Intergiciels

Applications réparties

Chapitre 2

Polytech Marseille

Département Informatique

Formation par alternance / HUGo

5ème année

Introduction

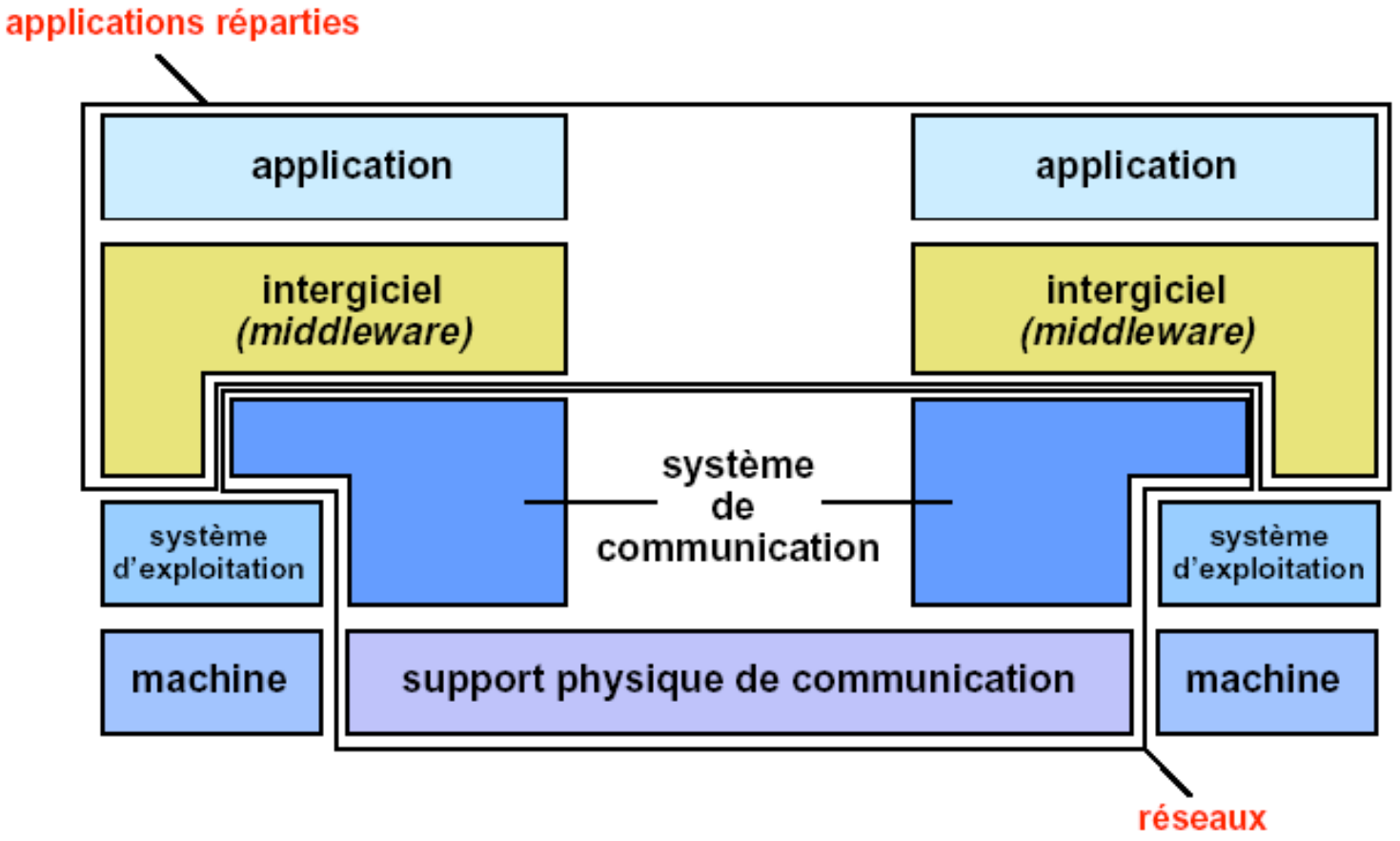
Réalisation d'une application répartie (fournissant un ou plusieurs services) :

- Par des opérations de "bas niveau"
  - Utilisation de primitives du système de communication.
  - Exemple : sockets
- Par des opérations de "haut niveau"
  - Utilisation d'un **intergiciel** (middleware) spécialisé.

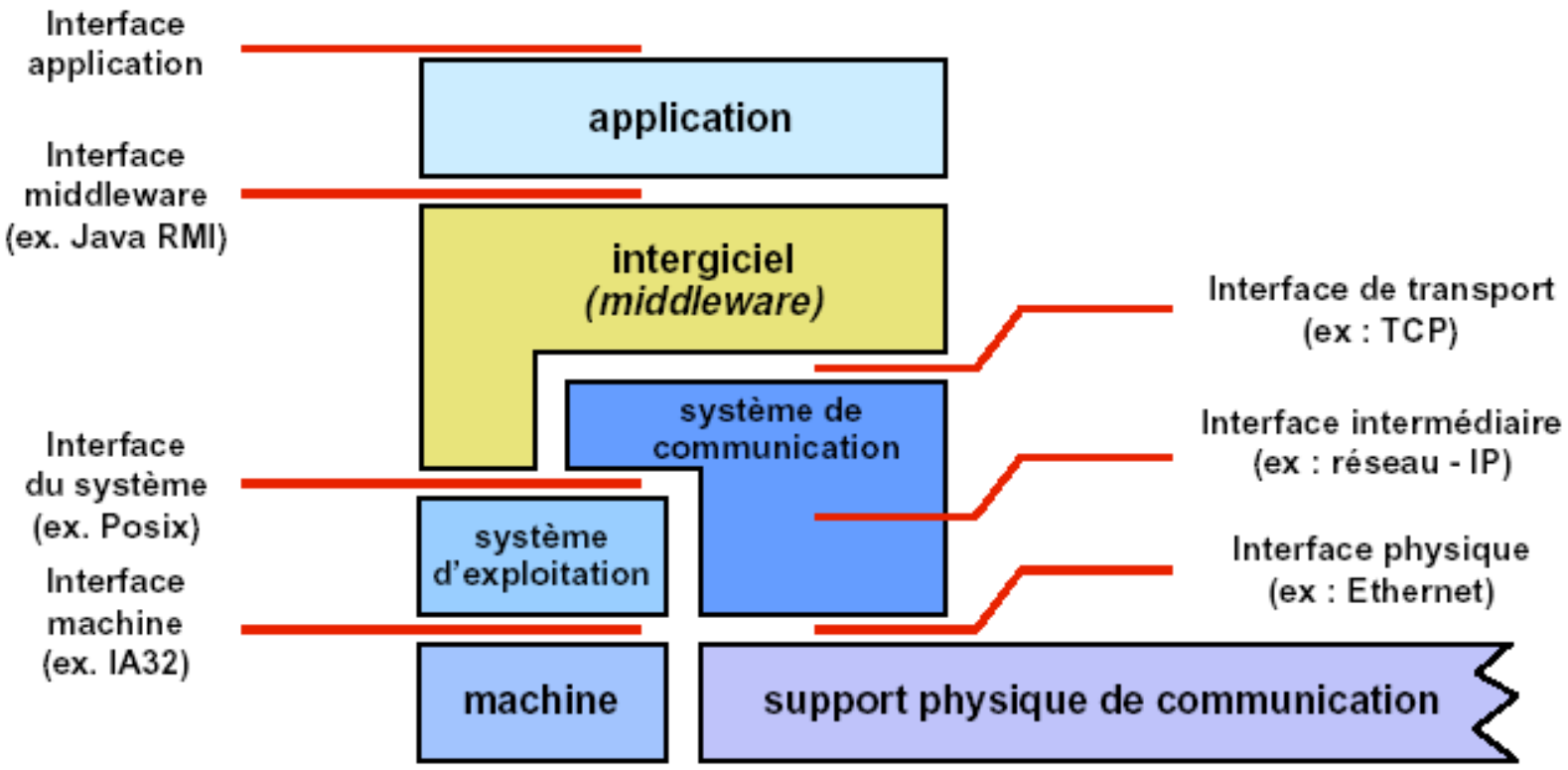
Intergiciel Définition

- **Intergiciel (Middleware)** = couche de logiciel (réparti) destinée à :
  - Masquer l'**hétérogénéité** des machines et des systèmes,
  - Masquer la répartition des traitements et données (**transparence** de localisation),
  - Fournir une interface aux applications (**modèle de programmation** + API).

Intergiciel
Situation (couche)



Intergiciel
Exemple d'interfaces



Chaque interface cache les interfaces de niveau inférieur

Catégories d'intergiciels

- Procédures distantes : RPC
- Orienté objet (objets répartis)
  - Java RMI, CORBA, PYRO, DCOM, ...
- Orienté composants
  - Enterprise Java Beans (EJB), .NET, …
- Orienté messages (MOM)
  - Java Message Service (JMS), ActiveMQ, …
- Orienté Web
  - Web Services (XML-RPC, SOAP), …

RPC (Remote Procedure Call)

- Appel de procédures à distance
- Avantages
  - Formes et effets identiques à un appel local
    - Simplicité conceptuelle et mise au point
  - Abstraction
    - Vis-à-vis protocole de communication
      - Distribution masquée
- Fin 70 début 80
- Utilisée dans le protocole NFS (entre autres)
- Exemples : SUN RPC, XML-RPC, …

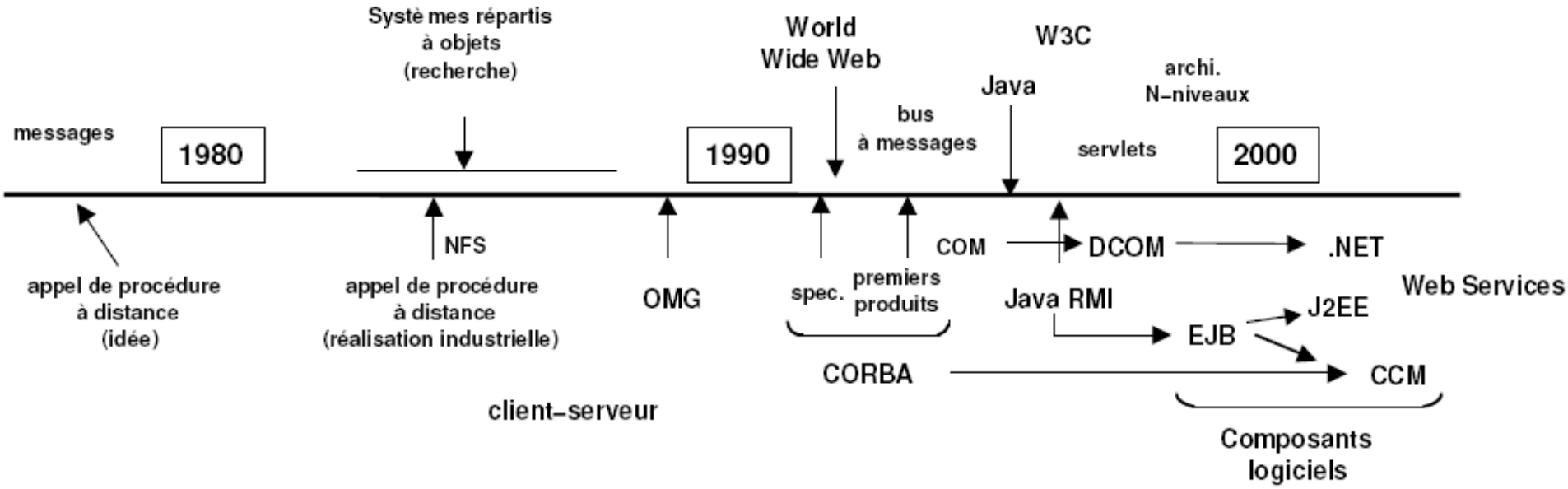
Java RMI (Remote Method Invocation)

- Invocation de méthodes à distance
- RMI : implémentation de RPC pour le langage orienté objet **Java**
- Appel de méthodes à distance sur "**objets serveurs**"
- Présent dans le JDK
- Autres implémentations : NinjaRMI, Jeremie, …

CORBA
(Common Object Request Broker Architecture)

- Norme rédigée par l'Object Management Group (OMG) : 1989, plus de 800 membres (IBM, Microsoft, ...)
- **Multi-langages** de programmations
- Appel de méthodes à distance (sur objets serveurs : **objets CORBA**)
- Nombreux **services**
- Exemples d'implémentations : ORBacus (Java/C++), MICO (C++), JDK (<=10), …

Historique des intergiciels



Services et interfaces

- Un **service**, c'est quoi ?
  - ***"un comportement défini par contrat, qui peut être implémenté et fourni par un composant pour être utilisé par un autre composant, sur la base exclusive du contrat"***

  (Bieber and Carpenter, "Introduction to Service-Oriented Programming")
- Mise en œuvre
  - Un service est accessible via une ou plusieurs **interfaces**.
  - Une **interface** décrit l'interaction entre client et fournisseur du service.
    - Point de vue *opérationnel* : définition des opérations et structures de données qui contribuent à la réalisation du service.
    - Point de vue *contractuel* : définition du contrat entre client et fournisseur.

Définition d'interfaces

- Partie "opérationnelle"
  - *Interface Definition Language* (**IDL**)
    - Pas de standard, mais s'appuie sur un langage existant
    - IDL propre à CORBA
    - Java pour Java RMI
- Partie "contractuelle"
  - Plusieurs niveaux de contrats
    - Sur la forme : conformité syntaxique
    - Sur le comportement (1 méthode) : conformité sémantique
    - Sur les interactions entre méthodes : synchronisation
    - Sur les aspects non fonctionnels (performances, …) : QoS

Intergiciels orientés objet

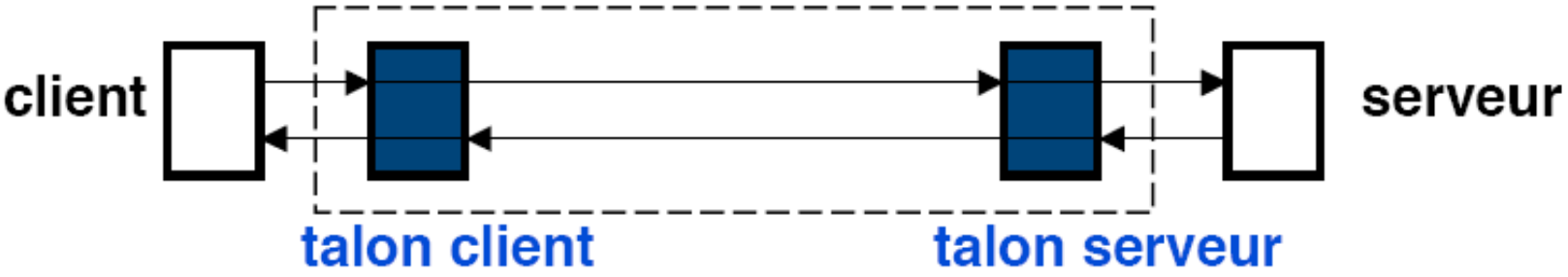
- Objet réparti / Objet serveur / Servant :

  objet manipulable simultanément par plusieurs clients (comme en local) bien que situé (à distance) sur le serveur
- Comment un client peut-il manipuler un objet serveur comme s'il en disposait en local ?

  → Talons

Talons

- A partir d'une définition d'interface (en IDL), production du **talon client** et du **talon serveur** pour un langage donné.
- Talon : relai (**proxy**), un passage obligatoire pour communiquer avec une entité distante



Talons client et serveur

- Talon client : procédure d'interface du site client

  Stub : **souche**.

- Talon serveur : procédure sur le site serveur

  Skeleton : **squelette**.

- Fonctions des talons :
  - Emballage/déballage des paramètres/résultats.
  - Conversion de données.
  - Gestion des processus exécutants (côté serveur).
  - Gestion des erreurs.

→ **Masquage de l'hétérogénéité**

Talons
Exemple avec RPC

Talon client (souche)

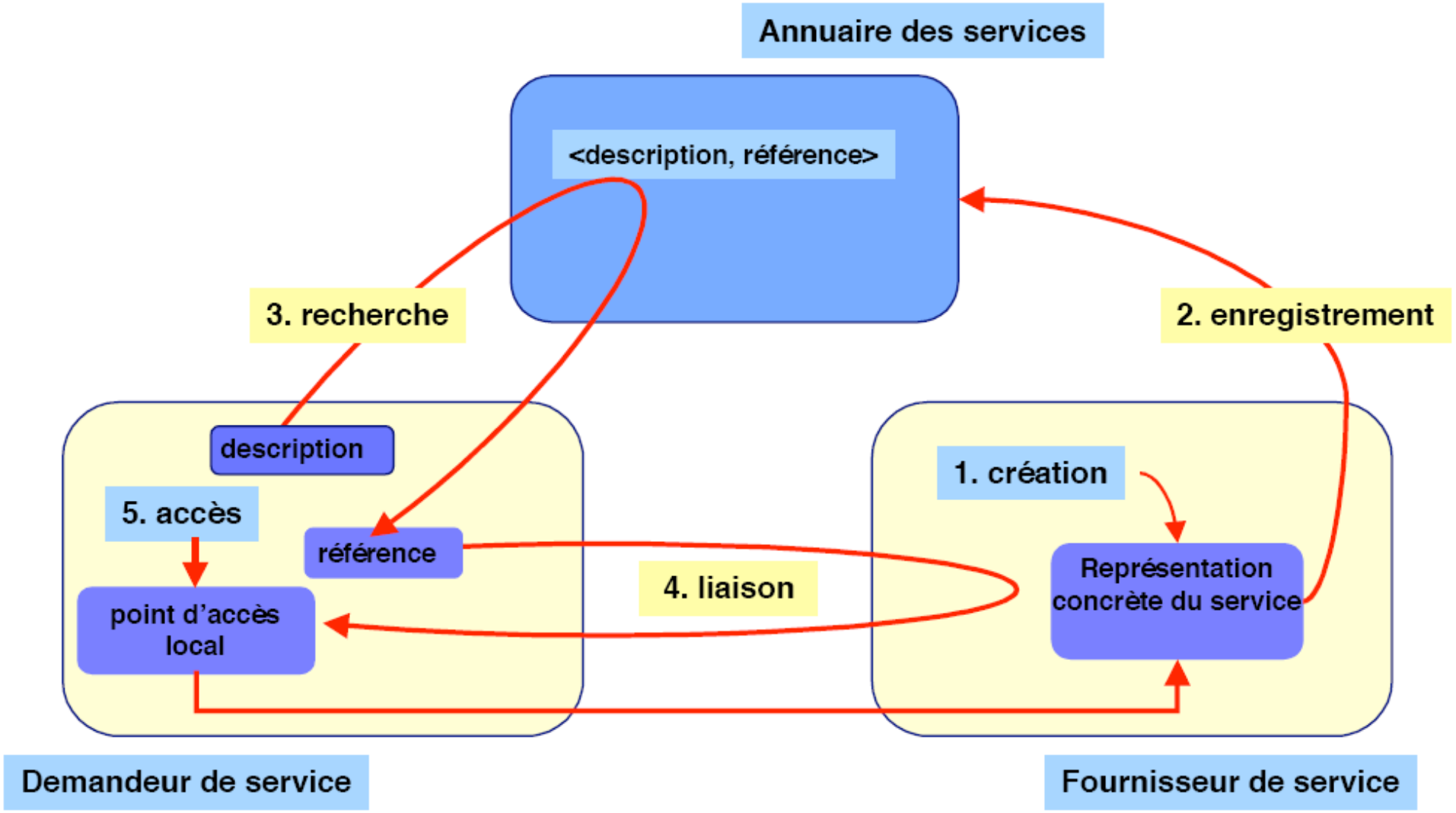
1 Reçoit l'appel local
2 Emballe les paramètres
3 Exécute l'appel distant
4 Place le client en attente

9 Reçoit et déballe résultats
10 Les retourne au client

Talon serveur

5 Reçoit le message d'appel
6 Déballe les paramètres
7 Fait exécuter la procédure
8 Emballe, transmet résultat

Paradigme SOA
Architecture Orientée Services



Exemple
Intergiciels orientés objets

- Côté serveur
  - Création (d'un objet serveur (ou servant))
  - Enregistrement auprès d'un annuaire
- Côté client
  - Recherche auprès de l'annuaire (à partir du nom symbolique)
  - Récupération de la référence distante (de l'objet serveur)
  - Utilisation (invocation de méthodes distantes de cet objet)

La suite

- Java RMI
- Algorithmes répartis
- [pour vous : CORBA]

Annexes

Comparatif d'intergiciels

	RPC	RMI	CORBA	.NET Remoting	SOAP
Qui	SUN/OSF	SUN	OMG	MicroSoft/ECMA	W3C
Plate-formes	Multi	Multi	Multi	Win32, FreeBSD, Linux	Multi
Langages de Programmation	C, C++, …	Java	Multi	C#, VB, J#, …	Multi
Langages de Définition de Service	RPCGEN	Java	IDL	CLR	XML
Réseau	TCP, UDP	TCP, HTTP, IIOP customisable	GIOP, IIOP, Pluggable Transport Layer	TCP,HTTP, *IIOP*	RPC,HTTP
Marshalling		Sérialisation Java	Représentation IIOP	Formatteurs Binaire, SOAP	SOAP
Nommage	IP+Port	RMI, JNDI,JINI	CosNaming	IP+Nom	IP+Port, URL
Intercepteur	Non	depuis 1.4	Oui	Oui CallContext	Extension applicative dans le header
Extra		Chargement dynamique des classes	Services Communs Services Sectoriels	Pas de Chargement dynamique des classes	

Annexes

Schémas de conception

(design patterns)

Schémas/Patrons de conception (*design patterns*)

- Schémas de conception utilisés
  - Proxy
  - Adapter (Wrapper)
  - Interceptor
- Autres Schémas de conception utilisés
  - Fabrique

Schéma *Proxy* (Mandataire)

- Contexte
  - Applications reparties.
  - Client accède à des services distants.
- Problème
  - Masquer au client
    - Localisation du service, protocole de communication.
  - Propriétés souhaitables
    - Accès efficace et sûr, masquer au client la complexité.

Schéma *Proxy* (Mandataire)

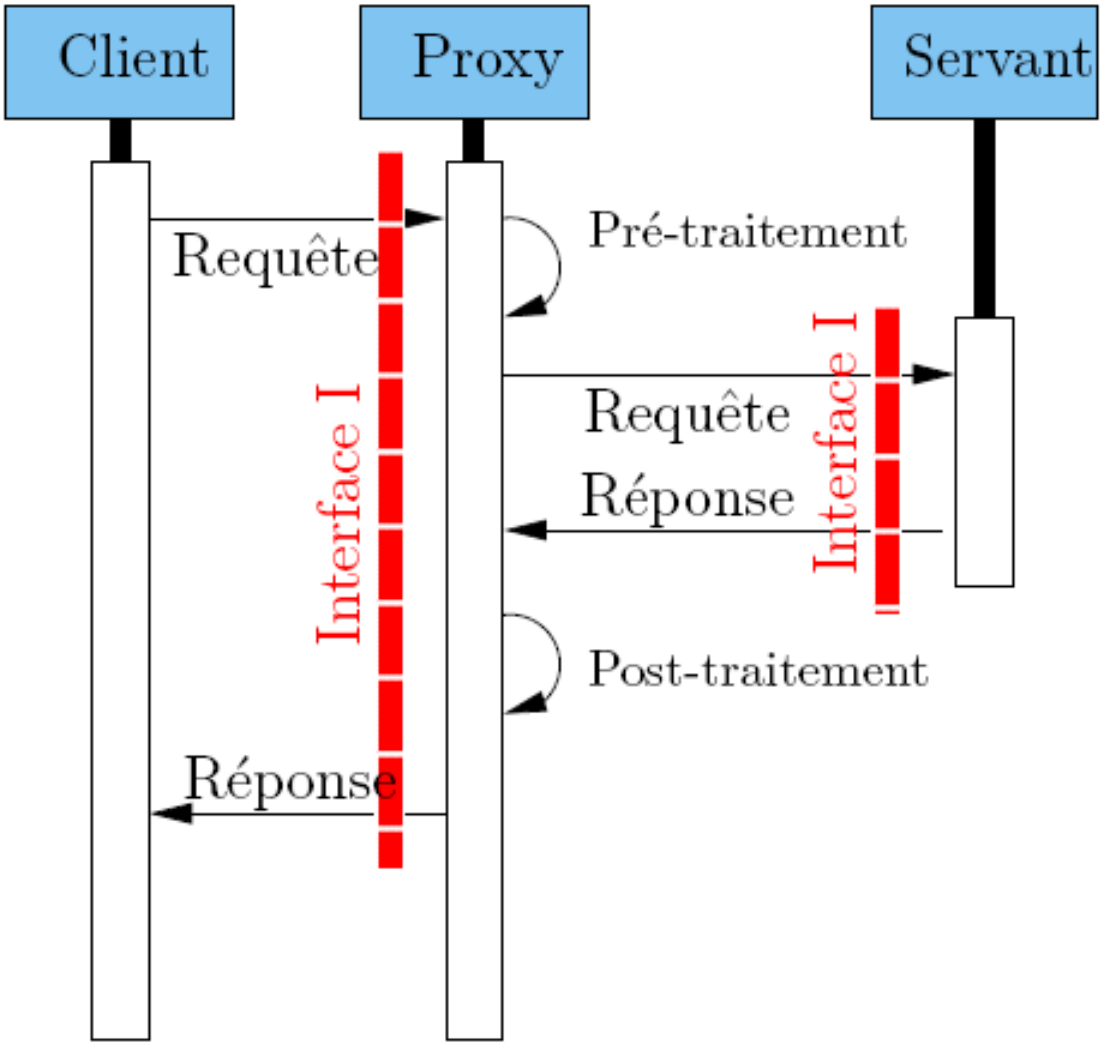


Schéma *Adapter* (ou Wrapper)

- Contexte
  - Services fournis par servants, utilisés par clients et définis par interfaces.
- Problème
  - Modifier l'interface d'un servant existant.
  - Propriétés souhaitables : efficacité ; adaptivité ; généricité.

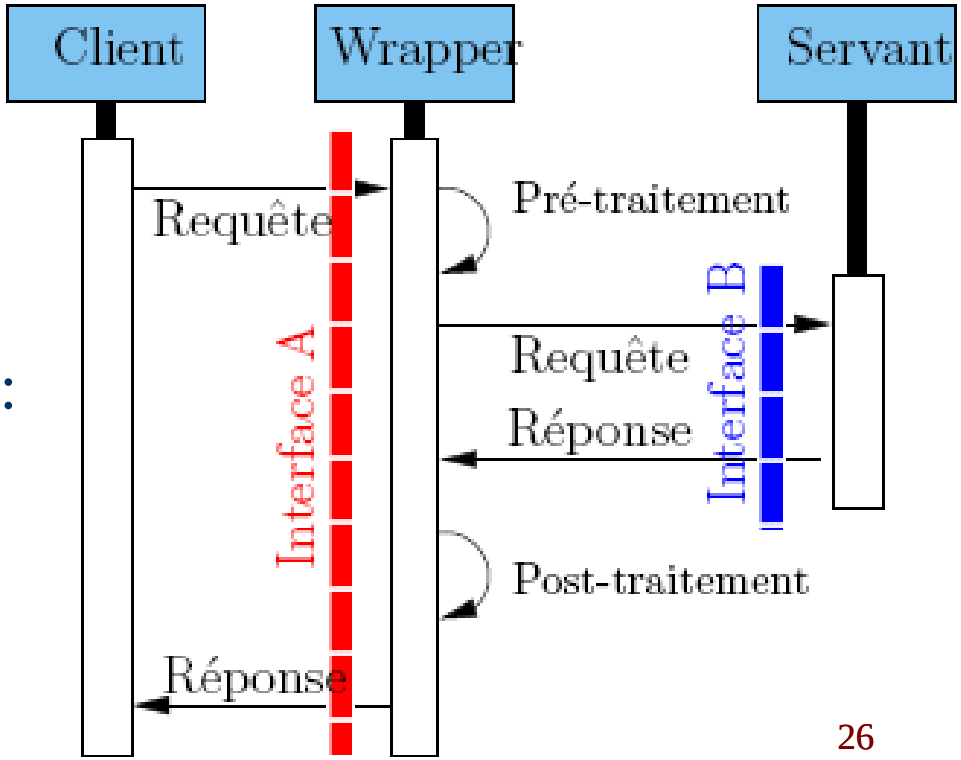
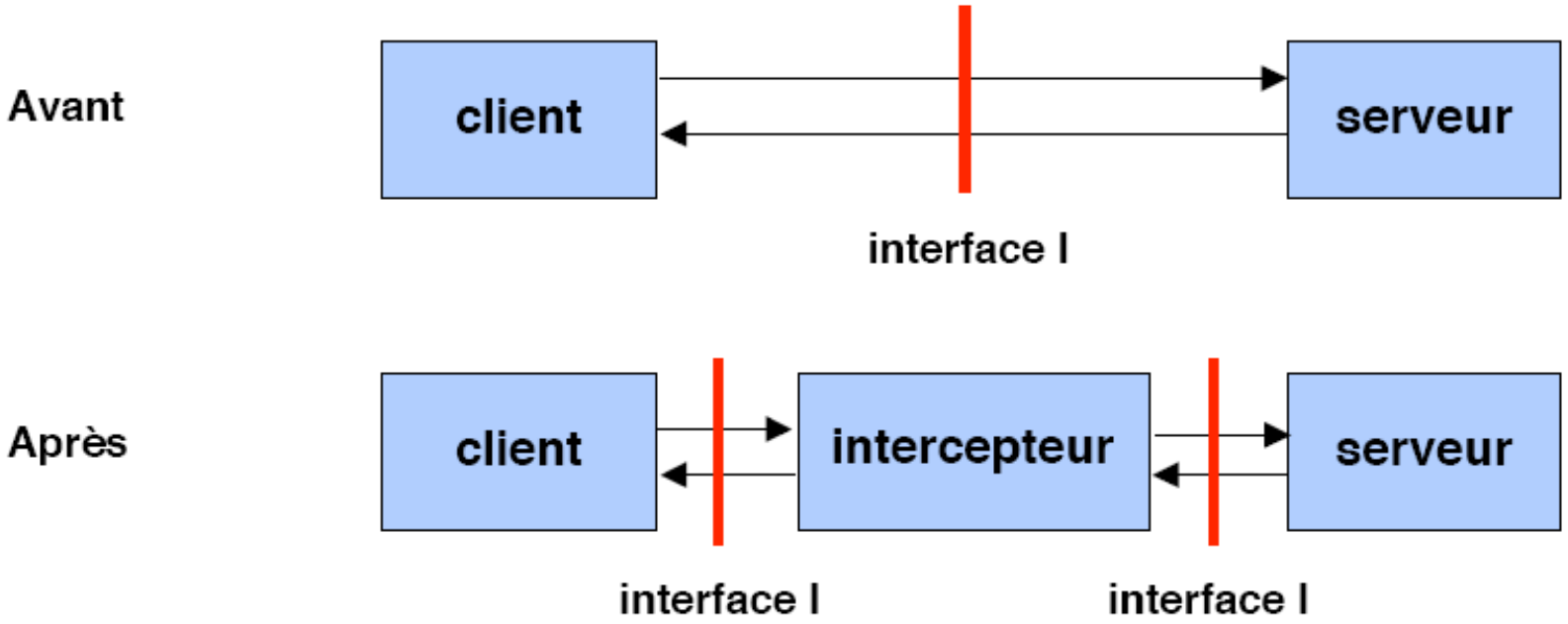


Schéma *Interceptor* (Intercepteur)

- Contexte : cadre général de la fourniture de services
  - Client-serveur, P2P, uni/bi-directionnel, synchrone ou non.
- Problème : transformer le service
  - Modifier la requête, modifier la réponse, ajouter un traitement, rediriger la requête, prélever des mesures, …
- Contraintes
  - Pas de modification des clients et serveurs.
  - Doit être "transparent" (ne pas modifier l'interface).
  - Ajout/suppression dynamique de service.

Schéma *Interceptor* (Intercepteur)

- Interposition d'un traitement entre client et serveur.



Comparaison des schémas

- *Adapter vs. Proxy*
  - Adapter et Proxy ont une structure similaire.
  - Proxy préserve l'interface ; Adapter transforme l'interface.
  - Proxy accès souvent distant ; Adapter accès souvent local.
- *Adapter vs. Interceptor*
  - Adapter et Interceptor ont une fonction similaire.
  - Adapter transforme l'interface.
  - Interceptor transforme la fonction (voire la cible).
- *Proxy vs. Interceptor*
  - Proxy est une forme simplifiée d'Interceptor.
  - Ajouter un Interceptor à un proxy → *Smart Proxy*.

Schéma *Factory* (Fabrique)

- Contexte
  - Application = ensemble d'objets.
- Problème
  - Créer à distance et dynamiquement des instances d'une classe d'objets.
  - Propriétés souhaitables :

    Instances paramétrables ; évolution facile (rien "en dur").
- Solutions
  - Factory factory : création de créateurs paramétrés.
  - Abstract factory : interface et organisation génériques de création d'objets.

    Création effective déléguée à des fabriques concrètes dérivées.

Annexes

RPC

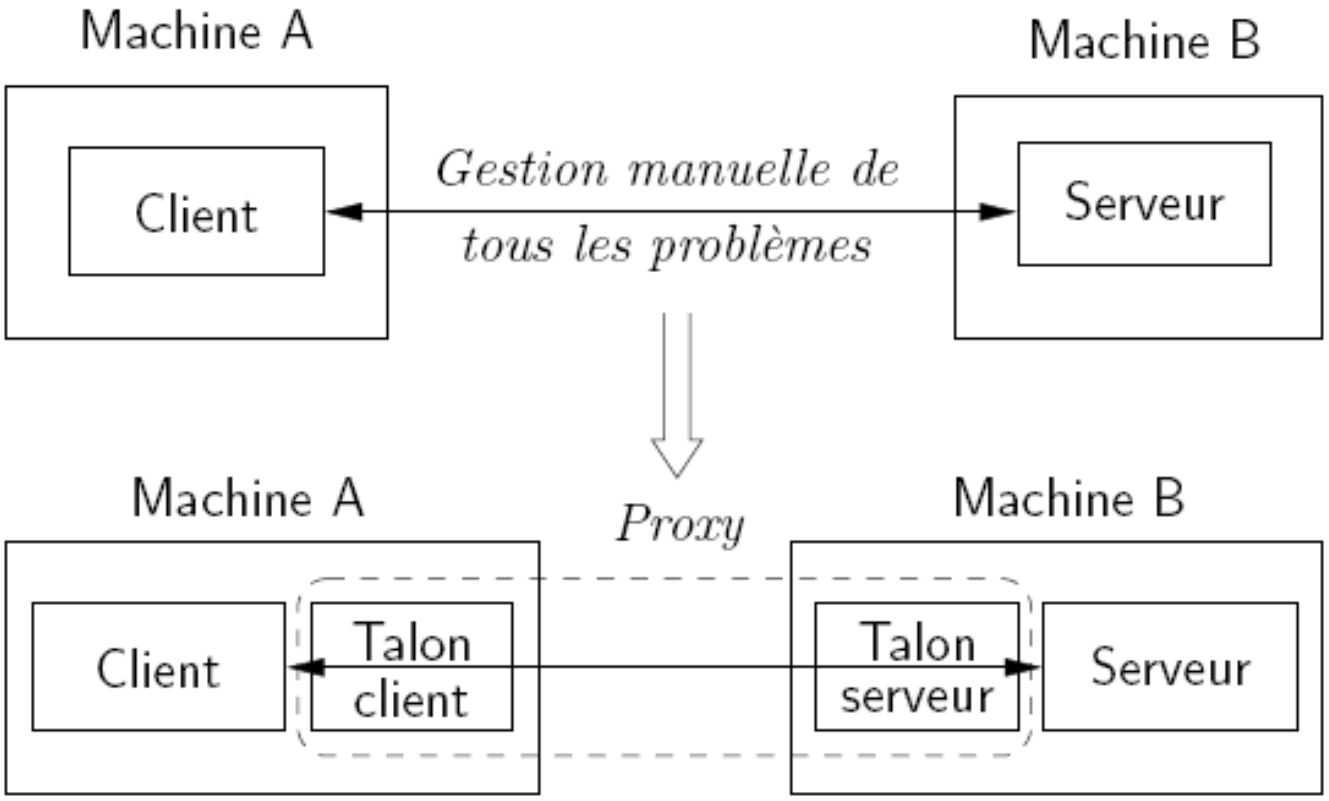
Un petit exemple

Principe du RPC

- Transparence réseau assuré par les talons
  - Interception local de l'appel de procédure par la souche
  - Emballage (empaquetage) des arguments et transport vers le serveur
  - ...

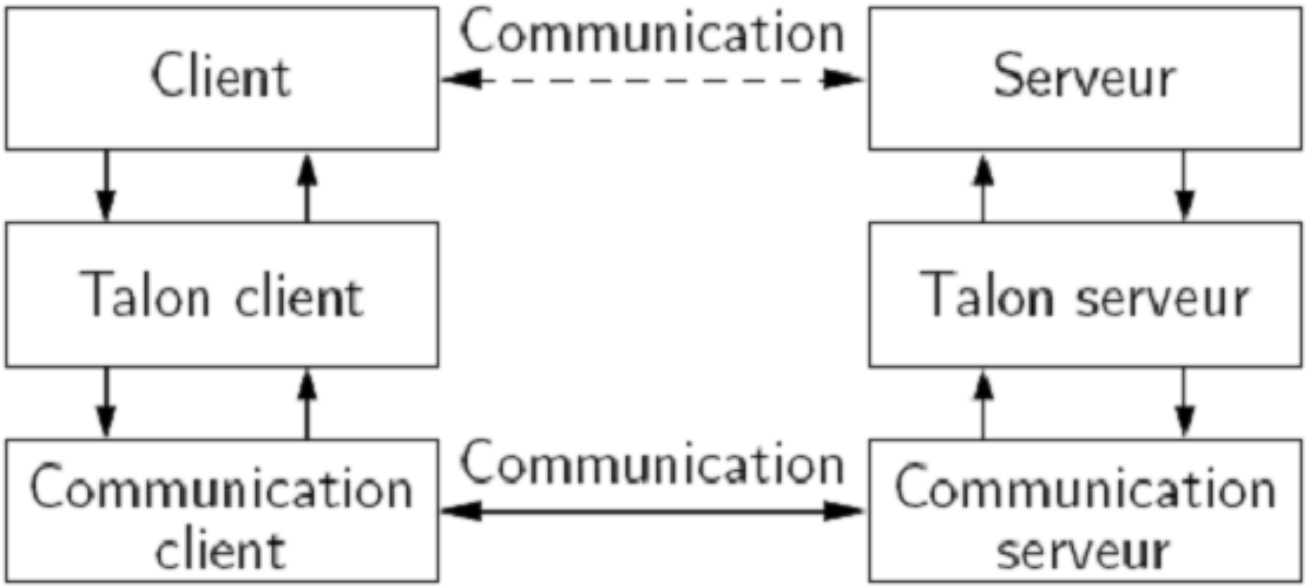
SUN RPC

- Idée : génération automatique du code commun à tous les RPC



SUN RPC

- Vision en couches



SUN RPC

Talon client

1 Reçoit l'appel local
2 Emballe les paramètres
3 Crée un identificateur
4 Exécute l'appel distant
5 Place le client en attente

10 Reçoit et déballe résultats
11 Les "retourne" au client

Talon serveur

6 Reçoit le message d'appel
7 Déballe les paramètres
8 Fait exécuter la procédure
9 Emballe, transmet résultat

SUN RPC

- Sérialisation des données via XDR
  - XDR = eXternal Data Representation
- Transport via TCP ou UDP
- Déploiement : accès aux services RPC via le *port mapper* qui écoute sur le port 111
- Outils : rpcgen, rpcinfo

SUN RPC

- Un service réseau est fourni par plusieurs programmes
- Un programme fourni plusieurs procédures
- Un client appelle une procédure d'un programme
- Une procédure est identifiée par 3 entiers positifs
  - numéro du programme
  - numéro de version du programme
  - numéro de procédure au sein du programme
    - réservé au programmeur : 0x20000000 - 0x3FFFFFFF

XDR

- RPC basé sur l'échange de message XDR
- XDR (eXternal Data Representation)
  - Représentation portable des données à travers le réseau
  - Définition de type de données pour les arguments des RPCs
  - Syntaxe proche du C
    - void, int, float, double, ...
    - enum, struct, string, ...
    - tableaux …

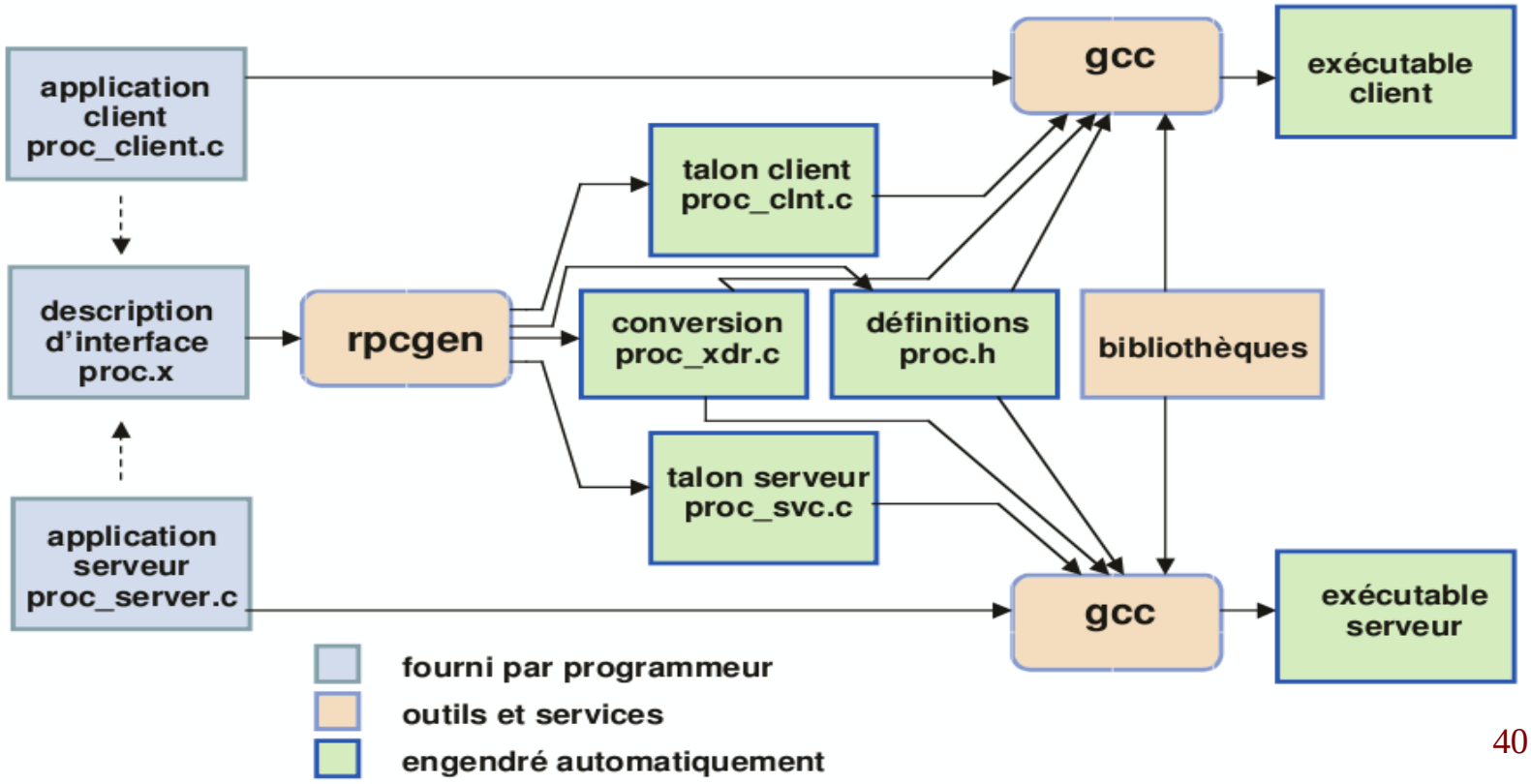
Exemple

- Interface "calculation.x"

```
struct two_int {
    int a;
    int b;
};

program CALCULATION_PROG {
    version CALCULATION_VERS {
int sum(two_int)=1;
    }=1;
}=0x20000000;
```

Compilateur : rpcgen

- Génération des talons client et serveur.
- Génération des fonctions de traduction XDR.



Exemple

- Code fourni par le programmeur
  - client : “calculation_client.c”
  - serveur : “calculation_serveur.c“
    - implantation du service défini dans “calculation.h”
    - fonction main() fournie dans “calculation_svc.c”
  - Génération automatique d'un exemple de code client / serveur avec “rpcgen -a”
- Compilation en vrai...

```
rpcgen  calculation.x
cc -g    -c -o calculation_clnt.o calculation_clnt.c
cc -g    -c -o calculation_xdr.o calculation_xdr.c
cc -g    -c -o calculation_client.o calculation_client.c
cc -g     -o calculation_client  calculation_clnt.o calculation_xdr.o calculation_client.o -lnsl
cc -g    -c -o calculation_svc.o calculation_svc.c
cc -g    -c -o calculation_server.o calculation_server.c
cc -g     -o calculation_server  calculation_svc.o calculation_xdr.o calculation_server.o -lnsl
```

Exemple

- Extrait du code généré "calculation.h"

```
struct two_int {
int a;    int b;
};

typedef struct two_int two_int;

#define CALCULATION_PROG 0x20000000
#define CALCULATION_VERS 1

extern  int * sum_1(two_int *, CLIENT *);
extern  int * sum_1_svc(two_int *, struct svc_req *);
extern int calculation_prog_1_freeresult (SVCXPRT *, xdrproc_t, caddr_t);

/* the xdr functions */
extern  bool_t xdr_two_int (XDR *, two_int*);
```

Exemple

- Code serveur "calculation_server.c"

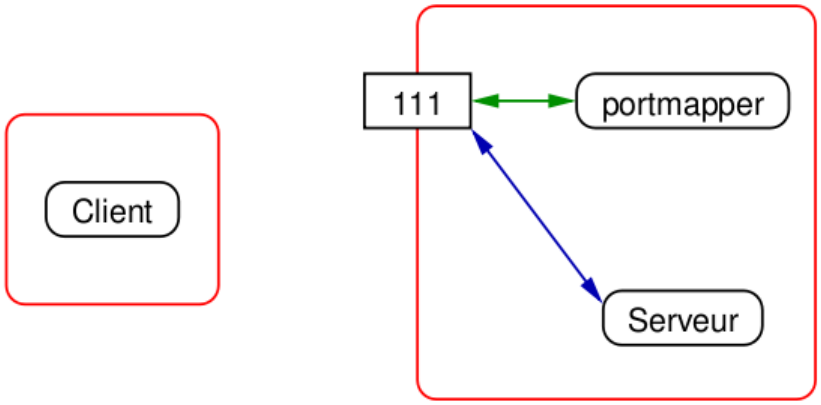
```
#include "calculation.h"

int *  sum_1_svc(two_int *argp, struct svc_req *rqstp)
{
static int  result;
result=argp->a+argp->b;
return &result;
}
```

Exemple

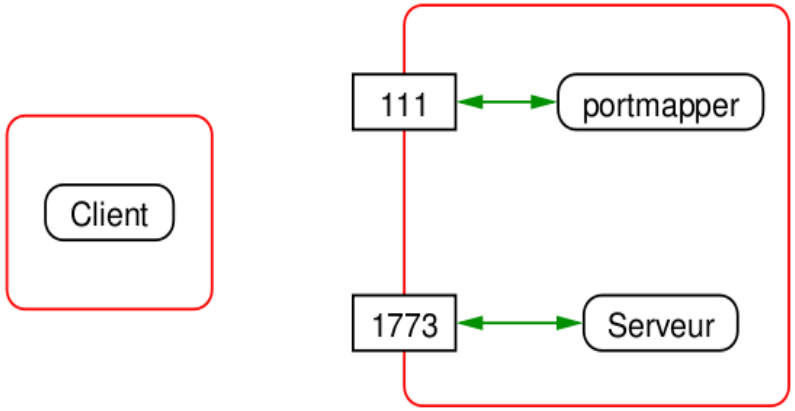
- Code client "calculation_client.c"

```
CLIENT *clnt;
int  *result_1;
two_int  sum_1_arg;
sum_1_arg.a=15;
sum_1_arg.b=35;
clnt = clnt_create (host, CALCULATION_PROG, CALCULATION_VERS, "udp");
if (clnt == NULL) {
  clnt_pcreateerror (host);
  exit (1);
}
result_1 = sum_1(&sum_1_arg, clnt);
if (result_1 == (int *) NULL) {
  clnt_perror (clnt, "call failed");
}
else {
  printf("%d+%d=%d\n",sum_1_arg.a,sum_1_arg.b,*result_1);
}
clnt_destroy (clnt);
```

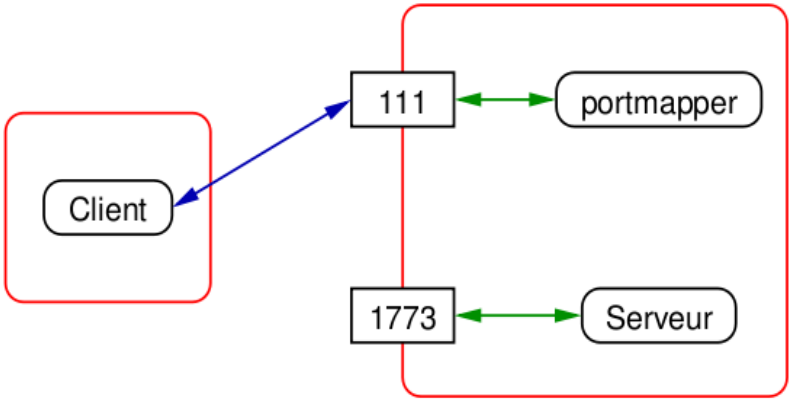
Déploiement : port mapper



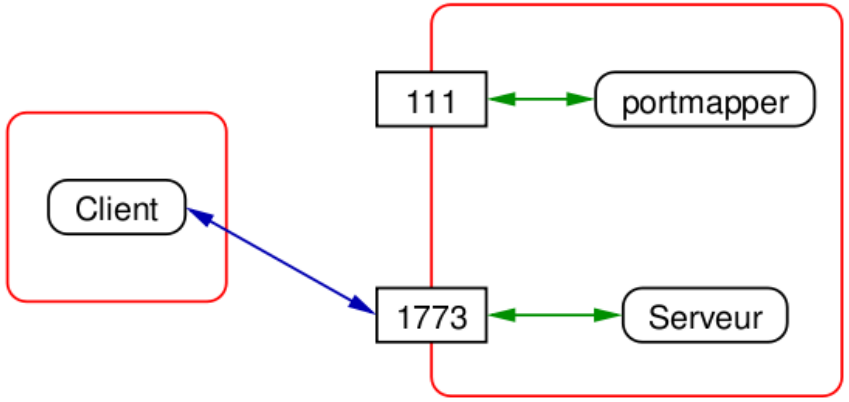
Publication du programme (RPC)



Le programme écoute sur le port 1773



Demande de résolution (RPC)



Appel de la fonction recherchée