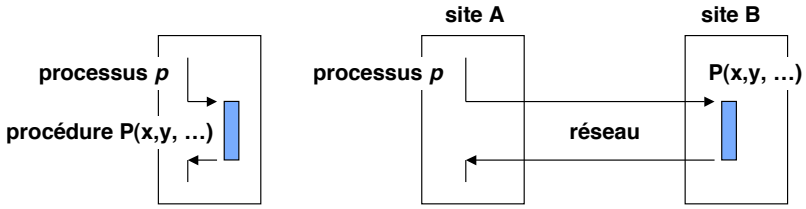
Appel de procédure à distance

Sacha Krakowiak
Université Joseph Fourier
Projet Sardes (INRIA et IMAG-LSR)

http://sardes.inrialpes.fr/people/krakowia

Appel de procédure à distance
Remote Procedure Call (RPC)

- **Outil de haut niveau pour la réalisation du schéma client-serveur**
- **Principe**

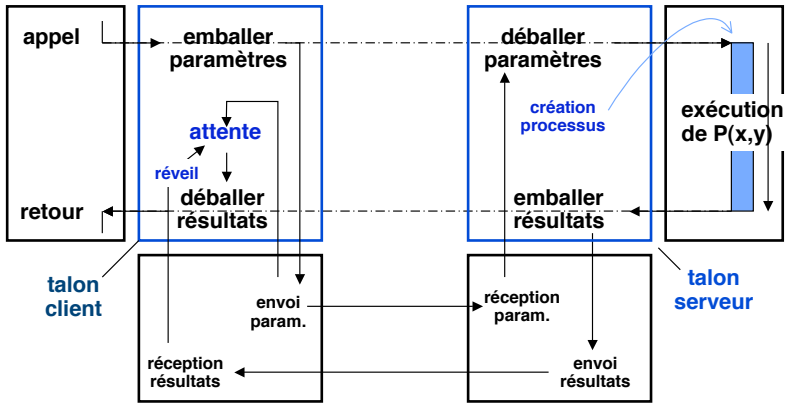


L'effet doit être identique dans les deux cas, vu de *p* Dans la pratique, ce n'est pas vrai en toute rigueur

Appel de procédure à distance

- **Avantages attendus**
  - Forme et effet identiques à ceux d'un appel local
    - Applications non modifiées lors du passage à l'appel distant
    - Mise au point locale avant répartition
    - Simplicité d'utilisation
  - Abstraction
    - Indépendance par rapport aux protocoles de communication
    - Réutilisation, y compris en environnement hétérogène
- **Problèmes**
  - Sémantique complexe en cas de panne
    - Les processus client et serveur peuvent tomber en panne indépendamment
    - Le réseau introduit une incertitude
  - Restrictions sur les paramètres
    - Passage de structures complexes, possible mais compliqué

Appel de procédure à distance : principe de réalisation



logiciel de communication (*sockets*)

[Birrell & Nelson, 1984]

Fonctions des talons (*stubs*)

- **Talon client**
  - Représente le serveur sur le site client
  - Reçoit l'appel local
  - Emballe les paramètres
  - Crée un identificateur unique pour l'appel
  - Exécute l'appel distant
  - Met le proc. client en attente
  - Reçoit et déballe les résultats
  - Exécute le retour vers l'appelant (comme retour local de procédure)
- **Talon serveur**
  - Représente le client sur le site serveur
  - Reçoit l'appel sous forme de message et déballe les paramètres
  - Crée ou sélectionne un processus (ou *thread*) et lui fait exécuter la procédure
  - Emballe les résultats et les transmet à l'appelant

En outre, les talons doivent détecter et traiter les erreurs (délais de garde)

Appel de procédure à distance
Problèmes de mise en œuvre

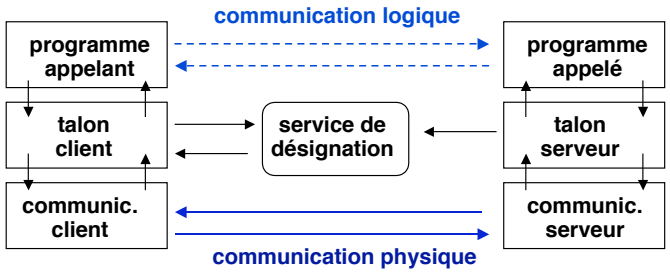
- Comment le client connaît-il l'adresse du serveur ?
- Comment fonctionne la transmission des paramètres et résultats
  - Modes de passage (valeur, référence ?)
  - Structures complexes
  - Représentation, emballage/déballage
  - Traitement de l'hétérogénéité
- Comment sont traitées les erreurs ?
  - Hypothèses sur les erreurs, mode de détection
  - "Sémantique" de l'appel en cas d'erreur
- Construction des talons
  - Génération automatique
- Gestion à l'exécution
  - Lancement et arrêt du serveur, etc.

RPC : Désignation et liaison

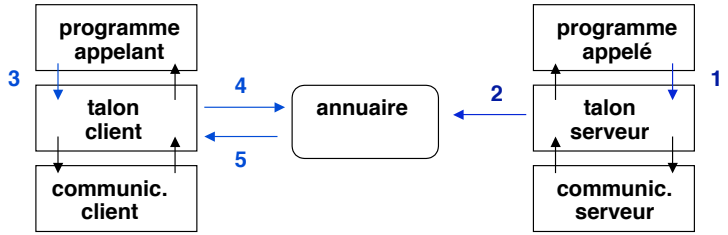
- **Objets à désigner**
  - Procédure appelée, site serveur
  - Propriétés souhaitées : désignation indépendante de la localisation
    - possibilité de reconfiguration (pannes,régulation de charge)
- **Moment de liaison**
  - Liaison précoce (statique) ou tardive (dynamique)
  - Statique : localisation du serveur connue à la compilation
  - Dynamique : localisation non connue à la compilation
    - Désignation symbolique des services (non liée à un site d 'exécution
    - Possibilité d 'implémentation ou de sélection retardée

RPC : désignation et liaison

- Liaison statique : pas d'appel à un serveur de noms (ou appel à la compilation
- Liaison au premier appel : consultation du serveur de noms au premier appel seulement
- Liaison à chaque appel : consultation du serveur de noms à chaque appel



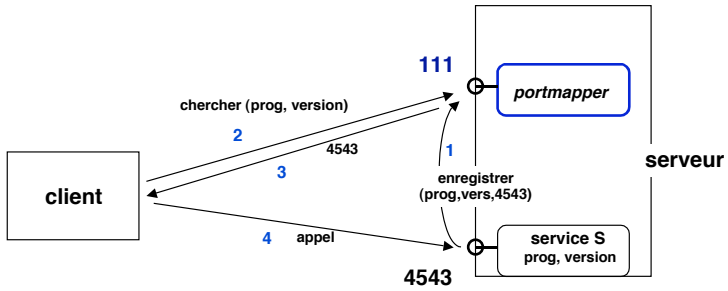
RPC : désignation et liaison utilisant un annuaire



- 1, 2 : le serveur s'enregistre auprès de l'annuaire avec <nom, adr. serveur, n° port>
- 3, 4, 5 : le client consulte l'annuaire pour trouver <adr. serveur, n° port> à partir de <nom>
- L'appel peut alors avoir lieu
- Schémas plus élaborés : attributs (critères de choix)

RPC : désignation et liaison utilisant le *portmapper*

- Si le serveur est connu (cas fréquent) : on peut utiliser un service de nommage local au serveur, le *portmapper*
- Un service enregistre le n° de port de son veilleur auprès du *portmapper* et le veilleur se met en attente sur ce port
- Le *portmapper* a un n° de port fixé par convention (111)



RPC : gestion des paramètres

Problèmes des paramètres

- Les espaces d'adressage de l'appelant et de l'appelé sont distincts
  - Pas de passage par référence
  - Les pointeurs perdent leur signification (difficulté pour passer des structures complexes)
- Les paramètres sont transmis sur le réseau
  - Représentation sérialisée des structures
- Les machines client et serveur peuvent être hétérogènes
  - Conversion de format

Problèmes de représentation des paramètres

Conventions différentes sur client et serveur

- Exemple : *little endian* vs *big endian*
  - Sens des octets d'une chaîne de caractères

  - Poids fort des nombres à gauche ou à droite

  - Alignement des données sur les frontières de mots
- Conventions de représentation des nombres flottants
- Conventions de représentation des structures complexes

Limitations de taille

- Exemple : entiers sur 64 bits vs 32 bits

RPC : représentation des paramètres

- **Solution normalisée (ASN.1)**
  - Syntaxe abstraite (Abstract Syntax Notation) pour représenter des structure de données
  - Codage indépendant des machines
  - Outils de génération disponibles pour des machines spécifiques
- **Autres solutions**
  - Représentation externe commune. Exemple : Sun XDR (*External Data Representation*) utilisée dans NFS
    - … conversions inutiles si client et serveur de même type
  - Conversion explicite (par le serveur) à partir d'une représentation locale au client
  - Négociation entre client et serveur

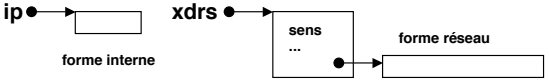
Un exemple de convention de représentation de paramètres : XDR

- **XDR : *External Data Representation* (Sun)**
  - Format de transmission sur réseau
  - Bibliothèque de programmes de conversion
- **Ensemble de routines disponibles pour les types de base (de C)**
  - Données élémentaires
  - Structures simples (tableaux, chaînes, unions)
  - Traitement de pointeurs, gestion de mémoire
- **Utilisation : programmes d'emballage/déballage**
  - Courante : par les générateurs automatiques de talons, dans tous les cas usuels
  - Avancée : directement par le programmeur, pour le traitement de structures de données complexes

XDR : exemples de routines de conversion

Paramètres : pointeurs vers zones contenant les variables sous forme interne et sous forme "réseau" (transmissible). La même routine sert dans les deux sens (un paramètre spécifie le sens).

```
bool_t xdr_int(xdrs, ip)
        XDR *xdrs
        int *ip
```



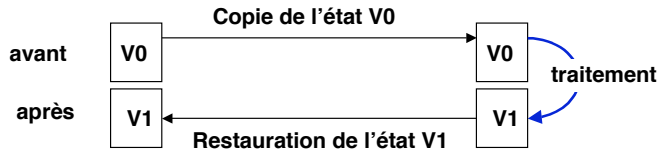

```
bool_t xdr_array (xdrs, arrp, sizep, maxsize, elsize, elproc)
        XDR *xdrs
        char **arrp
        u_int *sizep, maxsize, elsize
        xdrproc_t elproc;
```

arrp : pointeur vers tableau
**sizep* : nombre d'éléments de taille *elsize*
maxsize : nombre max d'éléments
elproc : filtre xdr pour convertir l'élément (exemple : xdr_int pour tableau d'entiers, etc

RPC : passage des paramètres

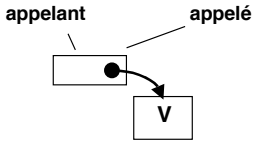
Plusieurs modes de passage sont possibles

- **Passage par valeur**
  - pas de problème
- **Passage par copie-restauration**
  - A l'appel, copie des valeurs des paramètres de l'appelant vers l'appelé
  - Au retour, copie de nouvelles valeurs pour les paramètres de l'appelé vers l'appelant



RPC : passage des paramètres

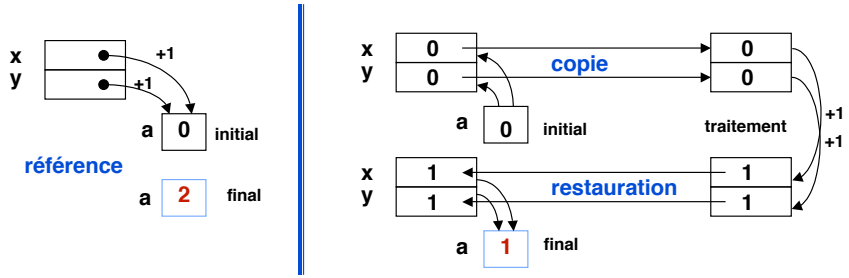
- **Passage par référence (adresse)**
  - Impossible puisque espaces distincts
- **Palliatifs possibles**
  - Interdire l'appel par référence
  - Utiliser l'appel par copie-restauration
    - Équivalent le plus souvent …
    - … mais pas dans tous les cas (*aliasing*)
  - Programmer à la main les procédures d'emballage-déballage pour les structures complexes
  - Nouveaux modèles de communication (mémoire virtuelle répartie) : revient à un espace commun



Passage par copie-restauration et passage par référence

```
procedure double_incr (x, y)
    x := x+1
    y := y+1
```

Quel est l'effet de :
a:= 0; double_incr (a, a) ?



RPC : "sémantique" en cas de panne

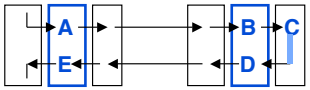
Détection de panne = expiration de délai de garde. On tente de réexecuter l'appel. Combien de fois la procédure a-t-elle été exécutée ?

- **La sémantique dépend des hypothèses de pannes et du mécanisme de reprise**
  - Indéfini (pas d'hypothèses)
  - Au moins une fois (appel répété, plusieurs exécutions possibles, au moins une réussit)
    - acceptable si opération idempotente (2 appels successifs ont le même effet que 1 appel)
  - Au plus une fois (0 ou 1 fois)
    - on évite les exécutions répétées, mais on ne garantit pas le succès
  - Exactement une fois (c'est l'idéal, difficile à atteindre)

RPC : traitement des défaillances

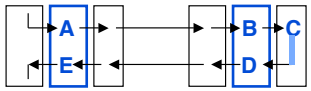
- **Hypothèses de défaillances**
  - Système de communication
    - Congestion du réseau, messages retardés
    - Perte de messages
    - Altération de messages
  - Serveur
    - Défaillance avant l'exécution de la procédure
    - Défaillance pendant l'exécution de la procédure
    - Défaillance après l'exécution de la procédure
    - Défaillance définitive ou reprise possible
  - Client
    - Défaillance pendant le traitement de la requête

RPC : traitement des défaillances



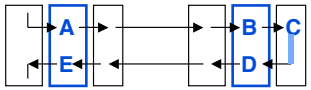
- **Congestion du réseau ou du serveur**
  - Panne transitoire (ne nécessite pas d'action)
  - Détection : expiration du délai de garde A ou D
  - Reprise (délai de garde A)
    - Le talon client (A) réémet l 'appel (même identificateur) sans intervention de l 'application
    - Le service d'exécution (C) détecte que c'est une réémission
      - Appel en cours : aucun effet
      - Retour déjà effectué : réémission du résultat
  - Reprise (délai de garde D) : réémission du résultat
  - Sémantique
    - Si défaillance transitoire : exactement une fois
    - Si défaillance permanente : détection, exception vers application

RPC : traitement des défaillances



- **Panne du serveur après émission de l 'appel**
  - Plusieurs moments possibles
    - Avant B, entre C et D, entre fin traitement et D
    - Traitement : pas fait, partiel, total
  - Détection : expiration du délai de garde A
  - Reprise
    - Le client réémet l'appel dès que le serveur redémarre
    - Sémantique : au moins une fois
      - Le client ne connaît pas l'endroit de la panne
    - On peut faire mieux avec un service transactionnel
      - Mémorise identificateur de requête et état avant exécution

RPC : traitement des défaillances

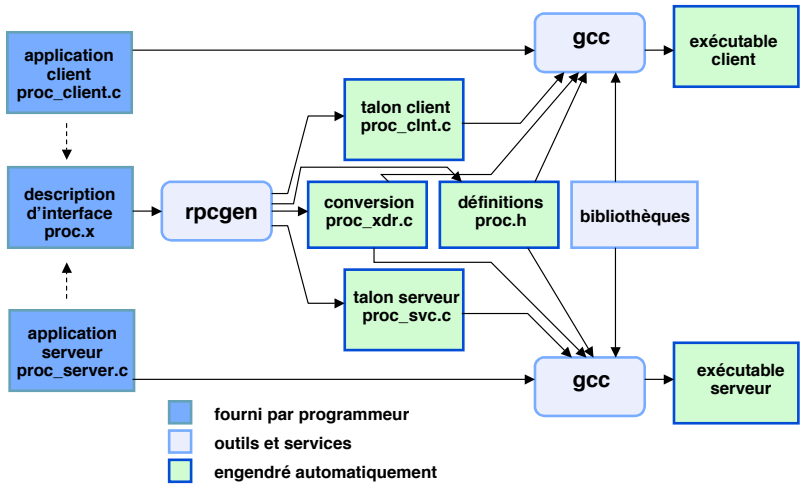


- **Panne du client après émission de l'appel**
  - L'appel est correctement traité
    - Changement d'état du serveur
    - Le processus exécutant courant est déclaré "orphelin"
  - Détection : expiration du délai de garde D, *n* réémissions infructueuses
    - Action du serveur : élimination des orphelins
      - Consomment des ressources
  - Reprise (après redémarrage du client)
    - L'application cliente réémet l'appel (id. différent)
      - Sémantique : au moins une fois
      - Le serveur ne peut pas détecter qu 'il s 'agit d'une répétition
        - Pas d 'incidence si idempotent
      - On peut faire mieux, si le client a un service de transactions

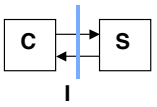
RPC : génération des talons

- **Principe**
  - La structure des talons est bien définie et ils peuvent être créés automatiquement
  - Renseignements nécessaires
    - Dépendants de l'environnement local : procédures de conversion, protocole de communication
    - Dépendants de l'application : formats des paramètres et résultats (pour emballage-déballage) ; c 'est l'interface de la procédure
- **Mise en œuvre**
  - Les informations nécessaires dépendantes de l'application sont décrites dans un langage de définition d'interfaces (IDL)

RPC : génération des talons



Description d'interfaces



- **Interface = "contrat" entre client et serveur**
  - Définition commune abstraite
    - Indépendante d'un langage particulier (adaptée à des langages multiples)
    - Indépendante de la représentation des types
    - Indépendante de la machine (hétérogénéité)
  - Contenu minimal
    - Identification des procédures (nom, version)
    - Définition des types des paramètres, résultats, exceptions
    - Définition du mode de passage (IN, OUT, IN-OUT)
  - Extensions possibles
    - Procédures de conversion pour types complexes
    - Propriétés non-fonctionnelles (qualité de service), non standard)

RPC : exemple de description d'interface

fichier "annuaire.x"

```
/* constantes et types */

const max_nom = 20 ;
const max_adr = 50 ;
const max_numero = 16 ;

typedef string typenom<max_nom>
typedef string typeadr<max_adr>
typedef string
        typenumero<max_numero>

struct personne {
        typenumero numero ;
        typenom nom ;
        typeadr adresse ;
        } ;

typedef struct personne personne ;
```

```
/* description de l 'interface */

program ANNUAIRE {
   version V1 {
        void INIT(void) = 1;
        int AJOUTER(personne p) = 2 ;
        int SUPPRIMER (personne p) = 3 ;
        personne CONSULTER (typenom nom) = 4 ;
   } = 1 ;
} = 0x23456789 ;
```


Utilisation de *rpcgen*

Fichiers engendrés à partir de annuaire.x :

annuaire.h	include
annuaire_clnt.c	talon client
annuaire_svc.c	talon serveur
annuaire_xdr.c	proc. conversion

Aides à la construction :

annuaire_client.c	modèle de programme client
annuaire_server.c	modèle de programme client
makefile.annuaire	modele de makefile

Illustration : annuaire_xdr.c

```
bool_t xdr_typeadr(xdrs, objp)
   register XDR *xdrs;
   typeadr *objp;
{
   register long *buf;

   if (!xdr_string(xdrs, objp, max_numero))
          return (FALSE) ;
   return (TRUE) ;
}
```

```
bool_t xdr_typenom(xdrs, objp)
...
bool_t xdr_typenumero(xdrs, objp)
...
bool_t xdr_personne(xdrs, objp)

  register XDR *xdrs;
  typeadr *objp;
{
  register long *buf;

   if (!xdr_typenumero(xdrs, &objp->numero))
          return (FALSE) ;
   if (!xdr_typenom(xdrs, &objp->nom))
          return (FALSE) ;
   if (!xdr_typeadr (xdrs, &objp->adresse))
          return (FALSE) ;
   return (TRUE) ;
}
```

Utilisation de *rpcgen*

Exemple : modèle de programme client : annuaire_client.c

```
…
main(argc, argv)
int argc ; char * argv ;
{
        CLIENT *clnt ;
        char * host
…
        if (argc <2) {
                printf("usage: %s server_host\n", argv[0]) ;
                exit(1)
        {
        host = argv[1] ;

        clnt = clnt_create (host, ANNUAIRE, V1, "netpath") ;   /* "poignée" d 'accès au serveur */
        if (clnt == (CLIENT *) NULL {
                {clnt_pcreateerror(host) ;
                exit(1) ;
        {
…                                                              /* saisir paramètres */
        result_2 = ajouter_1(&ajouter_1_arg, clnt)
        if (result_2 == (int *) NULL) {
                clnt_perror(clnt, "call failed") ;
        {
…                                                              /* afficher résultats */
}
```