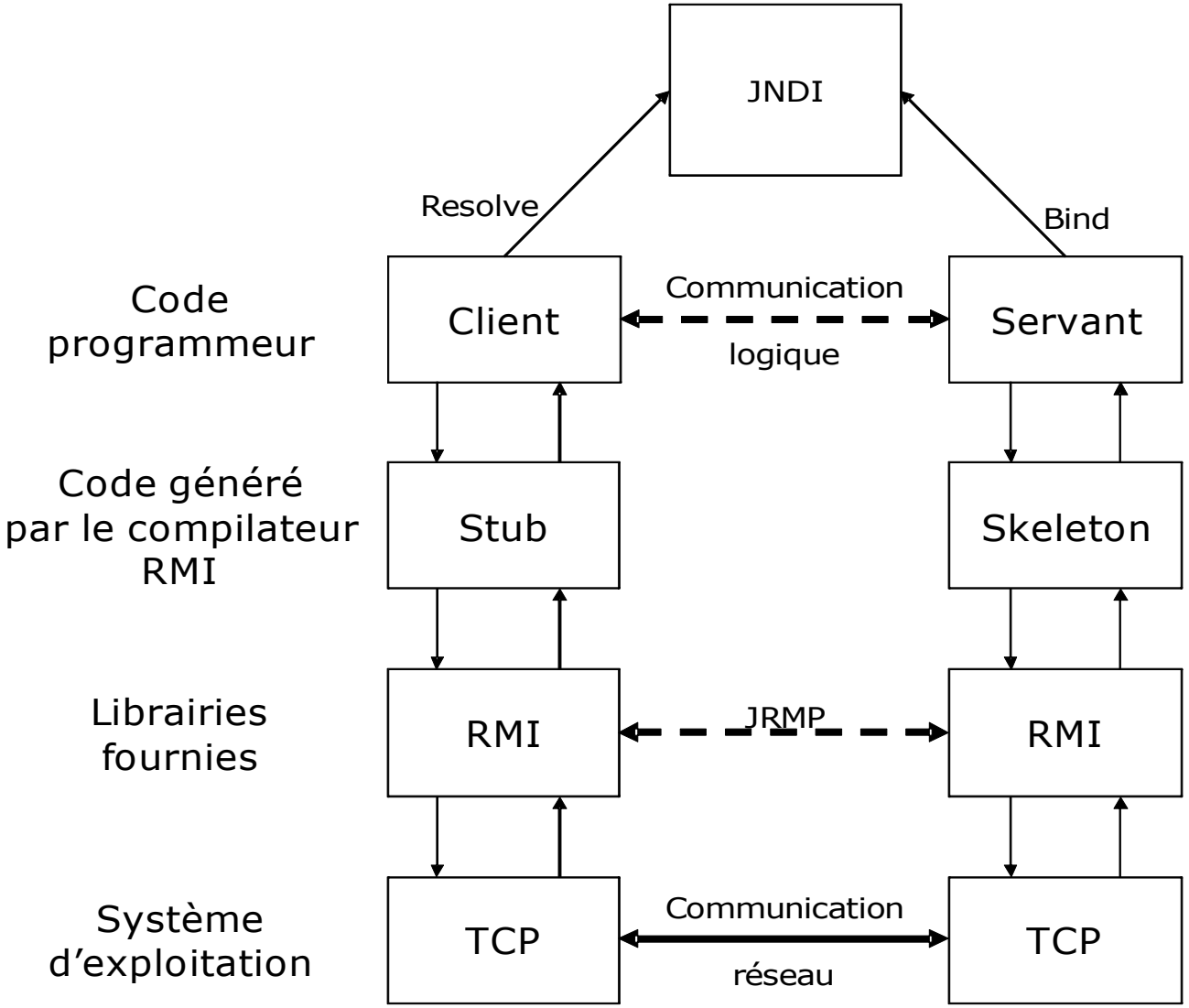
# RMI



- Mêmes principes de base que CORBA
- Limité à Java (objets non hétérogènes)
- Plus de langage de spécification d'interfaces
- Protocole de communication : JRMP

# RMI





# RMI



- Exemple d'utilisation
  - Interface Java

```
package monchat;
import java.rmi.*;
 public interface Chat extends Remote {
    public void setMessage (String auteur, String texte)
throws RemoteException;}
```

  - Classe skeleton

```
ChatServant_Skel
```

  - Classe stub

```
ChatServant_Stub
```

# RMI



- Exemple d'utilisation
  - À Programmer

```
class ChatServant extends UnicastRemoteObject implements
  Chat {
  void setMessages (String auteur, String texte){…} }
class ChatServer {
  public static void main (String args[]){…} }
class ChatClient {
  public static void main (String args[]){…} }
```

  - Compilation

```
rmic monchat.Chat
```

# RMI

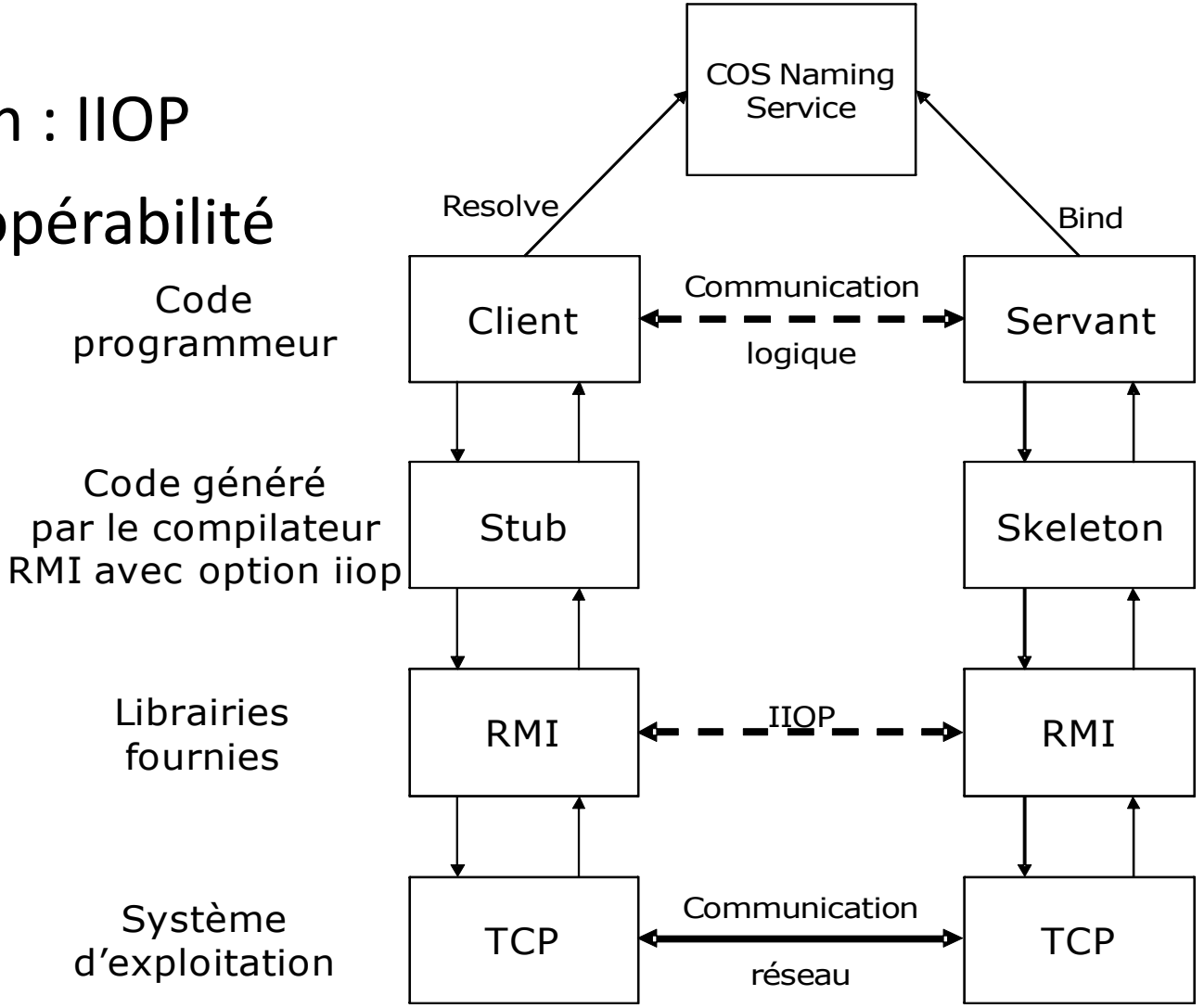


- Exemple d'utilisation
  - Serveur de noms
    `rmiregistry` (méthodes `lookup()` et `bind()`)
  - Lancement
    - Serveur de noms (machine m1)
      `rmiregistry`
    - Serveur (machine m2)
      `java -Djava.rmi.server.codebase=http://m1/ ChatServer`
    - Client(machine m3)
      `java -Djava.rmi.server.codebase=http://m1/ ChatClient`

# RMI/IIOP



- Protocole de communication : IIOP
- Permet l'interopérabilité RMI / CORBA



# RMI/IIOP



- Exemple d'utilisation (différences avec RMI)
  - Implémentation du servant

```
class ChatServant extends PortableRemoteObject
  implements Chat {
  void setMessages (String auteur, String
  texte){…} }
```

  - Transtypage complexe
  - Compilation : `rmic –iiop monchat.Chat`
  - Packages à importer
    - Javax.rmi (servant, serveur, client)
    - Javax.naming (serveur, client)

# RMI/IIOP



- Exemple d'utilisation (différences avec RMI)
  - Lancement
    - Serveur de noms (machine m1)

      ```
      tnameserv
      ```

    - Serveur (machine m2) :

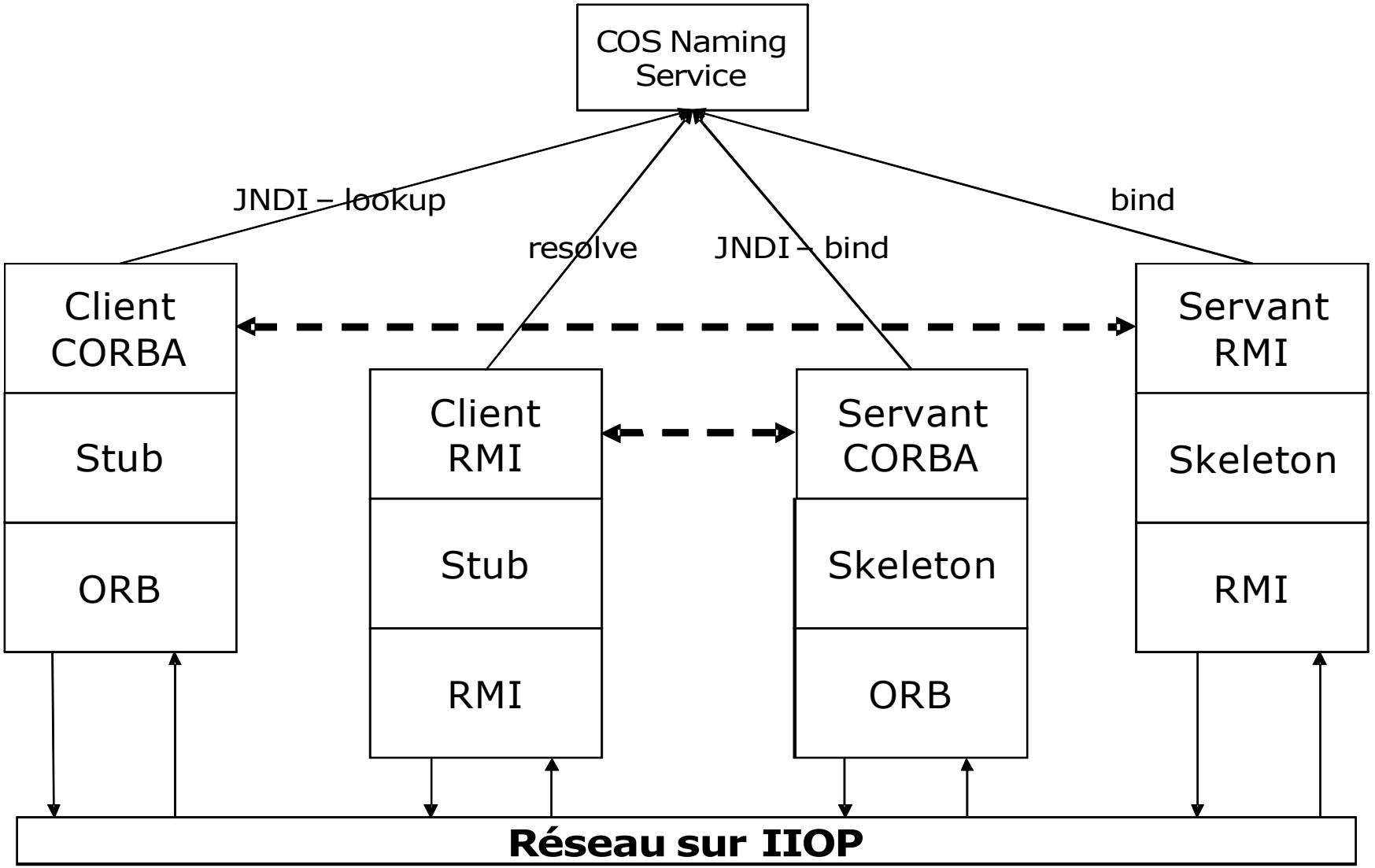
      ```
      java -Djava.rmi.server.codebase=http://m1/ ChatServer
      ```

    - Client(machine m3) :

      ```
      java -Djava.rmi.server.codebase=http://m1/ ChatClient
      ```

# CORBA + RMI/IIOP





# Conclusion



- Principes fondamentaux
  - Objets client, servant et serveur
  - Invocation d'objets distants transparente pour le client
  - Échange de messages conformes à des descriptions d'interfaces
  - Service de nommage
- Limites
  - Requiert la programmation du serveur
  - Requiert l'inscription dans le serveur de noms
  - Interfaces Générées à la compilation (CORBA non-DII)
- Mise en place « lourde » pour le développeur

# Conclusion



- Géré : accès aux services transversaux
  - Nommage (serveur de noms)
  - Transactions
  - Persistance
  - Sécurité…
- Non géré : optimisation des accès aux ressources
  - Pools de connexion ou de threads
  - Activation et désactivation des objets
  - Répartition de la charge
  - Tolérance aux pannes

# Références



- CORBA
  - http://www.omg.org
  - http://corba.developpez.com/cours/
  - http://corba.developpez.com/presentation.htm
- RMI et RMI/IIOP
  - http://java.sun.com/products/jdk/rmi/
  - http://java.sun.com/j2se/1.4.2/docs/api/
- Exemples de code RMI et RMI/IIOP
  - http://java.sun.com/developer/codesamples/index.html
  - http://thomasfly.com/RMI/rmi_tutorial.html
  - http://www-128.ibm.com/developerworks/java/rmi-iiop/space.html

# Struts

## http://struts.apache.org/



- Caractéristiques
  - Développement d'applications Web
  - Fondé sur le pattern MVC 2
    - Un contrôleur unique (ActionServlet)
    - Un délégateur de requêtes (ActionMapping)
    - Des « workers » (Action, *.do)
    - Des composants (Beans) pour le modèle métier
  - S'appuie sur les technologies servlets, JSP, JSTL...
- Avantage
  - Très connu / utilisé
- Inconvénients
  - Limité aux applications simples (modèle métier = quelques classes Java)
  - Difficilement testable

# Spring

## http://www.springframework.org/



- Caractéristiques
  - Framework fondé sur MVC 2
  - Conteneur léger
  - Support AOP : AspectJ
  - Intégration d'autres frameworks : Struts, JSF, AJAX DWR, support de portlets, ORM, JUnit...
- Avantages
  - Framework complet de développement d'applications
  - S'appuie sur d'autres frameworks ayant fait leurs preuves
- Inconvénient
  - Choix des solutions *a priori*

# JEE5

## http://java.sun.com/javaee/



- Caractéristiques
  - Plutôt une spécification qu'un framework
  - Intégré aux serveurs d'applications Java
  - Permet le développement d'applications intégrant des EJB
    - Conteneur EJB
    - Serveur JNDI
    - Gestion des services spécifiques (JPA, JMS...)
- Avantage
  - Très complet
  - Simplifié depuis la spécification EJB 3.0
- Inconvénient
  - Trop complet ?

# JEE5

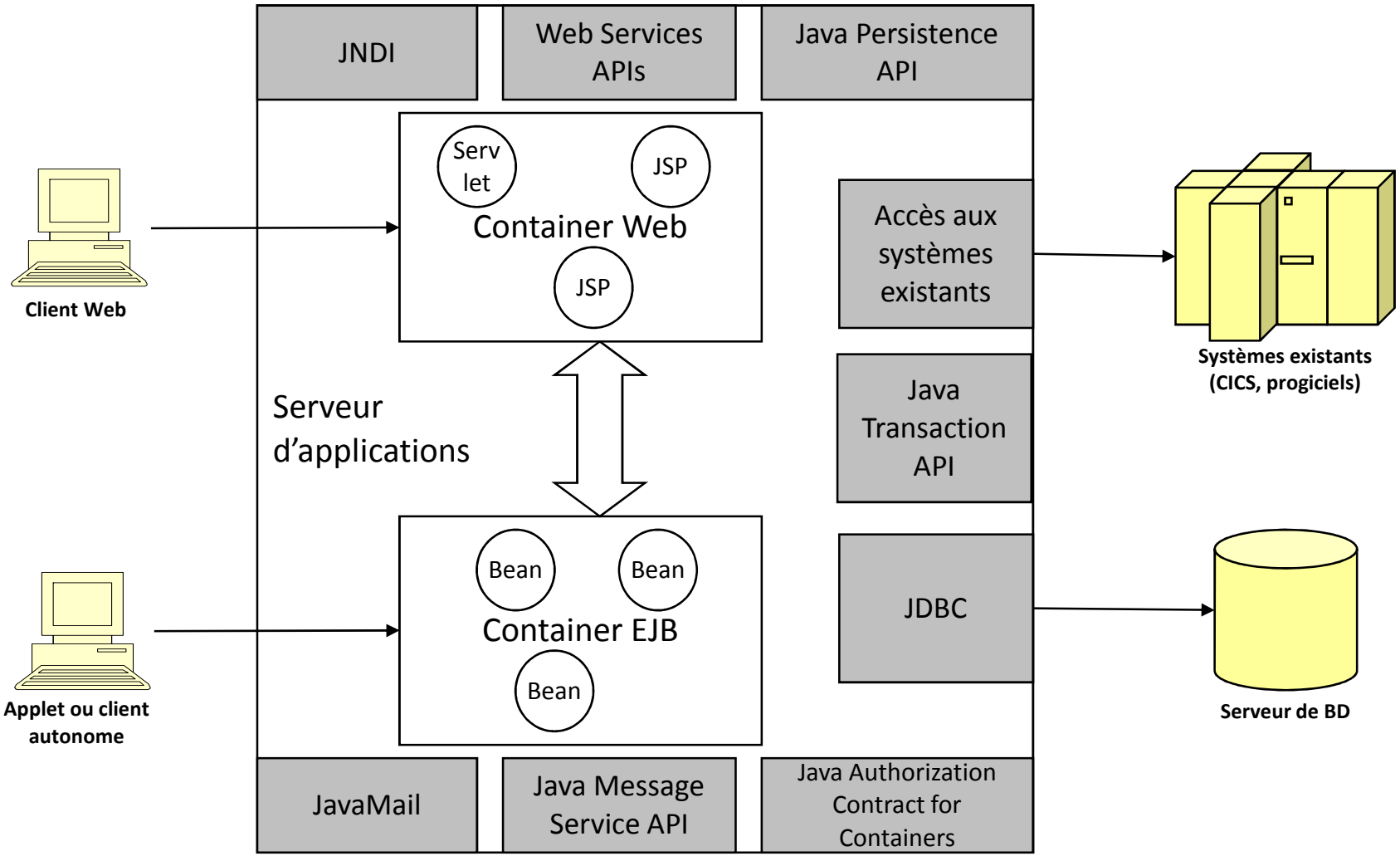
## http://java.sun.com/javaee/



- But : développement, déploiement et exécution des applications distribuées
- Mêmes principes de base que les infrastructures middleware…
  - Communication synchrone (RMI/IIOP) et asynchrone (JMS) entre objets distribués, serveurs de nom (JNDI), intégration d'objets CORBA (JavaIDL)…
- …plus de nombreux services techniques supplémentaires
  - Génération d'objets transactionnels distribués (EJB), gestion du cycle de vie des objets, gestion des transactions (JTA et JTS), sécurité, accès aux BD (JDBC), interfaces graphiques dynamiques (Servlets , JSP, JSF)…

# L'architecture JEE5





# Exemple de service JEE5



- Java Persistence API (JSR 220)
  - The Java Persistence API provides a POJO persistence model for object-relational mapping.
  - The Java Persistence API was developed by the EJB 3.0 software expert group as part of JSR 220, but its use is not limited to EJB software components. It can also be used directly by web applications and application clients, and even outside the Java EE platform, for example, in Java SE applications.
  - http://java.sun.com/javaee/technologies/persistence.jsp