Rhum

Un ORB réactif

Laurent Hazard
FT R&D DTL/ASR
02/01

Sommaire

- Motivation
  - exemple et discussion
- DPE / ORB : plateformes réparties
- Présentation de Rhum
  - et quelques détails d'implémentation
  - Retour sur l'exemple
- Conclusion

Motivation

- Programmation réactive/synchrone
  - systèmes réactifs, interactions, //, synchro.
- Contexte de la programmation objet
  - Java, ORBs
  - interaction procédurale -> RPC
  - séquence d'actions
- Carences, difficultés...

Exemple

- Un serveur

```
public interface ServeurI {
      public int add (int a, int b);
}
```

- Un client

```
...
ServeurI serveur = <ref. sur un serveur>;
...
res = serveur.add (x1, x2);
```

Requêtes parallèles

- Comment faire pour programmer :

```
...
ServeurI serveur1 = <ref. sur un serveur>;
ServeurI serveur2 = <ref. sur un serveur>;
...
res = serveur1.add (x1, x2)
      ou
      serveur2.add (x1, x2)
      (le premier qui - y - arrive) ;
```

- ... Parallèlisme

Requête gardée

- Comment faire pour programmer :

```
...
    ServeurI serveur = <ref. sur un serveur>;
    ...
    res = serveur.add (x1, x2)
          en moins de 3 secondes sinon 0;
```

- ...Concurrence

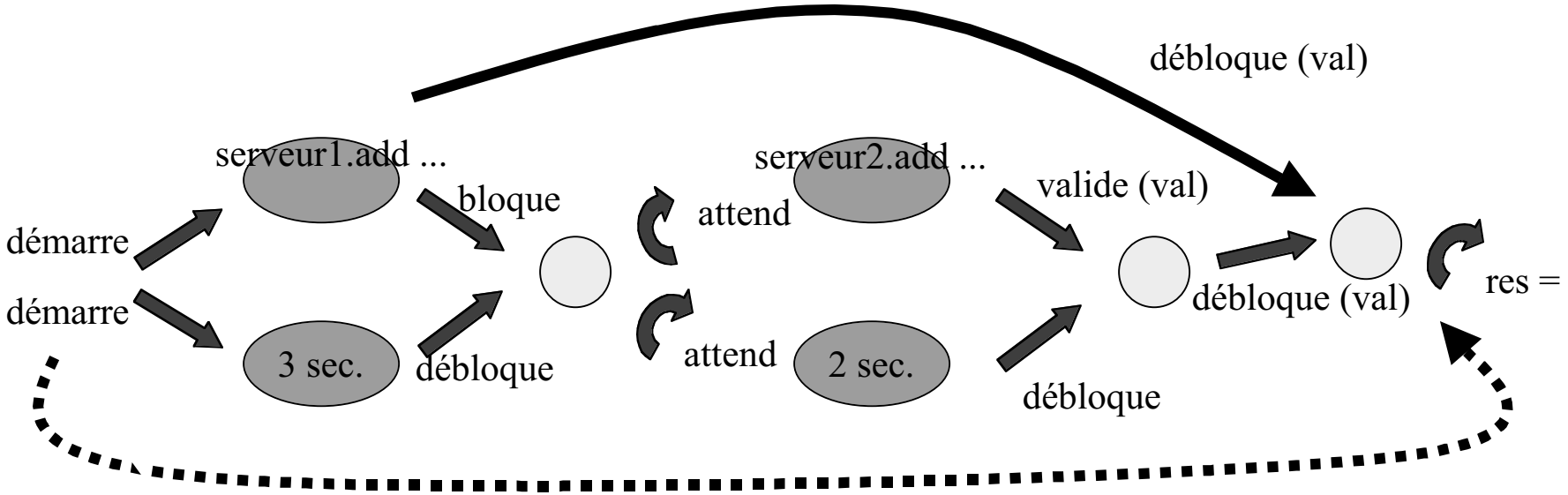
Requêtes gardées en //

- Comment faire pour programmer :

```
...
ServeurI serveur1 = <ref. sur un serveur>;
ServeurI serveur2 = <ref. sur un serveur>;
...
res = serveur1.add (x1, x2)
       si après 3 secondes pas de résultat,
      lancer serveur2.add (x1, x2)
      si après 2 sec. supp.
      serveur1 n'a pas rendu de résultat
      prendre celui de serveur2 si présent,
      sinon 0;
```

A coeur vaillant...

- En Java, pour les activités // et concurrentes, => on a les Threads !
- Essayons :



Discussion...

- Faisable ? mouais...
  - pas simple
  - pas modulaire
  - pas « scalable »
- Faisable ? pas sûr...
  - a-t-on vraiment «programmé» ce qu'on voulait?
    - sémantique d'approximation
    - modèle d'activité implicite et... inconnu...

Discussion (2)

- Programmation ?
  - on reste en séquence... seule « garantie » !

```
...
requeteServeur1.start ();
requeteServeur2.start ();
...
res = serveur.add (a, b);
serveur.add (a, b, moi);
...
timer.start ();
requeteServeur.start ();
```

Discussion (3)

- Valeur de retour (RPC) ou pas ?
  - en partie, un problème (mais il y a aussi d'autres problèmes)
  - et c'est quand même pratique.
- Si retour « asynchrone »
  => le client doit être aussi serveur
  …et alors là… :
  - threads de l 'ORB, sans contrôle de l'appli
  - protection des accès concurrents

Discussion (4)

- Manque un modèle explicite
  - d'activité
  - d'activation
    pour les clients et les serveurs
- Manque une définition de contexte
  - pour y appliquer un modèle
  - commun avec les autres activités du système

Plateformes réparties

... qqs mots, en général et en particulier !

Principes

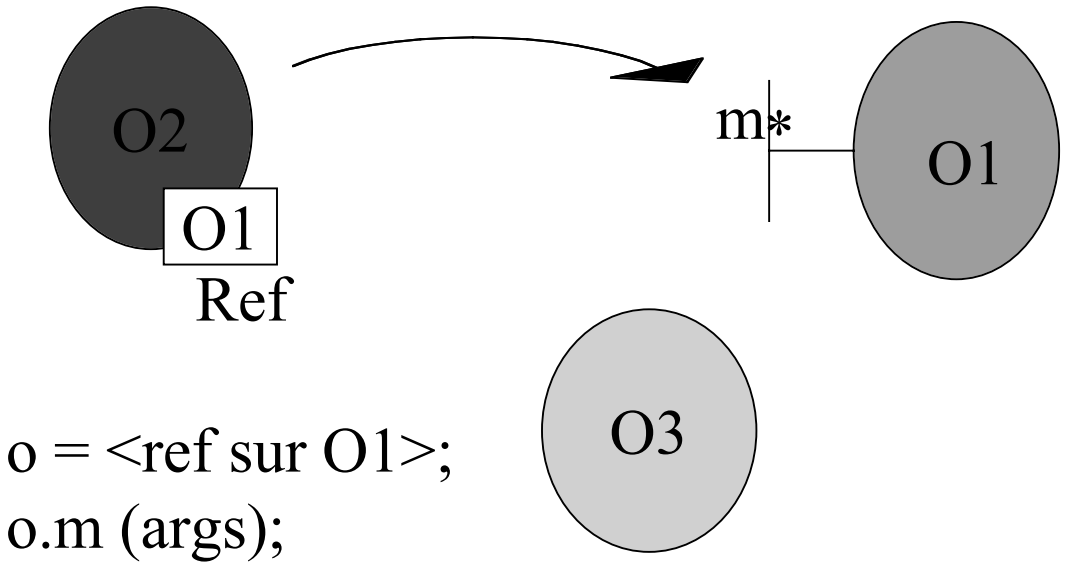
- Contexte physiquement réparti
- Pas d'horloge globale
  - l'asynchronisme rend les décisions difficiles (voire impossibles)
  - protocoles
- Nécessité d'abstraire le contexte physique
  - constructions logiques : « modèle »
  - transparences / masquages

Historique

- RPC
  - sockets
- Programmation par objets
  - RM-ODP (iso – officiel !)
  - Corba (consortium – de fait !)
  - Java RMI (succès populaire !)
  - anecd. : DCOM (+)… ?
- (Algorithmique répartie)
    - indép. des plateformes
    - utilisée dans les protocoles ou les services

Objets dans contextes répartis

- Objet, Interface, Référence
- Export, Bind (implicite or explicite, 3rd party)



ORBs

- Transparences
  - accès
  - localisation
- Contrôle
  - interfaces typées
  - services : nommage, persistence, migration, …

O.m (...args...);

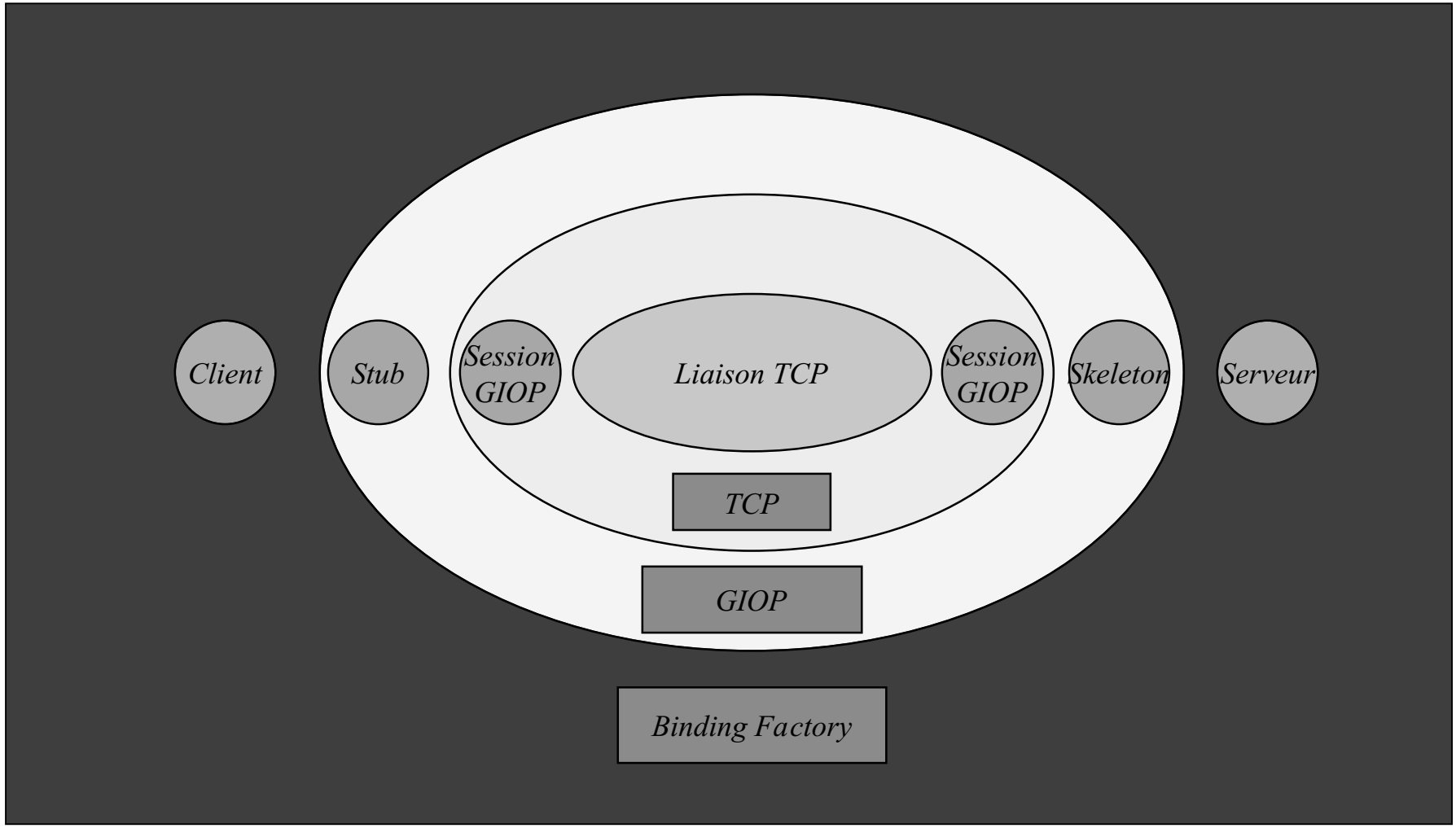
ORBs – 2

- Qqs problèmes de base
  - « annonces » et « interrogations »
    - en Corba, mot-clé « oneway »
  - parallèlisme et concurrence ?…
    - exécution séquentielle
    - appel de procédure
    - pas de gestion d'activités :
      - création de threads
      - paramétrages de « timeout »

Exportation / Liaison

- Implicite / Explicite
- Référence
  - construire : Contexte de nommage
    - NamingContext
    - référence = collections de noms
  - lier : Usine à liaisons
    - BindingFactory
    - utilise un des noms de la référence

Liaison


Client
Stub
Session GIOP
Liaison TCP
Session GIOP
Skeleton
Serveur
TCP
GIOP
Binding Factory

Jonathan

- Un ORB pour les télécommunications
  - ouvert
  - adaptable, configurable
  - basé sur le modèle ODP
    - objet, interface, référence
  - implémenté en Java
  - 2 « personnalités » : David, Jeremie

Rhum (1)

- Modèle d'Objet Réactif...
  - Papier « Reactive Objects »
  - ROM/DROM
    - appel procédural vs. message (send and forget)
    - messages <=> appels de méthodes de l'interface
    - objets exécutent des instants
      - domaine d 'exécution
    - communication synchrone
      - au plus, 1 exécution de chaque méthode par instant

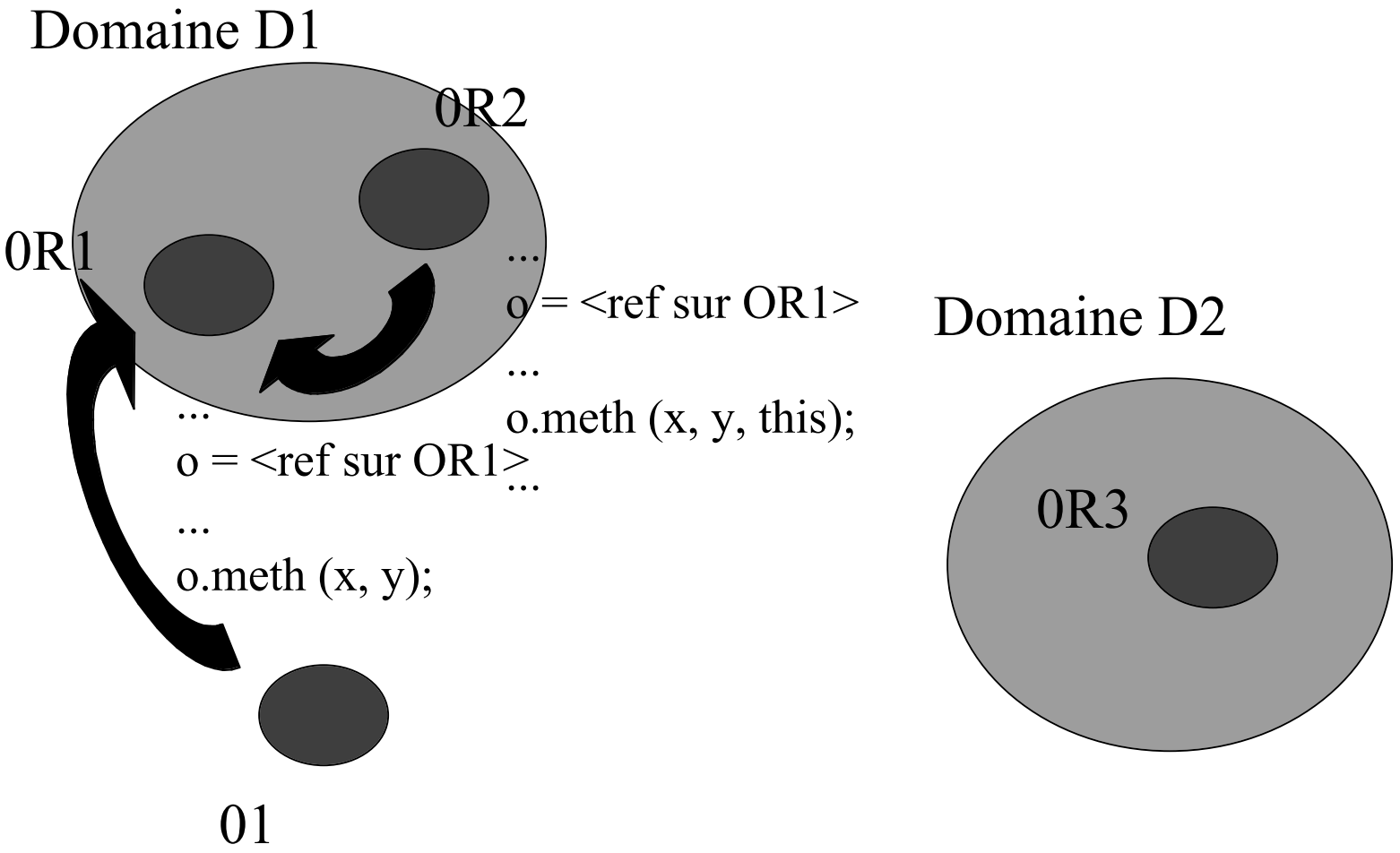
Rhum (2)

- ... appliqué à des objets Java ...
  - interface Java « normale »
    - méthodes obligatoirement « `void ... » -`
  - **`OR.m (args);`**
- ...dans une machine réactive « classique ».
  - des événements, des instants, ...
  - exécution de comportements (pas de threads).

Rhum (3)

- Domaine (machine réactive)
  - lieu d'exécution des comportements
  - lieu de diffusion des événements
  - une invocation ou un evt. de l'environnement => un événement - diffusé -
  - présence des événements :
    - intra-domaine : synchrone
    - inter-domaine (ou environnement) : asynchrone
    - possib. de groupage/synch. de plusieurs domaines

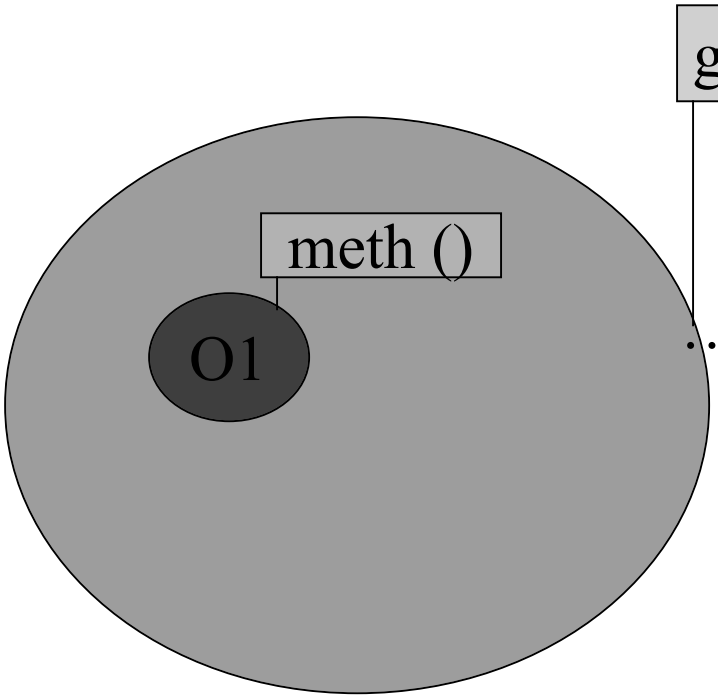
Binding


Domaine D1
0R2
0R1
...
o = <ref sur OR1>
...
o.meth (x, y, this);
...
...
o = <ref sur OR1>
...
o.meth (x, y);
01
Domaine D2
0R3

Rhum : personnalité Jonathan

- Référence d'interface réactive
  - sur un objet réactif
  - invocation uniforme
  - fournie par l 'infrastructure
    - transparence d'accès, à la localisation.
- Les domaines sont intégrés dans Jonathan
  - export : attacher un objet à un domaine
  - lier : fabriquer la chaine pour générer un événement pour l'objet dans son domaine

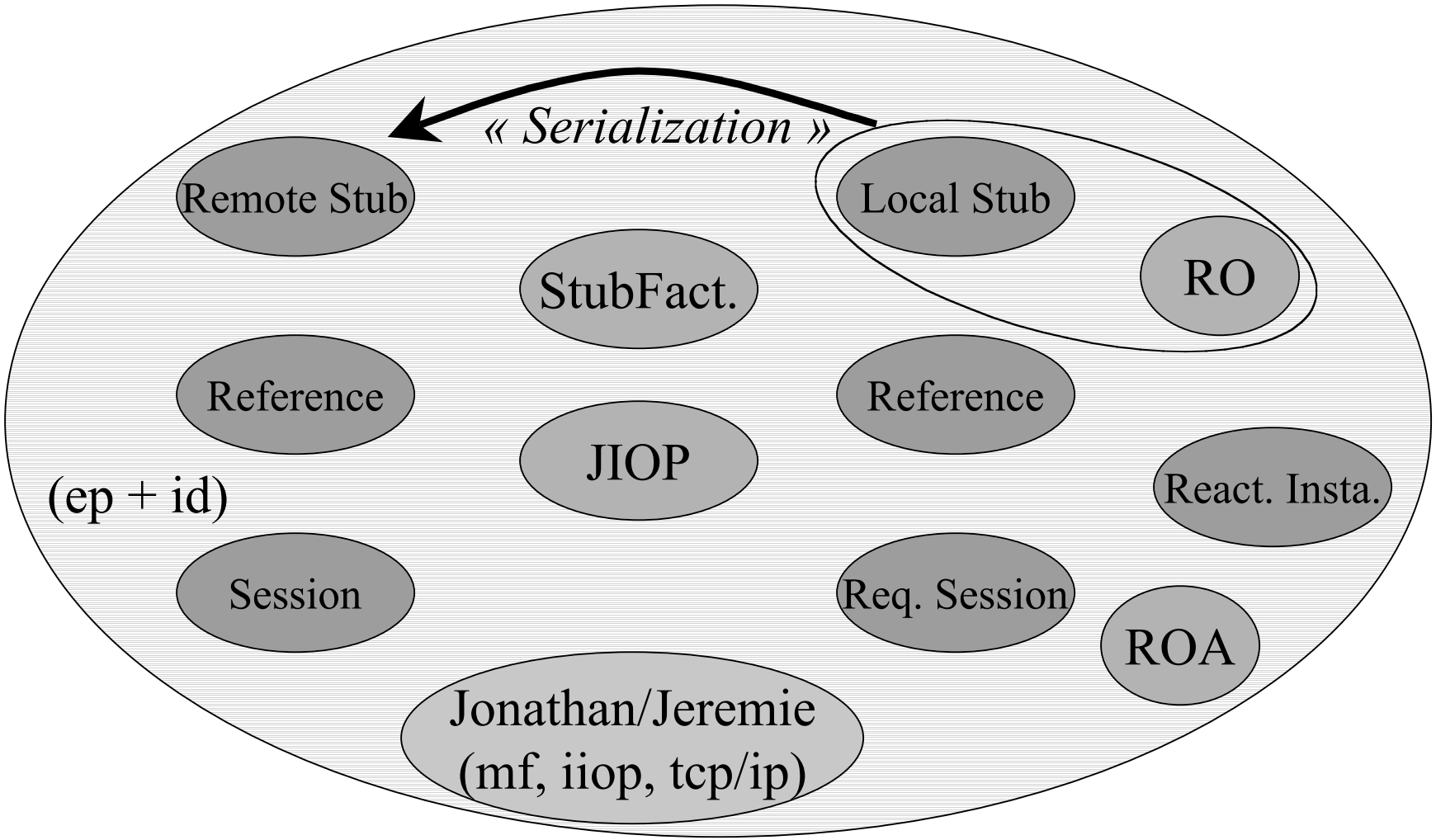
En résumé…



… *dans un contexte quelconque…* :

```
…
o = (ref sur O1)
…
o.meth ();
…
```

En détail…


« Serialization »
Remote Stub
Local Stub
StubFact.
RO
Reference
Reference
JIOP
(ep + id)
React. Insta.
Session
Req. Session
ROA
Jonathan/Jeremie
(mf, iiop, tcp/ip)

Mise en oeuvre (2)

- Spécification d'objet réactif
  - langage réactif (à la « script réactif »)
  - classe Java « passive »
- Codage de la plate-forme
  - ORB Jonathan
    - StubFactory, ObjectAdaptor, binder JIOP
  - comportement en Junior, domaine = Jr.Machine
  - parsing de la spécif en JavaCC - JJTree

Spécif. d'objet réactif (1)

- Déclare/utilise une interface
- Utilise une instance « passive »
- Spécifie un comportement
  - exécute des instants (séquence, parallèle, etc.)
  - connaît des événements (attente, garde, etc.)
    - méthodes de l 'interface
    - « channel »
    - environnement

Spécif. d'objet réactif (2)

Exemple :

```
public class Client extends RBase implements ClientI {
        ServeurI serveur;
        int res;
        public void setServ (ServeurI serv) {serveur = serv;}
        public void request (int a, int b) {
                if (serveur != null) serveur.addFor (a, b, this);
        }
        public void print (String msg) {System.out.println (msg);}
        public void receive (int resul) {print (" Client recoit :"+ resul);}
}

public class Serveur implements ServeurI {
        public void addFor (int a, int b, ClientI r) {
                r.receive (a+b);
        }
}
```

Spécif. d'objet réactif (3)

Exemple:

```
rclass RClient {

interface /* ClientI */ {
 public void setServ (Serveur serv)
 public void request (int a, int b);
 public void receive (int resul);
}

uses Client;

environment {
 ms; // signal de milliseconde
}
```

```
behavior {
 control loop {setServ (setServ::serv);}; stop end by setServ;
 ||
 loop
  await request;
  ((
   {request (request::a, request::b);};
   do
    await receive;
    {receive (receive::resul);};
   until 1000 ms actual {print ("1 sec. et pas de réponse ! ")} end
  ) || (
   stop; // pas de boucle instantanée
  ))
 end
}

}
```

L’exemple revisité

- Une solution à notre petit problème :

```
...
do
    { request (serveur1, a, b);} ;
    await Receive;
until 3000 ms
actual { request (serveur2, a, b);} end;
do
    await Receive;
    { res = Receive::val;}
until 2000 ms
actual { res = <dflt>;} end;
...
```

Remarques

- Le code est simple, clair et précis...
- Les instants doivent être de durée < 1 ms
- Le serveur doit aussi être un objet réactif
    - si « addFor » dure, ce doit être sur plusieurs instants ou dans un autre domaine
- Les arguments possibles sont :
  - d'un type de base (String, int, boolean, Object…)
  - des référerences d'objets réactifs

Bilan

- intégration d'un modèle d'activités… réussi (?)
- 2 « langages »
  - traitement de données + comportement
- repose un Orb « classique » pour les comms.:
  - configuration des protocoles
  - dépend du système sous-jacent
    - threads, sockets, etc...
  - gestion (complexe) de la concurrence
- questions ?