Middleware

Le 12 septembre 2012 , SVN-ID 231

Table des matières

1 Introduction **2**
- 1.1 Applications simples . . . 2
- 1.2 Applications type . . . 2
- 1.3 Outils réseau . . . 3

2 Générateur de PDU **3**
- 2.1 Objectifs . . . 3
- 2.2 Principe . . . 3
- 2.3 Principaux systèmes . . . 3
- 2.4 Comparaison . . . 6

3 Middleware **6**
- 3.1 Concept . . . 6
- 3.2 Types . . . 6

4 ONC RPC : Open Network Computing **7**
- 4.1 Eléments . . . 7
- 4.2 Exemple C du server financier . . . 7
- 4.3 Exemple Java/Remotetea du server financier . . . 9
- 4.4 Localisation . . . 10

5 XML RPC **10**
- 5.1 Eléments . . . 10
- 5.2 Principe . . . 11

6 RMI : Remote Method Invocation **11**
- 6.1 Eléments . . . 11
- 6.2 Exemple du server financier . . . 12

7 CORBA **13**
- 7.1 Introduction . . . 13
- 7.2 Bases . . . 14
- 7.3 Exemple de l'application finance . . . 17
- 7.4 Spécificités de programmation . . . 21
- 7.5 Services . . . 24
- 7.6 Conclusion . . . 28

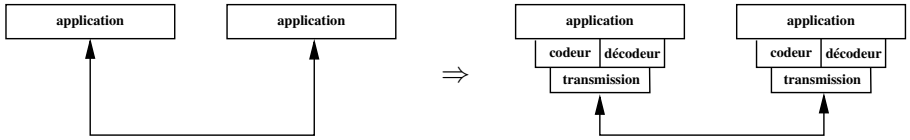
8 Travaux dirigés et pratiques **28**
- 8.1 TD sur les RPC ONC . . . 29
- 8.2 TD de RMI . . . 32
- 8.3 TP sur les RPC . . . 32
- 8.4 TD Initiation à CORBA . . . 34
- 8.5 TP Initiation à CORBA . . . 38
- 8.6 TD Services CORBA . . . 39
- 8.7 TP Service CORBA . . . 40

1 Introduction

1.1 Applications simples

1.1.1 Problème

2 entités s'échangent des données:



Codeur/décodeur petit/grand indien, ASCII/EBCDIC, flottant, alignements, langages ⟶ PDU obligatoires

Transmission supports physiques de types et de qualités variés (timeout, retransmission, code détecteur d'erreur)

⟹ nombre d'occurrences élevé: M x L [x S] avec M:type de machine ; L:langage ; S:support
exemple: 24 avec M=4 (386 32/64bit Unix/Windows), L=6 (C, C++, java, python, php, basic)

1.1.2 Exemple: Mini-serveur financier

Cet exemple servira de trame à de nombreuses partie de ce cours.

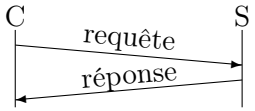
les indices: CAC40, DOW JONES, NIKKEI, ...

les services:
1) obtention de la valeur du jour d'un indice.
2) obtention des valeurs des 10 derniers jours d'un indice.

modèle réseau:

protocole:

question/réponse



services 1
- PDU_0 requête: getcur, indice
- PDU_1 reponse: status, réel

services 2
- DU_2 requête: getlast, indice
- DU_3 reponse: status, nb, {$réel_0$, $réel_1$, ...}

définition des 4 PDUs

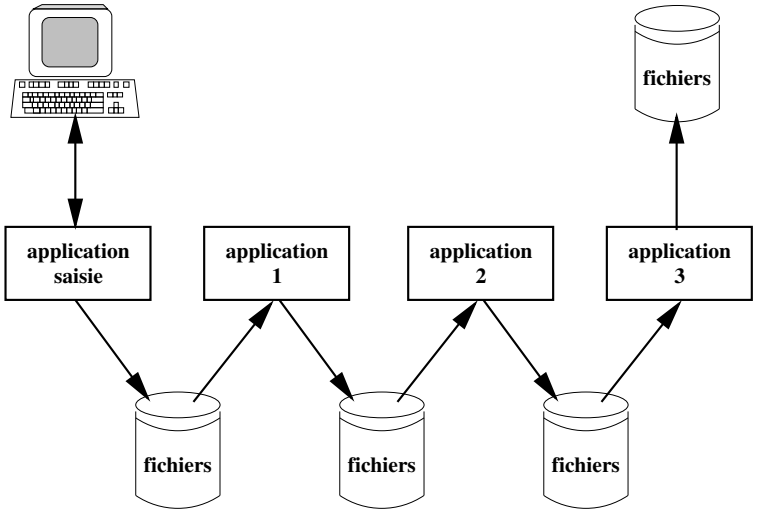
octets	0	1 2	3	....
	C	ID	indice (10)	
	L	ID	indice (10)	
	R	ID	statux (1)	valeur (4)
	M	ID	statux (1)	nb (1) | valeur (4*nb)

Réalisation des codeurs/décodeurs
1 à 3 homme-jours pour 1 langage (spécification + développement + test).

1.2 Applications type

1.2.1 Dans le passé

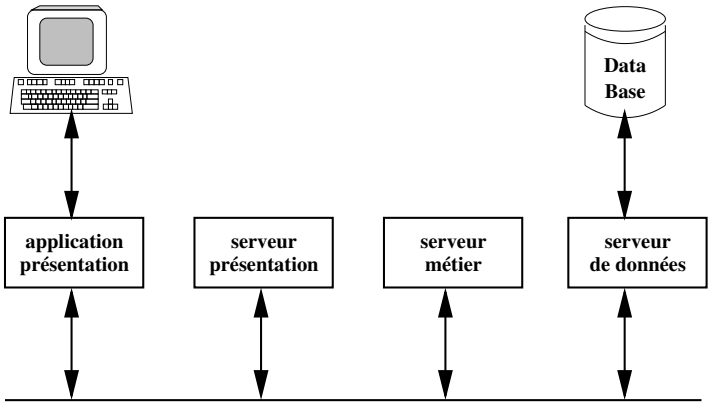
- machine centrale puissante
- applications batch séquentialisées
- communication par fichiers



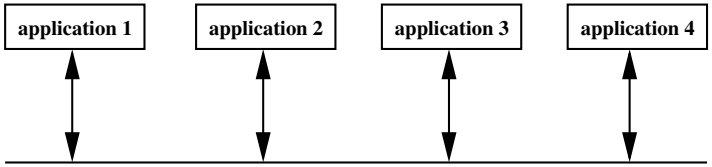
1.2.2 Aujourd'hui

- Applications 3 tiers
  - séparation des différentes tâches

- applications décentralisées
- communication par réseau



- Applications liées
  - ftp et telnet:
  - word, exel et gimp:
  - dans une entreprise: les applications de gestion comptable, du personnel (Ressource Humaine), les outils de travail, ... partagent les informations (nom, mail, grade, localisation, ...)



1.3 Outils réseau

développer de telles applications

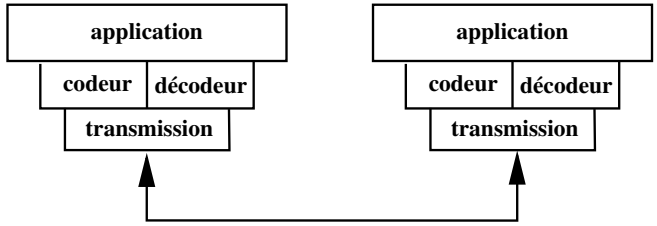
⇓

besoin d'outils masquant l'utilisation d'un réseau

Générateur de PDUs
Middleware
Annuaire

2 Générateur de PDU

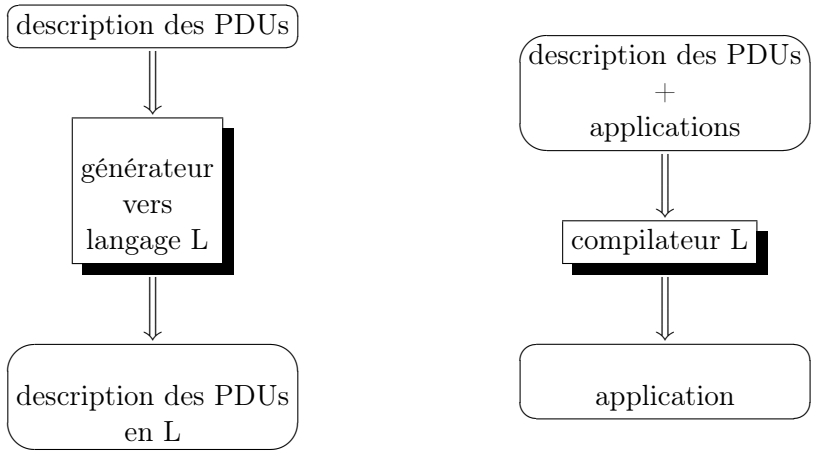
2.1 Objectifs



Générer automatiquement les codeurs et les décodeurs

- Ne diminue pas le nombre d'occurrences (**M x L**).
- Facilite l'intéropérabilité des applications entre machines et langages.

2.2 Principe



2.3 Principaux systèmes

2.3.1 XDR

Caractéristiques

- Sun Microsystem
- Fichier source décrit la structure de donnée

- types de base: enum, u/char, u/int, string, opaque ...
- types composés: tableau, liste, structure, union

– Utilisé dans NFS, NIS.

Fichiers source & générés du Mini-serveur(page 2)

```
  enum Tmesstype {
     REQUEST = 0,
     ANSWER  = 1
  };

enum Treqtype {
   GetCourant  = 0,
   GetPasse    = 1
};

struct Trequest {
   Treqtype type;
   char     indice[10];
};

struct Tanswer {
  uint    status;
  float  value<10>;
};



union Tmessage  {
  uint  ident;
  switch (Tmesstype kind) {
    case REQUEST:
      Trequest    req;
    case ANSWER:
      Tanswer     ans;
  }
};
```

```
  enum Tmesstype {
      REQUEST = 0,
      ANSWER = 1,
  };
  typedef enum Tmesstype Tmesstype;
enum Treqtype {
   GetCourant  = 0,
   GetPasse    = 1
};
typedef enum Treqtype Treqtype;
struct Trequest{
  Treqtype type;
  char     indice[10];
};
typedef struct Trequest Trequest;
struct Tanswer {
  uint status;
  struct {
     u_int value_len;
     float *value_val;
  } value;
};
typedef struct Tanswer Tanswer;
struct Tmessage {
   uint ident;
   Tmesstype kind;
   union {
      Trequest  req;
      Tanswer   ans;
   } u;
}
typedef struct Tmessage Tmessage;
```

Interface générée

```
extern  bool_t xdr_Tmesstype (XDR *, Tmesstype*);
extern  bool_t xdr_Treqtype (XDR *, Treqtype*);
extern  bool_t xdr_Trequest (XDR *, Trequest*);
extern  bool_t xdr_Tanswer (XDR *, Tanswer*);
extern  bool_t xdr_Tmessage (XDR *, Tmessage*);
```

Encodage

Tout est aligné sur 32 bits

requête CAC40 $\Longrightarrow$

0	00	00	10	00	ident =4096
4	00	00	00	00	kind=REQUEST
8	00	00	00	00	kind=GetCurr
C	C	A	C	4	
10	0	00	00	00	
14	00	00			

réponse avec 2 floats $\Longrightarrow$

0	00	00	10	00	ident =4096
4	00	00	00	01	kind=ANSWER
8	00	00	00	00	status=0
C	00	00	00	02	2
10	01	10	00	0A	float 0
14	01	22	01	EE	float 1

2.3.2 ASN.1 (Abstract Syntax Notation)

Caractéristiques

- issue de OSI
- fichier source décrit la structure de donnée
  - types de base: BOOLEAN, INTEGER, REAL, bit string, octet string, ...
  - types normalisés: PrintableString, NumericString, date, ...
  - types composés: tableau, liste, structure, union, ensemble
  - possibilité de définir un gabarit de message (généricité)
  - gestion des versions
    - définition des messages (version N)
    - $APP1(N) \longrightarrow APP2$ (N)
    - redéfinition des messages (ajout de champs) (version N+1)
    - mise à jour de APP1 • $APP1(N+1) \longrightarrow APP2$ (N)
- Utilisé dans SNMP.

Fichier source

```
mesPDUs DEFINITION EXPLICIT TAGS := BEGIN
  TrequestData ::= SEQUENCE {
    type    ENUMERATED{GetCourant,GetPasse},
    indice  PrintableString( SIZE(1..10)),
  }
  TanswerData ::= SEQUENCE {
    status   INTEGER,
    value    INTEGER (SIZE(1..10))
  }
  Tmessage ::= CLASS {
    &msgtype INTEGER UNIQUE,
    &msgdata OPTIONAL
  } WITH SYNTAX {
    type    &msgtype,
    data    &msgdata
  }
  Trequest Tmessage ::= {
    type 0,
    data TrequestData
  }
  Tanswer Tmessage ::= {
    type 1,
    data TanswerData
  }
END
```

Codage plusieurs formats d'encodage (binaire: BER, PER ; texte: XML)

Encodege BER

- tout type est codé par: "tag taille valeur"
- le tag est un identifiant de type. Pour les types de base, le tag est prédéfini, pour les autres il est soit généré, soit fixé par l'utilisateur.

tag < 128	1 octet	1xxxxxxx
tag < 2**14	2 octets	0xxxxxxx 1xxxxxxx
tag < 2**21	3 octets	0xxxxxxx 0xxxxxxx 1xxxxxxx
...	...	...

- la taille est codée comme le tag.
- contient la valeur si type de base ou constante, d'autres types "tag taille valeur" si type composé.

2.3.3 XML

Caractéristiques

- De plus en plus utilisé
- Définition des PDU: une dtd
- Emission d'un PDU: 1) construire l'arbre XML du PDU à partir de sa SD, 2) le sauver en XML, 3) l'envoyer.
- Réception d'un PDU: 1) lire une description XML, 2) construire l'arbre XML, 3) remplir sa SD à partir de l'arbre XML.
- Outils: Bibliothèques XML pour construruire des arbres XML (envoie) et rechercher des données (réception).

DTD du Mini-serveur financier(page 2)

```
  <?xml version="1.0" encoding="ISO-8859-1" ?>
<!ELEMENT pdu (request | answer)>
  <!ELEMENT request EMPTY>
    <!ATTLIST request type (courant|passe) #REQUIRED>
    <!ATTLIST request indice (CAC40|NIKKEI|...) #REQUIRED>
  <!ELEMENT answer (value*)>
    <!ATTLIST answer status (OK|ERR) #REQUIRED>
    <!ELEMENT value (#PCDATA)>
```

Encodage

requêtes

```
<?xml version="1.0"?>
<!DOCTYPE pdu SYSTEM "pdu.dtd">
<request type="courant" indice="NIKKEI"/>
```

```
<?xml version="1.0"?>
<!DOCTYPE pdu SYSTEM "pdu.dtd">
<request type="passe" indice="NIKKEI"/>
```

réponses

```
<?xml version="1.0"?>
<!DOCTYPE pdu SYSTEM "pdu.dtd">
<answer status="ERR"/>
```

```
<?xml version="1.0"?>
<!DOCTYPE pdu SYSTEM "pdu.dtd">
<answer status="OK">
 <value>2.3</value>
 <value>2.5</value>
 <value>2.4</value>
</answer>
```

2.4 Comparaison

- Puissance de description
- Efficacité du codage
- Coût CPU du codage/décodage
- Contrôle d'erreurs de programmation
- Facilité d'utilisation

3 Middleware

3.1 Concept

La figure 1 représente un middleware ou bus logiciel. Les objectifs d'un middleware sont:

- abstraction des PDUs
- abstraction des supports de communication
- abstraction de la location physique
- abstraction des langages

Il en découle les avantages:

$\Longrightarrow$ Conception plus rapide

$\Longrightarrow$ Interopérabilité des systèmes

$\Longrightarrow$ Interopérabilité des langages

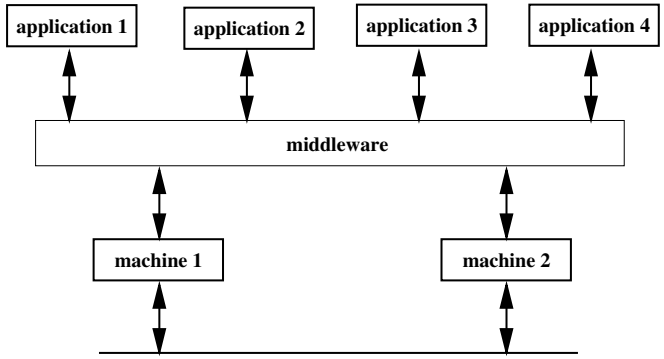


FIGURE 1: Un middleware

3.2 Types

Basés sur messages et événements

Basés sur appels de procdures (RPC, RMI)

Basés sur objet (CORBA, ISE, DCOM, DOTNET)

3.2.1 Basés sur messages et événements

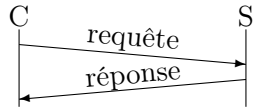
La figure 2 représente un middleware basé sur messages. Le fonctionnement général est:

- les serveurs déclarent leurs services
- les clients s'abonnent aux services
- le middleware gére la diffusion au travers de queues
- le middleware est permanent

3.2.2 Basés sur appels de procédures

Principe

protocole: question(client)/réponse(serveur)



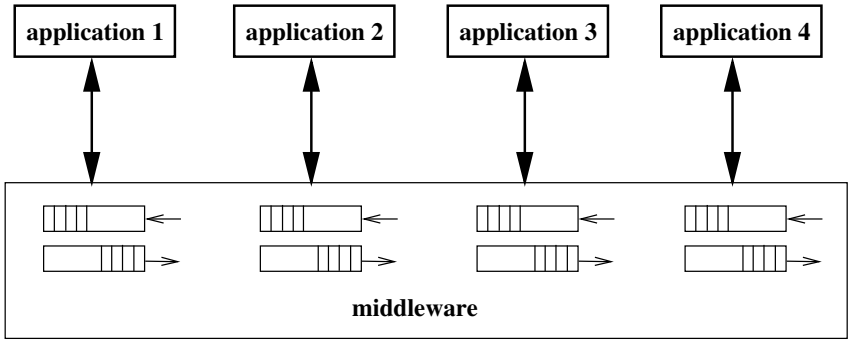


FIGURE 2: Un middleware basé sur messages et événements

coté client: une question est faite par un appel de fonction

coté serveur: le serveur exécute la fonction.

Outil:

- Outil génère la fonction à 100% coté client (souche cliente),
- Outil génére le squelette du serveur

limitations:

1. Error handling:
2. Parameters, Global variables and side-effects:
3. Performance:
4. Authentication:

3.2.3 Basés sur objet

Une extension des RPC où les clients manipulent des objets qui tournent sur des serveurs (voir section 7.1).

4 ONC RPC: Open Network Computing

Les RPC ONC (**O**pen **N**etwork **C**omputing) ou Sun RPC sont à l'origine des RPC. Elles sont décrites dans la RFC 1831. La première spécification fut faite par Sun en 1989. Elles sont encore utilisées dans de nombreux outils Unix tels NFS et NIS.

4.1 Eléments

Langage d'entrée XDR + extension

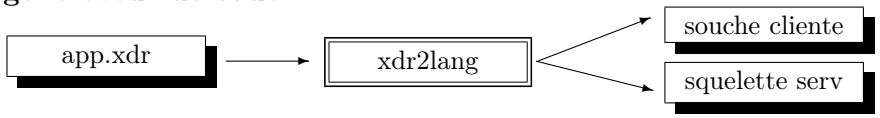
```
struct Tanswer { ... }
struct Trequest { ... }
program Exemple {
  version ExVers {
    float fonction_1(int r) = 1;
    Tanswer fonction_2(Trequest r) = 2;
  } = 3;
} = 0x20002000;
```

Identifiant RPC 32 bits, appellé port RPC

0-1fffffff	defined by rpc@sun.com
20000000-3fffffff	defined by user
40000000-5fffffff	transient (serveur dynamique)
60000000- ffffffff	reserved

Serveur d'enregistrement Croise les référence entre les ports UDP/TCP et RPC.

Un générateur de code



Génération d'un serveur

- Ecrire le code des fonctions RPC.
- Le compiler avec le squelette du serveur.

Génération d'un client

- Ecrire un programme principal en utilsant les fonctions RPC.
- Le compiler avec la souche cliente.

4.2 Exemple C du server financier

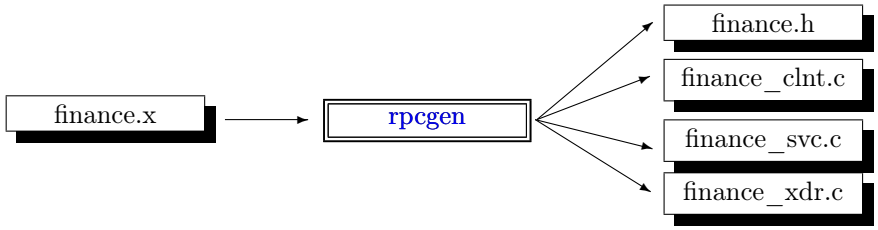
4.2.1 Source XDR & rpcgen

Fichier d'entrée: finance.x

```
/* finance.x: source rpc */
struct Tanswer { int status; float value<10>; }
program miniserv {
  version VERS {
```

```
    Tanswer getCurr(string indice) = 1;
    Tanswer getLast(string indice) = 2;
  } = 2;
} = 0x20002000 ;
```

rpcgen



finance.h header commun à la souche cliente et au squelette du serveur

finance_clnt.c la souche cliente

finance_svc.c le squelette du serveur

finance_xdr.c les codeurs/décodeurs des PDUs

finance.h

```
struct Tanswer {
  int status;
  struct {
    u_int value_len;
    float *value_val;
  } value;
};
...
extern  bool_t xdr_Tanswer (XDR *, Tanswer*);
...
extern Tanswer* getCurr_2(char**, CLIENT*);
extern Tanswer* getLast_2(char**, CLIENT*);
extern Tanswer* getCurr_2_svc(char**, struct svc_req*);
extern Tanswer* getLast_2_svc(char**, struct svc_req*);
```

4.2.2 Implantation d'un serveur

"finance_svc.c" contient le squelette du serveur complet. Ce qui manque c'est ce que font les fonctions RPC (getcurr_2_svc et getlast_2_svc). Il suffit de les implanter dans un fichier.

Description du serveur (serveur.c)

```
#include "finance.h"
Tanswer* getcurr_2_svc(char **indice, struct svc_req *cli)
{
  static Tanswer  a;
  static float    ft[1]; a.value.value_val=ft;
  if ( strcmp(*indice,"CAC40")==0 ) {
    a.status=0; a.value.value_len=1; a.value.value_val[0] = 3500;
  } else if ( strcmp(*indice,"NIKKEI")==0 ) {
    a.status=0; a.value.value_len=1; a.value.value_val[0] = 9500;
  } else {
    a.status=-1; a.value.value_len=0; a.value.value_val = 0;
  }
  return &a;
}
Tanswer* getlast_2_svc(char **indice, struct svc_req *cli)
{
  static Tanswer  a;
  ...
  return &a;
}
```

Génération & lancement

```
⟶ cc -o serveur finance_svc.c serveur.c
⟶ ./serveur
```

4.2.3 Implantation d'un client

"finance_clnt.c" contient les fonctions RPC (getcurr_2 et getlast_2).On peut les utiliser à volonté.

Description du client (client.c)

```
#include "finance.h"
int main (int argc, char *argv[])
{
  // vérification des paramètres
  if (argc < 3) {
    printf ("usage: %s serveur indice\n", argv[0]);
    exit (1);
  }
  // initialisation du RPC
  CLIENT *serv;
  CLIENT* serv=clnt_create (SERVEUR,miniserv,VERS,"udp");
  if ( (serv=clnt_create (argv[1],miniserv,VERS,"udp"))==0 ) {
    clnt_pcreateerror (argv[1]);
    exit (1);
  }
  // appel RPC
  Tanswer* a;
  if ( (a=getcurr_2(&argv[2], serv))==0 ) {
    clnt_perror (serv, "call failed");
    exit(1);
```

```
  }
  // traitement du résultat de l'appel
  if (a->status<0) {
    fprintf(stderr,"indice inconnu: %s\n",argv[2]);
  } else {
    fprintf(stderr,"valeur de %s: %7.2f\n",
        argv[2],a->value.value_val[0]);
  }
  return 0;
}
```

Génération & lancement

```
⟶ cc -o client finance_clnt.c client.c
⟶ ./client host CAC40
```

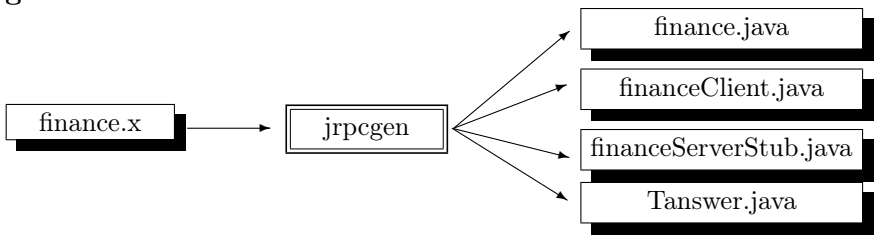
4.3 Exemple Java/Remotetea du server financier

On utilise ici l'implantation Remote Tea (URL: http://remotetea.sourceforge.net/) des RPC ONC.

4.3.1 Source XDR & jrpcgen

Fichier d'entrée: finance.x le même que pour le langage C (voir section 4.2.1 page 7).

jrpcgen



finance.java les constantes du XDR.

financeClient.java la souche cliente

financeServerStub.java le squelette du serveur

Tanswer.java les codeurs/décodeurs des PDUs

Tanswer.java

```
public class Tanswer implements XdrAble {
    public int status;
    public float [] value;
    public Tanswer() { ... }
    ...
}
```

financeClient.java

```
public class financeClient extends OncRpcClientStub {
    public financeClient(InetAddress host, int protocol)
           throws OncRpcException, IOException {
        super(host, finance.miniserv, 2, 0, protocol);
    }
    public Tanswer getCurr_2(String indice)
           throws OncRpcException, IOException { ... }
    public Tanswer getLast_2(String indice)
           throws OncRpcException, IOException { ... }
    ...
}
```

4.3.2 Implantation d'un client

La classe "financeClient" contient les membres RPC (getCurr_2 et getLast_2). Dès qu'un objet est créé, on peut les utiliser à volonté.

Description du client (client.java)

```
import org.acplt.oncrpc.*;
import java.io.IOException;
import java.net.InetAddress;
class client {
 public static void main(String argv[]) {
  // vérification des paramètres
  if ( argv.length!=2 ) {
    System.err.println("usage: java client host ind");
    System.exit(1);
  }
  try {
   // initialisation du RPC
   InetAddress host = InetAddress.getAllByName(argv[0])[0];
   financeClient server = new
       financeClient(host, OncRpcProtocols.ONCRPC_TCP);
   // appel RPC
   Tanswer a=server.getCurr_2(argv[1]);
   // traitement du résultat de l'appel
   if (a.status<0)
    System.out.println("java: status:" + a.status
                       + " unknown:" + argv[1]);
   ...
  } catch (Exception e) {
   ...
}}};
```

Génération & lancement

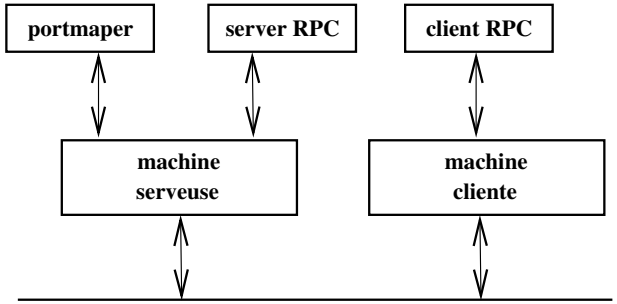
⟶ `javac finance.javac financeClient.java client.java`
⟶ `java client host CAC40`

4.4 Localisation

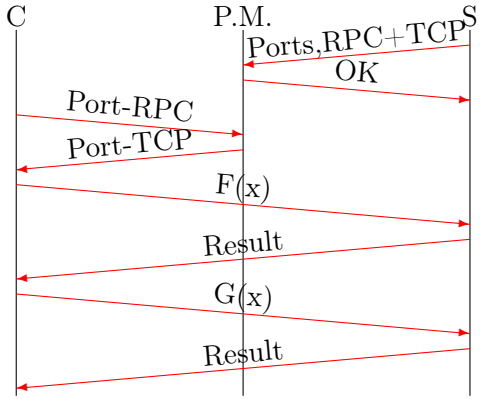
Un client à besoin de connaître:

1. l'addresse réseau de la machine du serveur.
2. l'identifiant du RPC (port RPC).

⟹ Le rôle du port mapper (portmap ou rpcbind sous Unix) est de cross-référencer les ports RPC avec les ports TCP ou IDP. le port mapper et le serveur doivent être sur la même machine.



MSC



5 XML RPC

Les XML RPC font parties des WEB services. Elles datent du début des années 2000.

5.1 Eléments

Protocole & identifiant

Protocole de transport: HTTP (post pour question)

Identifiant RPC serveur port nom

PDU Ils sont décrits en XML sans DTD.
⟹ pas de contrôle

PDU

Quelques balises XML pour définir:

types de base entier (32 bits), chaîne de caractères, réels, date, ...

```
<i4>41</i4>
<string>bonjour monsieur</string>
```

types composés tableau, structure.

PDU requête nom de la méthode à appeler et ses paramètres

PDU réponse paramètre retourné

PDU d'erreur

Introspection

Un serveur XML RPC doit founir au travers de méthodes dont les noms sont prédéfinis les informations suivantes:

1. La liste des fonctions RPC.
2. Pour chaque fonction la structure du paramètre de son PDU requête.
3. Pour chaque fonction la structure du paramètre de son PDU réponse.
4. Pour chaque fonction une documentation.

5.2 Principe

Les outils XML RPC fournissent des bibliothèques dont les services sont définis ci-dessous:

1. Construire un arbre XML.
2. Extraire des valeurs d'un arbre XML.

Création d'un serveur

- Ecrire les fonctions RPC: Elles ont un argument qui est un arbre XML et elles renvoient un arbre XML.
- Enregistrer les fonctions RPC avec leurs aides et leurs signatures.
- Lire une requête, appeler une fonction qui aiguille sur la bonne fonction ou renvoie un arbre XML erreur.

Création d'un client

langages interprétés

- Se connecter au serveur $\Longrightarrow$ crée la souche cliente par introspection.
- Construire le XML du paramètre de la requête.
- Appeler la fonction RPC avec ce XML et récupérer le XML de la réponse.
- Extraire du XML renvoyé les valeurs.

langages compilés Très lourd et aucun contrôle.

6 RMI: Remote Method Invocation

6.1 Eléments

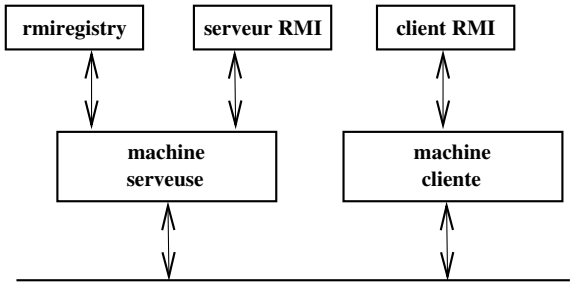
Langage d'entrée Pas de langage spécifique mais une Interface Java.

```
import java.rmi.*;
public interface exemple extends java.rmi.Remote
{
  public double add(double x, double y)
    throws RemoteException;
  public int log2(int x) throws RemoteException;
}
```

- Calcule la valeur retournée en fonction des arguments.
- Types des arguments et de la valeur retournée doivent être sérialisables.

Localisation

- Identifiant: une chaîne de caractères
- Rmiregistry: un serveur de localisation RMI tourne sur la machine du serveur.



$\Rightarrow$ les serveurs s'enregistrent sous un identifiant

$\Rightarrow$ il génère la souches clientes des serveurs

$\Rightarrow$ les clients l'interrogent pour récupérer la souche cliente

Génération d'un serveur

- Définir une classe implantant l'interface.
- Un main
  - créant une instance de la classe
  - s'enregistrant sur le serveur RMI.
  - entrant dans la boucle d'écoute

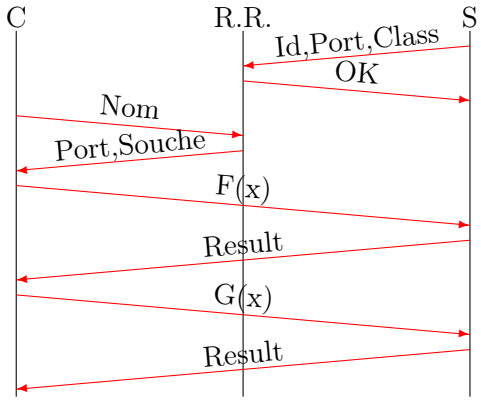
Génération d'un client

- Se connecter au serveur de localisation

  $\Rightarrow$ les parametre réseau du serveur

  $\Rightarrow$ la souche cliente
- Utiliser souche cliente

MSC



6.2 Exemple du server financier

6.2.1 Interface

finance_ansver.java On a besoin du type réponse.

```
class finance_answer
    implements java.io.Serializable {
  public finance_answer() { values= new double[10]; }
  public int status;
  public double [] values;
}
```

finance.java L'interface avec les 2 fonctions du serveur.

```
import java.rmi.*;

public interface finance extends java.rmi.Remote {
  public finance_answer getcurr(String indice)
                throws RemoteException;
  public finance_answer getlast(String indice)
                throws RemoteException;
}
```

6.2.2 Implantation d'un serveur

finance_export.java La classe serveur.

```
import java.rmi.*;
import java.rmi.server.*;

public class finance_export extends UnicastRemoteObject
                            implements finance {
  public finance_export() throws RemoteException { }

  public finance_answer
         getcurr(String indice) throws RemoteException {
    finance_answer a = new finance_answer();
    if (indice.equals("CAC40") ) {
        a.status = 0;
        a.values[0] = 3900;
    } else if ( indice.equals("NIKKEI") ) {
        ...
    } else {
        a.status = -1;
    }
    return a;
  }

  public finance_answer
         getlast(String indice) throws RemoteException {
    finance_answer a = new finance_answer();
    ...
    return a;
  }
}
```

finance_server.java Le programme principal qui crée un serveur et l'enregistre.

```
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;

public class finance_server {

  public static void main(String[] args) throws Exception {
    finance_export o = new finance_export();
    Naming.rebind("myobj", o);
  }
}
```

Génération

```
⟶ javac finance_answer.java
⟶ javac finance.java
⟶ javac finance_export.java
⟶ javac finance_server.java
```

Lancement

```
⟶ rmiregistry &
⟶ java finance_server
```

rmiregistry doit pouvoir accéder à finance.class.

6.2.3 Implantation d'un client

finance_client.java Un client qui fait une requête getcurr sur un objet (dont l'identifiant est le second argument) tournant sur la machine spécifiée par le premier argument.

```
import java.rmi.*;
import java.rmi.server.*;
public class finance_client {
  public static void main(String[] args) throws Exception {
    // verification des arguments
    if (args.length!=2) {
      System.err.println("usage: java client <ser> <obj>");
      System.exit(2);
    }
    // obtention d'un serveur
    String  url = "rmi://" + args[0] + "/" + args[1];
    finance f = (finance) Naming.lookup(url);
    // requête RPC
    finance_answer a;
    a = f.getcurr("CAC40");
    // utilisation de la réponse
    System.out.println("sta= " + a.status);
    System.out.println("val= " + a.values[0]);
   }
}
```

Génération

```
⟶ javac finance_answer.java
⟶ javac finance.java
⟶ javac finance_client.java
```

Lancement

```
⟶ java finance_client nom-serveur myobj
```

7 CORBA

7.1 Introduction

7.1.1 Vue programmation

```
class personne {
  string nom();
};
class parent : personne {
  personne* enfant(int i);
};
class enfant : personne {
  parent* pere();
  parent* mere();
};
```

```
x= new ...  P1     | P2    | P3
y= new ...  x= new |       |
z= new ...         |y= new|
                   |       | z=new
```

```
enfant*   penf;
penf= x;

enfant*   penf;
penf= ...

personne* p;
personne* pere;
pere= penf->pere();
i=0;
while (pere->enfant(i)) {
  p= pere->enfant(i++);
  printf("%s\n",p->nom());
}
```

- Serveur crée des objets utilisables par les clients.
- Client utilise les objets fournis par les serveurs.
- Limitations Les objets sont des interfaces, pas d'attribut donnée d'où:

OO local	OO middleware
`obj->v1= 0;`	interdit
`obj->ptr= obj2;`	interdit

- Principe
  création de serveur d'interface
  création de classe cliente
- Middleware
  les paramètres réseaux sont complètement masqués
  multi-langages: clients et serveurs peuvent être écrits dans des langages différents.

7.1.2 Environnement

Définition d'un ensemble d'utilitaires pour aider le développement et la programmation d'applications réparties.

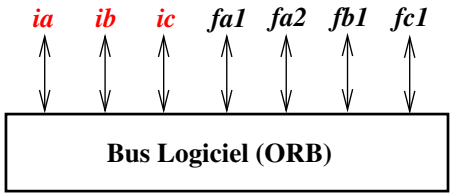
- Recherche d'un objet: page blanche, page jaune.

OO local	OO corba
`class X* obj;`	`class X* obj;`
`obj= new X();`	`obj= quelques_lignes_corba`

- Moteur transactionnel:
- Gestionnaire de persistence:
- Gestionnaire de propriétés:
- ...

$\Longrightarrow$ Ces utilitaires sont appelés les services.
$\Longrightarrow$ Ces services sont des objets CORBA.

7.1.3 ORB: Object Request Broker



2 types d'objets sur le bus

ia ib ic objets réels (instances)

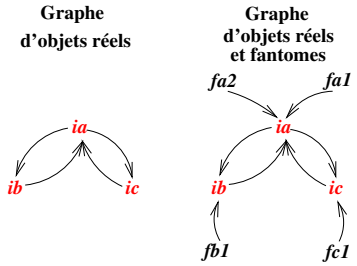
- ils peuvent être dans le même processus ou dans plusieurs.
- ces processus sont dit "serveur"
- les données de l'objet sont stockées dans le processus

fa1 fa2 fb1 fc1 référence réseau à ia, ib, ic (*fantomes*)

- ils peuvent être dans le même processus ou dans plusieurs.
- ces processus sont dits "clients".
- ces objets *fantomes* n'ont pas de données, ce sont des interfaces fonctionnelles.

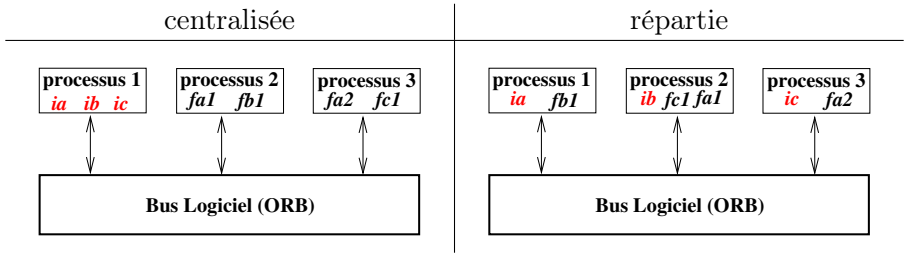
Graphe d'objets

les objets fantômes sont des références sur les objets réels.



Répartition des objets

toutes les répartitions des objets sur les processus sont posibles.



7.2 Bases

7.2.1 Eléments CORBA

Conceptuel

- IDL: Interface Description Langage
- IOR: Interface Object Reference

Logiciel

- compilateurs d'IDL vers différents langages
- bibliothèques d'objets contenant les souches clientes des utilitaires CORBA
- bibliothèques des serveurs des utilitaires CORBA

Génération d'une application CORBA

1. Ecrire la description IDL.
2. La compiler vers un langage.

3. Ecrire le client en utilisant les classes de la souche cliente, puis compiler le tout pour obtenir le client.
4. Compléter le squelette du serveur puis le compiler avec la squelette du serveur.

7.2.2 Un exemple

Etapes de conception

1) Description IDL, 2) Génération de la souche cliente et du squelette du serveur, 3) Implantation d'un serveur, 4) Implantation d'un client, 5) Mise en oeuvre.

Description IDL

Fichier: name.idl

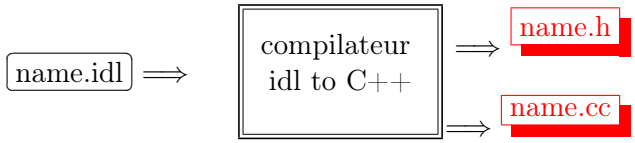
```
interface personne {
  string nom();
};
```

Principales caractéristiques de l'IDL

- héritage simple et multiple
- pas de redéfinition
- pas de surcharge
- pas de conflit de nom
- pas de membre données

Génération de la souche cliente et du squelette du serveur

Compilation de l'IDL (vers $C++$)



Souche cliente

```
class personne : public CORBA::Object {
  ...
  char* nom();
  ...
}
```

Squelette du serveur

```
class POA_personne :
    public PortableServer::StaticImplementation {
  ...
  virtual char* nom() = 0;
  ...
}
```

name.cc

- Implantation de la souche cliente (classe personne)
- Implantation du squelette du serveur (classe POA_personne)

Implantation d'un serveur

Algorithme général

1. Définir une classe implantant les fonctions abstraites du squelette du serveur
2. Initialisation d'un ORB serveur
3. Créer des objets
4. Mettre les IOR des objets créés quelque part
5. Lancer le serveur

Code (fichier: ivan.cc)

```
#include <CORBA.h>
#include "name.h"

class ivan : public POA_personne {
public:
    ivan() {}
    char* nom() { return strdup("ivan"); }
};

int main(int argc, char** argv)
{
    CORBA::ORB_var orb = CORBA::ORB_init(argc,argv);
    PortableServer::POA_var poa;
```

```
    CORBA::Object* obj;
    obj = orb->resolve_initial_references("RootPOA");
    poa= PortableServer::POA::_narrow(obj);
    ivan* i = new ivan();
    CORBA::Object* fi = i->_this();
    char* ior = orb->object_to_string(fi);
    printf("%s\n",ior);

    fprintf(stderr,"Serveur pret\n");
    poa->the_POAManager()->activate();
    orb->run();
    return 0;
}
```

Implantation d'un client

Algorithme général

1. Initialisation d'un ORB client
2. Créer des objets fantômes
3. Utiliser les objets fantômes

Code C++ (fichier: client.cc)

```
#include <CORBA.h>
#include "name.h"

int main(int argc, char** argv)
{
  CORBA::ORB_var orb= CORBA::ORB_init(argc,argv);
  if (argc!=2) {
    fprintf(stderr,"usage: %s ior\n",argv[0]);
    exit(1);
  }
  char* ior = argv[1];
  CORBA::Object* o = orb->string_to_object(ior);
  personne* f = personne::_narrow(o);
  if ( f == 0 ) { ... exit(1); }
  if ( f == 0 ) {
    fprintf(stderr,"%s: invalid IOR\n",argv[0],ior);
    exit(1);
  }
  char* n = f->nom();
  printf("nom=%s\n",n);
  return 0;
}
```

Code Java (fichier: client.java)

```
import org.omg.CORBA.*;
public class client {
public static void main(String args[]) {
  try{
    // initialisation de l'ORB
    ORB orb = ORB.init(args, null);
    // creation d'un fantome
    String ior = args[0];
    org.omg.CORBA.Object o = orb.string_to_object(ior);
    personne f = personneHelper.narrow(o);
    // utilisation du fantome
    String n = f.nom();
    System.out.println("nom=" + n);
  } catch (Exception e) {
    System.out.println("ERROR : " + e) ;
    e.printStackTrace(System.out);
  }
 }
}
```

Mise en oeuvre

serveur: (name.idl, ivan.cc)

```
⟶ idl name.idl && ls
⟶ name.cc name.h name.idl ivan.cc
⟶ g++ -I ... ivan.cc name.cc -L ... -l... -l... -o serv
⟶ ./serv
IOR:0100000011000000494....
Serveur pret
```

client: (name.idl client.cc)

```
⟶ idl name.idl && ls
name.idl client.cc name.cc name.h
⟶ g++ -I ... client.cc name.cc -L ... -l... -l... -o cli
⟶ ./cli IOR:0100000011000000494....
nom=ivan
⟶
```

client: (name.idl client.java)

```
⟶ idlj name.idl && ls
name.idl client.java _personneStub.java personne.java ...
⟶ javac client.java
⟶ java client IOR:0100000011000000494....
nom=van
⟶
```

7.2.3 Fonctions principales

ior $\Longleftrightarrow$ objet-fantôme

```
CORBA::Object* f = orb->string_to_object(ior);
const char* ior = object_to_string(f);
```

objet-fantôme général $\Longrightarrow$ objet-fantôme spécialisé

```
CORBA::Object* f;
XXX_client* fs = XXX_client::_narrow(f);
```

objet-serveur $\Longrightarrow$ objet-fantôme

```
XXX_seveur* os;
XXX_client* fs = os->_this();
CORBA::Object* f = os->_this();
```

et par héritage

```
XXX_client* fs;
CORBA::Object* f = fs;
```

7.3 Exemple de l'application finance

7.3.1 Version centralisée

IDL

Code: (finnance.idl)

```
module MF {

  struct Tanswer {
      long             status;
      sequence<double> values;
  };

  interface finance {
      Tanswer getcurr(in string indice);
      Tanswer getlast(in string indice);
  };
};
```

Un peu plus d'IDL

- struct: Définition de types. CORBA génère les codeurs et les décodeurs pour les transporter.
- sequence: CORBA définit des containeurs génériques. textcolorbluesequence est un tableau dynamique.
- module: encapsulation, similaire au package Java et namespase C++.

Implantation d'un serveur

classe squelette

```
namespace POA_MF {

class finance :
  virtual public PortableServer::StaticImplementation {
  ...
  virtual ::MF::Tanswer* getcurr( const char* indice ) = 0;
  virtual ::MF::Tanswer* getlast( const char* indice ) = 0;
};
} // namespace
```

Une implantation de la classe squelette

```
#include <CORBA.h>
#include "finance.h"

/*** Definition of the exported class                    ***/

class financeImp : public POA_MF::finance {
public:
  financeImp() {}
  ::MF::Tanswer* getcurr( const char* indice ) {
    ::MF::Tanswer* a = new Tanswer;
    if ( strcmp(*indice,"CAC40")==0 ) {
      a->status=0;
      a->values.length(1); a->values[0] = 3500;
    } else if ( strcmp(*indice,"NIKKEI")==0 ) {
      ...
    } else {
      a.status=-1;
    }
    return a;
  }
  ::MF::Tanswer* getlast( const char* indice ) {
  }
};
```

Le programme principal

```
int main(int argc, char** argv)
{
    CORBA::ORB_var orb = CORBA::ORB_init(argc,argv);
    PortableServer::POA_var poa;
    CORBA::Object* obj;
    obj = orb->resolve_initial_references("RootPOA");
    poa= PortableServer::POA::_narrow(obj);

    financeImp* fi = new financeImp;
    CORBA::Object* fi = i->_this();
    char* ior = orb->object_to_string(fi);
    printf("%s\n",ior);
```

```
    fprintf(stderr,"Serveur pret\n");
    poa->the_POAManager()->activate();
    orb->run();
    return 0;
}
```

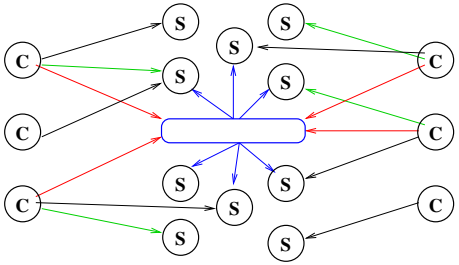
Qualités de cette version

Cette version est correcte et fonctionne mais:

- Le serveur central ⟶ pas de répartition de la charge.
- Ajout d'indices ⟶ arret du serveur.
- Evolution du logiciel ⟶ plus en plus complexe ⟶ de moins en moins modifiable
- Le serveur central ⟹ géré par 1 société ⟹ difficulté de regrouper toutes les sociétés du domaine.

bonne programation descendante quasi équivalente à la version RPC.

7.3.2 Une belle application répartie



Elements

- des millions de serveurs
- des 100 millions de clients
- une glue: HTTP et HTML
- quelques GPS: google, ...

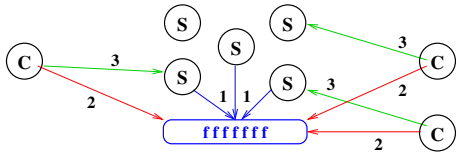
Avantages

- conception décentralisée (pas de maître d'oeuvre ou d'ouvrage).
- pas de contraines de développement (HTML, PHP, PYTHON, CGI, APPLET, ...).
- ajout de briques aisées.

conception ascendante par agrégation d'éléments.

7.3.3 Version répartie

Principe



1. Un GPS
2. Les seveurs s'enregistrent dans le GPS (1).
3. Les clients demandent les serveus au GPS (2).
4. Les clients peuvent communiquer avec les serveurs (3).

Séparation complète de la localisation et du métier.

IDL

Code: (finnance.idl)

```
typedef sequence<double> DoubleArray;

interface finance {
  long getcurr(out double value);
  long getlast(out DoubleArray values);
};

interface gps {
  finance get(in string indice);
  void    add(in string indice, in finance objet);
};
```

class finance plus besoin d'indice puisque l'objet ne s'occupe que d'un seul indice.

class gps une méthode d'enregistrement et un méthode de résolution

Implantation d'un serveur CAC40

Classe squelette (fichier: finance.h généré)

```
class POA_finance :
    virtual public PortableServer::StaticImplementation {
  ...
  virtual CORBA::Long getcurr( CORBA::Double_out value ) = 0;
  virtual CORBA::Long getlast( ::DoubleArray_out values ) = 0;
};
```

Classe serveur (fichier: cac40.cc)

```
#include <CORBA.h>
#include "finance.h"

class Cac40 : public POA_finance {
  public:

  CORBA::Long getcurr(CORBA::Double& value) {
    value = 2000;
    return 0;
  }

  CORBA::Long getlast(::DoubleArray_out values) {
    values = new DoubleArray;
    values->length(3);
    (*values)[0]=2001; (*values)[1]=2002; (*values)[2]=2003;
    return 0;
  }
};
```

Est reduite au métier.

Souche cliente du GPS (fichier: finance.h généré)

```
class gps : virtual public CORBA::Object {
  ...
  ::finance_ptr get( const char* indice );
  void          add( const char* indice, ::finance_ptr objet );
  ...
};
```

Programme principal (fichier: cac40.cc)

```
int main(int argc, char** argv)
{
  CORBA::Object* f;

  // initialisation de l'orb (serveur)
  CORBA::ORB_var orb= CORBA::ORB_init(argc,argv);
  f = orb->resolve_initial_references("RootPOA");
  PortableServer::POA_var poa= PortableServer::POA::_narrow(f);

  // vérification des arguments
  if ( argc!=2 ) {
    fprintf(stderr,"usage: %s ior\n",argv[0]);
    exit(1);
  }

  // récupération d'un fantome du GPS
  char* ior=argv[1];
  f= ........................
  gps* fpgs = ................

  // mise sur l'orb d'un objet.
  Cac40* icac40 = new Cac40();

  // publication de l'ior de l'objet
  ..................................

  // lancement du serveur
  printf("CAC40 object added\n");
  poa->the_POAManager()->activate();
  orb->run();
  return 0;
}
```

Implantation d'un serveur GPS

Classe squelette (fichier: finance.h généré)

```
class POA_gps :
    virtual public PortableServer::StaticImplementation {
  ...
  virtual ::finance_ptr get( const char* indice ) = 0;
  virtual void add( const char* indice, ::finance_ptr objet ) = 0;
  ...
};
```

Classe serveur (fichier: gps.cc)

```
1 #include <CORBA.h>
2 #include "finance.h"

3 class gpsImpl : public POA_gps {
4 private:
5   int lookup(const char* indice) { ...  }
```

```
 6   int           nb;
 7   char*         name[100];
 8   finance_ptr   objets[100];
 9 public:
10   gpsImpl() ...
11   virtual finance_ptr get(const char* indice)
12   {
13     int i = lookup(indice);
14     ...
15                 ...
16   }
17   virtual void add(const char* indice, finance_ptr obj)
18   {
19     int i = lookup(indice);
20     ...
21       ...
22       ...
23     ...
24     ...
25   }
26 };
```

rappel des étapes de conception d'une classe

1. Spécification des services de la classe (déjà fait ici: getcurr & getclass en grisé).
2. Définition d'un structure de données et d'utilitaires pour implanter les services (lignes 5 à 8).
3. Ecriture du/des constructeur/s dont la fonction est l'initialisation de la structure de données (ligne 10).
4. Ecriture des services (lignes 11 à 25).

Programme principal (fichier: gps.cc)

```
int main(int argc, char** argv)
{
  CORBA::Object* f;

  // initialisation de l'orb (serveur)
  CORBA::ORB_var orb= CORBA::ORB_init(argc,argv);
  PortableServer::POA_var poa= ...

  // vérification des arguments
  if ( argc!=1 ) {
    fprintf(stderr,"usage: %s\n",argv[0]);
    exit(1);
  }

  // mise sur l'orb d'un objet.
  gpsImpl* igps = new gpsImpl();

  // publication de l'ior de l'objet
  f = igps->_this();
  printf("%s\n", orb->object_to_string(f) );

  // lancement du serveur
  printf("GPS objet démarré\n");
  poa->the_POAManager()->activate();
  orb->run();

  return 0;
}
```

Implantation C++ d'un client

Souche cliente de finance (fichier: finance.h généré)

```
class finance : virtual public CORBA::Object {
  ...
  CORBA::Long getcurr( CORBA::Double_out value );
  CORBA::Long getlast( ::DoubleArray_out values );
  ...
};
```

Programme principal (fichier: client.cc)

```
#include <CORBA.h>
#include "finance.h"

int main(int argc, char** argv)
{
  CORBA::Object* f;

  // initialisation de l'orb (client)
  CORBA::ORB_var orb= CORBA::ORB_init(argc,argv);

  // vérification des arguments
  if ( argc!=3 ) {
    fprintf(stderr,"usage: %s ior indice\n",argv[0]);
    exit(1);
  }

  // récupération d'un fantôme du GPS
  char* ior=argv[1];
  f = ...
  gps* fgps = ...

  // récupération d'un fantôme d'indice
  finance* findice =  ...

  // utilisation de findice (getcurr)
  long status; double value;
  status = ...
  printf("%s:getcurr:status= %ld\n",argv[2],status);
  if ( status<0 )
    printf("%s:getcurr: non disponible.\n");
```

```
  else
    printf("%s:getcurr:value = %6.1f\n",argv[2],value);
  // utilisation de findice (getlast)
  DoubleArray_var values;
  status = ...
  printf("%s:getlast:status= %ld\n",argv[2],status);
  if ( status<0 )
    printf("%s:getlast: non disponible.\n");
  else {
    for (int i=0 ; i<.....  ; i+=1)
      printf("%s:getlast:values= %6.1f (i=%d)\n",argv[2], ..... ,i);
  }
  return 0;
}
```

Implantation Java d'un client

Souches clientes

du GPS

```
public interface gpsOperations {
  finance get (String indice);
  void add (String indice, finance objet);
}
public interface gps extends gpsOperations, ... { }
```

de finance

```
public interface financeOperations
{
  int getcurr (org.omg.CORBA.DoubleHolder value);
  int getlast (DoubleArrayHolder values);
}
public interface finance extends financeOperations, ... { }
```

Programme principal (fichier: client.java)

```
import org.omg.CORBA.*;
public class client {
public static void main(String args[]) {
  try{
   // initialisation de l'orb (client)
    ORB orb = ORB.init(args, null);
    // vérification des arguments
    if ( args.length!=2 ) {
      System.out.println("usage: java client %s ior indice\n");
      System.exit(1);
    }
    // récupération d'un fantôme du GPS
    String ior = args[0];
    org.omg.CORBA.Object f = ...
    gps fgps = ...
    // récupération d'un fantôme d'indice
    finance findice = .....................
    // utilisation de findice (getcurr)
    ....................................................
    if ( ................................<0 )
      System.out.println(args[1] + " : non disponible");
    else
      System.out.println(args[1] + " : " + ......................);
  } catch (Exception e) {
    System.out.println("ERROR : " + e) ;
    e.printStackTrace(System.out);
  }
 }
}
```

7.4 Spécificités de programmation

7.4.1 Obtention des IORs

Quelles IORs

```
interface X { | interface Y {
  ...         |    ...
  ...         |    X getX();
  ...         |    ...
}             | }
```

Seules les racines des graphes d'objets sont nécessaires.

Méthodes

- cablé en dur
- par fichiers
- par un autre protocole http, ldap, ...
- Naming Service: serveur CORBA stockant des couples (nom,IOR).
- Trader Service: serveur CORBA stockant des couples (mot clé, IOR).

7.4.2 Gestion des références

Langage OO

Java $\Longrightarrow$ c'est fait tout seul
C++ $\Longrightarrow$ à la charge du programmeur

CORBA

Java $\Longrightarrow$ c'est fait tout seul

C++ $\Longrightarrow$ CORBA ajoute un compteur de references dans les objets réels et *fantômes*.

Middleware OO

Quelque soit le langage, quand un serveur sert un objet il ne peut être détruit.

$\Longrightarrow$ le client doit indiquer qu'il n'en a plus besoin

$\Longrightarrow$ nécessite un nettoyage pèriodique pour les clients mal programmé ou pour les crashs

7.4.3 Actions périodiques sur un serveur

Problème

```
CORBA::ORB_var          orb;
PortableServer::POA_var poa;
...
poa->the_POAManager()->activate();
orb->run();
```

La boucle infinie d'écoute sur l'ORB empêche la possibilité de toute action.

Solution système

- Exemple UNIX: faire un callback sur un des signaux SIGUSR$_i$, SIGHUP.
- Exemple UNIX: faire un callback sur SIGALRM et programmer le timer.

$\longrightarrow$ non portable

Solution middleware Ajouter un objet avec une fonction "do()" dans le serveur, lancer un client periodiquement (crond) qui fait "do".

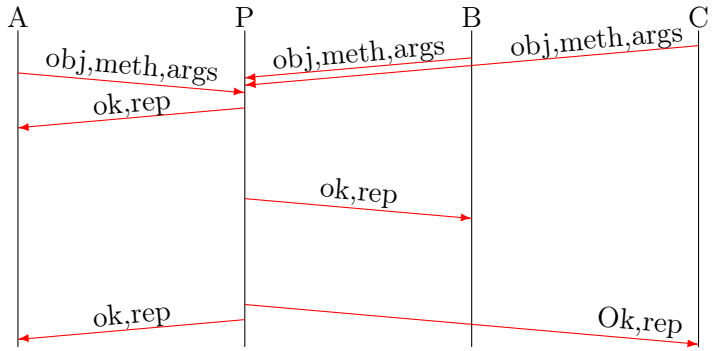
7.4.4 Politique de traitement des serveurs

Problèmes Réactivité, Inter-blocage, Ressources système

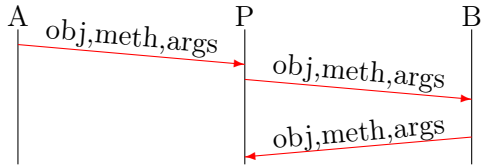
Implantation simple d'un client/serveur Le processus est soit client soit serveur:

pour toujours faire
  PDUR = lire-reseau(client)
  Obj,Meth,args $\Longleftarrow$ PDUR
  ret $\Longleftarrow$ Obj$->$Meth(args)
  PDUA $\Longleftarrow$ OK,ret
  ecrire-reseau(PDUA,client)
fait

Réactivité



Inter-blocage



- P attend une réponse sur la socket du serveur B.
- B attend une réponse sur la socket du serveur P.

Ressources système: SOLUTION-1

Principe Créer une thread pour chaque requête sortante et pour chaque requête entrante.

Avantages

- Résout les problemes d'inter-blocage.
- Résout partiellement les problemes de réactivité.

Inconvénients

- les créations/destructions de la threads ralentissent les fonctions simples
  $\Longrightarrow$ Réactivité constante mais dégradée.
- Grosse charge
  $\Longrightarrow$ Risque d'écroulement de la machine.
- Quitte le monde séquentiel.

Ressources système: SOLUTION-2

Principe Créer un pool de threads.

- prendre une thread du pool pour chaque requête sortante.
- prendre une thread du pool pour chaque requête entrante.
- s'il n'y a pas de thread libre dans le pool, alors attendre qu'une se libère.

Avantages

- Répousse les problemes d'inter-blocage.
- Résout les problemes de réactivité.
- Fixe une borne au ressources système utilisées.

Inconvénients

- Inter-blocages encore possibles.
- Quitte le monde séquentiel.

Solution CORBA

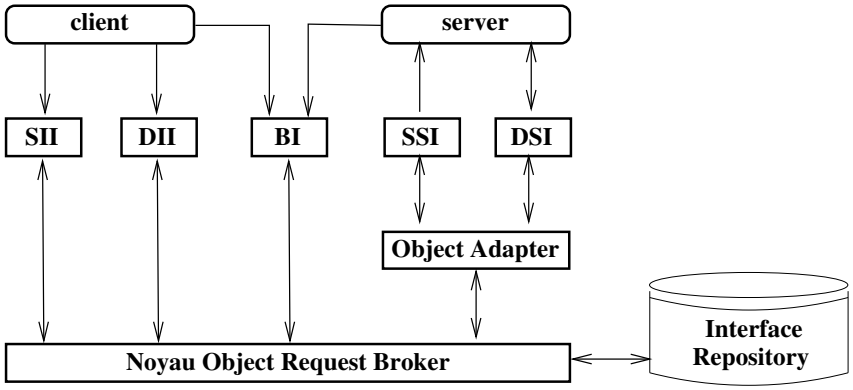
- Possibilité de créer des POAs (threads serveurs).
- Possibilité d'associer chaque objet à un POA.
- Pour des objets peu utilisés
  - Possibilité de désactiver un objet quand il n'est pas utilisé.
  - L'objet est réactivité automatiquement quand une requête arrive.

  $\Longrightarrow$ libère des ressources système.

  $\Longrightarrow$ permet de réaliser des serveurs avec beaucoup d'objets.

7.4.5 Vue interne

Architecture



ORB Noyau de communication (Object Request Broker)

BI/OA Bus Interface / Adadapteur d'objet

SII Static Invocation Interface

SSI Static Squeleton Interface

I. R. Description des interfaces des objets (introspection)

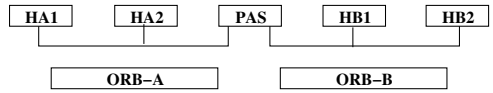
DII Dynamic Invocation Interface

DII Dynamic Squeleton Interface

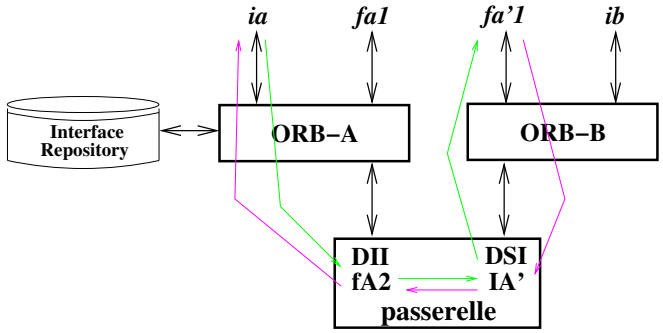
Ponts

utilité

- Explorateur du graphe d'objet (limité).
- Pont entre 2 ORBs.



implantation



7.5 Services

7.5.1 Généraliés

Principaux services

Collection, query gestion d'ensembles d'objet.

Concurrency accès en section critique à un objet.

Time synchronisation d'horloge.

EVoT enhanced view of time: Time + exécution périodique et diférée et batch.

Event, notification canaux de communication, signalisation.

Externalization persistence et internationnalisation.

Persistent state base de données d'objet.

Licensing gestionnaire de license.

naming associer un nom à un objet.

property associer des informations à des objets.

trading associer des mots clé à un objet.

transaction moteur transactionnel.

Spécification

Description idl Les services sont des interfaces CORBA.

Documentation utilisateur des interfaces utilisateurs.

$\Longrightarrow$ Chaque service est un ou un ensemble d'objets CORBA.

$\Longrightarrow$ Les implantations propriétaires CORBA fournissent:

- En librairie les souches clientes de ces interfaces.
- En librairie les squelette serveur de ces interfaces.
- Les exécutables de serveurs implantant ces interfaces.

Gabarit

```
module ServiceName {
    /* description des types internes */
    ...
    /* definition des enumeration */
    ...
    /* definition des exceptions */
    ...
    /* definition des interfaces  */
    ...
};
```

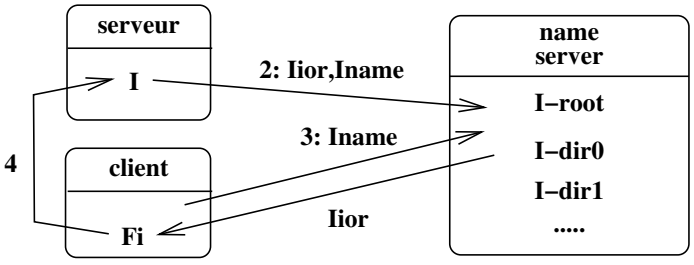
7.5.2 Service de nommage

Introduction

Principe

1. Un serveur gérant des couples (nom,IOR).
2. Les objets serveur s'enregistrent dans le NS sous un nom.
3. Les clients récupèrent du NS un objet en fonction du nom.

$\Longrightarrow$ NS est la racine du graphe d'objets.



Structure des noms

– structure arborescente (ex: /rep$_0$/rep$_1$/.../nom)

– nom de base: un couple (id.kind)

IDL

```
module CosNaming {
typedef string Istring;
 struct NameComponent {
   Istring id;
   Istring kind;
};
typedef sequence<NameComponent> Name;
interface NamingContext {
   ...
};
};
```

IDL de NamingContext

```
interface NamingContext {
void bind(in Name n, in Object obj)
   raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
 void rebind(in Name n, in Object obj)
   raises(NotFound, CannotProceed, InvalidName);
Object resolve (in Name n)
   raises(NotFound, CannotProceed, InvalidName);
void bind_context(in Name n, in NamingContext nc)
   raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
 NamingContext bind_new_context(in Name n)
   raises(NotFound, AlreadyBound, CannotProceed, InvalidName);
};
```

Utilisation

Lancement du serveur

```
lunix121:bash > nameserv -i -OAport 5000    # ORBACUS
lunix121:bash > tnameserv -ORBInitialPort 5000  # jdk
```

Lancement d'une application

```
unhost:bash > java app app-arg-0 app-arg-1 ...\
  --ORBInitialPort 5000 -ORBInitialHost lunix121
unhost:bash > app app-arg-0 app-arg-1 ...\
  -ORBservice  N-S iiop://lunix121:5000/DefaultNamingContext
unhost:bash > app app-arg-0 app-arg-1 ...\
  -ORBInitRef N-S=corbaloc::lunix121:5000/NameService
```

informations transmises

$\Longrightarrow$ le serveur est lancé sur un port donné.

$\Longrightarrow$ L'apllication récupère via N-S: host du server, port, le type de l'objet (NameService ou DefaultNamingContext).

Programmation

```
int main(int argc, char** argv) {
CORBA::ORB_var orb= CORBA::ORB_init(argc,argv);
CosNaming::NamingContext_var sn;
obj = orb -> resolve_initial_references("N-S");
sn = CosNaming::NamingContext::_narrow(obj);
CosNaming::Name name;
name.length(1);
name[0].id=   CORBA::string_dup("nom");
name[0].kind= CORBA::string_dup("objet");
// server                          // client
MonObj_impl* o = new MonObj_impl()obj = sn->resolve(name);
sn->rebind(name,o->_this());       MonObj_var f = tx::MonObj::_narrow(ob
...                                ...
}
```

7.5.3 Service de propriété

Fonction

Exemple

Une bibliothèque de livres, les attributs standards d'un livre sont: autheur,

titre, genre, ...

Problème

Comment utiliser cette bibliothèque pour faire une bibliothèque spécifique:

- personnelle en indiquand un jugement: coup de coeur, illisible, ...
- cormerciale en indiquant: prix, disponibilité, ...

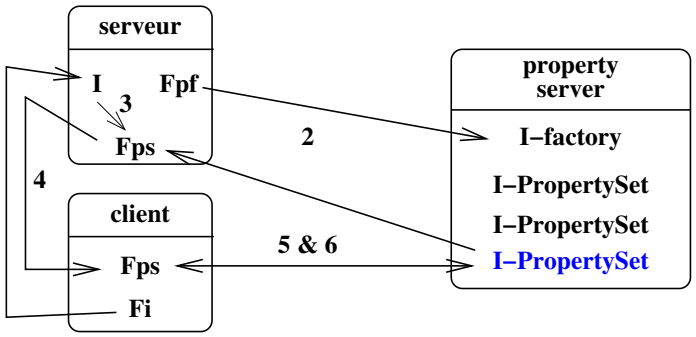
Fonction

Associer dynamiquement des attributs aux objets.

Introduction

Principe

1. Un serveur gérant des ensemples de propiétés (couple: nom,valeur).
2. Un objet serveur demande un ensemble de proriétés.
3. Le serveur associe cette ensemble à son objet.
4. Un objet fantôme récupère l'ensemble de proriétés.
5. L'objet fantôme peut alors récupérer une propriété.
6. L'objet fantôme peut alors ajouter une propriété.



Structure des Propriétés

- proprité de base: un couple (nom.valeur)
- ensemble de couples $\{(n_0, v_0), (n_1, v_1), \ldots\}$.

IDL

```
module PropertyService {
typedef string PropertyName;
struct Property {
  PropertyName property_name;
  any          property_value;
};
interface PropertySet        { ... };
interface PropertySetFactory { ... };
interface PropertySetDef : PropertySet { ... };
interface PropertySetDefFactory { ... };
};
```

IDL de PropertySet

```
interface PropertySet {
void define_property(
  in PropertyName property_name, in any property_value)
  raises(InvalidPropertyName, ConflictingProperty, ...);
boolean is_property_defined(in PropertyName property_name)
  raises(InvalidPropertyName);
any  get_property_value(in  PropertyName property_name)
  raises(PropertyNotFound, InvalidPropertyName);
void delete_property(in PropertyName property_name)
  raises(PropertyNotFound, InvalidPropertyName, ...);
unsigned long get_number_of_properties();
void get_all_property_names( ... )
};
```

IDL de PropertySetFactory

```
interface PropertySetFactory {
PropertySet create_propertyset ();
PropertySet create_initial_propertyset (
  in Properties initial_properties)
```

```
    raises (MultipleExceptions);
};
```

Utilisation

Lancement du serveur

```
    lunix121:bash > propserv -i -OAport 6000   # ORBACUS
    lunix121:bash > propertyd \
      --regname1 PropertySetFactory \
      --regname2 PropertySetDefFactory \
      -ORBInitRef N-S=corbaloc::lunix121:5000/NameService
```

Lancement d'une application

```
    unhost:bash > app app-arg-0 app-arg-1 ...\
      -ORBservice  P-S iiop://lunix121:6000/DefaultPropertySetFactory

    unhost:bash > app app-arg-0 app-arg-1 ...\
      -ORBInitRef N-S=corbaloc::lunix121:5000/NameService
```

Programmation

IDL

```
  #include <PropertyService.idl>
interface book {
  string title();
  PropertyService::PropertySet properties();
};
```

Serveur (classe squelette)

```
  struct book_exp : public POA::book {
  PropertyService::PropertySet_var propset;
  char* title() { ... }
  PropertyService::PropertySet_ptr properties() {
     return propset->_duplicate(propset);
  }
  book_exp(PropertyService::PropertySetFactory_var psf) {
    propset = psf->create_propertyset();
  }
};
```

Serveur (programme principal)

```
  // unhost:bash > app app-arg-0 app-arg-1 ...\
//   -ORBservice P-S iiop://lunix121:6000/DefaultPropertySetFactory

int main(int argc, char** argv) {
  CORBA::ORB_var orb= CORBA::ORB_init(argc,argv);
  PortableServer::POA_var poa = ...;
  PropertyService::PropertySetFactory_var psf;
  obj = orb -> resolve_initial_references("P-S");
  psf = PropertyService::PropertySetFactory::_narrow(obj);
  book_exp* obj = new book(psf);
  poa->the_POAManager()->activate();
  orb->run();
}
```

client

```
  int main(int argc, char** argv) {
  CORBA::ORB_var orb= CORBA::ORB_init(argc,argv);
  book_var* book = ...;
  // utilisation des proprités
  PropertyService::PropertyService_var ps=book->propertie();
  // écriture d'une proprité        // obtention d'une proprité
  long       note;                  long       note;
  CORBA::Any any ;                  CORBA::Any any ;
  note=12; any <<= note;            any = *ps->get_property_value(
  ps->define_property(                          "note-ia");
      "note-ia",any);               any >>= note;
  ...                               ...
}
```

utilisation du type CORBA::Any

- `<<=` surchargé pour les principaux types
  ```
  any <<= (char*) ; any <<= (double) ;
  any <<= (CORBA::Object*) ; ...
  ```
- `>>=` surchargé pour les principaux types
  ```
  any >>= (const char*) ; any >>= (double) ;
  any >>= (CORBA::Object*) ; ...
  ```

7.6 Conclusion

7.6.1 Vrai middleware

Définition d'un middleware

Faire comuniquer des processus en:

- abstraction de la location physique
- abstraction des supports de communication
- abstraction des PDUs
- abstraction des langages

CORBA

Une fois que l'on a une racine du graphe d'objet,
$\Longrightarrow$ on passe d'un objet à l'autre de facon 100% transparente.

7.6.2 Qu'est ce CORBA?

Specification fonctionnelle

- une description du moteur.
- la définition de l'IDL et du format des IORs.
- une description du mapping de l'IDL vers les langages courants.
- un protocole pour permettre l'interopérabilité des ORBs IIOP (Internet Inter-ORB Protocol).
- une boite à outils généraux (services).
  Ce sont des descriptions IDL, documentées
- des boites à outils métiers Ce sont des descriptions IDL, documentées

ORB CORBA compliant

Une implantation (propriétaire) respectant les spécifications.
Elles sont appelées ORB (ex: ORB Mico, ORB Orbix, ...).

7.6.3 Propriétés

Interopérabilité d'ORB

$\Longrightarrow$ une aplication CORBA tourne sur n'importe implantation CORBA.

$\Longrightarrow$ 2 aplications CORBA développées sur le même ORB peuvent comuniquer.

$\Longrightarrow$ 2 applications CORBA développées sur des ORBs différents peuvent communiquer.

Protocole ouvert

D'autres protocoles peuvent être définis en plus de l'IIOP. Ceci permet de faire un ORB:

- optimisé pour mono machine.
- optimisé pour un ensemble de machine identiques.
- optimisé pour un protocole de transport.

7.6.4 Domaines d'applications

Domaines

$\Longrightarrow$ Milieu hétérogène.

$\Longrightarrow$ Existence d'un ORB pour les langages cibles.

$\Longrightarrow$ Pas de contraintes de performance réseau.

Applications

$\Longrightarrow$ Gestionnaire de fermes de calcul.

$\Longrightarrow$ Robotique/Controleur

$\Longrightarrow$ web & applet

$\Longrightarrow$ window manager

$\Longrightarrow$ widget manager

8 Travaux dirigés et pratiques

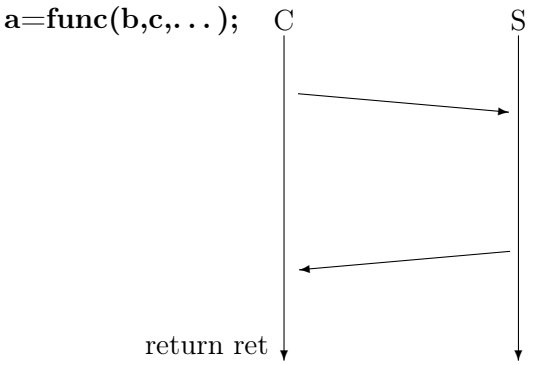


FIGURE 3: Message Sequence Chart

8.1 TD sur les RPC ONC

Le système RPC (RFC 1831) est une couche interface au-dessus de TCP-UDP/IP pour masquer et simplifier la communication entre clients et serveurs. L'idée de base est qu'une requête à un serveur est faite chez le client par un simple appel de fonction. Le résultat de la requête est la valeur retournée par la fonction.

8.1.1 Bases

L'objectif de ce TD est d'illustrer les concepts réseau **très importants** que sont les protocoles et les services. Ceci est fait au travers d'un exemple de transfert de fichiers en mode datagramme (non connecté).

Question.1 Considérons la fonction: `t_ret func(t_p0 p0, t_p1 p1, ...);` Indiquez les différentes étapes à effectuer pour réaliser un tel appel en RPC en complétant la figure 3.

Question.2 Quelles sont les limitations sur les types.

Question.3 La figure 4 montre qu'à partir d'un fichier "`ess.x`" contenant la déclaration "`int carre(int);`", "`rpcgen`" génère les fichiers décrit ci-dessous:

`ess.h` Il contient la déclaration de la fonction "`int carre(int)`" pour les clients et la déclaration de la fonction "`int carre_svc(int)`" pour le serveur.

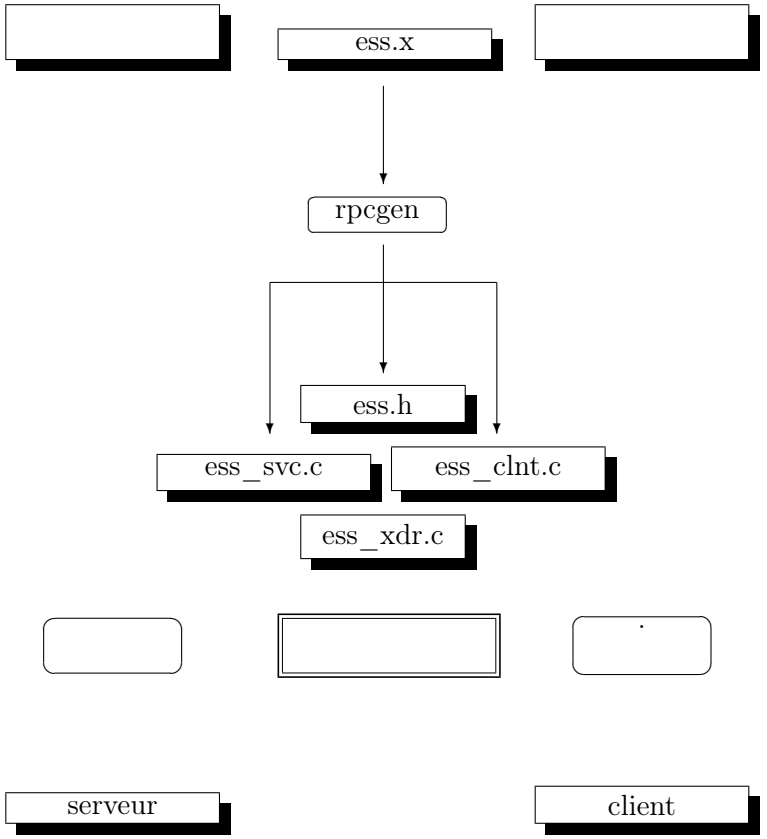


FIGURE 4: Schéma général de rpcgen

`ess_svc.c` C'est le squelette du serveur.

`ess_clnt.c` C'est la souche cliente.

`ess_xdr.c` Il contient les routines d'encodage et de décodage des PDU de l'application. Il n'existe pas si l'application n'a pas défini de types XDR.

a) Completez le schéma pour indiquer comment on décrit et génère un client et un serveur.

b) Si on souhaite générer un client sur une machine X, comment doit-on procéder?

8.1.2 Le fichier d'entrée

Le listing "`exemple.x`" ci dessous est un exemple de spécification d'application RPC. Ces définitions sont une extension de XDR. Les types des paramètres des fonctions doivent être les types de base de XDR ou tous types composés XDR pourvus qu'ils aient été définis préalablement dans le fichier.

Question.1 A quoi sert le numéro `0x31234567`.

Question.2 Pourquoi "`rpcgen`" supporte des numéros de version?

Question.3 Quel est l'identifiant d'une fonction sur un serveur?

Fichier: exemple.x

```
program EXEMPLE {
  version EXEMPLE_VERSION {
    string  conv(int)  =  5;
    int     carre(int) = 91;
  } = 3;
} = 0x31234567;
```

8.1.3 Le header généré

Le listing "`exemple.h`" ci-dessous est le header généré par "`rpcgen`" pour le fichier d'entrée "`exemple.x`".

Question.1 Indiquer le rôle de N dans les fonctions "`xxx_N`" et "`xxx_N_svc`".

Question.2 Indiquer les rôles des fonctions "`xxx_N`" et "`xxx_N_svc`".

Question.3 Que sont devenus les paramètres des fonctions? Quelles sont les raisons de cette transformation?

Fichier: exemple.h

```
1 /*
2  * Please do not edit this file.
3  * It was generated using rpcgen.
4  */
5 #ifndef _TD_H_RPCGEN
6 #define _TD_H_RPCGEN
7 #include <rpc/rpc.h>
8 #define EXEMPLE 0x31234567
9 #define EXEMPLE_VERSION 3
10 #define conv 5
11 extern  char ** conv_3(int *, CLIENT *);
12 extern  char ** conv_3_svc(int *, struct svc_req *);
13 #define carre 91
14 extern  int * carre_3(int *, CLIENT *);
15 extern  int * carre_3_svc(int *, struct svc_req *);
16 extern int exemple_3_freeresult (SVCXPRT *, xdrproc_t, caddr_t);
17 #endif /* !_TD_H_RPCGEN */
```

8.1.4 L'implantation des fonctions du côté serveur

Le listing "`server.c`" ci-dessous implante sur le serveur les fonctions définies dans le fichier d'entrée "`exemple.x`".

Question.1 Pourquoi la variables utilisée pour retourner la valeur elle est-elle déclarée statique? Comment pourrait-on aussi déclarer ces variables?

Question.2 Qui appelle ces fonctions?

Question.3 Donner l'algorithme du serveur.

Fichier: server.c

```
1 #include        <stdio.h>
2 #include        <rpc/rpc.h>       /* standard RPC include file */
3 #include        "exemple.h"       /* generated by rpcgen */
4 int * carre_3_svc(int *n, struct svc_req * svc)
5 {
6         static int     Carre;        /* must be static */
7         Carre= *n * *n;
8         return(&Carre);
9 }
10 char ** conv_3_svc(int *n, struct svc_req * svc)
11 {
12         static char ret_data[100];
13         static char* ret=ret_data;
```

```
14          sprintf(ret_data,"%d",*n);
15          return &ret;
16  }
```

8.1.5 Exemple de client

Le listing "`client.c`" ci-dessous donne un exemple d'application cliente.

Question.1 Commenter la séquence d'initialisation.

Question.2 Donner l'algorithme de "`carre_3`".

Question.3 Quelle est la durée de vie des pointeurs retournés?

Fichier: client.c

```
 1 /*************************************************************/
 2 /* client program                                            */
 3 /* usage:  client <server>                                   */

 4 #include        <stdio.h>
 5 #include        <rpc/rpc.h>      /* standard RPC include file */
 6 #include        "exemple.h"      /* generated by rpcgen */

 7 int main(int argc, char** argv)
 8 {
 9     CLIENT* cl;     /* RPC handle */
10     char*   server=argv[1];
11     int*    px;     /* return value from carre_1() */
12     char**  pc;     /* return value from conv_1()  */
13     int     x;      /* &x is used as parameter     */

14     if (argc != 2) {
15         fprintf(stderr, "usage: %s hostname\n", argv[0]);
16         exit(1);
17     }

18     /*  Create the client "handle." */
19     if ( (cl = clnt_create(server,
20                 EXEMPLE, EXEMPLE_VERSION, "udp")) == NULL) {
21         /* Couldn't establish connection with server.  */
22         clnt_pcreateerror(server);
23         exit(2);
24     }

25     /* First call the remote procedure "carre" (version 1). */
26     x=3;
27     if ( (px = carre_3(&x, cl)) == NULL) {
28         clnt_perror(cl, server);
29         exit(3);
30     }
31     printf("%d*%d-->%d\n",x,x,*px);

32     /* Now call the remote procedure "cov" (version 1).  */
33     x=101;
34     if ( (pc = conv_3(&x, cl)) == NULL) {
35         clnt_perror(cl, server);
36         exit(4);
37     }
38     printf("x=%d : pc=%s\n",x,*pc);

39     clnt_destroy(cl);
40     exit(0);
41 }
```

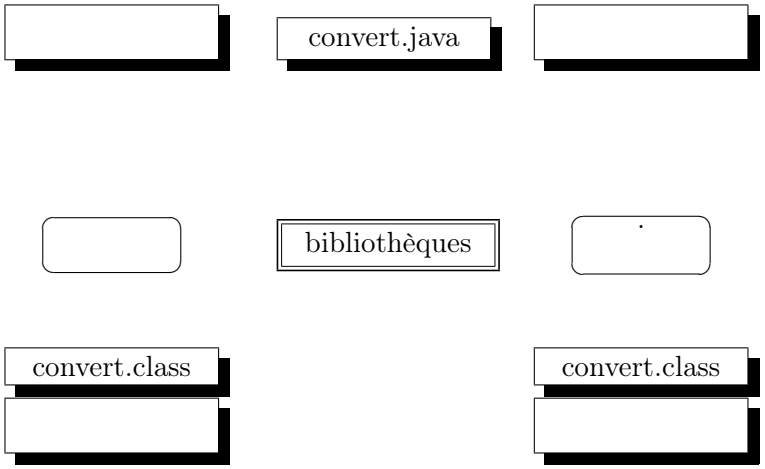


FIGURE 5: Schéma général de RMI

8.2 TD de RMI

L'objectif de ce TD est la conception d'un petit serveur financier et de quelques clients utilisant les services de ce serveur. Le serveur gère un taux de change entre 2 monnaies X et Y, il offre 2 services:

- La conversion de M unités de la monnaie X en monnaie Y (méthode convert, m étant le montant).
- La mise à jour du taux de change (méthode change, t est le nouveau taux et la fonction retourne l'ancien taux).

Question.1 Indiquez les taches pour réaliser une apllication RMI en complétant le schéma de la figure 6.

Question.2 Donnez une interface "`convert`" correspondant au services.

Question.3 Donnez le code d'un serveur.

Question.4 Donnez le code d'un client qui multiplie le taux par deux.

8.3 TP sur les RPC

Partie A: ONC RPC

L'objectif de ce TP est la réalisation d'un petit serveur financier et de quelques clients utilisant les services de ce serveur.

Le serveur gère un taux de change entre 2 monnaies X et Y, il offre 2 services:

- mise à jour du taux de change,
- conversion de M unités de la monnaie X en monnaie Y.

Question A.1 Ecrivez le fichier xdr/rpc "`x2y.xdr`" décrivant les services du serveur, compilez le puis appelez l'enseignant pour le valider.

Question A.2 Lancez rpcgen avec l'option "-a". A quoi correspondent le fichier "`x2y_server.c`"?

Question A.3 Ecrivez le fichier "`x2y.c`" décrivant le serveur, le taux sera initialisé à 1.

- Compilez le pour former l'exécutable "`x2y`".
- Dans un nouveau bureau, ouvrez une fenêtre et lancez le serveur.
- Revenez dans le bureau initial en laissant tourner le serveur.

Question A.4 Ecrivez le client "`x2y_conv.c`" qui toutes les 2 secondes affiche sur l'écran la conversion en Y d'une quantité de X passée en argument. L'impression peut se faire de la manière suivante:

```
printf("%5.2e X font %5.2e Y.\r", nbx, nby);
fflush(stdout);
sleep(2);
```

Ses arguments sont dans l'ordre: le montant, le serveur qui convertit les X en Y.

- Compilez le pour former l'exécutable "`x2y_conv`".
- Dans le bureau où tourne le serveur, ouvrez une fenêtre et lancez "`x2y_conv`" avec la quantité 50.
- Dans le bureau où tourne le serveur, ouvrez une autre fenêtre, connectez vous sur une autre machine (srvdev1-32 ou srvdev2-32) et lancez "`x2y_conv`" avec la quantité 100.

– Revenez dans le bureau initial en laissant tourner le serveur et les 2 clients.

Question A.5 Ecrivez le client "`x2y_set.c`" qui fixe le taux de change de X en Y à la valeur passée en argument. Ses arguments sont dans l'ordre: le nouveau taux, le serveur qui convertit les X en Y.

– Compilez le pour former l'exécutable "`x2y_set`".
– Dans le bureau où tourne le serveur, et les 2 clients, ouvrez une nouvelle fenêtre, et lancez "`x2y_set`" pour changez le taux à 2.
– Dans cette fenêtre connectez vous sur une autre machine (lunix121 ou lunix122) et lancez "`x2y_set`" pour changez le taux à 0.5.

Question A.6 Dans le bureau de test (1 serveur et 2 clients),

– créez une nouvelle fenêtre, connectez vous sur une autre machine et lancez "`x2y`" qui convertit les Y en Z.
– créez une nouvelle fenêtre, et lancez le client "`x2y_conv`" qui donne la conversion de 1000 Y en Z.
– changez les taux X/Y et Y/Z en 2 et .5.
– Revenez dans le bureau initial en laissant tourner les 2 serveurs et les 3 clients.

Paritie B: RMI

L'objectif de ce TD est la conception d'un petit serveur financier et de quelques clients utilisant les services de ce serveur. Le serveur gère un taux de change entre 2 monnaies X et Y, il offre 2 services:

– La conversion de M unités de la monnaie X en monnaie Y (méthode convert, m étant le montant).
– La mise à jour du taux de change (méthode change, t est le nouveau taux et la fonction retourne l'ancien taux).

Question B.1 Implantation du serveur du serveur

a) Ecrivez dans le fichier "`x2y.java`" la classe "`x2y`" qui implante l'interface convert. On y mettra aussi le `main`. Les arguments du main sont dans l'ordre le nom de l'objet, le taux de change.
NOTE: l'instruction JAVA ci-dessous permet de convertir une String en double.
`double d = Double.parseDouble("123.67");`

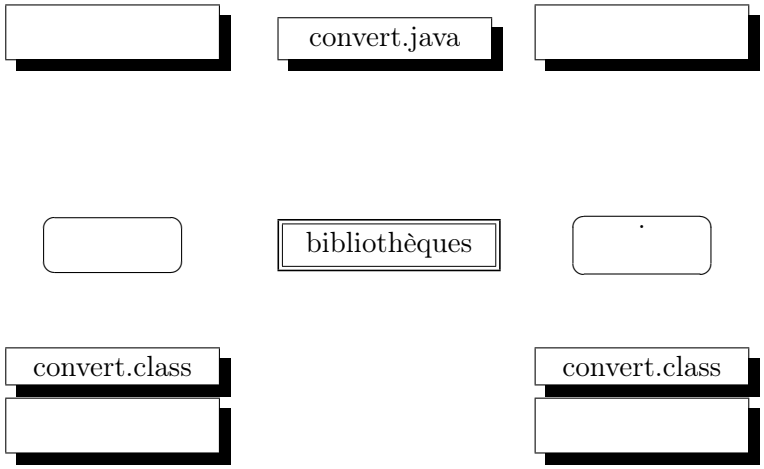


FIGURE 6: Schéma général de RMI

b) Compilez le.
c) Lancez le dans une fenêtre.

Question B.2 Implantation d'un client qui convertit une somme.

a) Ecrivez dans le fichier "`conv.java`" la classe "`conv`" qui contient le `main`. Les arguments du main sont dans l'ordre: le nom du serveur, le nom de l'objet, le montant à convertir. Le programme écrira "m X —> n Y" où m est le montant et n la conversion.
b) Compilez le.
c) testez le

Question B.3 Implantation d'un client qui change le taux.

a) Ecrivez dans le fichier "`change.java`" la classe "`change`" qui contient le `main`. Les arguments du main sont dans l'ordre: le nom du serveur, le nom de l'objet, le nouveau taux. Le programme écrira l'ancien taux et le nouveau taux.
b) Compilez le.
c) testez le avec "`conv`"

Paritie C: ONC RPC (suite)

L'objectif de ce TP est la réalisation d'un petit serveur financier et de quelques clients utilisant les services de ce serveur.

Le serveur gère un taux de change entre 2 monnaies X et Y, il offre 2 services:

- mise à jour du taux de change,
- conversion de M unités de la monnaie X en monnaie Y.

Question C.1 Ecrivez le client "`x2y2z_conv.c`" qui toutes les 2 secondes affiche sur l'écran la conversion en Z d'une quantité de X passée en argument. Ses arguments sont dans l'ordre: le montant, le serveur qui convertit les X en Y et le serveur qui convertit les Y en Z.
Testez le dans le bureau de test.

Question C.2 En utilisant l'implantation "`remotetea`" des ONC-RPC en java. écrivez le client "`x2y_conv.java`" et testez le avec les servers C existants. "`remotetea`" est installé dans le répertoire "`/pub/ia/remotetea`". L'équivalent du rpcgen se lance de la manière suivante:

```
⟶ java -classpath /pub/ia/remotetea/classes \
  -jar /pub/ia/remotetea/classes/jrpcgen.jar x2y.xdr
```

Question C.3 Refaire la question 1 mais le second serveur doit tourner sur la même machine que le premier. Pour cela on fera le script shell "`gen.sh`" dont les arguments sont 2 ports RPC et qui génère les exécutables:

- "`x2y_1`" similaire à "`x2y`" mais qui utilise le premier port RPC.
- "`x2y_2`" similaire à "`x2y`" mais qui utilise le second port RPC.
- "`x2Y2z_conv_12`" similaire à "`x2y2z_conv`" mais qui utilise les 2 ports RPC.

L'idée de ce script est: 1) lancer `rpcgen` pour regéner les fichiers; 2) de supprimer la macro qui définit le port TCP dans le fichier "`x2y.h`"(`sed -i -e "/define EXEMPLE/d" x2y.h`); 3) lancer les compilation des exécutables en fixant les macro des ports sur la ligne de commande grace à l'option `-DNomDeMacro=0xbeef`.

8.4 TD Initiation à CORBA

CORBA est un middleware objet. Il permet d'invoquer des objets distants. Bien qu'en théorie il soit possible d'implanter directement les mécanismes CORBA pour une application donnée, on utilise toujours une implantation CORBA qui contient déjà en librairie la plupart des mécanismes CORBA et génère le code CORBA pour la partie propre de l'application.
Dans ce TD on utilisera l'implantation **MICO**.

8.4.1 L'application

Ci-dessous est présentée l'IDL de l'application **tx**. Cette application est un gestionnaire de conversion de monnaie ou de capitalisation d'interêt. La classe "`taux`" utilise un taux d'intérêt t. On peut modifier ce taux (`inc_taux`: $t = t + i$), obtenir la valeur res d'une somme val (`calcul`: $res = val\ t$), obtenir une capitalisation d'intérêt sur a ans (capital: $res = val\ t^a$).

```
// fichier: taux.idl
module tx {
  interface convert {
    void calcul(
        out double res, in double val);
    void capital(out double res,
        in double val, in long an);
    void inc_taux(in double incr);
  };
};
```

On génère la souche cliente et le squelette du serveur en exécutant la commande "`(`"idl). Le fichier généré "`taux.h`" est présenté page 36.

```
⟶ ls
taux.idl
⟶ idl -codegen-c++ -I /usr/lang/mico/include/coss
taux.idl
⟶ ls
taux.cc taux.h taux.idl
⟶
```

Le fichier "`taux-serv.cc`" (page 36) est une implantation d'un serveur basique. Le fichier "`taux-cli1.cc`" (page 37) est une implantation d'un client qui affiche la conversion d'une somme donnée en temps "réel". Les commandes ci-dessous indiquent comment les compiler et les lancer.

fenêtre pour le serveur

```
⟶ g++ -o x2y \
-I. -I/usr/lang/include \
taux-serv.cc taux.cc \
-L /usr/lang/lib -lmico2.3.13 \
-lssl -lpthread
⟶ x2y 1.25 e d
serveur pret!
```

fenêtre pour le client

```
⟶ g++ -o cli1 \
  -I. -I/usr/lang/include \
  taux-cli1.cc taux.cc \
  -L /usr/lang/lib -lmico2.3.13 \
  -lssl -lpthread
⟶ cli1 80 e d
  80 e -> 100.00 d
```

8.4.2 Exercices

Question.1

a) Completez le schéma de la figure 7 pour indiquer comment on décrit et génère un client et un serveur.

b) Si on souhaite générer un client sur une machine X, comment doit-on procéder?

Question.2 Indiquez comment l'IOR de l'objet du serveur est transmise au client? Comment pourrait-on la lire?

Question.3 Quelle est la classe représentant la souche cliente?

Question.4 Quelle est la classe représentant le squelette du serveur? Comment est cette classe?

Question.5 Ecrivez cli2 qui modifie le taux. Expliquez comment l'essayer.

Question.6 On considère un nouveau client appelé "`cli3`". Celui-ci convertit un montant m de la monnaie x en la monnaie y de manière simulaire à "`cli1`". Pour cela il utilise le serveur convertissant la monnaie x en d ($) et le serveur convertissant la monnaie d en y. Les arguments de "`cli3`" sont les mêmes que ceux de "`cli1`".

a Donnez un MSC illustrant le fonctionnement de "`cli3`".

b Implantez "`cli3`".

Question.7 On considère un nouveau serveur appelé "`x2y2z`" qui convertit la monnaie x en z. Ses arguments sont la monnaie x, la monnaie y et la monnaie z. Ce serveur ne stocke pas le taux de conversion de la monnaie x en z mais pour chaque requête (calcul ou capital), il interroge le serveur convertissant la monnaie x en y et Celui convertissant la monnaie y en z.

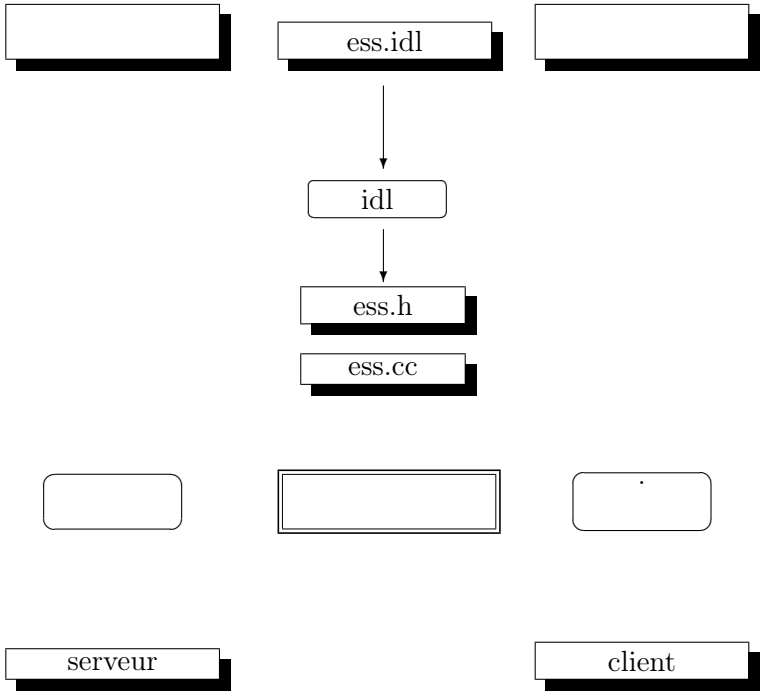


FIGURE 7: Schéma général de corba

a Donnez un MSC illustrant le fonctionnement de "x2y2z".

b Implantez "x2y2z" (La fonction "inc_taux" ne faisant rien).

c Un client "cli1" peut-il interroger un tel serveur?

d Un serveur "x2yZ" peut-il interroger un autre serveur "x2y2z" pour convertir la monnaie x en y ou la monnaie y en z.

Question.8

On modifie l'IDL de l'application **tx** comme montré ci-contre. La méthode "reverse" appliquée à un objet O, renvoie un objet O' qui a en permanence le taux inverse de O. C'est-à-dire, si la méthode "inc_taux" est appliquée à O, le taux de O' est ausi mis à jour et vice-versa.

```
// fichier: taux.idl
module tx {
  interface convert {
    void calcul(
      out double res, in double val);
    void capital(out double res,
      in double val, in long an);
    void inc_taux(in double incr);
  };
  interface convrev : convert {
    convrev reverse();
  }
};
```

a Créez le serveur "x2y_r" qui gère deux objets de type "convrev". "x2y_r" ne sauve l'IOR que du premier objet. La fonction "reverse" du premier renvoie le second et vice-versa.

b Créez le client "cli1_r" analogue au "cli1" précédent. Il a un argument suplémentaire indiquant s'il fonctionne sur l'objet direct ou son inverse.

c Est-ce que les anciens clients comme "cli1", "cli2" fonctionnent avec les nouveaux serveurs ("x2y_r").

d Si l'on désire pouvoir accèder à tous les objets d'un système, quels sont les objets dont il faut absolument connaitre les IORs?

8.4.3 Annexes

Fichier: taux.h

```
1 /*
2  *  MICO --- an Open Source CORBA implementation
3  *  Copyright (c) 1997-2006 by The Mico Team
4  *
5  *  This file was automatically generated. DO NOT EDIT!
6  */
7 namespace tx
8 {
9 class convert;
10 typedef convert *convert_ptr;
11 typedef convert_ptr convertRef;
12 typedef ObjVar< convert > convert_var;
13 typedef ObjOut< convert > convert_out;
14 }
15 namespace tx
16 {
17 class convert : virtual public CORBA::Object {
18     static convert_ptr _narrow( CORBA::Object_ptr obj );
19     ...
20     virtual void calcul( CORBA::Double_out res, CORBA::Double val ) = 0;
21     virtual void capitalisation( CORBA::Double_out res, CORBA::Double val,
22                                  CORBA::Long an ) = 0;
23     virtual void inc_taux( CORBA::Double incr ) = 0;
24     ...
25 };
26 class convert_stub: virtual public convert {
27     ...
28     void calcul( CORBA::Double_out res, CORBA::Double val );
29     void capitalisation( CORBA::Double_out res, CORBA::Double val,
30                          CORBA::Long an );
31     void inc_taux( CORBA::Double incr );
32     ...
33 };
34 }
35 namespace POA_tx
36 {
37 class convert : virtual public PortableServer::StaticImplementation
38 {
39     ...
40     tx::convert_ptr _this ();
41     ...
42     virtual void calcul( CORBA::Double_out res, CORBA::Double val ) = 0;
43     virtual void capitalisation( CORBA::Double_out res, CORBA::Double val,
44                                  CORBA::Long an ) = 0;
45     virtual void inc_taux( CORBA::Double incr ) = 0;
46     ...
47 };
48 }
```

Fichier: taux-serv.cc

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <CORBA.h>
4 #include "taux.h"
5 static char* progname;
6 CORBA::ORB_var          orb;
7 PortableServer::POA_var poa;
```

```
 8 /*********************************************************************/
 9 /*** Definition of the exported class                              ***/
10 class convert_exp : public POA_tx::convert {
11 public:
12     convert_exp( CORBA::Double t) : taux(t) {}
13     virtual void calcul(CORBA::Double& res, CORBA::Double val) {
14         res = val * taux;
15     }
16     virtual void capital(CORBA::Double& res, CORBA::Double val,
17             CORBA::Long an) {
18         res= val;
19         for (int i=0 ; i<an ; i++)
20             res += res*taux;
21     }
22     virtual void inc_taux(::CORBA::Double inc) {
23         taux += inc;
24     }
25 private:
26     CORBA::Double taux;
27 };
28 /*********************************************************************/
29 /*** writes the oid of obj into "src2dest.ref" file.               ***/
30 void generate_ref_file(char* src, char* dest, CORBA::Object_var obj)
31 {
32     char tmp[100]; sprintf(tmp,"%s2%s.ref",src,dest);
33     FILE* fid=fopen(tmp,"w");
34     if (fid==0) {
35         fprintf(stderr,"%s: can't open %s\n",progname,tmp);
36         exit(1);
37     }
38     CORBA::String_var s= orb->object_to_string(obj);
39     fprintf(fid,"%s",s._retn());
40     fclose(fid);
41 }
42 /*********************************************************************/
43 /*** main routine.                                                 ***/
44 int main(int argc, char** argv)
45 {
46     CORBA::Object_var obj;
47     progname=argv[0];
48     orb= CORBA::ORB_init(argc,argv);
49     obj = orb->resolve_initial_references("RootPOA");
50     poa= PortableServer::POA::_narrow(obj);
51     if (argc!=4) {
52         fprintf(stderr,"usage: %s taux src dest\n",progname);
53         exit(1);
54     }
55     convert_exp* o = new convert_exp(atof(argv[1]));
56     //tx::convert_var v = o->_this();
57     generate_ref_file(argv[2],argv[3],o->_this());
58     PortableServer::POAManager_var pman = poa->the_POAManager();
59     pman->activate();
60     fprintf(stderr,"Serveur pret\n");
61     orb->run();
62     return 0;
63 }
```

Fichier: taux-cli1.cc

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <unistd.h>
 4 #include <CORBA.h>
 5 #include "taux.h"
 6 char* progname;
 7 CORBA::ORB_var orb;
 8 /*********************************************************************/
 9 /*** reads a oid from the "srs2dest.ref" file, gets the associated  ***/
10 /*** objects and finaly tries to cast it to tx::convert.            ***/
11 tx::convert_var read_ref_file(char* src, char* dest)
12 {
13     char tmp[1000]; sprintf(tmp,"%s2%s.ref",src,dest);
14     FILE* fid=fopen(tmp,"r");
15     if (fid==0) {
16         fprintf(stderr,"%s: can't open %s\n",progname,tmp);
17         return 0;
18     }
19     int nb=fread(tmp,1,sizeof(tmp)-1,fid); tmp[nb]=0;
20     fclose(fid);
21     CORBA::Object_var o=orb->string_to_object(tmp);
22     if ( CORBA::is_nil(o) ) {
23         return 0;
24     }
25     return tx::convert::_narrow(o);
26 }
27 /*********************************************************************/
28 /*** main routine                                                  ***/
29 int main(int argc, char** argv, char **)
30 {
31     progname=argv[0];
32     orb= CORBA::ORB_init(argc,argv);
33     if (argc!=4) {
34         fprintf(stderr,"usage: %s montant src dest \n",progname);
35         exit(1);
36     }
37     double value=atof(argv[1]);
38     tx::convert_var conv;
39     conv= read_ref_file(argv[2],argv[3]);
40     if ( CORBA::is_nil(conv) ) {
```

```
41          fprintf(stderr,"%s: no object found in %s2%s.ref file\n",
42                          progname,argv[2],argv[3]);
43          exit(1);
44      }
45      while (1) {
46          double ret;
47          conv->calcul(ret,value);
48          printf("value=%5.2f conv=%5.2f\r",value,ret);
49          fflush(stdout);
50          sleep(1);
51      }
52      return 0;
53  }
```

8.5 TP Initiation à CORBA

Attention: Conservez vos fichiers pour le prochain TP.

Question 1 Générez le serveur `x2y` à partir des fichiers "`/pub/ia/corba/taux.idl`" et "`/pub/ia/corba/taux-serv.cc`" fournis. **Attention: il n'y a pas une ligne de code à écrire.**.

a) Lancez le dans une fenêtre.

b) Regardez l'IOR générée.

Question 2 Générez le client `cli1` à partir des fichiers "`/pub/ia/corba/taux.idl`" et "`/pub/ia/corba/taux-cli1.cc`" fournis. Puis lancez le dans une fenêtre. **Attention: il n'y a pas une ligne de code à écrire.**.

Question 3 Ecrivez et générez le client `cli2`. Pour le tester, mettez vous dans un nouveau bureau pui:

- Ouvrez une fenêtre et lancez un serveur `x2y`.
- Ouvrez une autre fenêtre et lancez le client `cli1` avec la quantité 50.
- Ouvrez une $3^{ième}$ fenêtre et lancez un client `cli2`.

Question 4 Faites le client `cli3` et testez le.

Question 5 Faites le server/client `x2y2z` et testez le.

Question 6 Faites le server `x2y_r` et vérifiez que les anciens clients `cli1` fonctionnent encore avec ce serveur.

Question 7 Faites le client `cli1_r` et testez avec `x2y_r`.

Question 8 Faites le client `cli1.class` en vous inspirant du fichier "`/pub/ia/corba/client.java`" fourni qui est l'implantation du client de l'application de base vue en cours (voir section 7.2.2 page 16).

Question 9 Réalisation d'un mini-service de nommage (MSN). **Note**:Vous pouvez vous inspirer de la classe `localisation` vue en cours.

- Ecrivez l'IDL de l'objet MSN. Les 2 fonctions étant:
  - enregistrer un IOR sous un nom,
  - récupérer l'IOR associé a un nom.
- Ecrivez le serveur `msn`. On stockera les couples (nom,IOR) dans un tableau de taille statique.
- Modifiez `x2y` et `cli1` pour qu'ils utilisent `msn`.

8.6 TD Services CORBA

Pour touver les objets d'une application répartie CORBA propose entre autre le service de nommage. Le service de nommage est un objet dont la fonction est de maintenir des références d'objets en fonction d'une clef (nom). Ainsi le service de nommage peut être vu comme une racine de tous les objets qu'il référence.

1. Donnez un MSC illustrant l'utilisation du service de nommage.
2. Le serveur du service de nommage écrit l'IOR de l'objet service de nommage sur le standard output. Utilisez cette option pour obtenir le service de nommage sans passer par "`resolve_initial_references`".
3. Ecrivez le serveur "`x2y_ns`" analogue à "`x2y`" (page 36, qui, au lieu d'écrire son IOR dans un fichier, s'inscrira auprès du service de nommage sous le couple (devise source, devise cible).
4. Ecrivez le client "`cli1_ns`" analogue à "`cli1`" (page 37, qui, au lieu de lire son IOR dans un fichier, va le chercher auprès du service de nommage sous le couple (devise source, devise cible).
5. Ecrivez un client "intelligent". Il a les même arguments que le précédent et son algorithme est le suivant:
   - Rechercher l'objet (source,cible) de type `convert`.
   - Rechercher l'objet (cible,source) de type `convrev` et utiliser la fonction reverse.
   - Rechercher d'abord (source,dollar) ou (dollar,source), puis (cible,dollar) ou (dollar,cible).
   - Rechercher d'abord (source,euro) ou (euro,source) puis (euro,cible) ou (euro,cible).

Le listing ci-dessous illustre l'utilisation et la connection à ce service. Ce service est lui même un objet de type "`CosNaming_NamingContext`".

```
1   // Commande pour lancer le service de nommage
2   //   lunix121:bash$ tnameserv -ORBInitialPort 5123
3   //   IOR:000000000000002b49444....
4   //   TransientNameServer: setting port for ... to: 5123
5   //   Ready
6
7   // Commande pour lancer une application CORBA utilisant le service de
8   // nommage (MICO)
9   //   lunix121:bash$ app app-arg-0 app-arg-1 ... \
10  //     -ORBInitRef NameService=corbaloc::lunix121:5123/NameService
11
12  // Commande pour lancer une application CORBA utilisant le service de
13  // nommage (JAVA)
14  //   lunix121:bash$ java app app-arg-0 app-arg-1 ... \
15  //     -ORBservice NameService iiop://lunix121:5123/DefaultNamingContext
16
17  // Code de base pour obtenir le service de nommage
18  int main(int argc, char** argv)
19  {
20    CORBA::ORB_var               orb;
21    CosNaming::NamingContext_var sn;
22    CORBA::Object_var            obj;
23
24    orb= CORBA::ORB_init(argc,argv);
25
26    try {
27      obj = orb -> resolve_initial_references("NameService");
28      sn = CosNaming::NamingContext::_narrow(obj);
29    } catch (...) {
30      fprintf(stderr,"%s:  can't resolve 'NameService'\n",argv[0]);
31      return 1;
32    }
33
34    ...
35  }
```

Le listing ci-dessous décrit une partie de l'interface du service de nommage. Ce service est lui même un objet de type "`CosNaming_NamingContext`".

```
1   Un composant de nom est essentiellement composé de 2 strings (id,kind):
2       // definition
3       struct CosNaming_NameComponent {
4         ...
5         CORBA::String_var id;
6         CORBA::String_var kind;
7       };
8
9       // exemple d'utilisation
10      CosNaming::NameComponent x;
11      x.id   = CORBA::string_dup("bonjour");
12      x.kind = CORBA::string_dup("monsieur");
13
14  Un nom est un "tableau" de composants de nom, il est
15  similaire au nom d'un fichier dans un système de fichiers
16  comme "/bonjour.monsieur/au.revoir/madame.soleil"
17      // definition
18      typedef OBVarSeq< CosNaming::NameComponent > CosNaming::Name;
19      // exemple d'utilisation
20      CosNaming_Name name;
21      name.length(3);
22      name[0].id  = CORBA::string_dup("bonjour");
23      name[0].kind= CORBA::string_dup("monsieur");
24      name[1].id  = CORBA::string_dup("au");
25      name[1].kind= CORBA::string_dup("revoir");
26      name[2].id  = CORBA::string_dup("madame");
27      name[2].kind= CORBA::string_dup("soleil");
28
29  Quelques services de CosNaming::NamingContext:
```

```
1) recherche d'un objet en fonction d'un nom
  // definition
  CORBA::Object_ptr resolve( CosNaming::Name& name );
  // exemple d'utilisation:
  CosNaming::NamingContext_var sn;
  CosNaming::Name              name;
  CORBA::Object_var            obj;
  try {
    obj= sn->resolve( name );
  } catch (const CosNaming::NamingContext::NotFound&) {
    fprintf(stderr,"can't find object\n");
  } catch (const CosNaming::NamingContext::InvalidName&) {
    fprintf(stderr,"can't find object\n");
  }

2) ajout d'un couple (objet,nom) avec bind
  // definition
  CORBA::Object_ptr bind( CosNaming::Name& name,
                          CORBA::Object_ptr obj);
  // exemple d'utilisation:
  CosNaming::NamingContext_var sn;
  CosNaming::Name              name;
  CORBA::Object_var            obj;
  try {
    sn->bind( name, obj );
  } catch (const CosNaming::NamingContext::NotFound&) {
    fprintf(stderr,"can't record object\n");
  } catch (const CosNaming::NamingContext::AlreadyBound&) {
    fprintf(stderr,"can't record object\n");
  } catch (const CosNaming::NamingContext::InvalidName&) {
    fprintf(stderr,"can't record object\n");
  }

3) ajout d'un couple (objet,nom) avec rebind.
  Idem que 2) (remplacez bind par rebind) sans l'exception AlreadyBound.

4) ajout d'un "répertoire"
  // definition
  CosNaming::NamingContext_ptr bind_new_context(CosNaming::Name& name);
  // exemple d'utilisation:
  CosNaming::NamingContext_var sn,sub_sn;
  CosNaming::Name              name;
  try {
    sub_sn = sn->bind_new_context(name);
  } // catch identiques que bind(...).
```

8.7 TP Service CORBA

Question 1 Faites le serveur `ns_x2y` et les clients `ns_cli1` et `ns_cli2` qui sont les adaptations de `x2y`, `cli1i` et `cli2` du TP précédent. Au lieu de stocker leurs IORs dans le fichier "x2y.ref", ils utilisent le service de nommage avec le nom "x.y".

Question 2 Faites le client `ns_cli` qui convertit les x en y. Dans le cas où l'objet "x.y" n'existe pas, il essaiera de chaîner "x.d" et "d.y" puis si ces derniers n'existent pas, il essaiera de chaîner "x.e" et "e.y".

Question 3 Ajoutez des propriétés à la classe `convert`. Adaptez `ns_x2y` pour qu'il supporte les propriétés.

Question 4 Faites le client `nsp_cli2` similaire au client `ns_cli2` précédent mais qui stocke dans une propriété la date de la dernière modification et dans une autre le nombre de modification depuis la création de l'objet. Ce client affichera ces deux propriétés avant de changer le taux et de modifier ces propriétés.

Notes: Si vous avez terminé le serveur `x2r_r` lors du tp précédent, vous pouvez partir de celui-ci pour la question 1 et faire le client "intelligent" vu en TD pour la question 2.

Ci dessous quelques rappels:

```
1   // Lancement du service de proprietes de MICO, dans l'exemple
2   // ci-dessus, le daemon enregistre dans le service de nommage
3   // à sa racine:
4   //   - sous le nom ("PropertySetFactory",""), la fabrique
5   //     d'ensembles de proprietes,
6   //   - sous le nom ("PropertySetDefFactory",""), la fabrique
7   //     d'ensembles de proprietes par defaut.
8   lunix121:bash$ propertyd \
9     --regname1 PropertySetFactory \
10    --regname2 PropertySetDefFactory \
11    -ORBInitRef NameService=corbaloc::<host>:<port>/NameService
12
13  // Gestion de la date sous UNIX en C
14    time_t t;   // date en second depuis l'Epoch
15    char*  d;   // date au format humain
16
17    t=time(0);   // obtention de la date en seconde
18    d=ctime(&t); // conversion de t au format humain
19
20    // d se termine par un \n, sa suppression:
21    int len=strlen(d);
22    d[len-1]=0;
```