Client /serveur

« Jeux 3D multi-joueurs sur PC

V. Gal/A. Topol

C/S : bibliographie

- Technologies et architectures internet – P.Y. Cloux, D. Doussot, A. Géron – Editions Dunod – 2ème édition 2002
- Cours d'introduction : modèles d'interactions pour le client serveur – G. Florin – CNAM
- Cours : Programmation client serveur : sockets – RPC – S. Krakowiak – Université J. Fourier
- Remote Procedure Call – F. Crevola –article paru dans GNU/Linux & Hurd Magazine France – septembre 2000
- RPC : Remote Procedure Call – Protocl specification version 2 – http://www.ietf.org/rfc/rfc1057.txt
- Divers articles sur le net

Client/serveur

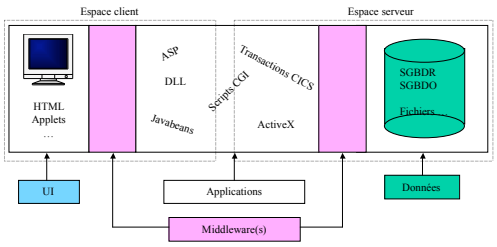
- Bibliographie
- Introduction
- Modèle général des architectures client-serveur
- Les architectures C/S
- Mise en œuvre
  - Programmation C/S (sockets – RPC)
  - Autre exemple de RPC (Sun RPC)
- Conclusion

C/S : introduction

- Interface utilisateur (UI) souvent graphique (GUI)
- Applications
- Manipulation des données
- Différentes architectures liées à ses composants

C/S : modèle général

Représentation serveur non aussi monolithique



C/S : les architectures

- Le modèle à messages
- Le modèle à événements
- Le modèle de base de données (ODBC)
- Le modèle en RPC
- Le modèle à base de code mobile
- Le modèle à mémoire virtuelle partagée répartie

C/S : les architectures
Le modèle à messages (1)

- Mode de communication basique par messages asynchrones
  - Asynchrone : émission non bloquante, réception bloquante
  - Communication par messages typés
  - Communication sur les processus destinataires ou sur boites à lettres (portes) de communication
- Exemples
  - Interfaces des piles réseaux : couche transport (sockets, TLI, NetBEUI)
  - Interfaces de systèmes répartis (Chorus, Mach)
  - Les MOM : mode messages asynchrones + mode asynchrone avec persistance (files de messages)

C/S : les architectures
Le modèle à messages (2)

- MOM (communication à files de messages)
  - Deux modes de fonctionnement :
    - Mode message (habituel)
    - Mode file de message (caractéristique)
  - Spécificité des messages
    - Identification unique
    - Structure : Entête, attributs, données
    - Paramètres d'acheminement : durée de vie, priorité, sécurité, confidentialité et contrôle d'accès
  - Apports des files de messages
    - Style synchrone : bloquant, terminant et avec acquittement. *Inconv. : si serveur ne répond pas => arrêt total du client*
    - Style asynchrone : non bloquant, acquittement et terminaison par protocole. *Inconv. : assez vite bloquant car possibilité de saturation du site distant ne pouvant tenir le rythme des services demandés (mauvais serveur)*
    - => utilisation des files de messages

C/S : les architectures
Le modèle à messages (3)

- MOM (communication à files de messages) suite
  - Files de messages : structures de données de stockage de messages asynchrones
    - Identification unique
    - Propriété de persistance (style transactionnel)
    - Partagées par les applications
    - Mode de réception variable selon le destinataire
  - Exemples :
    - IBM MQSeries, Microsoft Message Queue Server, …
  - Les 4 types de messages dans IBM MQSeries : Datagram, Request, Reply, Report.
  - Les 11 primitives : MQCONN, MQOPEN, MQGET, MQPUT, MQPUT1, MQCLOSE, MQDISC, MQINQ, MQSET, MQCMIT, MQBACK.
  - Avantages : interactions améliorées, meilleures interopérabilité et portabilité par l'isolement de l'application par rapport aux logiciels sous-jacents, infrastructures client serveur à MOM assez simples par rapport à aux autres styles de middleware
  - Inconvénients : sémantique limitée

C/S : les architectures
Le modèle à événements (1)

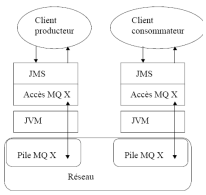
- **Fondé sur :**
  - Notions d'événements et de réactions
  - Attachement
  - Communication anonyme
- **Pourquoi ?**
  - Définir une approche "asynchrone" de communication
  - À base de programmation événementielle
  - Dans des environnements disposant par exemple de services d'événements (CORBA) de composants (CORBA, JMS Java Messaging Message) dans les environnements objets répartis
- **Schéma général**
  - Des sources produisent des événements : publish
  - Des consommateurs s'abonnent à des événements : subscribe

C/S : les architectures
Le modèle à événements (2)

- **Sémantique de consultation des boites à lettres :**
  - Mode "Pull" : le consommateur consulte sa boite pour traiter les événements en attente => contrôle moment de traitement des événements
  - Mode "Push" : le consommateur attache une méthode à chaque type d'événement => occurrence d'un événement déclenche automatiquement le traitement associé
- **Architecture du service d'événement centralisé**
  - Le service centralisé de gestion des événements dialogue en point à point avec les clients producteurs ou consommateurs
- **Architecture du service d'événement réparti**
  - Mode "Snowflake" : les courtiers coopèrent pour offrir un service réparti de gestion des événements
- **Architecture du service d'événement à bus de messages**
  - Les courtiers coopèrent au moyen d'un protocole à diffusion des événements

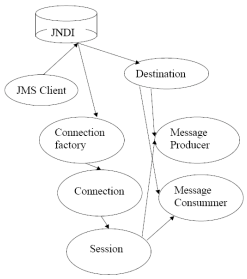
C/S : les architectures
Le modèle à événements (3)

- **Exemple d'environnement à événements : JMS**
  - API Java offrant un accès unique aux différents protocoles de files de messages (MOM MQSeries, …) : gestion événementielle, mode "publish"/"subscribe"



C/S : les architectures
Le modèle à événements (4)

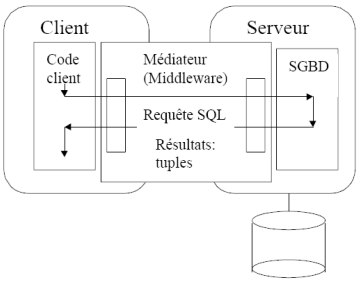
- **Architecture**



C/S : les architectures
Le modèle de données (1)

- **Fonctions du client**
  Traitements applicatifs non lié au stockage des données :
  - Dialogue avec l'utilisateur
  - Traitements divers
  - …
- **Fonctions du serveur**
  Stockage de données
  - Interprétation des requêtes
  - Optimisation des requêtes
  - Gestion de la persistance
  - Gestion de la sécurité des données
  - …

C/S : les architectures
Le modèle de données (2)

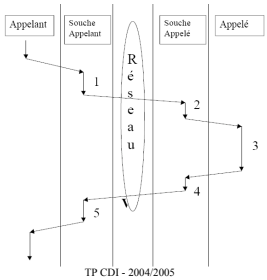


C/S : les architectures
Le modèle en RPC traditionnel (1)

- **"Remote Procedure Call"**
  - Permet de présenter les interactions distantes sous une forme (syntaxe) et un effet (sémantique) analogue à celui d'un appel de procédure local
  - Opération appelée (le serveur) = procédure située sur un site distant
  - Programme appelant (le client) = demande l'exécution
  - Interaction synchrone comme appel de procédure habituel
  - Offre un changement important / à la structuration au mode message (ne permet qu'un déclenchement asynchrone d'action à distance définie par une séquence de code)

C/S : les architectures
Le modèle en RPC traditionnel (2)

- **Birrell et Nelson 1984**



C/S : les architectures
Le modèle en RPC traditionnel (3)

- **Souche ou talon (stub)**
  - Deux procédures intermédiaires (chez le client et le serveur) pour l'adaptation de l'invocation à l'environnement réparti transformant un appel local en appel distant
  - **Souche appelant**
    - Reçoit l'appel en mode local
    - Emballe les paramètres d'appel dans un message de requête (marshalling)
    - Emet le message de requête
    - Se met en attente du message de réponse
    - Déballe les paramètres réponse
    - Retourne les paramètres réponse comme un retour local
  - **Souche appelé (skeleton)**
    - Reçoit le message d'appel
    - Déballe les paramètres d'appel (unmarshalling)
    - Fait réaliser l'exécution par la procédure appelé
    - Récupère par le retour local les résultats
    - Emballe les paramètres réponse dans le message de réponse
    - Emet le message réponse vers la souche appelant

C/S : les architectures
Le modèle en RPC traditionnel (4)

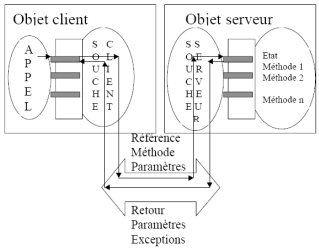
- **Invocations locale et distante**
  - Difficulté pour obtenir une transparence de la répartition
  - Syntaxe plus lourde en appel distant (description de la signature, usages d'interfaces systèmes, modes de désignation)
  - Sémantique de l'appel distant différente (performances, transmission des arguments, modes de panne)
  - **Nombreuses difficultés de mise en oeuvre**
    - Performances
    - Gestion des pannes
    - Ouverture (interopérabilité)
    - Désignation et liaison
    - Sécurité

C/S : les architectures
Le modèle à objets répartis (1)

- **Approche objet**
  - Objet : unité de programmation très utilisée
  - Objet : associe les traitements et les données
- **Répartition : intégration de l'univers objet et de l'univers réparti**
  - Placement des objets sur différents sites
  - Interface du système d'objet réparti = ensemble de primitives ou langage objet concurrent et réparti
- **Réalisation d'une invocation**
  - Référence d'objet
  - Identification de la méthode
  - Paramètres d'appel et de retour et signaux d'exception

C/S : les architectures
Le modèle à objets répartis (2)

- **Mise en oeuvre**

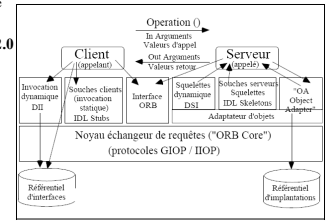


C/S : les architectures
Le modèle à objets répartis (3)

- **Système d'objets répartis homogènes : langages**
  - Représentation unique des objets propre à un langage : java (instanciation de classes)
  - Mode d'interaction optimisé (simplifié) : java RMI (Remote Method Invocation)
- **Système d'objets répartis hétérogènes**
  - Variété importante de formes d'objets supportée
  - Mode d'interaction unique
- OMG (Object Management Group) / CORBA (Common Object Requests Broker Architecture)
- Microsoft OLE (Object Link Embedding) / DCOM (Distributed Component Object Model)

C/S : les architectures
Le modèle à objets répartis (4)

- **Un modèle de référence pour une architecture dédiée aux objets répartis, un environnement coopératif de composants client/serveur**
  - CORBA, un exemple
- **Architecture CORBA 2.0**

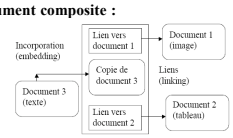


C/S : les architectures
Le modèle à composants (1)

- **Pour une simplification du processus :**
  - **De création**
  - **D'utilisation**

  des documents composites

  => environnement d'interaction unique en approche client/serveur, formats de données pivot, procédure de conversion
- **Exemple de document composite :**



C/S : les architectures
Le modèle à composants (2)

Composants / objets répartis
- Une unité de code de taille significative
- Une unité de réutilisation et d'intégration
- Une unité d'administration en réparti
- Pour associer des propriétés non fonctionnelles à un code

Avantages
- Construction de logiciels par assemblage
- Réutilisation de logiciels existants testés
- Amélioration de la modularité et des interfaces
- Cycle de développement raccourci
- Extensibilité en exécution (run-time) des applications
- Maintenance améliorée
- Amélioration des moyens de communication
- Plus portables
- Modèles de développement logiciels plus abstraits (scripts langages ADL)

C/S : les architectures
Le modèle à code mobile (1)

Buts :
- Assurer les interactions distantes en déplaçant le code à exécuter
- Rapprocher les traitements des données pour réduire le volume des échanges

Exemples évoqués :
- Collecte d'informations disséminées dans un réseau (recherche et filtrage)
- Surveillance de données
- Administration de réseaux
- Applications orientées flots de données (workflow)
- Négociation (agenda électronique, commerce électronique, …)
- Jeux en réseaux
- …

C/S : les architectures
Le modèle à code mobile (2)

Problèmes :
- Interopérabilité des codes exécutables
- Sécurité (avec méfiance mutuelle)
- Structuration des applications
- Partage des ressources (quelle cohérence ?)
- Migration faible ou forte (que déplace-t-on et comment ?)

Modèles d'exécution pour la mobilité
- Code à la demande – mobilité faible
- Agents mobiles – mobilité forte

C/S : les architectures
Le modèle à code mobile (3)

Etude de la mobilité forte :
- Notions utilisées
  - Objets (unités de codes exécutables [C/S, objets passifs]
  - Sites (environnement d'exécution)
  - Activités (flots de contrôle des appels s'étendant sur plusieurs objets ou sites)
  - Domaines (ensembles de droits sur des ressources d'une machine)
- Base de la classification
  - Activités dans même domaine
  - Activités dans domaines différents du même site
  - Activités dans domaines différents de sites différents

C/S : les architectures
Le modèle à code mobile (4)

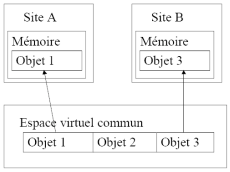
Etude de la mobilité forte (suite) :
- Type de partage
  - Partage dans le même domaine (notion de grappe, cluster d'objets)
  - Partage entre domaines différents (opération de changement de domaine, mécanisme de couplage entre domaines et objets)
  - Partage en exclusion mutuelle (une seule activité autorisée à utiliser un objet à un instant donné [cohérence de sérialisation])
  - Partage avec réplication (nécessité de mettre en œuvre une stratégie de cohérence par l'application entre les différentes copies)
- Exemple d'environnement à code mobile : les AGLETS
  Modèle proposé par IBM Japon étendant le modèle des applets Java
  - Une aglet permet la transmission de code java comme une applet
  - L'API Aglet permet le transport de l'état (seulement une partie)
  - La gestion de la sécurité est renforcée (gestionnaire de sécurité spécifique)

C/S : les architectures
Le modèle à mémoire partagée répartie (1)

Réalisation des interactions distantes en mémoire commune répartie comme espace de communication
- Buts :
  - Offrir aux développeurs les conditions de travail de l'univers centralisé
- Problèmes
  - Performance des implantations
  - Support des diverses cohérences (fortes, faibles)
  - Support des outils de développement existants (langages, compilateurs, …)

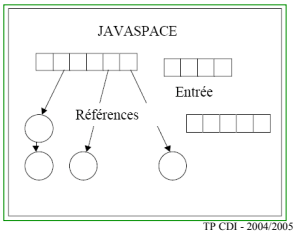
C/S : les architectures
Le modèle à mémoire partagée répartie (2)

- **Principes de réalisation : une simulation d'un espace mémoire unique dans un ensemble d'espaces mémoires réels distribués**
  - **Espace d'objets répartis partagés**
  - **Interface d'accès commune (nommage, invocation des méthodes, …)**
  - **Mode de réalisation par projection de l'espace virtuel partagé dans l'espace mémoire réel des différents sites**



C/S : les architectures
Le modèle à mémoire partagée répartie (3)

- **Exemple : javaspaces**
  - **Un javaspace = un espace de tuple ou entrées**
  - **Une entrée = un ensemble de références à des objets java**



Les opérations de base :
- Ecriture
- Lecture
- Retrait
- Notification
- Transaction

Les techniques d'utilisation :
- pas de modifications des données
- coopération
- sérialisabilité
- partage des objets réels
- duplication des objets réels

C/S : les architectures
Quel modèle ? Quel outil ?

Grande variabilité de l'offre aussi bien sur le plan des modèles que celui des plates-formes

- **Mode de synchronisation : synchrone ou asynchrone**
- **Approche langage ou système**
- **Développement d'un nouveau code pour le réparti ou intégration des solutions centralisées existantes**
- **Granularité des codes communicants**
  - **Grain moyen : objets répartis**
  - **Gros grain : composants**

C/S : Mise en oeuvre
Programmation client serveur

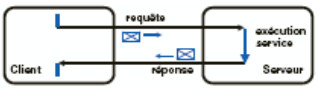
Problème : réaliser un service réparti en utilisant l'interface de transport (TCP, UDP)

Solutions

- **Les sockets**
- **L'appel de procédure à distance (RPC)**
- **L'appel de méthode à distance**
- **Le middleware intégré**

C/S : Mise en oeuvre
Programmation client serveur

- **Rappel du schéma client-serveur**
  - **Appel synchrone requête-réponse**



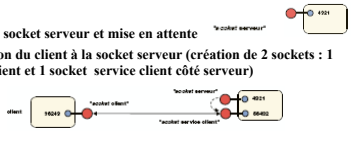
- **Mise en œuvre**
  - **Bas niveau : utilisation directe du transport (sockets)**
  - **Haut niveau : intégration dans un langage de programmation (RPC)**

C/S : Mise en oeuvre
Programmation client serveur

- **Rappel du principe de fonctionnement des sockets**

  Socket = mécanisme de communication permettant d'utiliser l'interface de transport (TCP/UDP)

  3 phases :
  - **Création socket serveur et mise en attente**
  - **Connexion du client à la socket serveur (création de 2 sockets : 1 socket client et 1 socket service client côté serveur)**



- **Communication établie**

C/S : Mise en oeuvre
Programmation client serveur

Client serveur avec sockets : deux réalisations possibles

Mode connecté (protocole TCP)

- **Ouverture d'une liaison, suite d'échanges, fermeture de liaison**
- **État du serveur préservé entre deux requêtes**
- **Garanties de TCP : ordre, contrôle de flux, fiabilité**
- **Adapté aux échanges d'une certaine durée (plusieurs messages)**

Mode non connecté (protocole UDP)

- **Requêtes successives indépendantes**
- **Pas de préservation de l'état entre les requêtes**
- **Client doit indiquer son adresse à chaque requête**
- **Pas de garanties particulières**
- **Adapté aux échanges brefs (réponse en 1 message)**

Points communs

- **Client à l'initiative de la communication – serveur à l'écoute**
- **Client doit connaître le serveur (adresse IP, n° port)**
- **Serveur pouvant servir plusieurs clients**

C/S : Mise en oeuvre
Programmation client serveur

Client serveur avec sockets : utilisation du mode connecté

Caractéristiques

- **Etablissement préalable de la connexion**
- **Fiabilité assurée par TCP**
- **Mode d'échange par flot d'octets**
- **Possibilité d'émettre et de recevoir des caractères urgents**
- **Après initialisation, serveur passif**
- **Plusieurs clients possibles (file d'attente)**

Contraintes

- **Client devant connaître l'adresse du serveur**

Modes de gestion des requêtes

- **Itératif**
- **Concurrent par création de processus fils pour les échanges de chaque requête**

C/S : Mise en oeuvre
Programmation client serveur

Client serveur avec sockets : serveur itératif en mode connecté

Ouverture de connexion avec serveur pour adresser des appels puis fermeture de connexion à la fin des opérations

- **Délimitation temporelle des échanges**
- **Maintien de l'état de connexion pour gestion des paramètres de qualité de service**

Orienté vers

- **Traitement ordonné d'une suite d'appels**
- **Gestion de données persistantes ou de protocoles avec état**

Client serveur avec sockets : programmation

Principes

- **Opérations de gestion de sockets fournies comme primitives dans Unix**
- **Utilisation directe de ces opérations mais des langages usuels permettent de les utiliser plus facilement en couvrant les cas courants aussi bien côté client que côté serveur (bibliothèques spécialisées en C, classes en Java)**

Exemple de programmation (voir plus loin)

C/S : Mise en oeuvre
Programmation client serveur

Client serveur avec sockets : programmation sockets (mode connecté) en Java

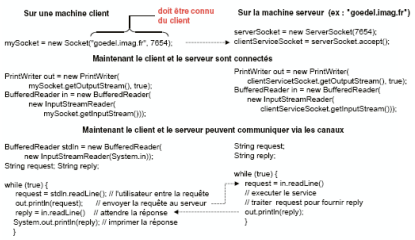
Deux classes de base

- **ServeurSocket : socket côté serveur**
- **Socket : sockets ordinaires pour les échanges qui fournissent des classes InputStream et OutputStream pour échanger de l'information**

Voir http://java.sun.com/docs/books/tutorial/networking/sockets/

C/S : Mise en oeuvre
Programmation client serveur

Client serveur avec sockets TCP en Java (version light)



C/S : Mise en oeuvre
Programmation client serveur

Client serveur avec sockets TCP en Java (version light) compléments

Importer les packages utiles

```
import java.io.*;
import java.net.*;
```

Prévoir les cas d'erreur

```
Socket mySocket = null;
try {
   mySocket = new Socket("goedel.imag.fr", 7654);
} catch (IOException e) {
   System.out.println("connexion impossible");
   System.exit(-1);
}
```

Prévoir terminaison

(dépend de l'application)

Terminer proprement

```
out.close();
in.close();
mySocket.close();
```

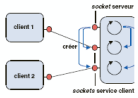
```
import java.io.*;
import java.net.*;

try {
   serverSocket = new ServerSocket(7654);
} catch (IOException e) {
   System.out.println("port 7654 non utilisable");
   System.exit(-1);
}
Socket clientServiceSocket = null;
try {
serverSocket.accept();
} catch (IOException e) {
   System.out.println(accept impossible sur port 7654");
   System.exit(-1);
}

out.close();
in.close();
clientServiceSocket.close();
serverSocket.close();
```

C/S : Mise en oeuvre
Programmation client serveur

Client serveur avec sockets TCP en Java (version light) Remarques

- **Dans le schéma précédent, un seul processus serveur**
  - **Si clients multiples : traitement en séquence (file d'attente)**
- **Utilisation d'un schéma veilleur-exécutants**
  - **Thread veilleur : création explicite d'un nouveau thread exécutant à chaque nouvelle connexion d'un client**
  - **Socket service client différente créée pour chaque client**
  - **Thread veilleur se remet ensuite en attente**

```
while (true) {
   accepter la connexion d'un nouveau client
      et créer une socket service client;
   créer un thread pour interagir avec ce client sur la
      nouvelle socket service client;
}
```



- **Voir aussi serveur multi thread**

C/S : Mise en oeuvre
Programmation client serveur

Client serveur avec sockets : utilisation du mode non connecté (UDP)

Caractéristiques
- **Pas d'établissement préalable d'une connexion**
- **Pas de garantie de fiabilité**
- **Adaptée aux applications avec requêtes courtes**
- **Récepteur reçoit les données en fonction du découpage réalisé par l'émetteur**

Contraintes
- **Client devant connaître l'adresse du serveur**
- **Serveur devant récupérer l'adresse de chaque client pour répondre**

Modes de gestion des requêtes
- **Itératif**
- **Concurrent (1 processus ou thread par client)**

C/S : Mise en oeuvre
Programmation client serveur

Client serveur avec sockets : serveur itératif en mode non connecté

Le client peut envoyer un appel au serveur à tout moment

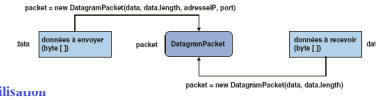
Mode léger permanent
- **Traitement non ordonné des appels**
- **Absence de mémorisation entre appels successifs**

Exemples
- **Calcul de fonction numérique**
- **DNS (service de noms)**
- **NFS (service de fichiers répartis)**

C/S : Mise en oeuvre
Programmation client serveur

Client serveur avec sockets : programmation sockets (mode non connecté) en Java

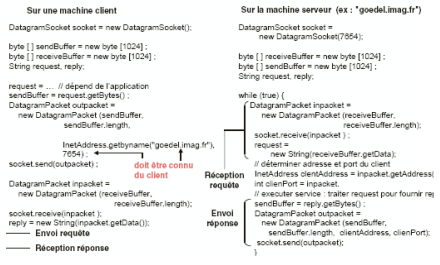
Deux classes de base
- **DatagramSocket : un seul type de socket**
- **DatagramPacket : format de message pour UDP**



Utilisation
- **Echanges simples (question/réponse)**
- **Messages brefs**
- **Streaming temps réel (performances)**

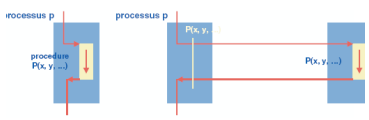
C/S : Mise en oeuvre
Programmation client serveur

Client serveur avec sockets en Java (mode non connecté)



C/S : Mise en oeuvre
Appel de procédure à distance (RPC)

- **Appel de procédure à distance (remote procedure call) = outil pour construire des applications client serveur dans un langage de haut niveau**
- **Appel et retour ont lieu sur un site, exécution sur un site distant**



C/S : Mise en oeuvre
Appel de procédure à distance (RPC)

Avantages attendus

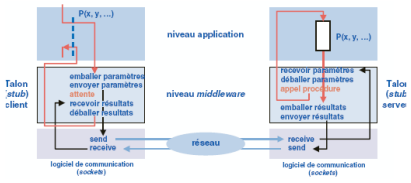
- **Facilité de programmation (complexité des protocoles de communication cachée)**
- **Facilité de mise au point (mise au point d'application sur un site puis déploiement sur autres sites)**
- **Portabilité (résultant de l'usage de langage de haut niveau *[indépendance par rapport au système de communication]*)**

Problèmes

- **Transmission des paramètres**
- **Gestion des processus**
- **Réactions aux défaillances**

C/S : Mise en oeuvre
Appel de procédure à distance (RPC)

Principes de mise en œuvre



- **Les talons client et serveur sont créés à partir d'une description d'interface**

C/S : Mise en oeuvre
Appel de procédure à distance (RPC)

Exemple d'utilisation de RPC en langage C d'un service d'annuaire

Les étapes :

- **Préparer une description de l'interface (sur transparent suivant) : fichier annuaire.x**
- **Compiler ce fichier à l'aide du générateur de talons**

```
rpcgen annuaire.x      résultat : les fichiers suivants :
        annuaire.h                  include
        annuaire_clnt.c             talon client
        annuaire_svc.c              talon serveur
        annuaire_xdr.c              proc. conversion données
fournit aussi des fichiers auxiliaires (modèles de prog. client, serveur, Makefile)
```

- **Sur le site client, construire l'exécutable client (programme : annuaire_client.c)**

```
gcc -o client annuaire_client.c annuaire_clnt.c
```

- **Sur le site serveur, construire l'exécutable serveur (programme : annuaire_serveur.c)**

```
gcc -o server annuaire_server.c annuaire_svc.c
```

- **Lancer l'exécutable serveur sur le site serveur**

```
server &
```

- **Lancer l'exécutable client sur le site client**

```
client
```


C/S : Mise en oeuvre
Appel de procédure à distance (RPC)

Description d'interface service d'annuaire

```
/* 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 ;
```


C/S : Mise en oeuvre
Appel de procédure à distance (RPC)

Interface = "contrat" entre client et serveur

- **Définition commune abstraite**
  - **Indépendante d'un langage particulier**
  - **Indépendante de la représentation des types**
  - **Indépendante de la machine**
- **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**

C/S : Mise en oeuvre
Appel de procédure à distance (RPC)

Structure du programme serveur

Ce programme ne contient que les fonctions à appeler. La procédure main est contenu dans le talon. Ce programme est à écrire par le développeur alors que le talon est engendré automatiquement.

```
#include "annuaire.h"

void *
nit_1_svc(void *argp, struct svc_req *rqstp)
{
        static char * result;
        /* le programme de la fonction init */
        return (void *) &result;
}

nt *
ajouter_1_svc(personne *argp, struct svc_req *rqstp)
{
        static int  result;
        /* le programme de la fonction ajouter */
        return &result;
```

```
int *
supprimer_1_svc(personne *argp, struct svc_req *rqstp)
{
        static int  result;
        /* le programme de la fonction supprimer */
        return &result;
}

personne *
consulter_1_svc(typenom *argp, struct svc_req *rqstp)
{
        static personne  result;
        /* le programme de la fonction consulter */
        return &result;
}
```

C/S : Mise en oeuvre
Appel de procédure à distance (RPC)

Structure du talon serveur

```
ss parties en gras ne sont pas détaillées
 includes ... */
atic void
nuaire_1(struct svc_req *rqstp, register SVCXPRT *transp){
  union {
    personne ajouter_1_arg;
    personne supprimer_1_arg;
    typenom consulter_1_arg;
  } argument;
  char *result;
  xdrproc_t _xdr_argument, _xdr_result;
  char *(*local)(char *, struct svc_req *);

  switch (rqstp->rq_proc) {
      case NULLPROC:
        (void) svc_sendreply (transp,
                  (xdrproc_t) xdr_void, (char *)NULL);
        return;
      case INIT:
        _xdr_argument = (xdrproc_t) xdr_void;
        _xdr_result = (xdrproc_t) xdr_void;
        local = (char *(*)(char *,
                  struct svc_req *)) init_1_svc;
        break;
      case AJOUTER:
        _xdr_argument = (xdrproc_t) xdr_personne;
        _xdr_result = (xdrproc_t) xdr_int;
        local = (char *(*)(char *,
                  struct svc_req *)) ajouter_1_svc;
        break;
      case SUPPRIMER:
                /* idem */
      case CONSULTER:
                /* idem */
      default:
        svcerr_noproc (transp);
        return;
  }

  /*  préparer arguments dans zone argument */
  result = (*local)((char *)&argument, rqstp);      /* exécution */
    if (result != NULL && !svc_sendreply(transp, /* envoi résultats */
                (xdrproc_t) _xdr_result, result)) {
      svcerr_systemerr (transp);
    }
  /*  libérer arguments */

  return;
}
int main (int argc, char **argv)
{
  /* créer une socket serveur TCP sur un port p  */
  /* créer une socket serveur UDP sur un port p1 */
  /* enregistrer le service annuaire_1 sous TCP (port p)  */
  /* enregistrer le service annuaire_1 sous UDP (port p1) */
  svc_run () ; /* se mettre en attente d'un appel du client */
}
```


C/S : Mise en oeuvre
Appel de procédure à distance (RPC)

Structure du programme client

```
annuaire_client.c
/* include */
/* déclarations */

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 */
...
}
```


C/S : Mise en oeuvre
Appel de procédure à distance (RPC)

Structure du talon client

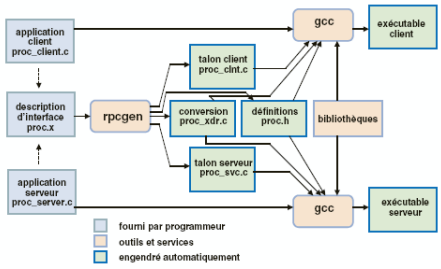
```
#include <memory.h> /* for memset */
#include "annuaire.h"
/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = {25, 0 };

void * init_1(void *argp, CLIENT *clnt) {
        static char clnt_res;

        memset((char *)&clnt_res, 0, sizeof(clnt_res));
        if (clnt_call (clnt, INIT,
                  (xdrproc_t) xdr_void, (caddr_t) argp,
                  (xdrproc_t) xdr_void, (caddr_t) &clnt_res,
                  TIMEOUT) != RPC_SUCCESS) {
                  return (NULL);
        }
        return ((void *)&clnt_res);
}

int * ajouter_1(personne *argp, CLIENT *clnt) {
        static int clnt_res;

        memset((char *)&clnt_res, 0, sizeof(clnt_res));
        if (clnt_call (clnt, AJOUTER,
                  (xdrproc_t) xdr_personne, (caddr_t) argp,
                  (xdrproc_t) xdr_int, (caddr_t) &clnt_res,
                  TIMEOUT) != RPC_SUCCESS) {
                  return (NULL);
        }
        return (&clnt_res);
}
```

L'appel via la socket est caché dans clnt_call

```
int * supprimer_1(personne *argp, CLIENT *clnt) {
        static int clnt_res;

        memset((char *)&clnt_res, 0, sizeof(clnt_res));
        if (clnt_call (clnt, SUPPRIMER,
                  (xdrproc_t) xdr_personne,
                              (caddr_t) argp,
                  (xdrproc_t) xdr_int,
                              (caddr_t) &clnt_res,
                  TIMEOUT) != RPC_SUCCESS) {
                  return (NULL);
        }
        return (&clnt_res);
}

personne * consulter_1(typenom *argp, CLIENT *clnt) {
        static personne clnt_res;

        memset((char *)&clnt_res, 0, sizeof(clnt_res));
        if (clnt_call (clnt, CONSULTER,
                  (xdrproc_t) xdr_typenom,
                              (caddr_t) argp,
                  (xdrproc_t) xdr_personne,
                              (caddr_t) &clnt_res,
                  TIMEOUT) != RPC_SUCCESS) {
                  return (NULL);
        }
        return (&clnt_res);
}
```


C/S : Mise en oeuvre
Appel de procédure à distance (RPC)

Chaîne de production



C/S : Mise en oeuvre
Appel de procédure à distance (RPC)

Détails de réalisation

La réalisation de l'appel de procédure à distance soulève divers problèmes techniques que nous n'examinerons pas ici.

- **Passage de paramètres**
- **Désignation**
- **Traitement des défaillances**
- **Serveur multi-threads**
- **Outils**

Une référence pour aller plus loin :
http://www.cs.cf.ac.uk/Dave/C/node33.html

C/S : Mise en oeuvre
Appel de procédure à distance (RPC)

Conclusion sur le RPC

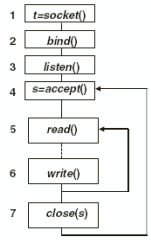
Avantages
- **Abstraction (détails de communication cachés)**
- **Intégration dans un langage : facilite la portabilité et la mise au point**
- **Outils de génération facilitent la mise en oeuvre**

Limitations
- **Structure application statique => pas de création dynamique de serveur, pas de possibilité de redéploiement entre sites**
- **Pas de passage de paramètres par référence**
- **Communication réduite à un schéma synchrone**
- **Persistance des données non assurée (réalisation par sauvegarde explicite des données dans les fichiers)**
- **Mécanismes plus évolués visant à remédier à ces limitations**
  - **Objets répartis (ex : Java RMI)**
  - **Composants répartis (ex : EJB, Corba CCM, .Net)**

C/S : mise en œuvre – Annexes utiles

Algorithme d'un serveur en mode connecté

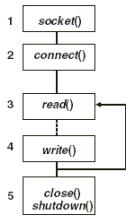
- 1. Création de la *socket* serveur
- 2. Récupération de l'adresse IP et du numéro de port du serveur ; lien de la *socket* à l'adresse du serveur
- 3. Mise en mode passif de la *socket* : elle est prête à accepter les requêtes des clients
- 4. (opération bloquante) : acceptation d'une connexion d'un client et création d'une *socket* service client, dont l'identité est rendue en retour
- 5. et 6. Lecture, traitement et écriture (selon algorithme du service)
- 7. Fermeture et remise en attente



C/S : mise en œuvre – Annexes utiles

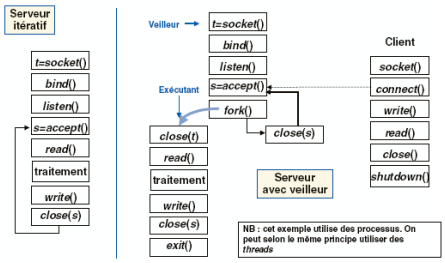
Algorithme d'un client en mode connecté

- 1. Création de la *socket*
- 2. Connexion de la *socket* au serveur
  - choix d'un port libre pour la *socket* par la couche TCP
  - attachement automatique de la *socket* à l'adresse (IP machine locale + n° de port)
  - connexion de la *socket* au serveur en passant en paramètre l'adresse IP et le n° de port du serveur
- 3. et 4. Dialogue avec le serveur (selon algorithme du service)
- 5. Fermeture de la connexion avec le serveur



C/S : mise en œuvre – Annexes utiles

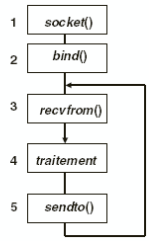
Gestion des processus en mode connecté



C/S : mise en œuvre – Annexes utiles

Algorithme d'un serveur en mode non connecté

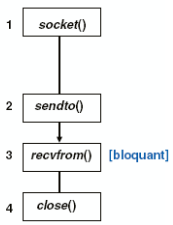
- 1. Création de la *socket*
- 2. Récupération de l'adresse IP et du numéro de port du serveur ; lien de la *socket* à l'adresse du serveur
- 3. Réception d'une requête de client
- 4. Traitement de la requête ; préparation de la réponse
- 5. Réponse à la requête en utilisant la *socket* et l'adresse du client obtenues par *recvfrom* ; retour pour attente d'une nouvelle requête



C/S : mise en œuvre – Annexes utiles

Algorithme d'un client en mode non connecté

- 1. Création de la *socket* (l'association à une adresse locale [adresse IP + n° port] est faite automatiquement lors de l'envoi de la requête)
- 2. Envoi d'une requête au serveur en spécifiant son adresse dans l'appel
- 3. Réception de la réponse à la requête
- 4. Fermeture de la *socket*



C/S : mise en œuvre – Annexes utiles

Gestion des processus en mode non connecté