SR03 2004 - Cours Architectures Internet - Introduction

1 SR03 2004 - Cours Architectures Internet - Introduction

1.1 De l'ordinateur au réseau, puis au réseau de réseaux

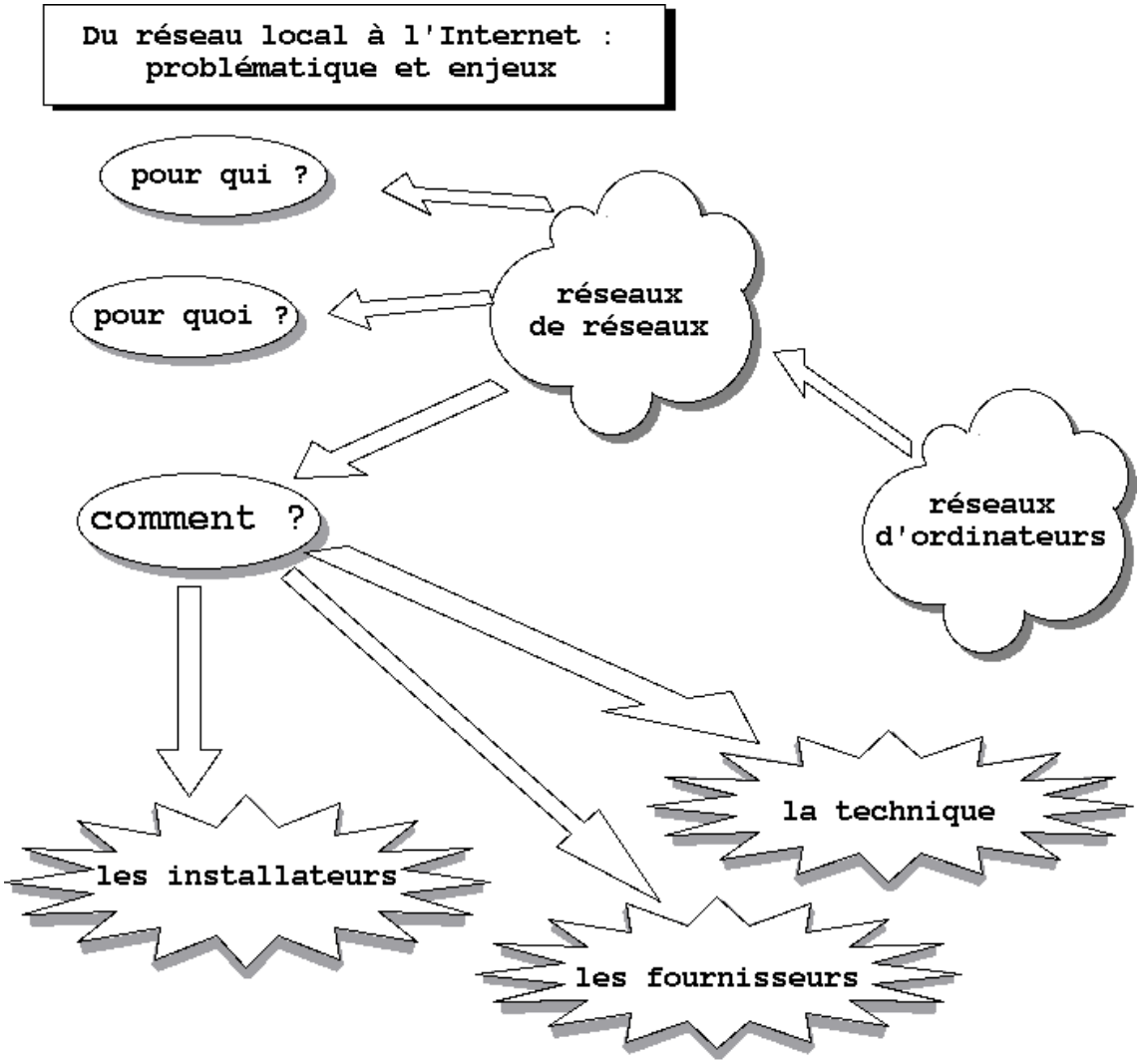


FIG. 1: enjeux 1/8

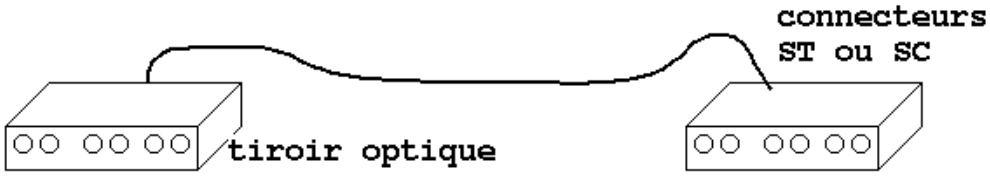
Du réseau local à l'Internet : problématique et enjeux

réseau d'ordinateurs -> réseau de réseaux

câblage local

coaxial --> 10B5, 10B2
paire torsadée --> 10BT (cat.5)
fibre optique --> 10B FX

câblage inter-bâtiments



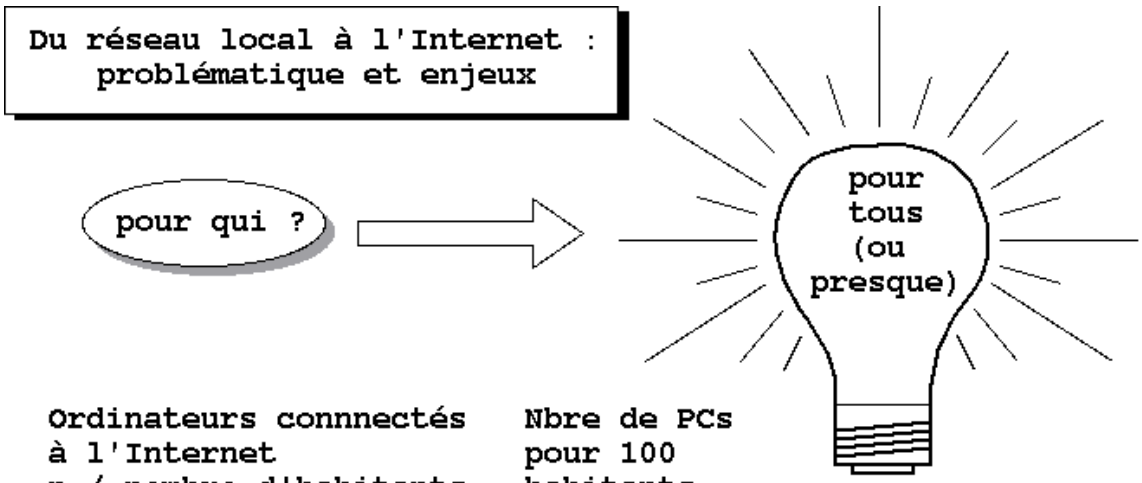
fibre optique multimode (d<=2km)
fibre optique monomode (d >2km)
laser infrarouge 2 Mbits (1 km) à 100 Mbits (100m)
faisceau micro-ondes : 2 à 150 Mbits (2 à 10 km)
wi-fi : 10 à 54 Mbits (< 1 km)

câblage longues distances

réseaux métropolitains :
- réseau "câble" (modem câble)
- fibre optique
- paire téléphonique (ADSL)
- wi-fi

réseaux inter-cités et international :
- opérateurs de télécommunication
- --> RTC, RNIS, LS, X25, SMDS, ATM, SDH

FIG. 2: enjeux 2/8



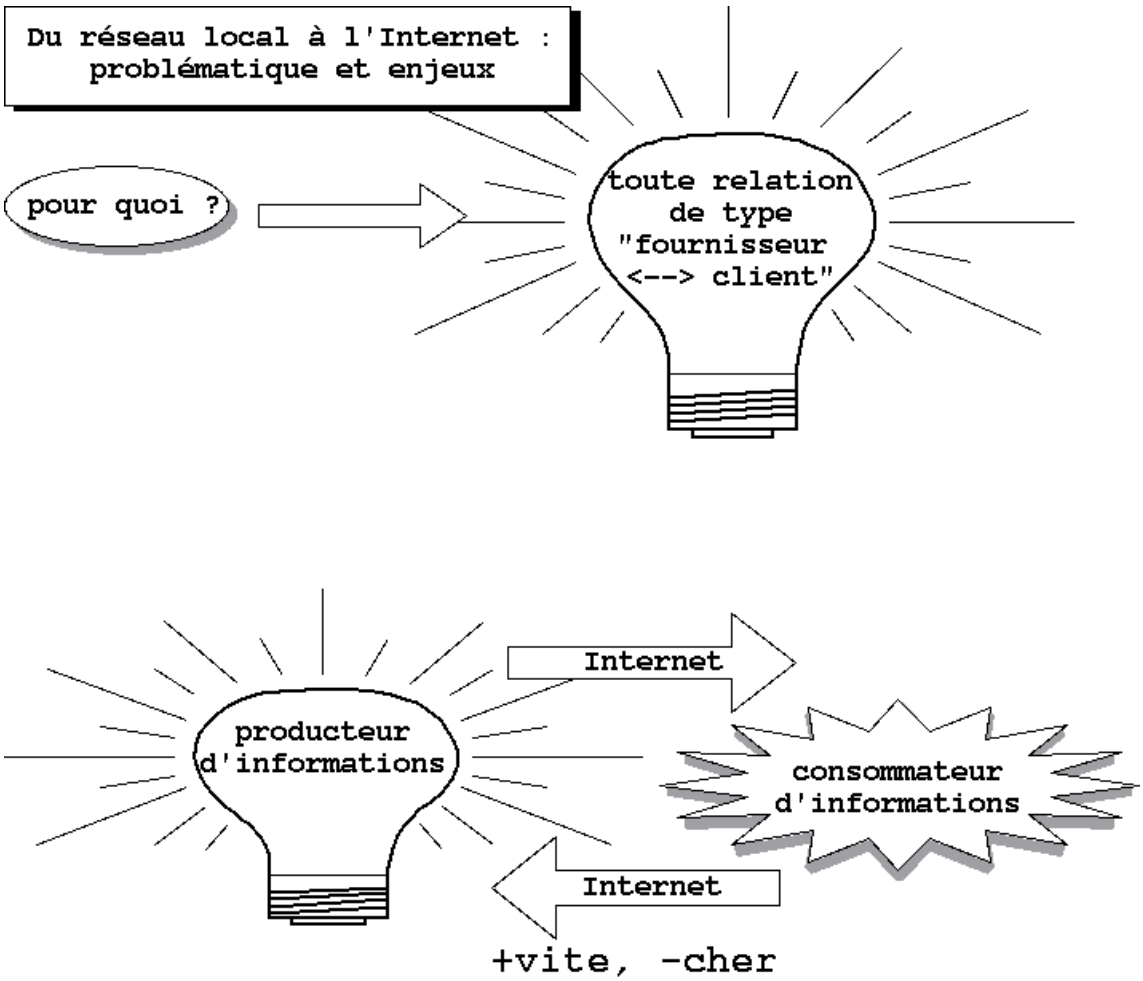
	Ordinateurs connnectés à l'Internet n / nombre d'habitants	Nbre de PCs pour 100 habitants
	1996	2001
Finlande	1 pour 18	42 pour 100
USA	1 pour 36	62 pour 100
France	1 pour 295	34 pour 100
Allemagne		34 pour 100

	Nbre de serveurs pour 10 000 hab. en 2001
Finlande	1700
USA	3700
France	133
Allemagne	294

La connectivité est sa propre récompense :
-> effet de "famille"
-> effet de "marché"

Cercles vertueux :
-> + on est => + intéressant => + attire
-> + on est => + clients => + rentable

FIG. 3: enjeux 3/8



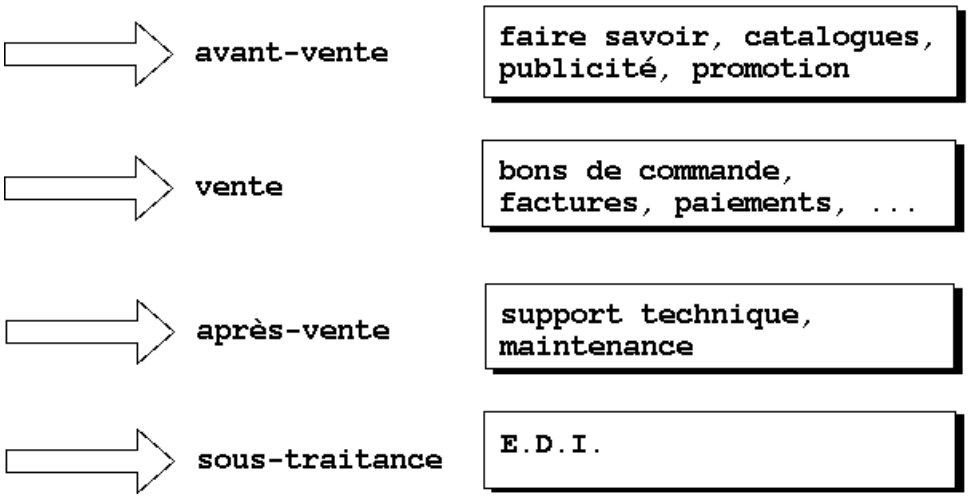
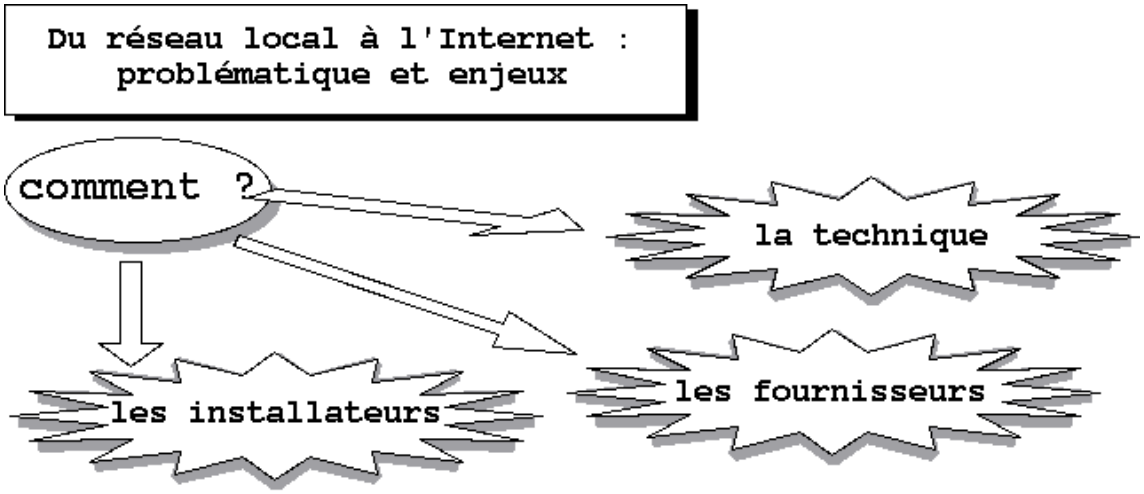


FIG. 4: enjeux 4/8



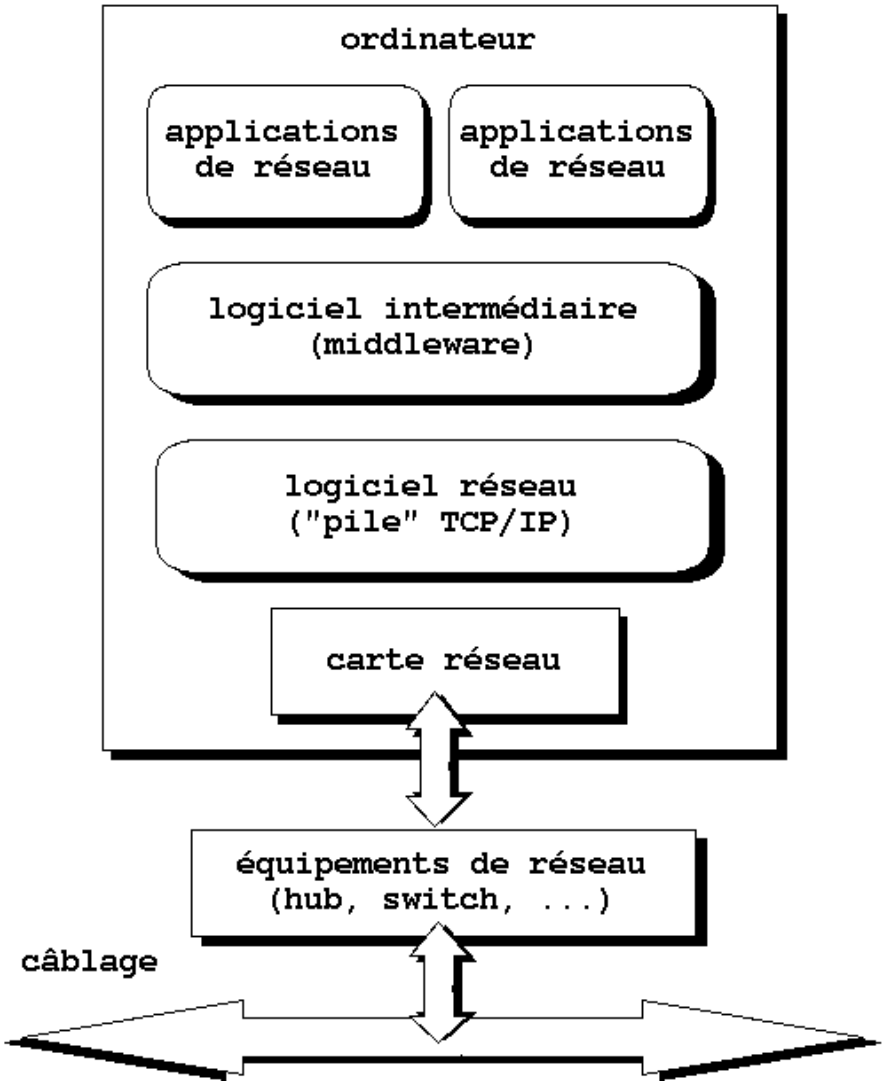


FIG. 5: enjeux 5/8

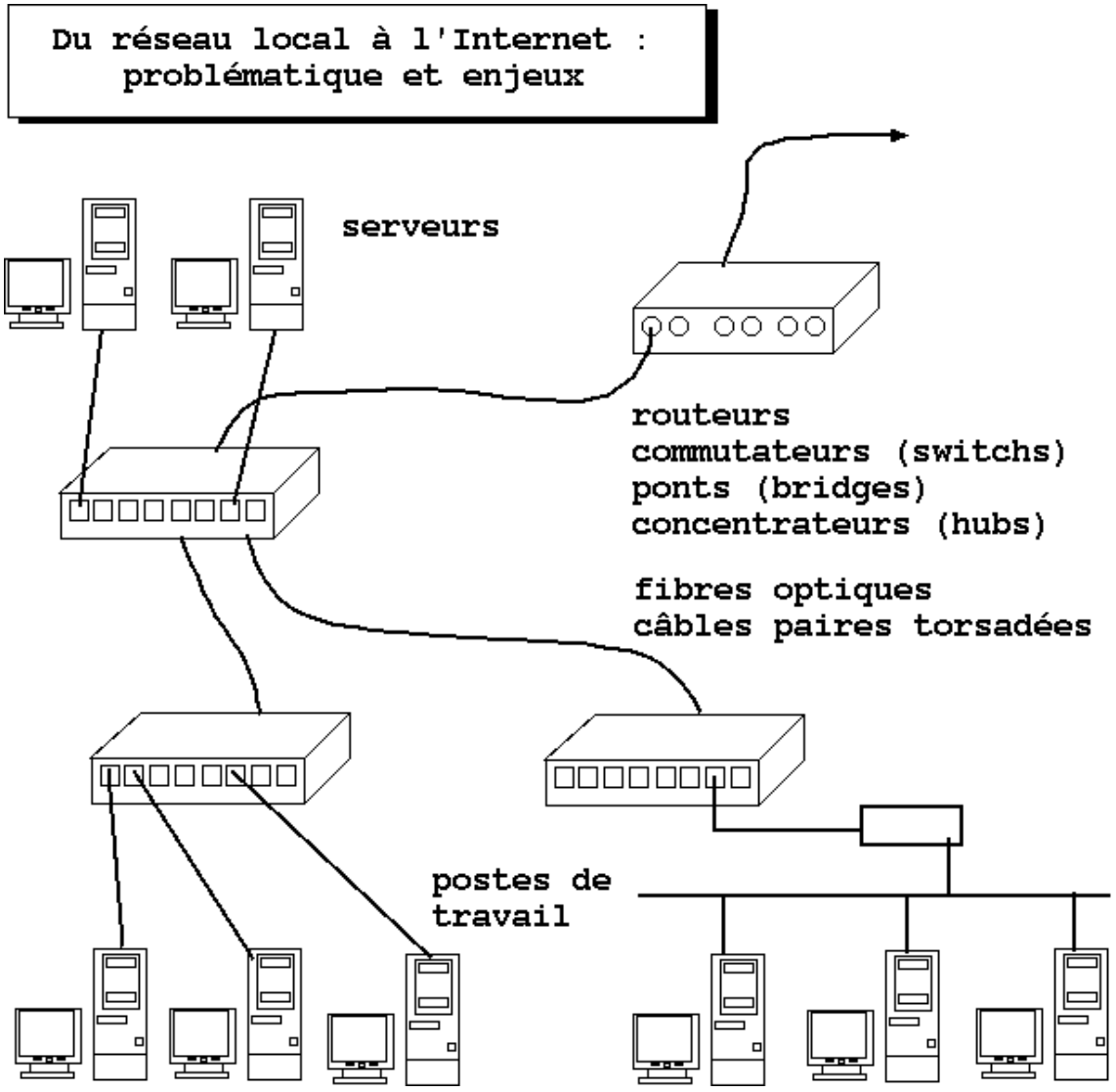


FIG. 6: enjeux 6/8

Du réseau local à l'Internet : problématique et enjeux

Échanges de données entre applications
Applications coopératives
Applications client-serveur

→ **Ensemble de protocoles standards, simples, universels (reconnus par tous)**

→ **Protocoles de l'Internet : TCP/IP, ftp, smtp, nntp, http, html, ...**

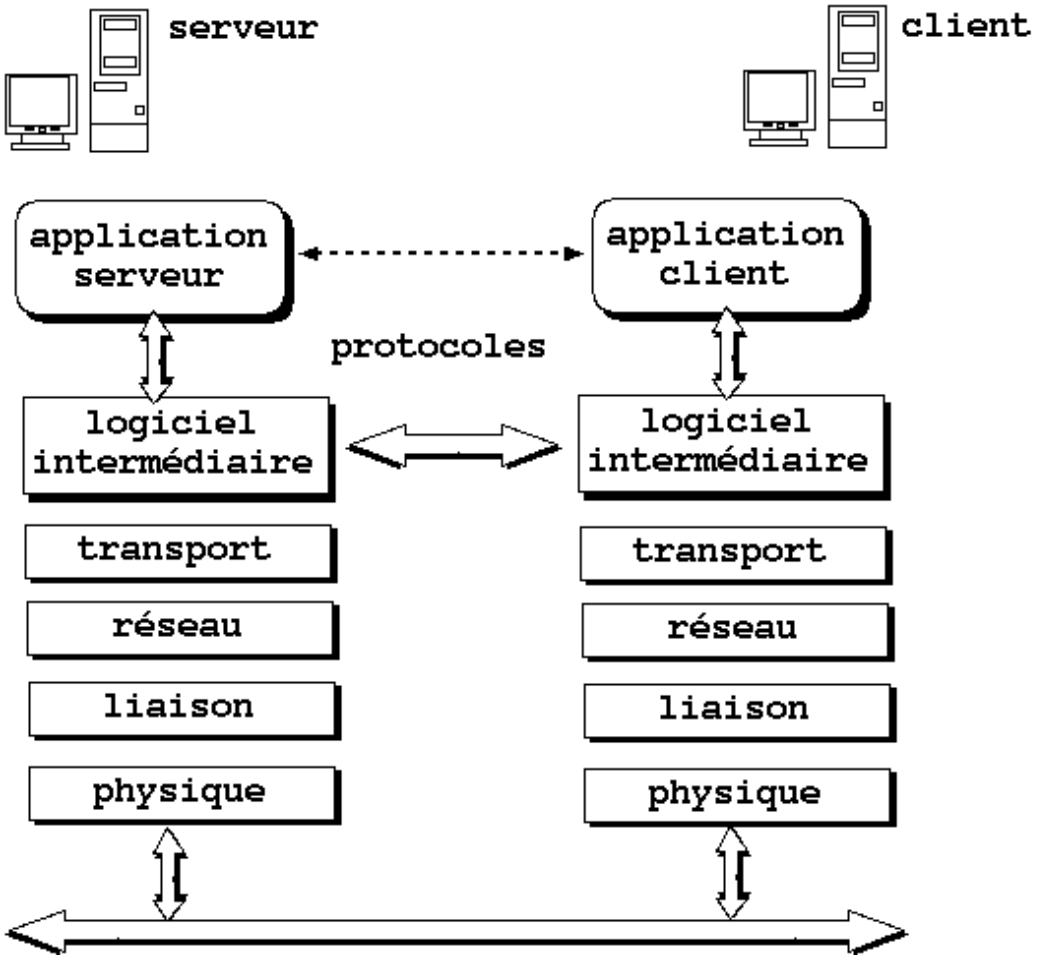


FIG. 7: enjeux 7/8

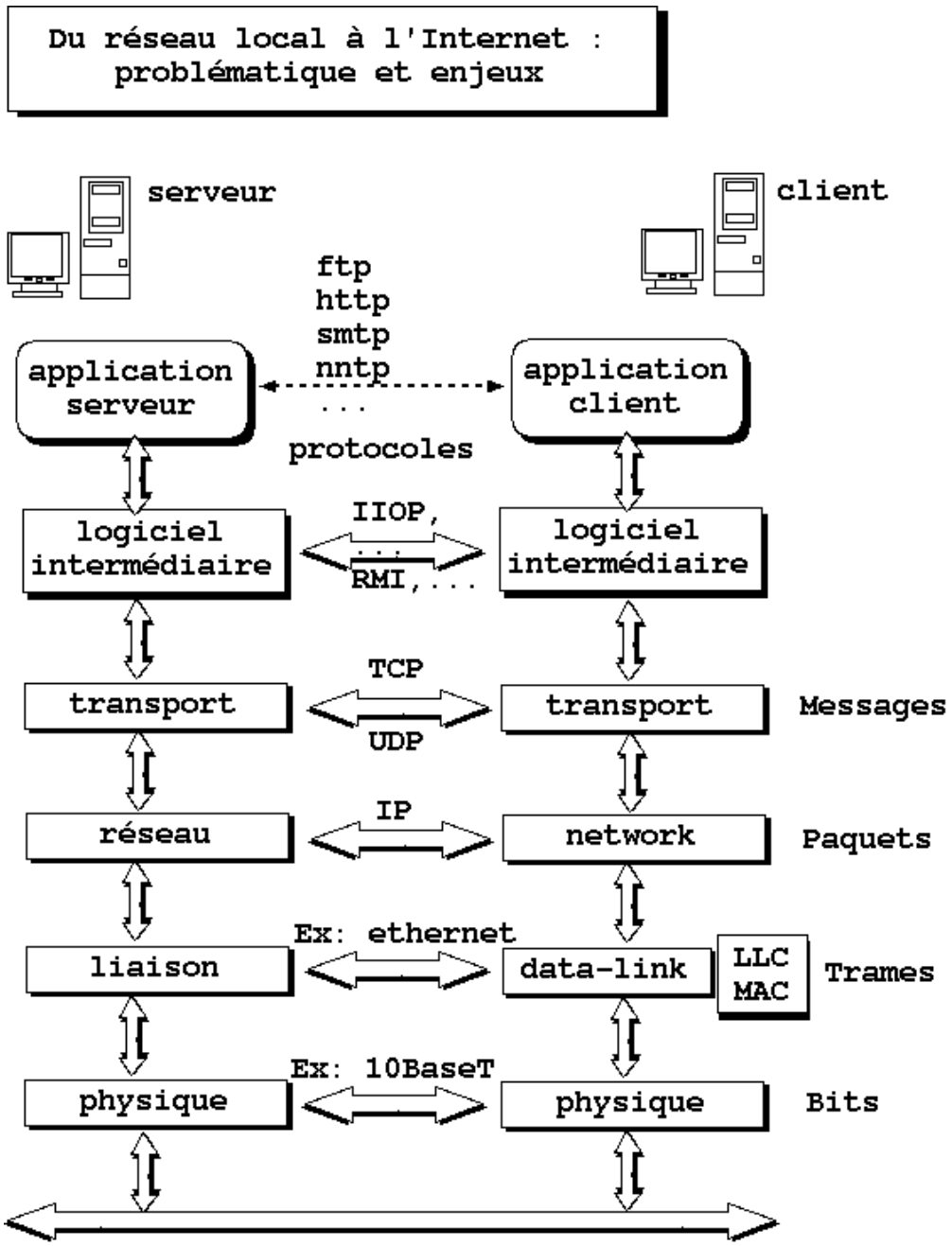


FIG. 8: enjeux 8/8

1.2 Extension de l'informatique et positionnement des UVs systèmes

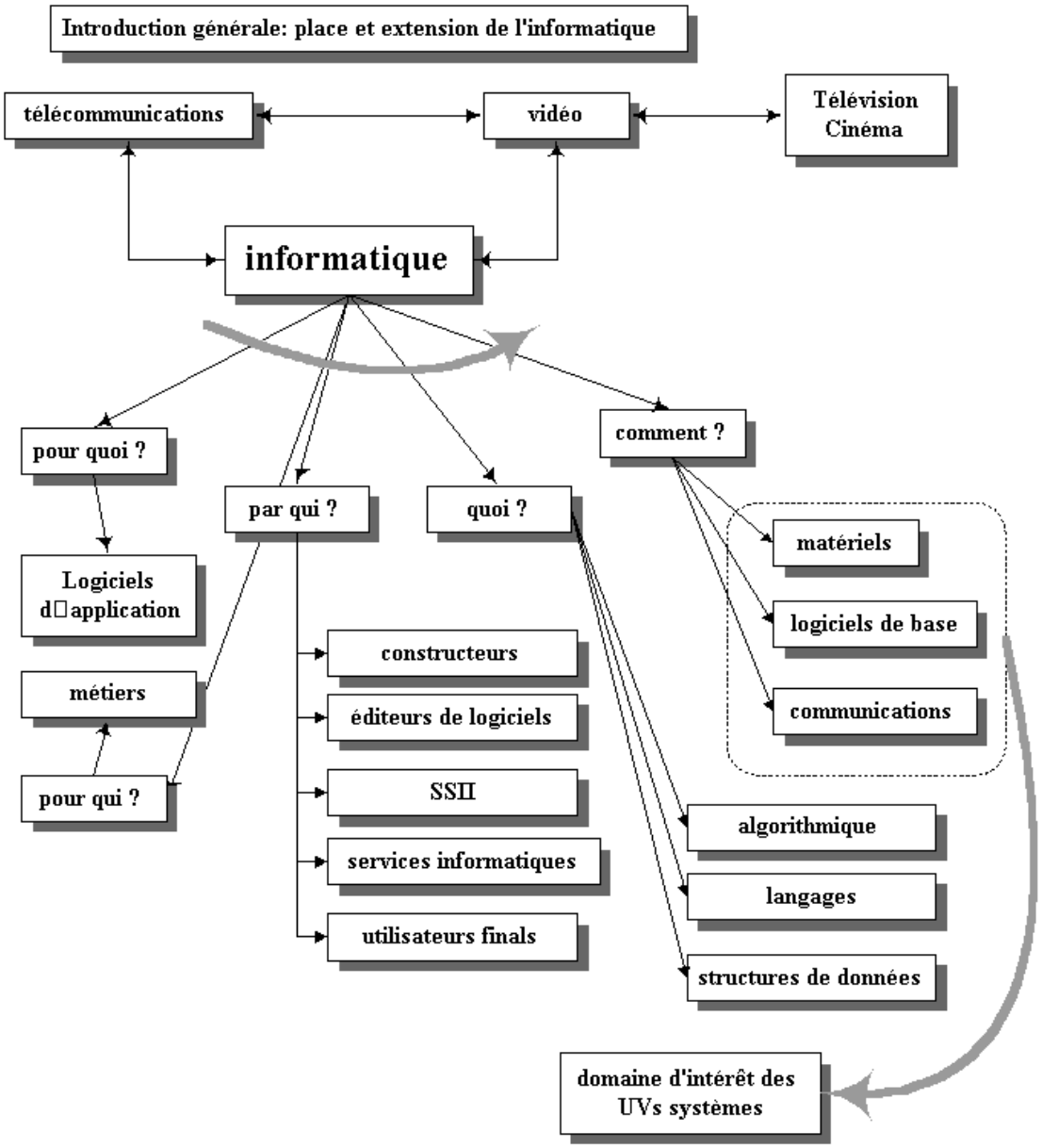


FIG. 9: L'informatique et les UVs systèmes 1/6

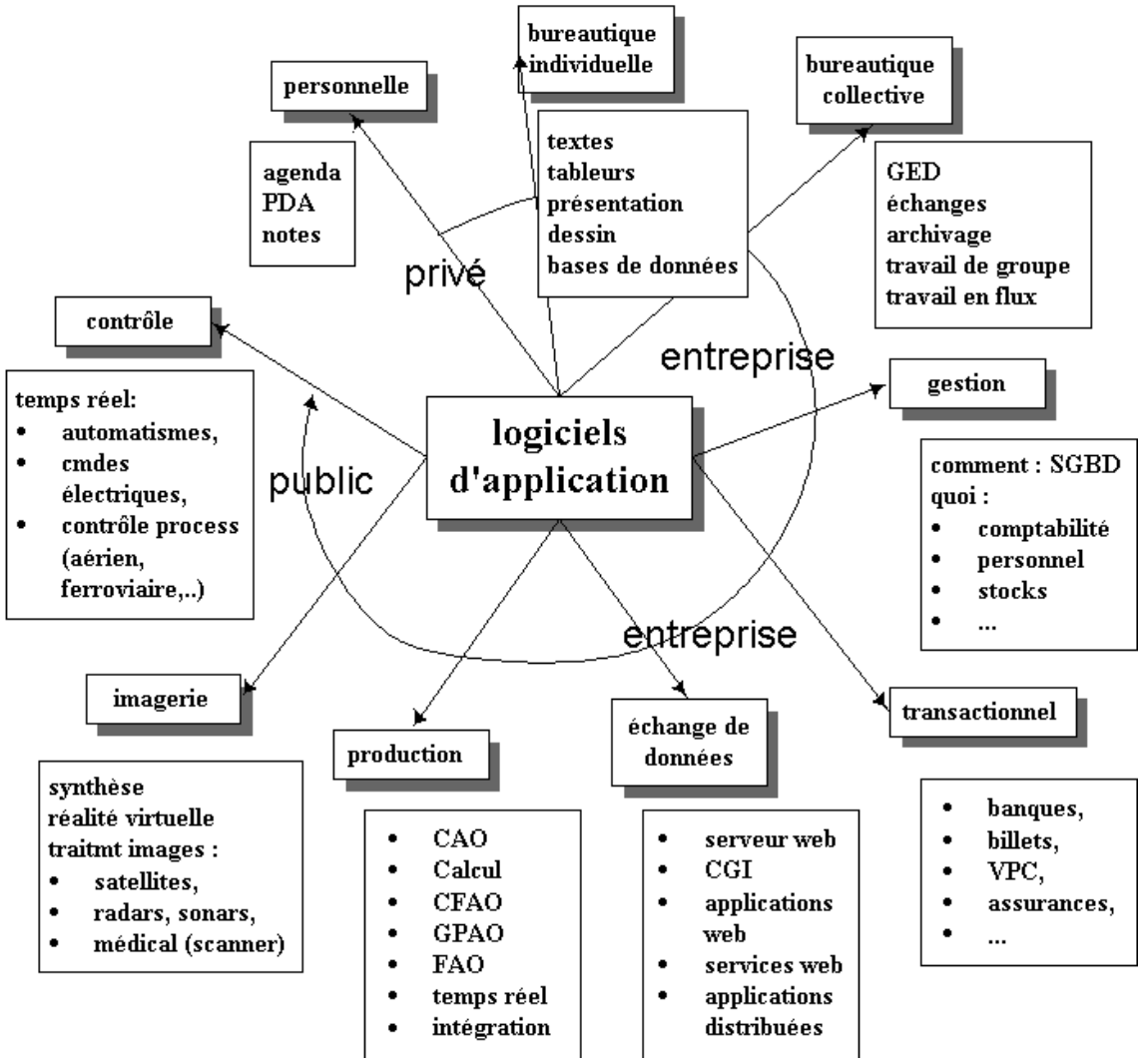


FIG. 10: L'informatique et les UVs systèmes 2/6

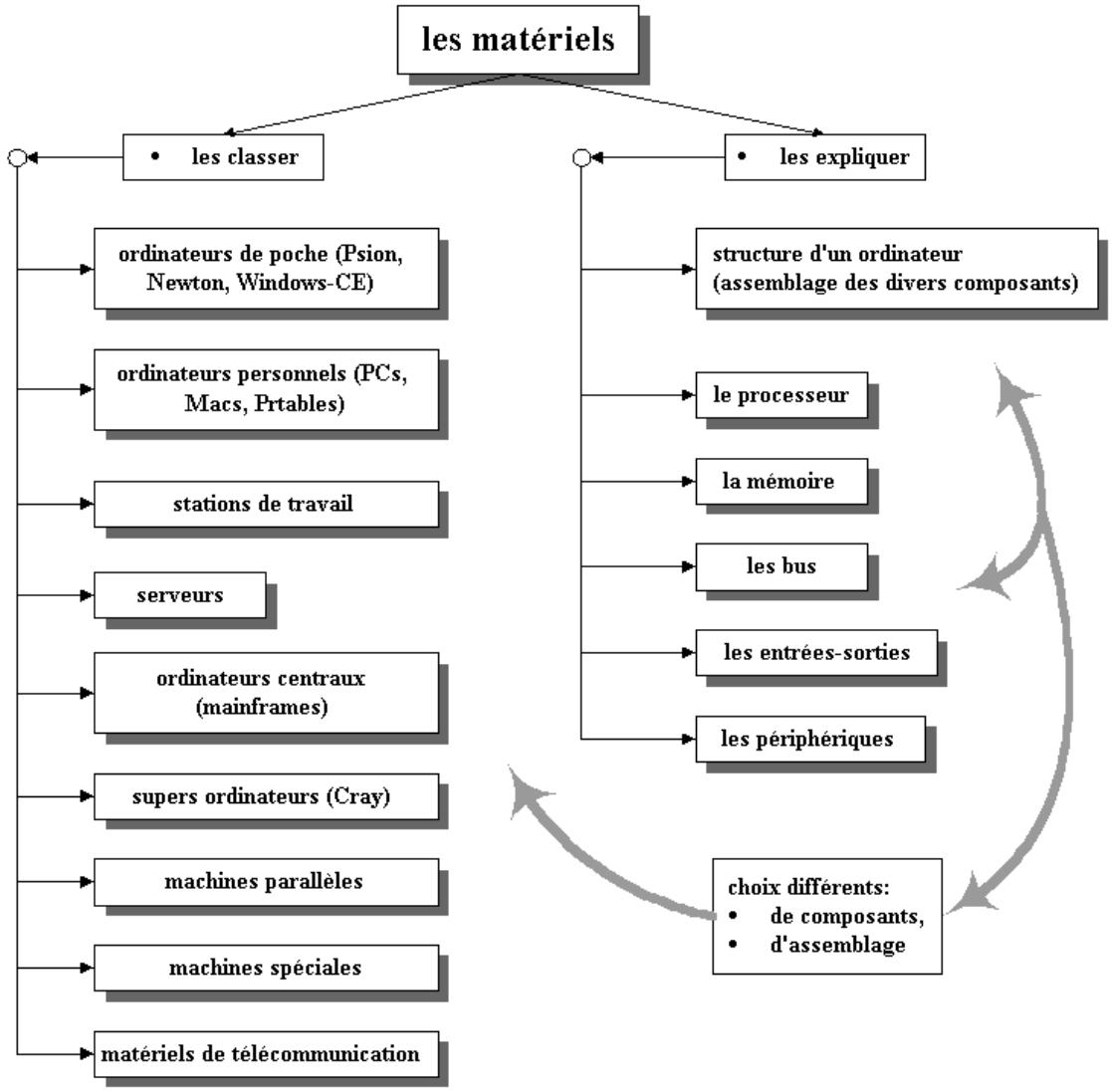


FIG. 11: L'informatique et les UVs systèmes 3/6

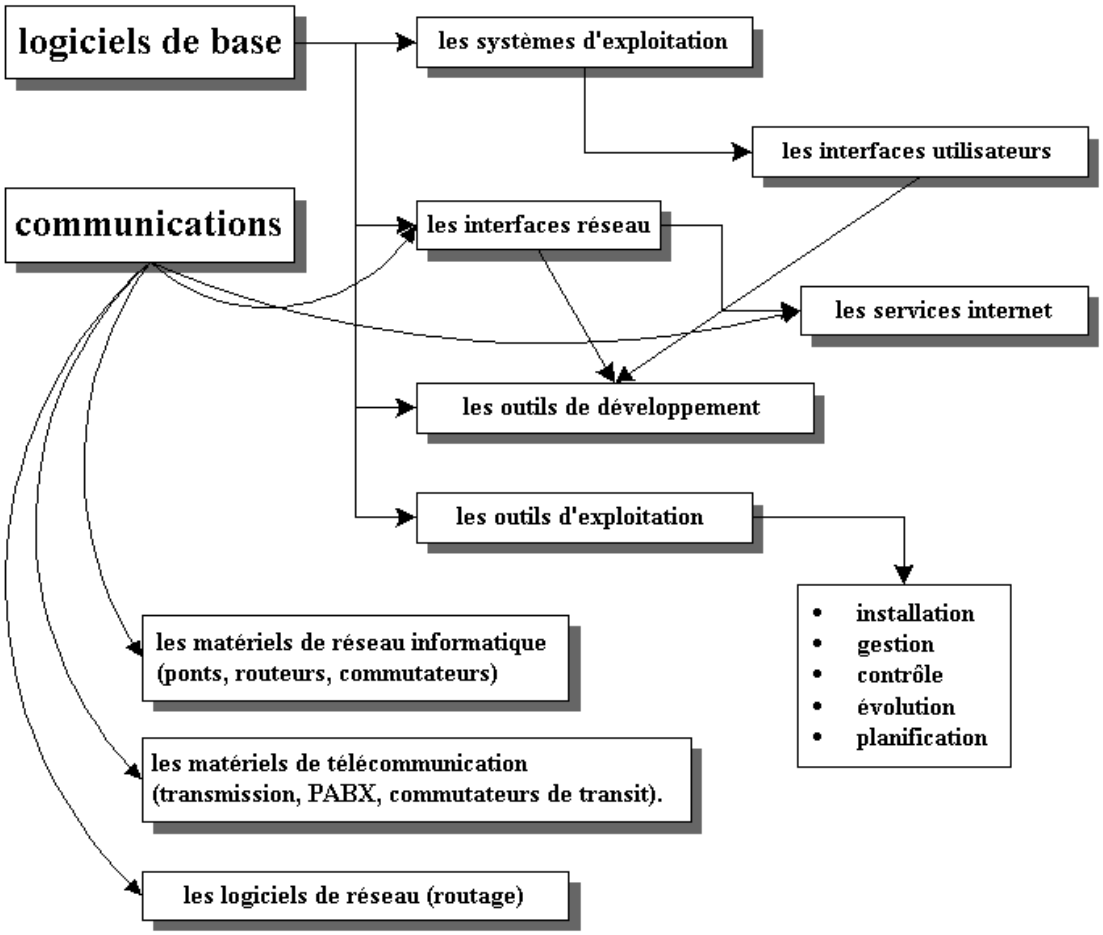


FIG. 12: L'informatique et les UVs systèmes 4/6

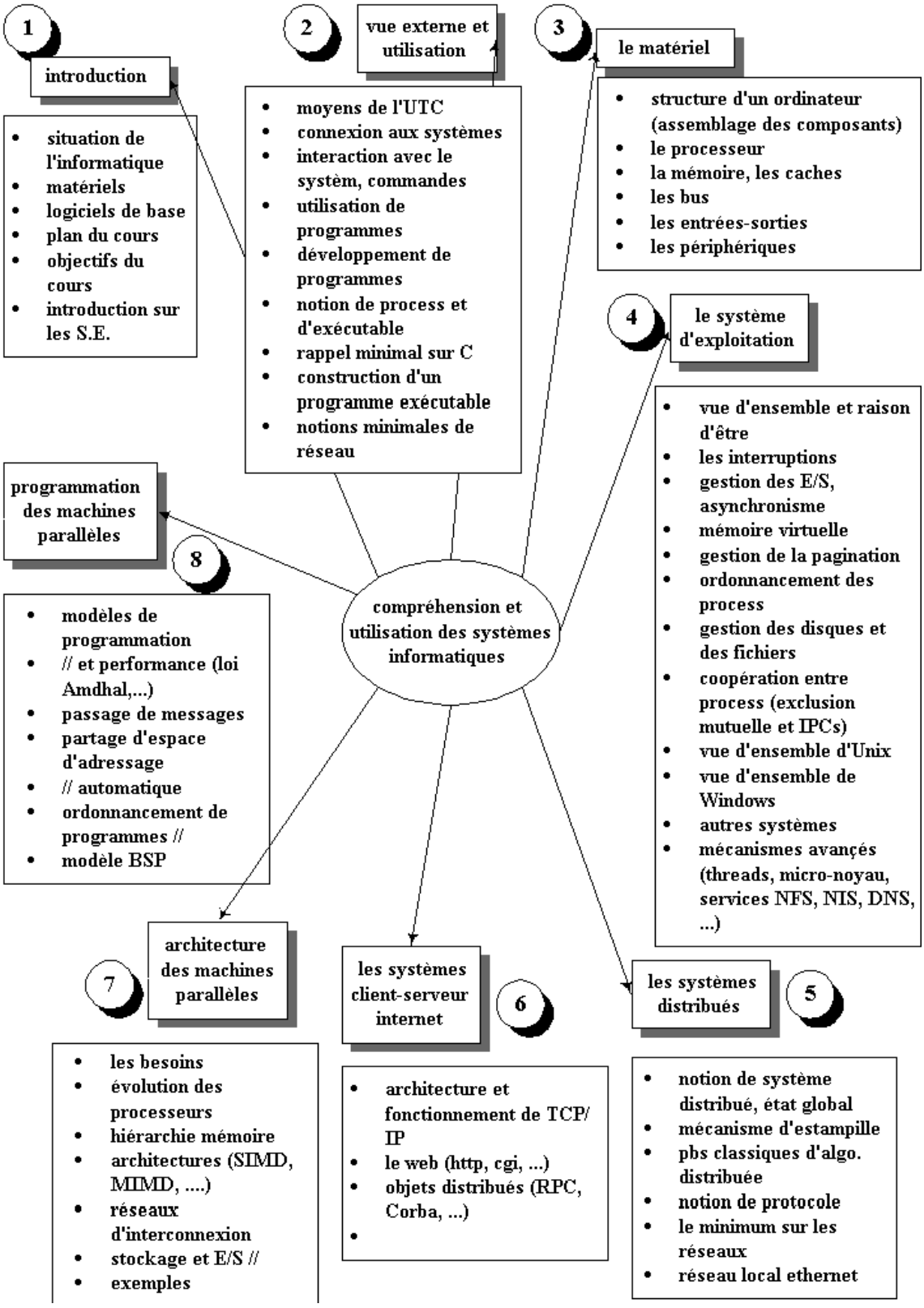


FIG. 13: L'informatique et les UVs systèmes 5/6

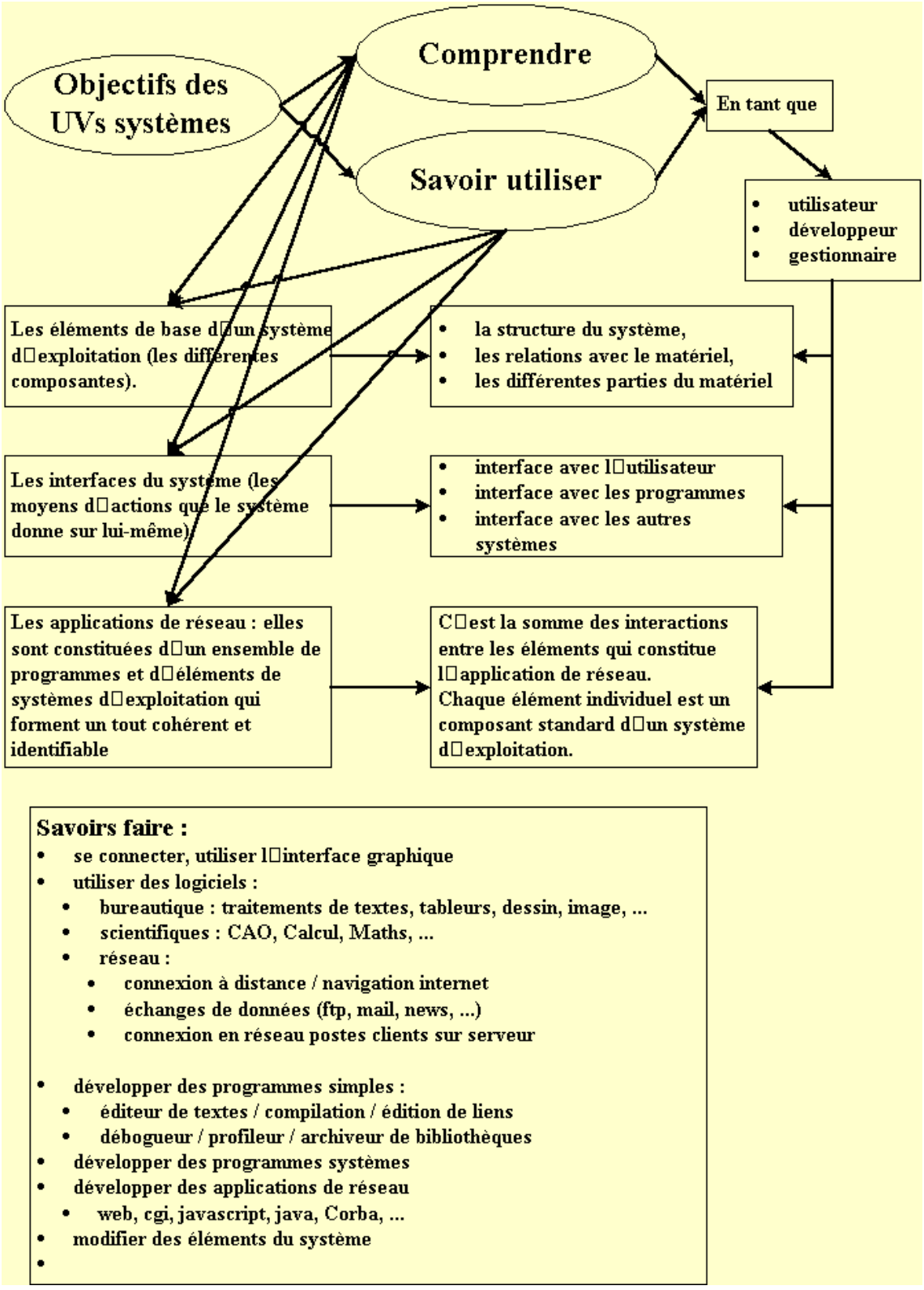


FIG. 14: L'informatique et les UVs systèmes 6/6

1.3 On en déduit le plan de l'UV

1. Enjeux de l'Internet, positionnement des UVs systèmes (ce chapitre)
2. Rappels et compléments sur les échanges entre process
3. Échanges entre process sur des machines différentes, modèles mémoire
4. Notion de protocole de communication
5. Introduction au fonctionnement d'un ethernet
6. Concepts de base d'un Internet, introduction à TCP/IP
7. Les protocoles UDP et TCP
8. Communication par RPC
9. Le web et http
10. Les CSS
11. Interactivité sur le client : javascript
12. Interaction entre client et serveur web : CGI, PHP
13. Client-serveur et objets distribués : javaRMI
14. Java, le client-serveur et le web
15. Le système Java
16. Programmation Java
17. Servlets et JSP
18. Client-serveur et objets distribués : CORBA
19. CORBA : exemple horloge ; IDL ; héritage et délégation
20. CORBA : le POA
21. XML et l'Internet, XSLT
22. XMLRPC, services web
23. Les Serveurs d'application Internet
24. Serveurs d'application java : (Servlets, JSP, J2EE)
25. Autre exemple de serveur d'application : ZOPE
26. Le parallélisme et PVM

Page blanche

SR03 2004 - Cours Architectures Internet -
Rappels et compléments sur les communications entre process

2 SR03 2004 - Cours Architectures Internet - Communications entre process

2.1 Les divers mécanismes de communications entre process sous unix

```
Pour communiquer entre processus, on a déja vu:
    - les signaux,
    - les pipes simples (créés par un process ancêtre),
    - les pipes nommés (ou FIFO).

Le Sys V offre 3 mécanismes de communication:

    - la mémoire partagée,
    - les sémaphores,
    - les files de messages.

Le système BSD offre un mécanisme de communication entre process
qui permet deux types de communications:
    - par deux process situés sur la même machine (UNIX DOMAIN),
    - par des process situés sur des machines différentes
      (INTERNET DOMAIN).
Dans les deux cas l'interface de programmation est la même:
    - les sockets.

La version 3 de System V offre également un mécanisme de
communication à travers le réseau: l'interface TLI ("Transport
Layer Interface").

Cette interface TLI a été normalisée par le groupe X/Open sous
le nom de "XTI".

En fait les systèmes livrés par les constructeurs
(DEC,SUN,HP,...) contiennent les deux types de mécanismes.

L'interface "socket" est de niveau un plus élevé que XTI. Par
exemple, dans Ultrix (l'u*x de DEC) les sockets sont implantés
par des appels à XTI, qui est également disponible pour le
programmeur.
```

2.2 Le fonctionnement des "pipes" sous unix

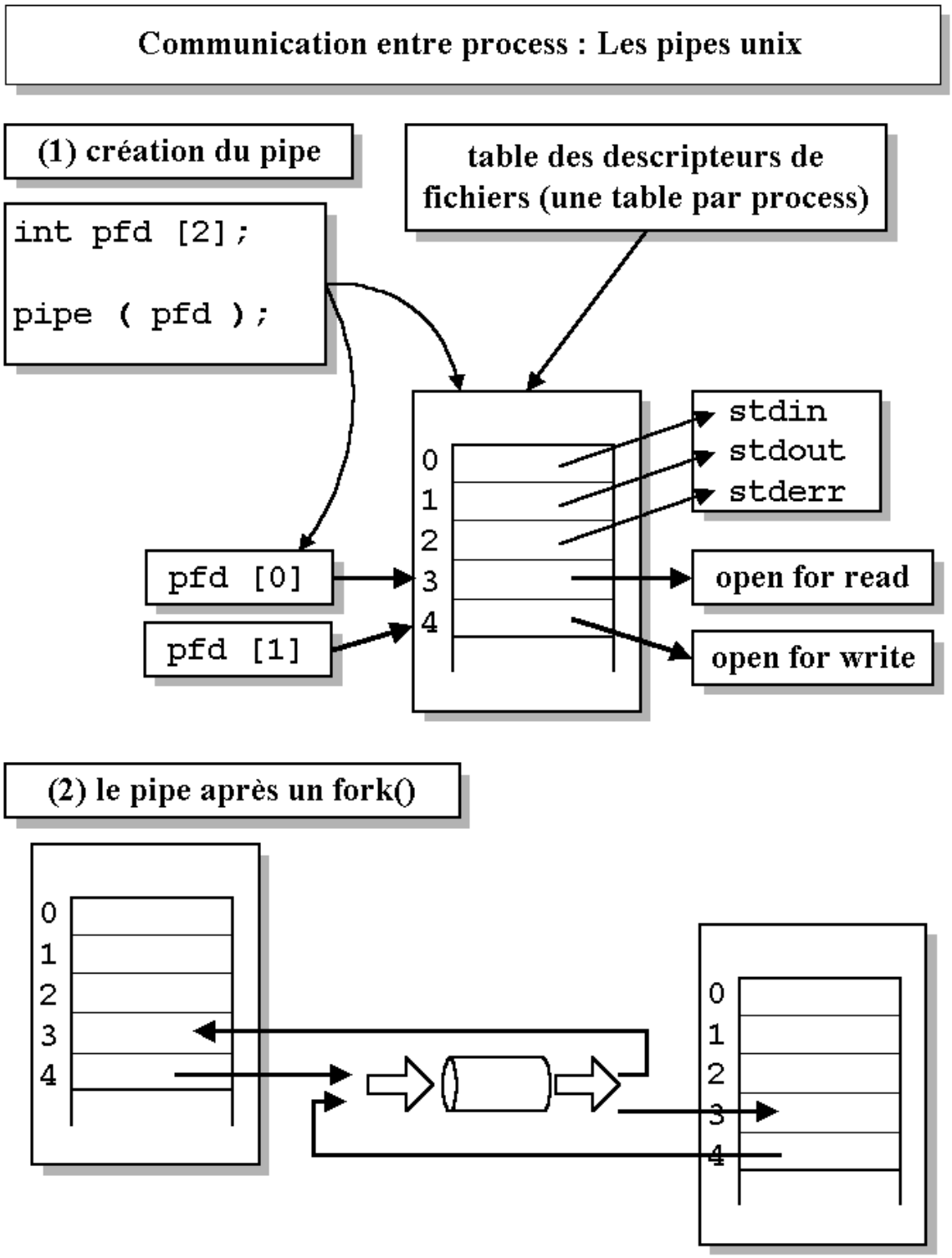


FIG. 15: Les pipes unix 1/4

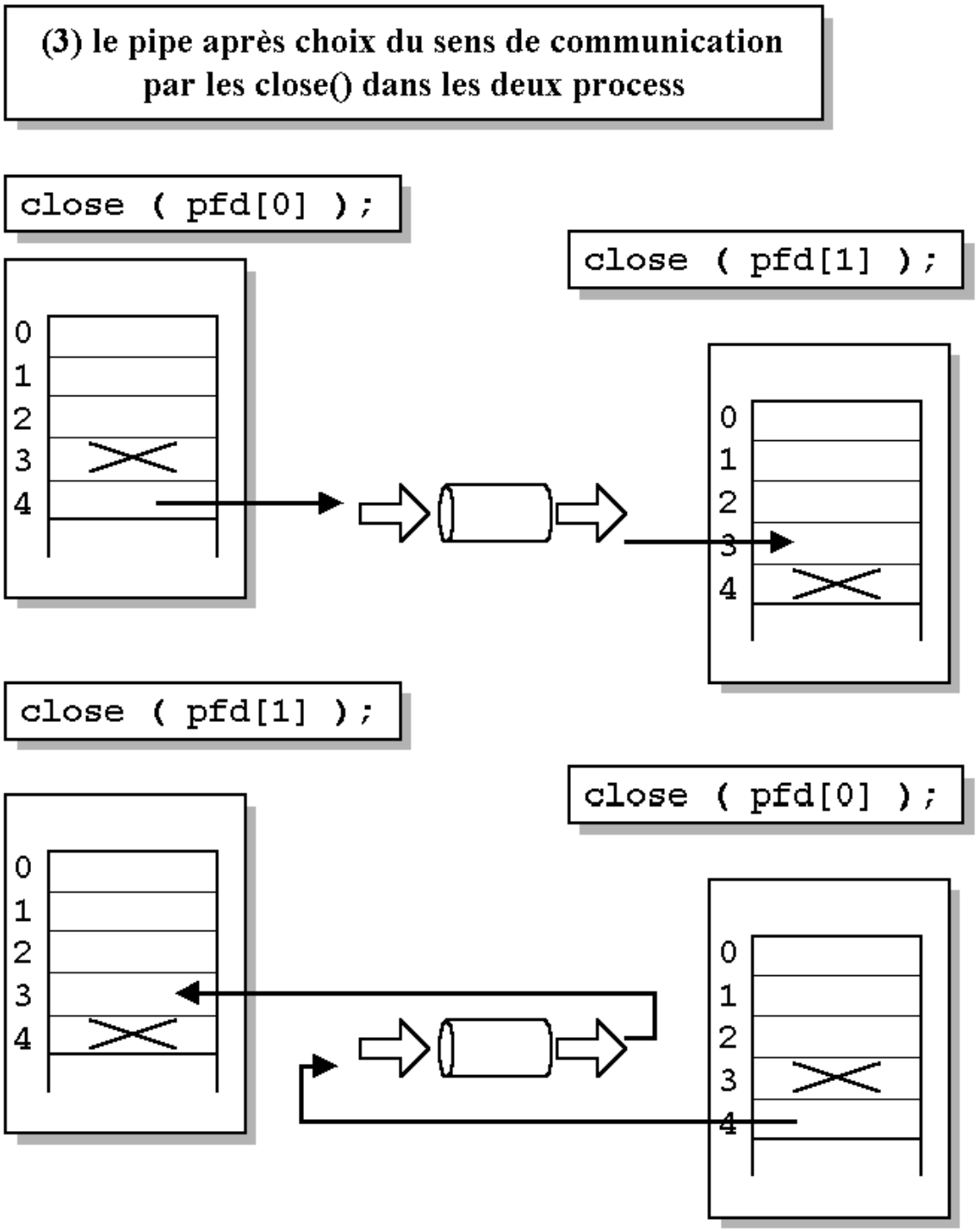


FIG. 16: Les pipes unix 2/4

(4) lecture dans le pipe

```
#define taille_buf 10;
int nbre_lu;
char buf[ taille_buf ];
...
nbre_lu = read ( pfd[0], buf, taille_buf );
...
```

1) si le tube n'es pas vide (et contient "contenu" caractères):
read va lire min (contenu, taille_buf)
2) si le tube est vide :
2.1) si le nombre d'écrivains est nul :
aucun car. lu, et retour de nbre_lu = 0
2.2) si le nombre d'écrivains n'es pas nul
2.2.1 -- si la lecture est BLOQUANTE
(cas par défaut) le processus est mis en sommeil
***** DANGER de bloquage infini si on a pas fait les close() correctement car le process croit à tord qu'il y a encore des écrivains**
2.2.2 -- si la lecture est NON-BLOQUANTE
retour avec
0 si on utilisé O_NODELAY
-1 et errno=EAGAIN si on a utilisé O_NONBLOCK

--> voir (5) lecture non bloquante

FIG. 17: Les pipes unix 3/4

(5) lecture non bloquante

```
#include <fcntl.h>
int status, pfd[2];
...
pipe (pfd);
...
...
status = fcntl ( pfd[0], G_GETFL );

fcntl (pfd[0], F_SETFL, status|O_NONBLOCK );
```

```
O_NONBLOCK  = POSIX

O_NODELAY   = System V
```

Synchronisation père / fils

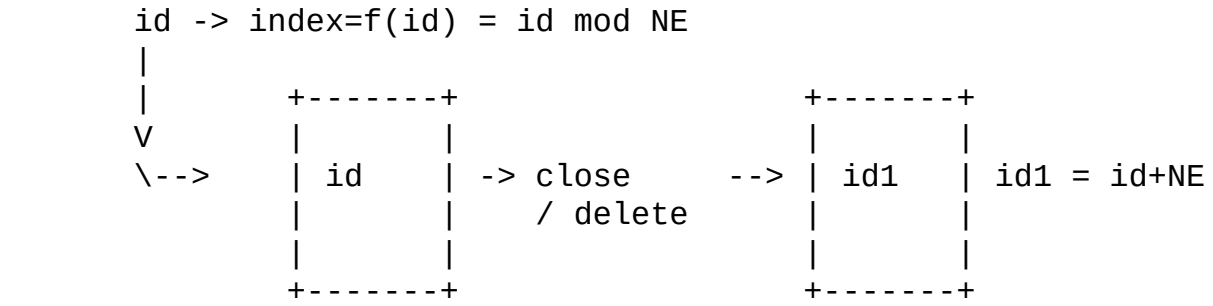
```
#include <fcntl.h>
int status, pfd[2];
...
pipe (pfd);
...
...
status = fcntl ( pfd[0], G_GETFL );

fcntl (pfd[0], F_SETFL, status|O_NONBLOCK );
```

FIG. 18: Les pipes unix 4/4

2.3 Caractéristiques communes aux mécanismes d'IPC de Sys V

```
Caractéristiques communes aux mécanismes Sys V
----------------------------------------------

o chaque mécanisme contient une table dont les entrées
  décrivent toutes les occurences du mécanisme:
      - il y a une table des segments de mémoire partagée,
      - une table des sémaphores,
      - une table des têtes de files de messages.

  Chacune de ces tables est GLOBALE pour tout le système.

o chaque entrée contient une "clé" numérique qui est le nom
  donné par l'utilisateur à cette entrée:
      - chaque mécanisme contient un appel "get" pour créer
        une nouvelle entrée ou retrouver une entrée existante,
        parmi les paramètres de cet appel ont trouve la "clé"
        et des drapeux (flags).
              id = xxxget(clé, ...)

      . les process peuvent appeler "get" avec la clé spéciale
        IPC_PRIVATE pour obtenir en retour une clé non
        utilisée.
      . les process peuvent appeler "get" avec le drapeau
        IPC_CREAT et une clé pour créer une nouvelle entrée
        pour cette clé s'il n'en existe pas déja.
      . les appels "get" renvoient un "descripteur" destiné à
        être utilisé lors des appels aux autres fonctions que
        get
             id = xxxget(clé, ...)
              |
              \--> descripteur --\
                                  |
                        xxxYYY(id, ...)

o réallocation des descripteurs:
     Afin d'éviter qu'un process ne réutilise un descripteur
     obsolète après l'avoir "fermé" et accède ainsi à un
     descripteur qui ne lui appartient pas, le système utilise
     la méthode suivante pour allouer les descripteurs:
      . supposons que la table du mécanisme contienne NE=100
        entrées,
      . lorsque le descripteur "1" est fermé après usage par
        un process, le système incrémente le descripteur
        assicié à cette entrée de la longueur de la table.
        Ainsi, les descripteurs successifs de l'entrée 1
        seront 101,201,301,...
```

. la séquence suivante est alors possible:
- process A crèe une entrée et reçoit desc. 101
- process A utilise le mécanisme en donnant 101 comme paramètre, le système calcul le numéro de l'entrée associée par:
index = descripteur modulo (nombre entrées dans table)
- process A ferme l'entrée associée au desc. 101
- le système incrémente le descripteur associé à l'entrée "1", qui devient donc 201
- process B crèe une entrée, si le système lui alloue l'entrée "1", il reçoit le descripteur 201,
- process A essaie de faire une opération sur le descripteur 101, le système regarde dans l'entrée 1 et trouve "201", il renvoie une erreur au process A.



o chaque entrée possède des champs d'autorisation qui contiennent:
- l'UID et le GID du process qui a créé cette entrée,
- un UID et un GID remplis par l'appel "control",
- un ensemble de bits rwx pour user-group-world semblables à ceux des fichiers.

o chaque entrée contient d'autres champs qui mémorisent des informa- tions d'état telles que:
- PID du dernier process ayant modifié cette entrée,
- date du dernier accès ou de la dernière mise à jour.

o chaque mécanisme contient un appel "control" qui permet:
- d'obtenir des infos sur l'état de l'entrée,
- de modifier ces infos d'état,
- de retirer l'entrée de la table.

appel : xxxctl()

Pour toutes ces opérations le système vérifie que le process appelant possède les droits correspondants (bits rwx et UID,GID du process).

Espace des noms

L'ensemble des noms possibles pour un type donné d'IPC est appelé son espace des noms (name space). Cet espace des noms est important car le "nom" est ce qui permet aux différents process coopérants d'accéder à la même instance du mécanisme de communication (par exemple le même sémaphore).

Type d'IPC	espace des noms	identification
pipe	(pas de nom)	descripteur fichier
pipe nommé (FIFO)	chemin de fichier	descripteur fichier
mémoire partagée	clé "key_t"	identificateur
sémaphores	clé "key_t"	identificateur
queue de messages	clé "key_t"	identificateur
socket - domaine u*x	chemin de fichier	descripteur fichier
socket - dom.internet	numéro port TCP	descripteur fichier

La fonction ftok est fournie pour convertir un chemin de fichier en une clé numérique (un entier sur 32 bits):

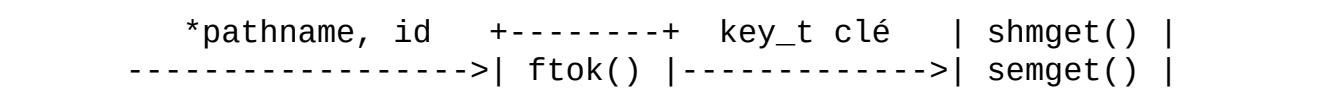
Name
ftok - standard interprocess communication package
Syntax

```
#include <sys/types.h>
#include <sys/ipc.h>

key_t ftok(char *pathname, char id);
```

Le pathname doit exister. S'il n'existe pas ou n'est pas accessible au process appelant, ftok retourne -1.

Les process qui veulent communiquer entre eux, ont seulement à se mettre d'accord sur le nom du pathname et à appeler chacun ftok avec LES mêmes arguments. "id" est aussi un argument d'entrée: c'est un octet (pas un pointeur). Si des process ont besoins de PLUSIEURS canaux de communication, ils vont appeler ftok plusieurs fois avec le même pathanme mais des valeurs différentes de "id" pour distinguer les appels.

Ftok() combine la valeur binaire de "id" avec le numéro de i-node du pathname ET avec le "minor device number" du système de fichier sur lequel réside le pathname. Cette combinaison forme un entier sur 32 bits qui est normalement unique. Cet entier va servir de clé lors de l'appel pour l'ouverture ou la création de l'IPC.



```
                        +--------+              | msgget() |
```

Opérations sur les IPC

	mémoire partagée	sémaphope	queue de messages
include file	<sys/shm.h>	<sys/sem.h>	<sys/msg.h>
créer/ouvrir	shmget	semget	msgget
opér.contrôle	shmctl	semctl	msgctl
IPC opérations	shmat	semop	msgsnd
	shmdt		msgrcv

la logique d'ouverture ou de création d'un canal IPC

Le comportement de l'appel suivant les drapeaux en paramètre et la place disponible dans la structure de gestion de l'IPC appelé:

	argument flag	clé n'existe pas	clé existe déja
1	aucun	error, errno= ENOENT	OK
2	IPC_CREAT	OK, crèe nvelle entrée	OK
3	IPC_CREAT|IPC_EXCL	OK, crèe entrée	errno=EEXIST

Rem: ligne 2, on ne sait pas si on a crée une nouvelle entrée
--- ou si on a utilisé une entrée existante
ligne 3: demande création seulement si entrée existe pas déja

Quand un nouveau canal est créé, les 9 bits de poids faible du paramètre "flag" initialisent le champ "mode protection" de l'entrée créée.
De plus, les champs suivants sont initialisés:
uid = cuid = effective UID du process créateur
gid = cgid = effective GID du process créateur
cuid et guid ne changent plus jamais jusqu'à destruction/ fermeture de l'IPC, les champs uid et gid peuvent être modifiés par un appel à la fonction xxxctl()

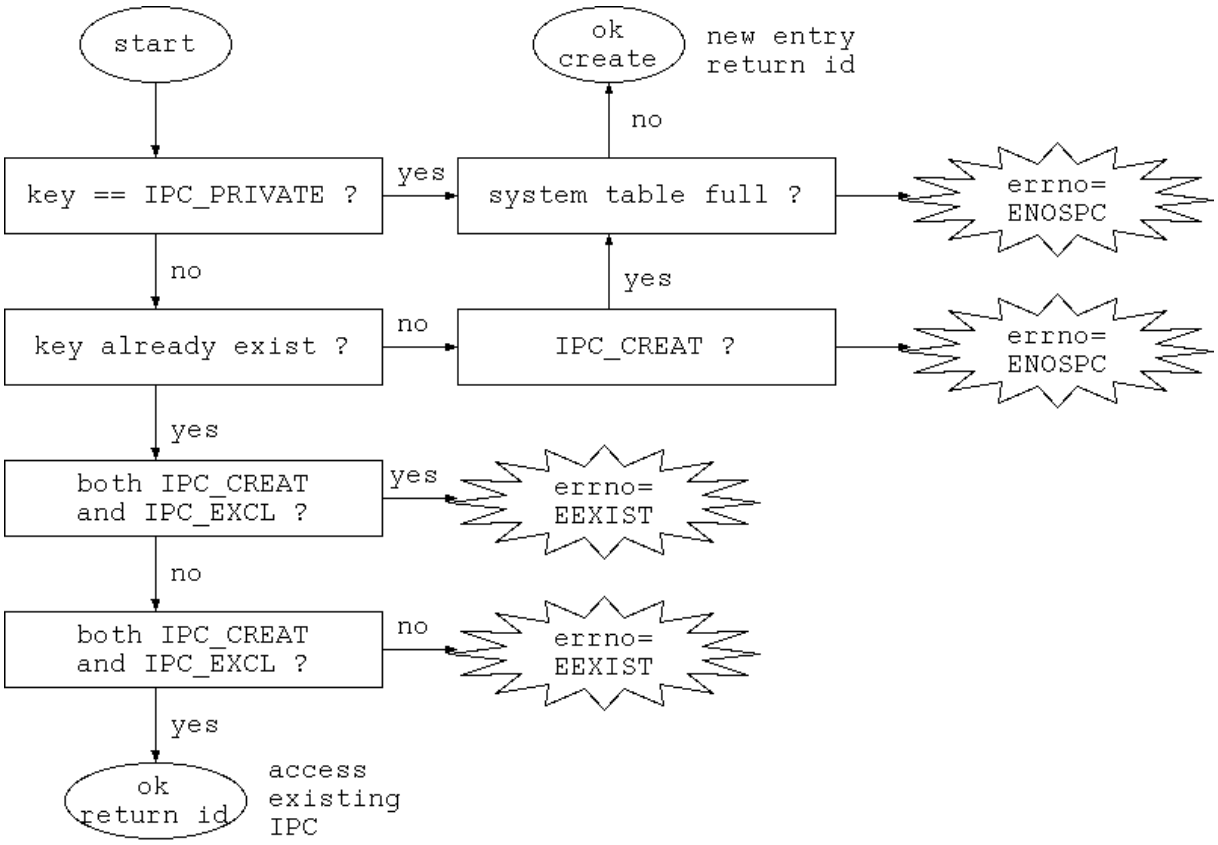


FIG. 19: Logique d'ouverture ou création d'un IPC

```
$ ipcs

------ Shared Memory Segments --------
key        shmid   owner     perms   bytes    nattch  status
0x00000000 65536   vayssade  600     393216   2       dest
0x00000000 98305   vayssade  600     393216   2       dest

------ Semaphore Arrays --------
key        semid    owner    perms   nsems

------ Message Queues --------
key        msqid    owner    perms   used-bytes  messages


$ apropos ipc

ftok      (3) - convert a pathname and a project identifier to
                a System V IPC key
ipc       (2) - System V IPC system calls
ipc       (5) - System V interprocess communication mechanisms
ipcrm     (8) - remove a message queue, semaphore set or
                shared memory id
ipcs      (8) - provide information on ipc facilities
perlipc   (1) - Perl interprocess communication (signals, fifos,
```

```
                pipes, safe subprocesses, sockets, semaphores)

ipcs -b  nbre d'octets ou nbre de process attachés
ipcs -c  créateur / owner
ipcs -p  pid dernier process ayant utilisé
ipcs -t  date dernier accès
```

```
ipcrm -q id détruire la queue d'identifiant "id"
ipcrm -m key détruire la file de message de clé "key"
```

2.4 Les files de messages

L'IPC System V de type "file de messages" (message queue) est implanté comme une structure dans l'espace système. Ainsi une fois qu'un message est déposé dans le file, il va être conservé même si le process émetteur se termine avant que le destinataire n'ai reçu le message.

Les opérations d'envoi (send) et de réception (receive) sont toujours à l'initiative des process utilisateurs.

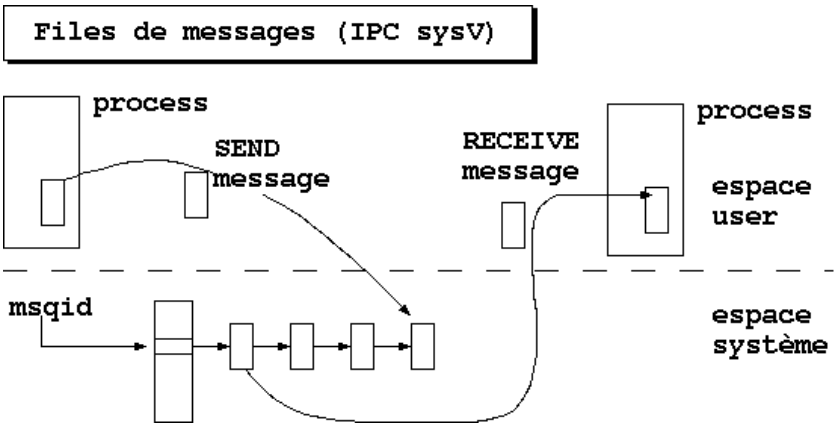


FIG. 20: L'IPC file de messages : implantation

```
rappel : le type contenu dans un message est obligatoirement >0.
MSGOP(2)            Linux Programmer's Manual           MSGOP(2)
NAME
     msgop - message operations
SYNOPSIS
     #include <sys/types.h>
     #include <sys/ipc.h>
     #include <sys/msg.h>

     int msgsnd(int msqid, struct msgbuf *msgp, size_t msgsz,
     int msgflg);
```

```
     ssize_t  msgrcv(int msqid, struct msgbuf *msgp, size_t msgsz,
     long msgtyp, int msgflg);

DESCRIPTION
     To send or receive a message, the calling process allocates
     a structure of the following general form:

        struct msgbuf {
             long mtype;     /* message type, must be > 0 */
             char mtext[1];  /* message data */
        };
```

Caractéristiques des files de messages :
- tous les messages sont stockés dans le noyau,
- un int msqid identifie la file,
- les process lisent ou écrivent dans n'importe quelle queue,
- il n'y a pas de synchronisation écrivain/lecteur nécessaire comme dans le cas des PIPES ou des FIFOs,
- un process peut écrire un ou plusieurs messages, un autre les lire plus tard, à son rythme.

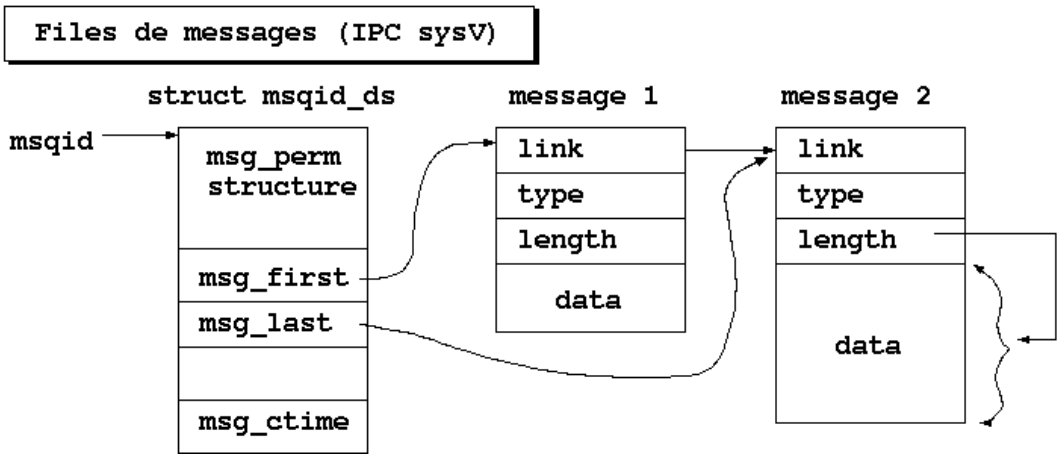


FIG. 21: L'IPC file de messages : structure interne

```
     struct msgbuf {  long mtype;
                      char mtext[n];}
     struct msgbuf buf;

     id = msgget(MA_cle, PROT|IPC_CREAT);
      \_______
        |      |
        |     msgsnd (id, &buf, lng, flag);
        |_______________
        |               |
        |     msgrcv (id, &buf, lng, type, flag);
        |_____________
                      |
             msgctl(id, IPC_RMID,0);
```

```
argument "type" dans msgrcv :
      type de message demandé
      si type=0 : demande le 1er message de la file
      si type>0 : demande le 1er message de ce type
      si type<0 : demande le 1er message dont le type est
                  le plus petit qui soit aussi <= au type demandé
rappel : le type contenu dans un message est obligatoirement >0.
```

Page blanche

3 SR03 2004 - Cours Architectures Internet - Les sockets

3.1 Origine, présentation et conception des sockets

Les sockets sont apparus avec BSD 4.2 et se sont stabilisés avec BSD 4.3. Leur utilité a fait qu'ils sont maintenant disponible sur tous les systèmes unix et sur de nombreux autres systèmes.

Ce sont en effet, le support privilégié de la plupart des communications ebtre process sur l'Internet.

C'est une interface de communication générale permettant de créer des applications distribuées.

Il crèe un canal unique bidirectionnel (full duplex), contrairement aux "pipes" unix qui ne fonctionnent que dans un seul sens.

Les sockets couvrent deux domaines et sont de deux types différents.

Domaines :

- domaine AF_UNIX : permet à des process de la même machine de communiquer entre eux ;
- domaine AF_INET : permet à des process situés sur deux machines différentes d'un même internet de communiquer. Ceci suppose que les deux machines communiquent par la suite de protocoles réseau TCP/IP.

Un domaine défini donc l'ensemble des autres sockets avec lesquels un process donné pourra communiquer.

Types :

le type du socket défini les propriétés de la communication. Les deux types principaux sont :

- le type datagramme, permettant l'échange de messages complets et structurés, sans négociation préalable (communication sans connexion) ;
- le type connecté, permettant l'échange de flux de données sur un circuit virtuel préalablement établi.

3.2 Objectifs de conception de l'interface socket

Les sockets ont été conçus pour permettre à des process de communiquer à travers un réseau tout en respectant :

- **transparence :** que la communication soit identique si les process sont sur la même machine ou non ;
- **efficacité :** implantés comme une bibliothèque système incluse dans le noyau, ils ont accès direct au driver de l'interface réseau évitant ainsi une double recopie du buffer de données et ils travaillebt comme un appel système depuis le process utilisateur, dans le contexte de celui-ci, sans obliger à un changement de contexte comme cela aurait été le cas s'ils avaient été implantés sous forme de process indépendant ;
- **compatibilité :** les sockets sont vus des process comme des descripteurs de fichiers, ce qui permet les opérations habituelles de redirection des E/S (dup()) ;

3.3 Mode client-serveur et établissement d'une communication

La caractéristique la plus fréquente des connexions en mode "client-serveur" est d'être asymétrique :

- **d'abord**, un process **serveur** se prépare ; il se mets en écoute sur un **port** de communication. Ce n'est encore qu'une "moitié" de la liaison. C'est le robinet sur lequel le tuyau n'a pas encore été branché.
- **ensuite**, un process **client** va envoyer à ce port serveur déja existant une demande de connexion.
- **alors**, la connexion est établie et les données peuvent circuler dans les deux sens.

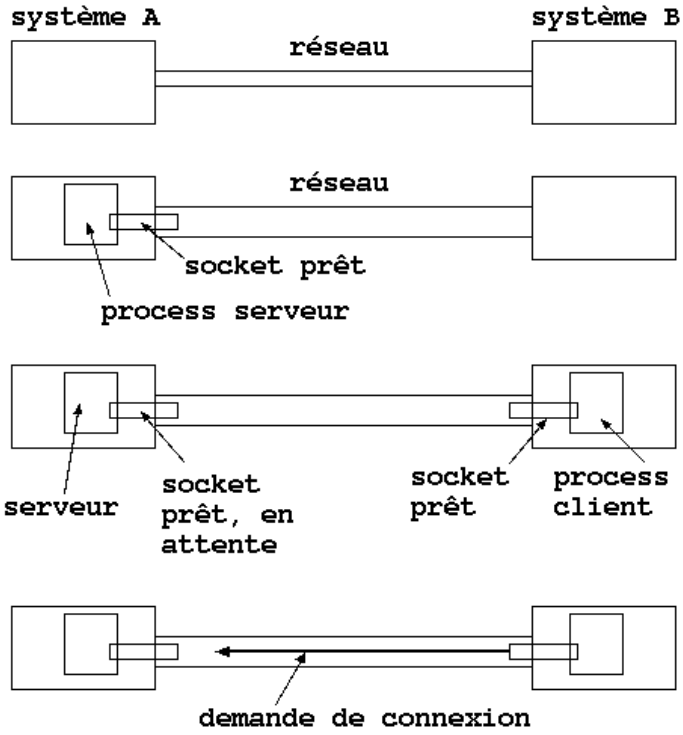


FIG. 22: Fonctionnement d'une communication en client-serveur 1/2

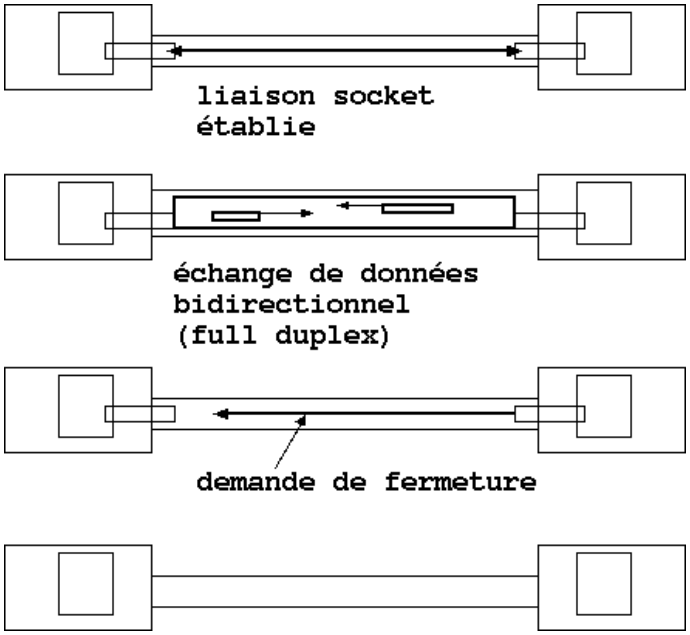


FIG. 23: Fonctionnement d'une communication en client-serveur 2/2

3.4 Les types de sockets

Les deux types principaux de sockets correspondent aux protocoles **sans connexion** et aux protocoles **orientés connexion** :

- protocoles **sans connexion** : ("connectionless") il y a échange de messages autonomes, complets et structurés ; chaque envoi doit contenir le destinataire ; ceci peut se comparer à l'envoi de lettres par la poste.
- protocoles **orientés connexion** : ("connection oriented") après établissement de la connexion (circuit virtuel), tous les envois sont implicitement pour le même destinatire ; ceci peut se comparer au téléphone.

1. **sockets SOCK_DGRAM** : envoi de datagrammes (messages), mode non connecté ; protocole **UDP** au-dessus de **IP** :
   - messages de taille bornée,
   - préserve les frontières de message,
   - pas de garantie de remise ("best effort"),
   - pas de garantie de remise dans le même ordre que l'envoi ;
2. **sockets SOCK_STREAM** : flot de données, mode connecté, protocole **TCP** au-dessus de **IP** :
   - transfert fiable (pas de perte ni d'altération de données),
   - données délivrées dans l'ordre d'envoi,
   - pas de duplication,
   - supporte la notion de messages urgents ("out-of-band") pouvant être lus avant les données "normales" ;
3. **sockets SOCK_RAW** : permet d'accéder au protocole de plus bas niveau (IP) pour construire d'autres protocoles. Réservé au mode superviseur.
4. **sockets SOCK_SEQPACKET** : paquets avec préservation des frontières, délivrés de façon fiable et en séquence (dans l'ordre). Cité pour la complétude, était utilisé dans le protocole Decnet.

3.5 Caractéristiques d'un socket

Vu "de l'intérieur" du process utilisateur, un socket est un descripteur de fichiers. On peut ainsi rediriger vers un socket les E/S standard d'un programme développé dansun cadre local.

Une autre conséquence est que les sockets sont hérités par le fils lors d'un fork.

Un socket est créé par la primitive **socket()**. La valeur de retour étant le descripteur sur lequel on va faire les opérations de lecture et d'écriture.

La différence avec un fichier est qu'il est ensuite possible (et nécessaire !) **d'attacher** au socket une adresse du domaine auquel il appartient. Ceci se fait avec la primitive **bind()**. Cette adresse est communément appelée **numéro de port**.

3.6 Création et attachement d'un socket

```
NAME
       socket - create an endpoint for communication
SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>

       int socket(int domain, int type, int protocol);
```

Le troisième paramètre est souvent donné à 0, car souvent il n'y a qu'un seul type de protocole disponible (par exemple pour un SOCK_DGRAM dans le domaine AF_INET, il n'y a que UDP, et pour SOCK_STREAM que TCP).

```
Exemple:   sd = socket (AF_INET, SOCK_DGRAM, 0);
     ou:   sd = socket (AF_INET, SOCK_STREAM, 0);
```

Attachement (bind)

```
$ man 2 bind
NAME
       bind - bind a name to a socket
SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>

       int bind(int sockfd, struct sockaddr *my_addr,
       socklen_t addrlen);

(addrlen est la longueur de la structure my_addr)
```

3.7 Échange de données entre client UDP et serveur UDP

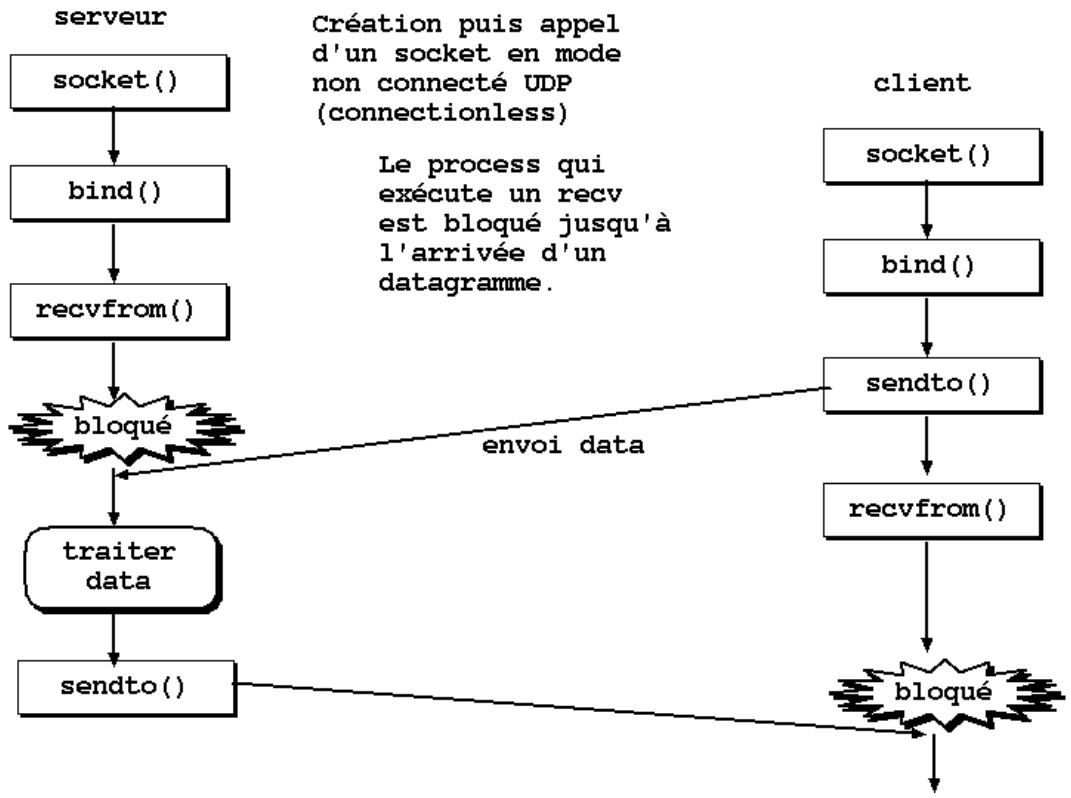


FIG. 24: Échange par socket UDP

```
$ man 2 send
$ man 2 recv

SEND(2)           Linux Programmer's Manual              SEND(2)
NAME
       send, sendto, sendmsg - send a message from a socket
SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>
       ssize_t send(int s, const void *msg, size_t len,
                    int flags);
       ssize_t  sendto(int  s,  const  void *msg, size_t len,
         int flags, const struct sockaddr *to, socklen_t tolen);
       ssize_t sendmsg(int s, const struct msghdr *msg,
                       int flags);
DESCRIPTION
       Send, sendto, and sendmsg are used to transmit a message
       to another socket. Send may be used only when the socket
       is in a connected state, while sendto and sendmsg may be
       used at any time.

       The address of the target is given by to with tolen
       specifying  its size. The length of the message is given
```

```
       by len. If the message is too long to pass atomically
       through the underlying protocol, the  error EMSGSIZE is
       returned, and the message is not transmitted.
```

L'émetteur peut être soit le client, soit le serveur. La seule contrainte est qu'ils doivent être programmés pour qu'il y ait autant de recv() exécutés par l'un que de send() exécutés par l'autre.

```
Exemples :
sendto(sa, bufa, nbufa, flag, &to, lento)
recvfrom( sb, bufb, nbufb, flag, &from, lenfrom)
ou:
send(sa, bufa, nbufa, flag)
recv(sb, bufb, nbufb, flag)
```

Remarque : le "man" précide que "Send may be used only when the socket is in a connected state". Or, on est ici dans un contexte UDP, c'est-à-dire sans connexion. Qu'en est-il ?

En fait il s'agit d'une facilité de programmation : il est possible, afin d'éviter de remettre dans chaque appel "sendto()" l'adresse du destinataire, si c'est toujours le même, de stocker cette adresse dans la structure socket. On peut alors utiliser simplement send() vers ce destinataire implicite. Il s'agit d'une **pseudo**-connexion dont l'autre entité n'a pas connaissance. C'est juste une facilité de programmation locale.

On peut ainsi faire un send() vers un process qui lui-même est uniquement programmé avec des recvfrom(), et inversement.

Flags :

- **MSG_OOB** send or recv Out-of-Band data
- **MSG_PEEK** permet de "regarder" les data à lire sans que le système ne les enlève du buffer d'arrivée.

3.8 Exemple de serveur et de client UDP

```
        Un serveur UDP  et  un client UDP
(1)  le serveur se met en attente sur un port UDP,
(2)  le client crèe son socket local,
(3)  le client envoie un message sur le host et le port UDP
     du serveur,
(4)  à récpetion du message, le serveur récupère l'adresse
     (host,port) de l'émetteur et traite le message,
     puis envoie une réponse.
```

udpser.c

```
 1  /* udpser.c  */
 2  /* UTC UV SR03 - (c) Michel.Vayssade@utc.fr
 3  linux :          gcc -o udpser udpser.c
 4  solaris :        gcc -o udpser udpser.c -lsocket -lnsl */
 5  #include <stdio.h>
 6  #include <sys/types.h>
```

udpser.c (suite)

```
 7  #include <sys/socket.h>
 8  #include <netdb.h>
 9  #include <netinet/in.h>
10  #include <errno.h>
11
12  #define PORT 4677       /* port du serveur */
13  int sd,i,n;
14  int *fromlen;
15  struct sockaddr_in mon_s;
16  struct sockaddr_in son_s;
17
18  typedef struct messages {
19          char mtext[50];
20  } message;
21  message *mess;
22
23  void    affiche_errno() { printf("\nerrno=%d",errno); }
24  main(){
25
26  sd=socket(PF_INET,SOCK_DGRAM ,0);
27
28  if (sd==-1) perror("udpser :err socket"),affiche_errno(),exit(1);
29  bzero(&mon_s,sizeof(mon_s));
30  /* accept input from any interface */
31  mon_s.sin_addr.s_addr = INADDR_ANY;
32  mon_s.sin_family = PF_INET;     /* sur IP */
33  mon_s.sin_port = htons(PORT);   /* port du serveur */
34
35  n=bind(sd,(struct sockaddr*)&mon_s, sizeof(mon_s));
36
37  if (n==-1) perror("udpser :err bind"),affiche_errno(),exit(1);
38  mess=(message*)malloc(sizeof(message));
39
40  fromlen=(int*)malloc(sizeof(int));
41  *fromlen=sizeof(son_s);
42
43  n=recvfrom(sd,mess,sizeof(message),0,
44            (struct sockaddr*)&son_s,fromlen);
45
46  if (n==-1) perror("udpser :err recvfrom"),affiche_errno(),exit(1);
47  printf("Message recu= %s\n",mess->mtext);
48
49  strcpy(mess->mtext,"serveur repond et envoie un message");
50  printf("serveur envoi message\n");
51
52  n=sendto(sd,mess,sizeof(message),0,
```

udpser.c (suite)

```
53          (struct sockaddr*)&son_s,sizeof(son_s));
54
55  if (n==-1) perror("udpser : err sendto "),affiche_errno(),exit(1);
56  }
```

udpcli.c

```
 1  /* udpcli.c   */
 2  /* UTC UV SR03 - (c) Michel.Vayssade@utc.fr
 3   linux :         gcc -o udpcli udpcli.c
 4  solaris :        gcc -o udpcli udpcli.c -lsocket -lnsl  */
 5  #include <stdio.h>
 6  #include <sys/types.h>
 7  #include <sys/socket.h>
 8  #include <netdb.h>
 9  #include <netinet/in.h>
10  #include <errno.h>
11
12  #define PORT 4677       /* port du serveur */
13  int sd,i,n;
14  int *fromlen;
15  struct sockaddr_in mon_s;
16  struct sockaddr_in son_s;
17  struct hostent *hs;
18
19  typedef struct messages {
20          char mtext[50];
21  } message;
22  message *mess;
23
24  void    affiche_errno() { printf("\nerrno=%d",errno); }
25
26  main(){
27  hs=gethostbyname("localhost");
28
29  sd=socket(PF_INET,SOCK_DGRAM ,0);
30
31  if(sd==-1)perror("udpcli :err socket"),affiche_errno(),exit(1);
32  bzero(&son_s,sizeof(son_s));
33  bcopy(hs->h_addr,&son_s.sin_addr,hs->h_length); /* adr IP serveur */
34  son_s.sin_family = PF_INET;     /* sur IP */
35  son_s.sin_port = htons(PORT);   /* port du serveur */
36
37  bzero(&mon_s,sizeof(mon_s));
38  /* adr client choisie par système */
```

udpcli.c (suite)

```
39  mon_s.sin_addr.s_addr = INADDR_ANY;
40  mon_s.sin_family = PF_INET;  /* sur IP */
41  mon_s.sin_port = htons(0);   /* port client choisi par le système */
42  /* si on impose meme que serveur et meme machine => déja utilisé */
43
44  n=bind(sd,(struct sockaddr*)&mon_s,sizeof(mon_s));
45
46  if (n==-1) perror("udpcli :err bind"),affiche_errno(),exit(1);
47  mess=(message*)malloc(sizeof(message));
48  strcpy(mess->mtext,"j'envoie un message");
49
50  n=sendto(sd,mess,sizeof(message),0,
51          (struct sockaddr*)&son_s,sizeof(mon_s));
52
53  if (n==-1) perror("udpcli :err sendto"),affiche_errno(),exit(1);
54  fromlen=(int*)malloc(sizeof(int));
55  *fromlen=sizeof(mon_s);
56  printf("lire message\n");
57
58  n=recvfrom(sd,mess,sizeof(message),0,
59            (struct sockaddr*)&mon_s,fromlen);
60
61  if (n==-1) perror("udpcli : err recvfrom "),affiche_errno(),exit(1);
62  printf("Message recu= %s\n",mess->mtext);
63  }
```

```
$ gcc -o udpser udpser.c
$ gcc -o udpcli udpcli.c
$ ./udpser                                  (1)
Message recu= j'envoie un message            |  (3)--\
serveur envoi message                        |   ^   |
                                             |   |   |
Dans une autre fenêtre :                     |   |   |
$ ./udpcli                                  (2)--/   V
lire message                                        (4)
Message recu= serveur repond et envoie un message
```

3.9 Échange de données entre client TCP et serveur TCP

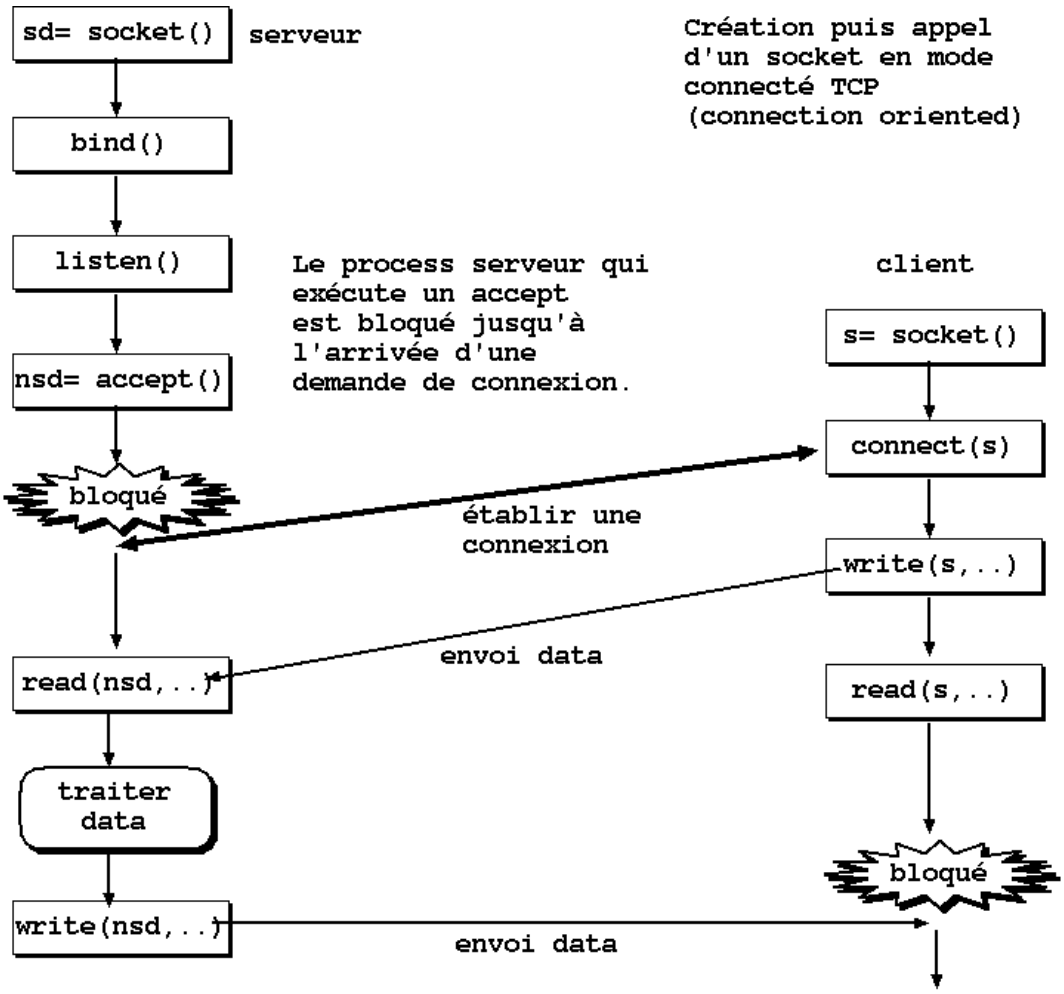


FIG. 25: Échange par socket TCP

Utilisation :

```
#include <sys/types.h>
#include <sys/socket.h>

sd = socket(AF_INET, SOCK_STREAM, 0);
|              |          |
descripteur    domaine    type
=-1 si erreur

#define PORT 0x1234
#include <netinet/in.h>  ! in: domaine internet
struct sockaddr_in mon_soc;
bzero(&mon_soc, sizeof(mon_soc)); /* init mon_soc */
mon_soc.sin_family = AF_INET;
mon_soc.sin_port = htons(PORT);

bind(sd, &mon_soc, sizeof(mon_soc));
```

```
/* bind() donne un identifiant (le numéro de port
   dans le domaine AF_INET) au socket pour que les
   clients puissent lui envoyer une demande de
   connexion en utilisant cet identifiant */

    listen(sd, 5);
/* listen indique au système que l'on va se mettre
   en attente de demandes de connexion */

    newsd = accept(sd, 0, 0);
/* on se met en attente d'arrivées de demandes de
   connexion de clients qui exécutent l'appel connect()
   Cet appel est BLOQUANT. On en ressort seulement si
   un client appelle le serveur.
   accept() renvoie un nouveau descripteur sur lequel
   se feront tous les échanges de données avec ce client.
   Le descripteur initial est réservé à l'écoute des
   demandes de connexion.  */
```

Différents types de serveurs TCP : itératifs et concourants

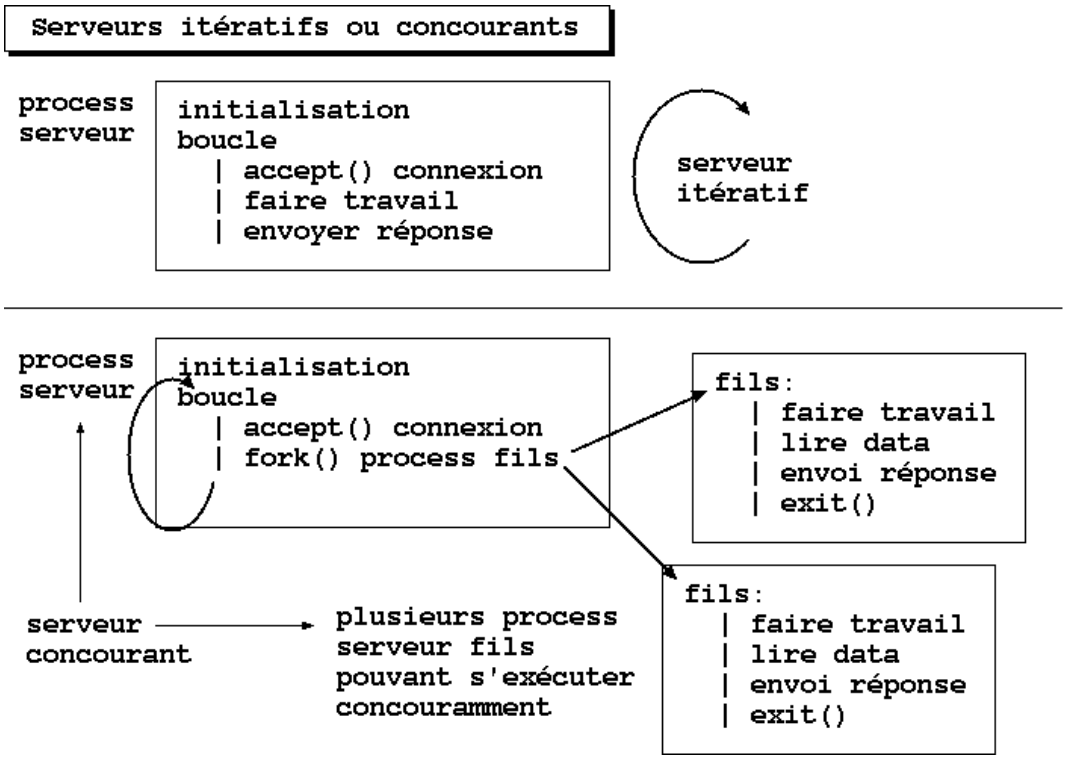


FIG. 26: Serveurs TCP itératifs et concourants

- **itératifs :** si le travail à faire est petit et prévisible, cette méthode est plus rapide et moins coûteuse.
- **concourants :** si le travail à chaque connexion est plus long ou non prévisible, il y a un risque de perte de demandes de connexion dans le mode itératif. On crèe alors un process fils pour faire le travail et le serveur principal reboucle aussitôt sur le accept().

3.10 Exemple de serveur et de client TCP

```
        Un serveur TCP  et  un client TCP
(1)  le serveur se met en attente sur un port TCP
(2)  le client se connecte au serveur
(3)  le serveur accepte la connexion sur un nouveau socket newsd
(4)  client boucle sur envoi "n" messages de tailles 8, 16,...
(5)  le serveur boucle sur la lecture du socket,
     pour récupérer des messages complets, de 8,16,32, il doit
     implanter une boucle (while(attendu)) pour chaque message
     car TCP mets tous les messages à la file, sans frontière.
```

tcpser.c

```
 1  /*  tcpser.c  */
 2  /* UTC UV SR03 - (c) Michel.Vayssade@utc.fr
 3   linux :         gcc -o tcpser tcpser.c
 4  solaris :        gcc -o tcpser tcpser.c -lsocket -lnsl
 5  - creer socket SOCK_STREAM
 6  - init structure sockaddr
 7  - bind socket to PORT
 8  - listen socket : accepte demandes connexions
 9  - accept une connexion demandee par un client (par ex. cli.c)
10  - boucle
11    . recv message on socket
12    . IF valeur[0] = -1 sortie de boucle, FIN programme
13    . afficher qques valeurs
14  */
15  #include <sys/types.h>
16  #include <sys/socket.h>
17  #include <netinet/in.h>
18  #include <netdb.h>
19  #include <signal.h>
20  #include <errno.h>
21
22  #define NBENREG         10  /* transferer NBENREG messages */
23  #define ENREGSIZE       8   /* de longueurs 8,16,24,32,40,..*/
24  #define END_OF_SEND     -1
25  #define SLEEPTIME       0
26  #define PORT            0x2001
27
28  union   type_uni {      char    txt[ENREGSIZE*NBENREG];
29                          double  valeur[NBENREG];  } msg_src;
30  static  int     nbErreurTotal = 0 , recu , attendu , fois;
31  void    affiche_errno() { printf("\nerrno=%d",errno); }
32  /*---------------------------------------*/
33  main() {
34          struct  sockaddr_in     sin;
```

tcpser.c (suite)

```
35          int     sd, i, j, newsd, count, nbInfo , d1=0;
36          errno = -1;
37          printf("\nser - init socket");
38  /*- socket SOCK_STREAM ----------------------------*/
39          if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
40                  perror("\nser : error init socket"),exit(1);
41          affiche_errno();
42  /*- init structure sockaddr -----------------------*/
43          bzero(&sin, sizeof(sin));
44          sin.sin_family = AF_INET;
45          sin.sin_port   = htons(PORT);
46  /*- bind socket to PORT ---------------------------*/
47          if(bind(sd, (struct sockaddr*)&sin,sizeof(sin))==-1)
48                  perror("ser : error bind"), exit(1);
49          affiche_errno();
50  /*- listen socket : accepte demandes connexions -----------*/
51          if (listen(sd, 5) == -1)
52                  perror("ser : error listen"), exit(1);
53          affiche_errno();
54          printf("\nAttachement reussi- Attente connexion.\n");
55  /*- accept une connexion demandee par un client ---*/
56          if ((newsd = accept(sd, 0, 0)) == -1)
57                  perror("ser : error accept"), exit(1);
58          affiche_errno();
59          printf("\nConnexion etablie\n");
60  /*- raz buffer lecture ----------------------------*/
61          for ( i=0; i<NBENREG; i++)
62                  msg_src.valeur[i] = 0;
63  /*- BOUCLE - lecture messages ----------------------*/
64          for ( nbInfo=1; nbInfo<= NBENREG+5; nbInfo++)
65      {
66          sleep(1);
67          attendu = nbInfo*ENREGSIZE; recu = 0; fois = 0;
68          while (attendu) {
69          count = recv(newsd, &msg_src.txt[recu],attendu,0);
70          if ((int)msg_src.valeur[0] == END_OF_SEND)
71          {       printf("\nRecu END_OF_SEND.");
72          printf("\n mess coupes= %6d. d1= %5d. Exit\n",
73                 nbErreurTotal,d1);
74                  exit(0); }
75          if (count==-1) perror("ser : error recv"),exit(1);
76          recu = recu + count; attendu= attendu - count;
77          fois = fois +1;
78          }
79              if (fois=1) d1= nbInfo; else nbErreurTotal++;
80  printf(\
81  "\nmsg # %4d, errno=%5d, recu= %4d,pre=%6d der=%6d en %3d fois",
```

tcpser.c (suite)

```
82  nbInfo,errno,recu,
83  (int)msg_src.valeur[0],(int)msg_src.valeur[nbInfo-1],fois);
84      }
85  printf("\n Fin bcle lecture. Mess coupes= %6d. d1= %5d. Exit\n",
86    nbErreurTotal,d1);
87  }
```

tcpcli.c

```
 1  /*  tcpcli.c  */
 2  /* UTC UV SR03 - (c) Michel.Vayssade@utc.fr
 3   linux :         gcc -o tcpcli tcpcli.c
 4  solaris :        gcc -o tcpcli tcpcli.c -lsocket -lnsl
 5  - lire adresse host destinataire
 6  - init structure sockaddr
 7  - ouvrir socket SOCK_STREAM
 8  - demande connexion sur PORT
 9  - BOUCLE
10      . prepare message
11      . send message lng 8, 16, 24, 32,...
12  - envoi 3 messages de fin
13  */
14  #include <sys/types.h>
15  #include <sys/socket.h>
16  #include <netinet/in.h>
17  #include <netdb.h>
18  #include <signal.h>
19  #include <errno.h>
20
21  #define NBENREG         10
22  #define ENREGSIZE       8
23  #define END_OF_SEND     -1
24  #define SLEEPTIME       0
25  #define PORT            0x2001
26
27  union   type_uni {      char    txt[ENREGSIZE*NBENREG];
28                          double  valeur[NBENREG];  } msg_src;
29  void    init_table(int taille);
30  void    affiche_errno() { printf("\nerrno=%d",errno); }
31  /*--------------------------------------*/
32  main() {
33          struct  sockaddr_in     sin;
34          struct  hostent         *hp;
35          int     sd, i, j, nbInfo, result, count;
36
```

tcpcli.c (suite)

```
37          errno = -1;
38  /*- lire adresse destinataire dans "/etc/hosts" --------*/
39          if ((hp = gethostbyname("localhost")) == 0)
40                  perror("cli : gethostbyname"), exit(1);
41  /*- init structure sockaddr ----------------------*/
42          bzero(&sin, sizeof(sin));
43          bcopy(hp->h_addr, &sin.sin_addr, hp->h_length);
44          sin.sin_family = hp->h_addrtype;
45          sin.sin_port   = htons(PORT);
46  /*- ouvrir socket ---------------------------*/
47          sd = socket(AF_INET, SOCK_STREAM, 0);
48          affiche_errno();
49          if (sd == -1) perror("cli : socket"), exit(1);
50  /*- connect sur PORT ------------------------*/
51          result = connect(sd,(struct sockaddr*)&sin,sizeof(sin));
52          affiche_errno();
53          if (result == -1) perror("cli : connect"), exit(1);
54          printf("\nConnexion etablie. Envoi messages.");
55  /*- BOUCLE envoi successif d'une table de 8,16,... octets -*/
56          for ( nbInfo=1; nbInfo<=NBENREG; nbInfo++)
57          {
58               init_table(nbInfo);
59               count = send(sd, msg_src.txt,nbInfo*ENREGSIZE, 0);
60               if (count == -1) perror("cli : send"), exit(1);
61  printf("\nmsg fin # %4d errno=%4d envoi %5d octets pre=%6d der=%6d",
62  nbInfo,errno,nbInfo*ENREGSIZE,(int)msg_src.valeur[0],
63      (int)msg_src.valeur[nbInfo-1]);
64          }
65  /*- Envoi message terminaison ---------------------*/
66          nbInfo = 1;
67          msg_src.valeur[0] = END_OF_SEND;
68          for ( i=1; i<=3; i++)
69          {   count = send(sd, msg_src.txt,nbInfo*ENREGSIZE,0);
70              if (count==-1) perror("cli :send termin."),exit(1);
71  printf("\nmsg fin # %4d errno=%4d envoi %5d octets pre=%6d der=%6d",
72  i,errno,nbInfo*ENREGSIZE,(int)msg_src.valeur[0],
73      (int)msg_src.valeur[nbInfo-1]);
74          }
75  /*- Fermer connexion ------------------------*/
76          close(sd); printf("\nExit.\n"); exit(0);
77  }
78
79  void  init_table(taille)
80  int   taille;
81  {     int     j, delay;
82        msg_src.valeur[0] = (double)taille;
83        for ( j=1; j<=taille-1; j++) msg_src.valeur[j]= (double)j;
```

tcpcli.c (suite)

```
 84        for ( j=taille+1; j<NBENREG; j++) msg_src.valeur[j] = 0.;
 85        sleep(SLEEPTIME);
 86  }
```

```
$ gcc -o tcpcli tcpcli.c
$ gcc -o tcpser tcpser.c

$ ./tcpser                              (1)
                                         |
ser - init socket                        |
errno=-1                                 |
errno=-1                                 |
errno=-1                                 |
Attachmt reussi- Attente connexion      (2)
                                         |
errno=-1                                 |
Connexion etablie                        | (4)------\  (6) réception
                                         |  |       |  |
msg #    1, errno= -1, recu=   8,pre=    |  | 1 der= |  | 1 en 1 fois
msg #    2, errno= -1, recu=  16,pre=    |  | 2 der= |  | 1 en 1 fois
.............                            |  |       |  |
msg #   10, errno= -1, recu=  80,pre=    |  |10 der= |  | 9 en 1 fois
Recu END_OF_SEND.                        |  |       |  |
 mess coupes=      0. d1=    10.Exit     |  |       |  |
                                         |  |       |  |
Dans une autre fenêtre :                 |  |       |  |
$ ./tcpcli                               |  |       |  |
                                         |  |       |  |
errno=0                                  |  |       |  |
errno=0                                  V  |       |  |
Connexion etablie. Envoi messages.      (3)-/      (5)-/ envoi

msg fin #    1 errno=   0 envoi     8 octets pre=   1 der=   1
msg fin #    2 errno=   0 envoi    16 octets pre=   2 der=   1
.............
msg fin #   10 errno=   0 envoi    80 octets pre=  10 der=   9
msg fin #    1 errno=   0 envoi     8 octets pre=  -1 der=  -1
msg fin #    2 errno=   0 envoi     8 octets pre=  -1 der=  -1
msg fin #    3 errno=   0 envoi     8 octets pre=  -1 der=  -1
Exit.
```

3.11 Modèles mémoires différents et échanges de données

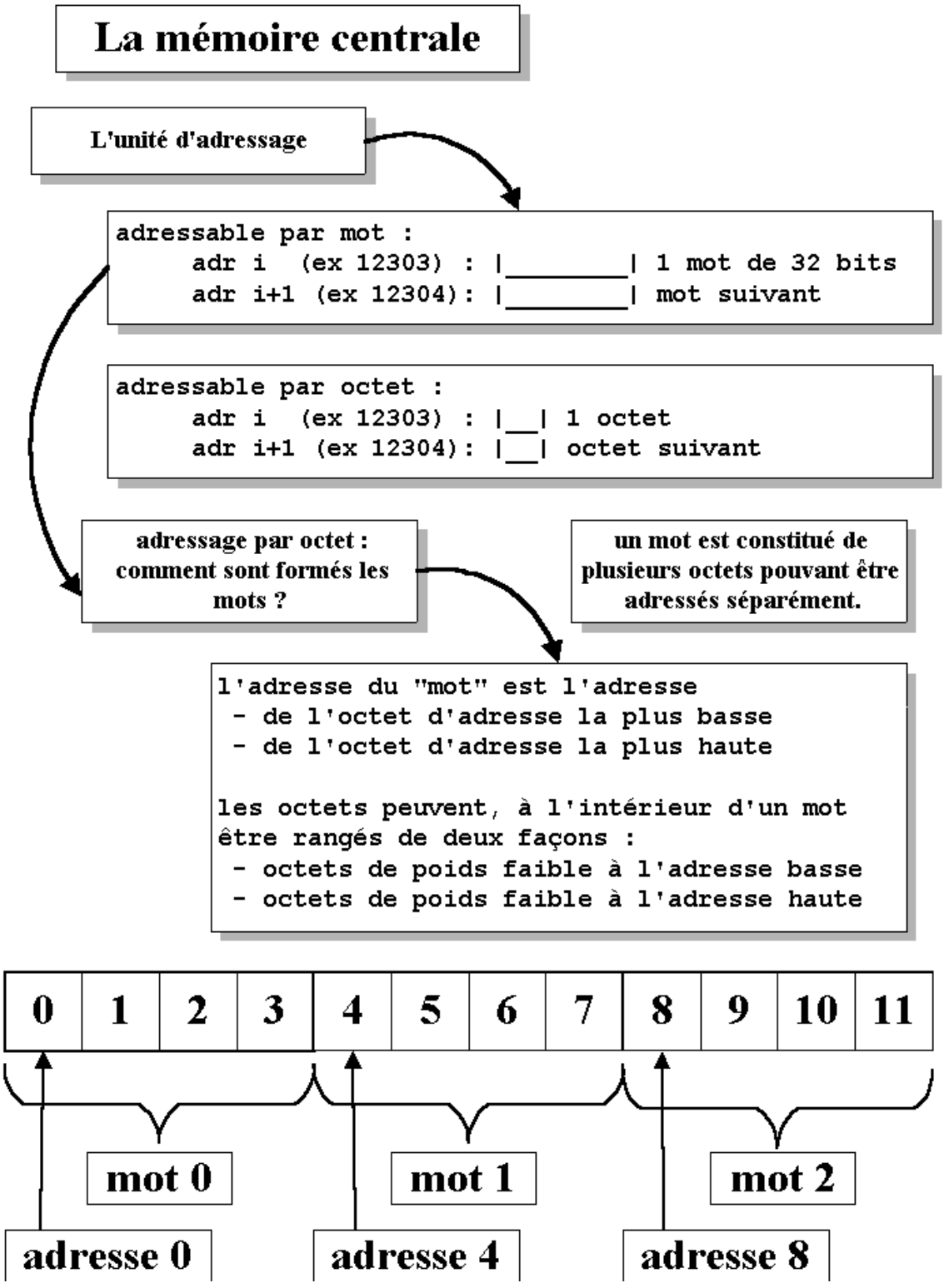


FIG. 27: Modèles mémoires 1/7

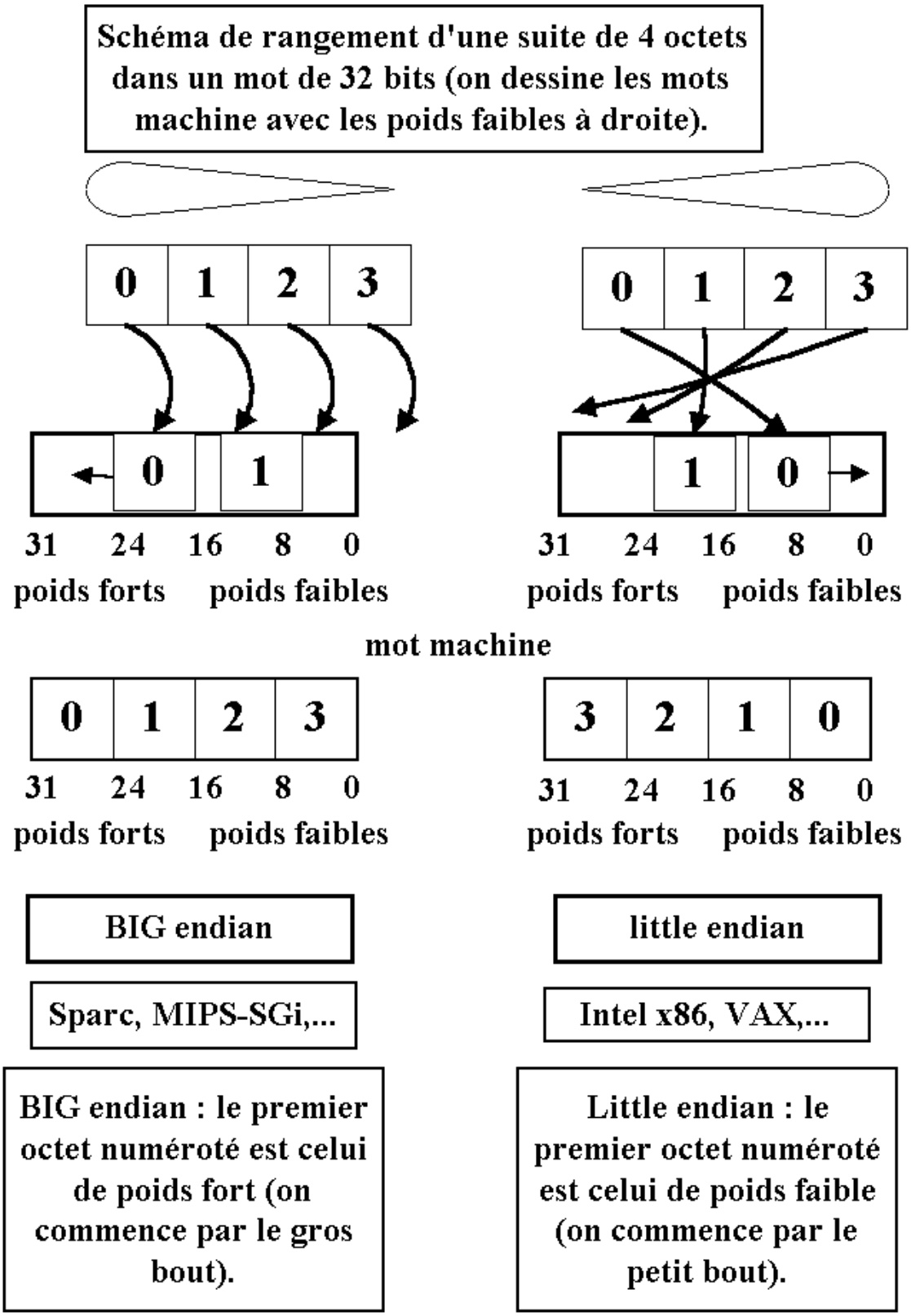


FIG. 28: Modèles mémoires 2/7

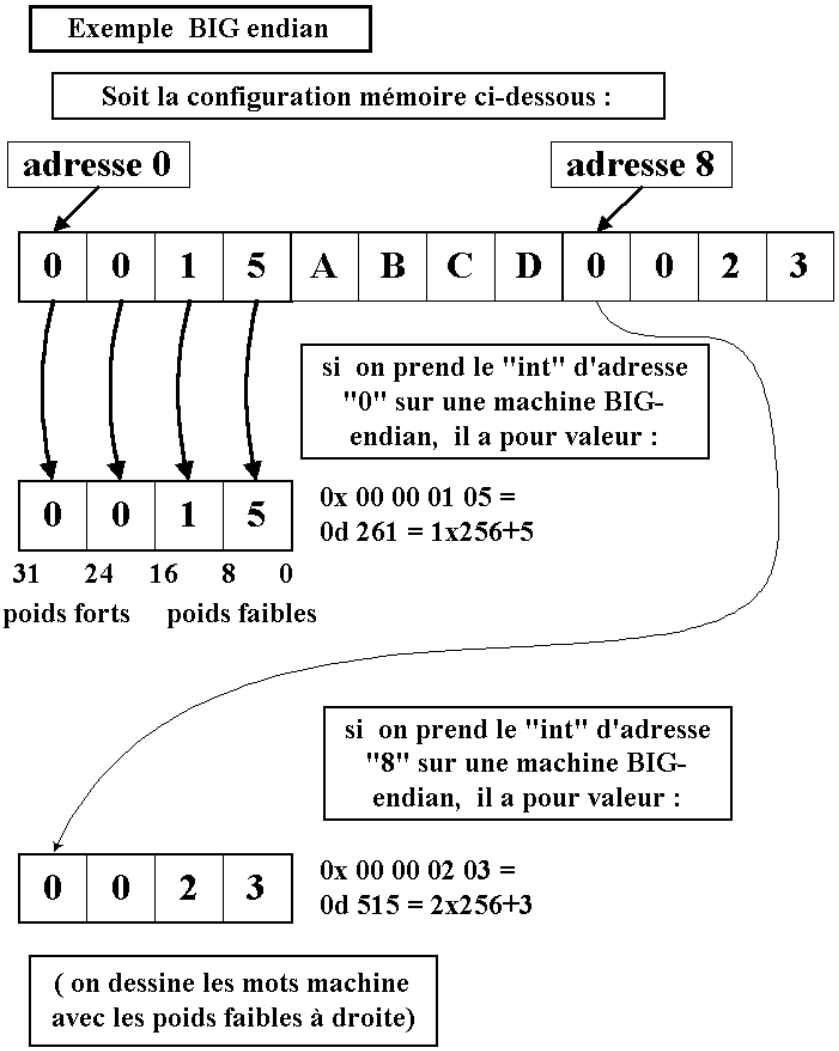


FIG. 29: Modèles mémoires 3/7

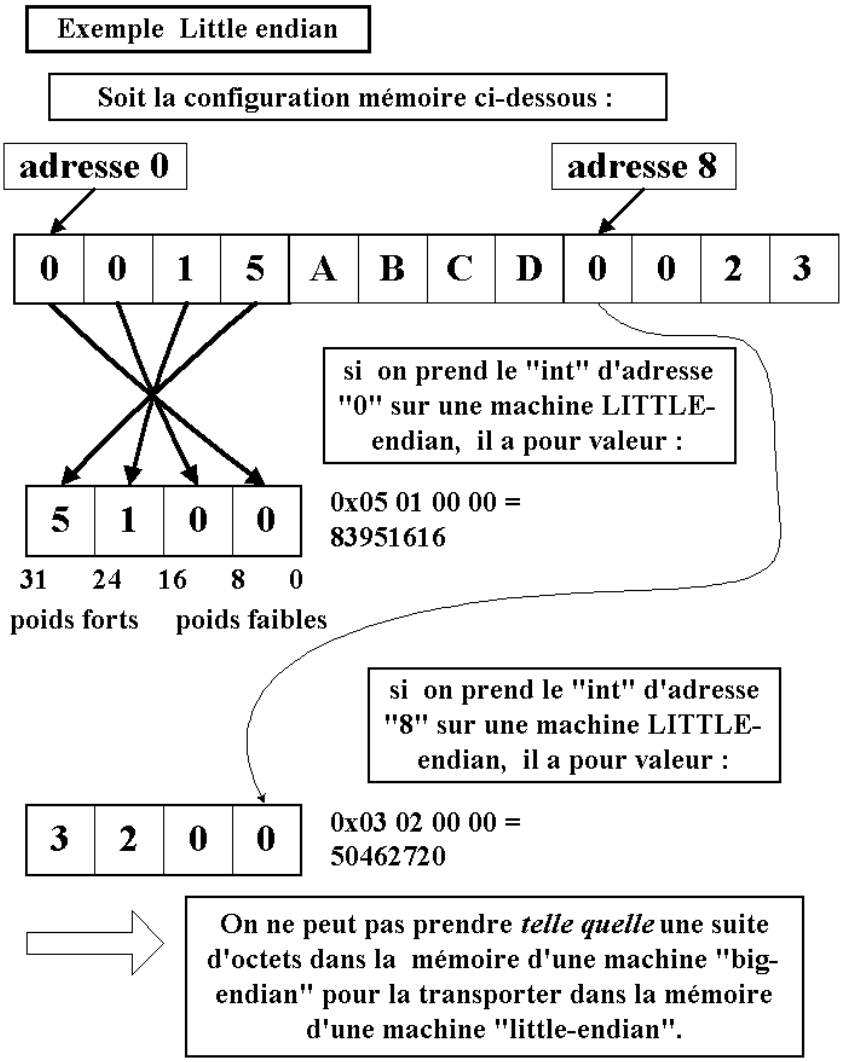


FIG. 30: Modèles mémoires 4/7

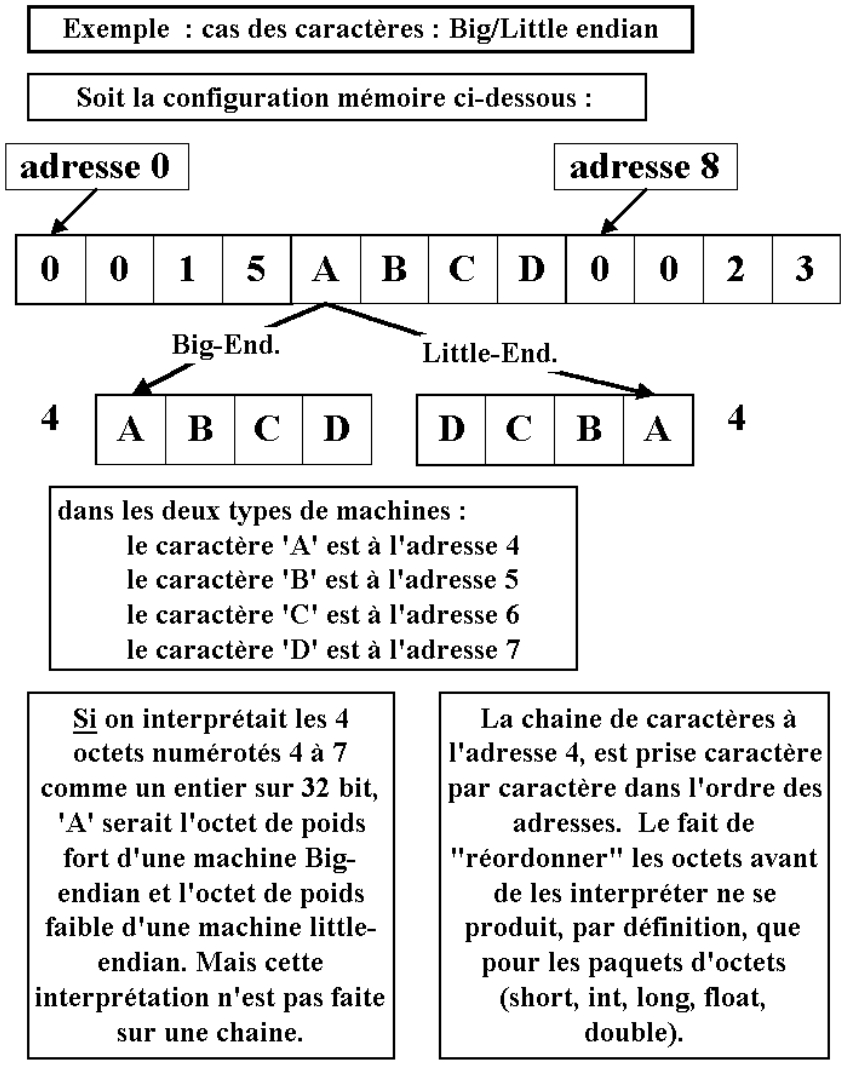


FIG. 31: Modèles mémoires 5/7

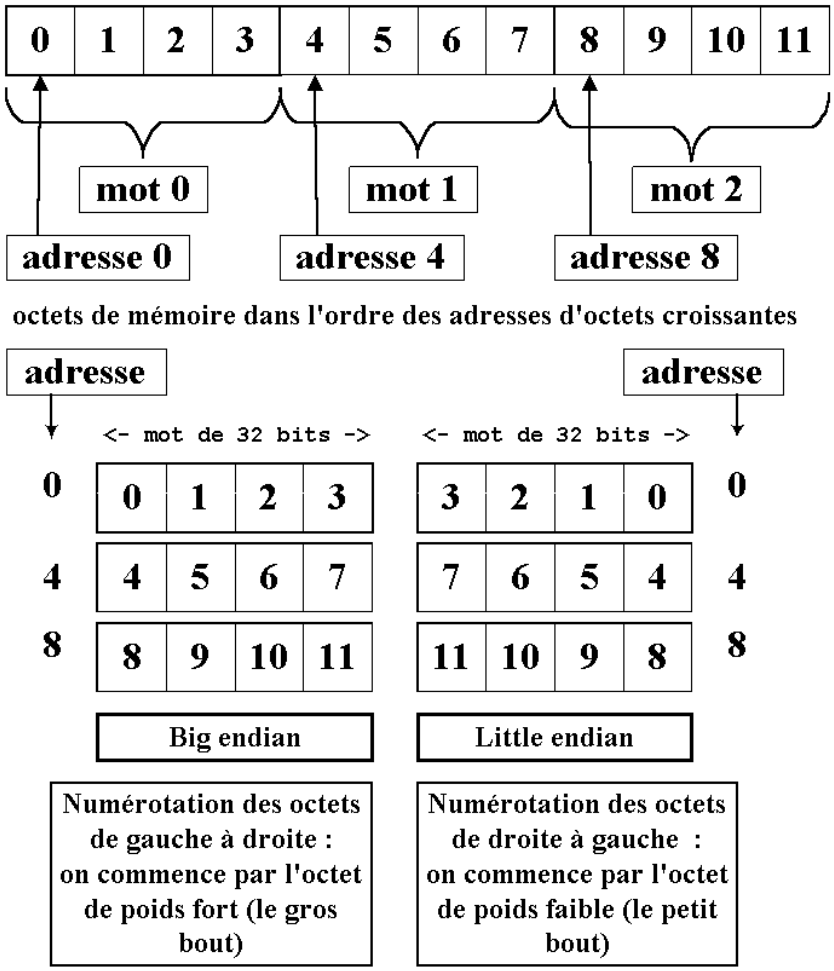


FIG. 32: Modèles mémoires 6/7

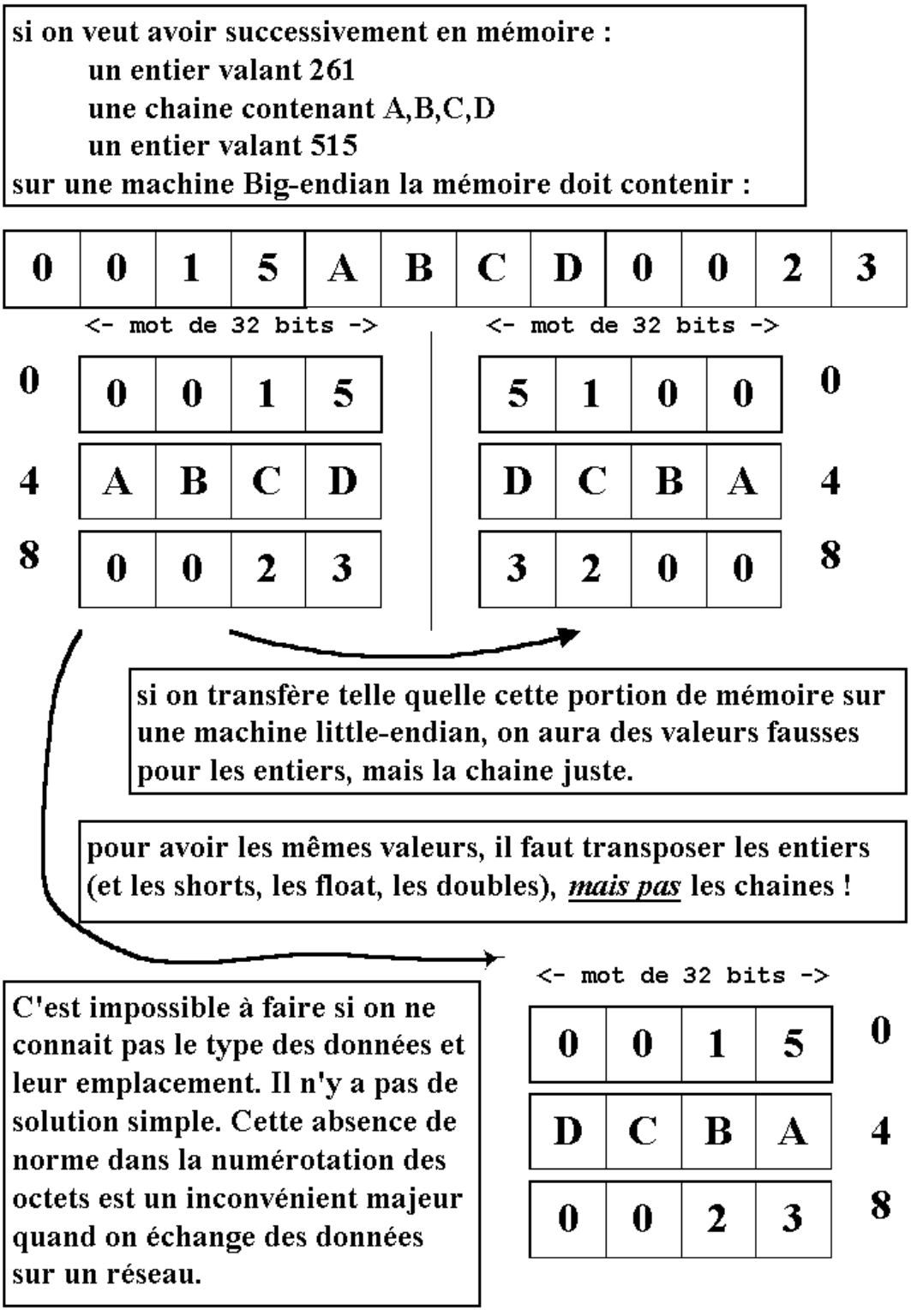


FIG. 33: Modèles mémoires 7/7

wbl.c

```
 1  /* wbl.c */
 2  /* gcc -o wbl wbl.c */
 3  #include <stdio.h>
 4  /* #include <sys/file.h> define O_CREAT et O_RDWR */
 5  #include <string.h> /* pour memcpy */
 6                          /* pour solaris */
 7  #include <sys/fcntl.h> /* define O_CREAT et O_RDWR */
 8
 9  typedef struct {
10          short i;
11          short j;
12          int k;
13          char ch[3];/* */
14          int f;
15  } biglit;
16  extern int errno;
17
18  main()
19  {  biglit test;
20     char *name = "wbl.dat";
21     int fd,i;
22
23     test.i = 10;
24     test.j = 10*256+1;
25     test.k = 10*256*256+11*256+12;
26     test.ch[0]= 'A'; test.ch[1]= 'B'; test.ch[2]= 'C';
27     test.f = 11*256*256+12*256+13;
28
29     if ((fd = open(name,O_CREAT|O_RDWR,0600)) <=0)
30     { printf("system err.in open, errno= %d\n",errno);
31       exit(0); }
32
33     i = write (fd,&test,sizeof(test));
34     printf("écrit %d\n",i);
35     if (i==-1)  {
36          printf("system err.in write test, errno= %d\n",errno);
37          perror("erreur="); }
38     close (fd);
39  }
```

```
linux : Intel x86 32 bits, little endian
sunserv : Sparc 32 bits, big endian

[vayssade@linux endian]$ od -b wbl.dat
0000000 012 000 001 012 014 013 012 000 \
101 102 103 277 015 014 013 000

lo33 sunserv:~/mv/endian> od -b wbl.dat
0000000 000 012 012 001 000 012 013 014 \
101 102 103 004 000 013 014 015

1ère moitié  test.k = 10*256*256+11*256+12

linux   012 000 001 012 014 013 012 000

sunserv 000 012 012 001 000 012 013 014

2ème moitié

linux   101 102 103 277 015 014 013 000

sunserv 101 102 103 004 000 013 014 015


[vayssade@linux endian]$ od -t x1 wbl.dat
0000000 0a 00 01 0a 0c 0b 0a 00 41 42 43 bf 0d 0c 0b 00

33 lo33 sunserv:~/mv/endian> od -t x1 wbl.dat
0000000 00 0a 0a 01 00 0a 0b 0c 41 42 43 04 00 0b 0c 0d
                    test.k                  test.f

[vayssade@linux endian]$ od -t x2 wbl.dat
0000000 000a 0a01 0b0c 000a 4241 bf43 0c0d 000b

34 lo33 sunserv:~/mv/endian> od -t x2 wbl.dat
0000000 000a 0a01 000a 0b0c 4142 4304 000b 0c0d

[vayssade@linux endian]$ od -t x4 wbl.dat
0000000 0a01000a 000a0b0c bf434241 000b0c0d

35 lo33 sunserv:~/mv/endian> od -t x4 wbl.dat
0000000 000a0a01 000a0b0c 41424304 000b0c0d
```

FIG. 34: Les valeurs binaires stockées

SR03 2004 - Cours Architectures Internet - Introduction
Fin du chapitre.

SR03 2004 - Cours Architectures Internet - Notion de protocole de communication
©Michel.Vayssade@utc.fr – Université de Technologie de Compiègne.

4 SR03 2004 - Cours Architectures Internet - Notion de protocole de communication

4.1 Primitives d'échange de données sur une liaison

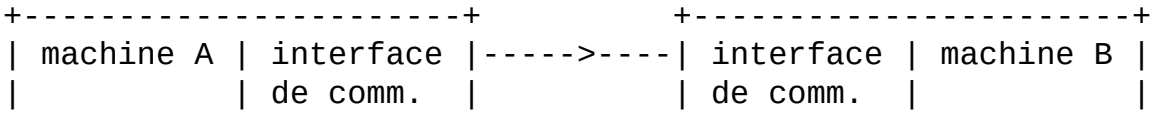
Sur les systèmes u*x, on a vu que les échanges de messages entre processus à travers un "socket" pouvaient être soit fiables (socket "stream"), soit non fiables (socket "datagram").

De plus, on a vu que:
- dans le cas non fiable (datagram), la communication était du type non connextée,
- dans le cas fiable (stream) la communication était du type connectée (connection oriented), c'est-à-dire qu'à l'ouverture du socket, le système établit un "circuit virtuel" par lequel les informations échangées transistent comme dans un "pipe" unix.

Ces deux types de communication reflètent l'état des liens qui sont utilisés pour véhiculer cette communication.

Le but des logiciels de contrôle des lignes et des liens physiques qui relient entre elles les machines formant un réseau, est justement de "cacher" aux programmes utilisateurs le fait que, dans le cas général, les transmissions sur un réseau sont sujettes à des erreurs, des pertes de messages, des altérations, des duplications de messages, etc ...

Le logiciel de réseau doit tenir compte de ce manque de fiabilité de la transmission et utiliser des techniques de dialogue entre les machines qui lui permettent de transférer tout de même des informations de façon fiable. Ces techniques de dialogues sont appelées des "protocoles". Elles sont fondées sur la détection d'erreurs de transmission, et la ré-émission des données altérées, après requête par le destinataire. La détection des erreurs est, elle, basée sur des calculs de parité plus ou moins complets.



+----------------------+ +----------------------+

Une machine A veut envoyer des données à une machine B.

On pourrait penser que l'on va faire comme dans le cas de deux process communiquant à travers un "pipe".

Mais, il y a ici plusieurs différences majeures avec le cas du pipe:

- les machines A et B n'ont pas de mémoire commune, A ne peut donc pas savoir si le "pipe" est plein ou non pour arrêter ou continuer à envoyer des données.
  . si A continue à envoyer des données alors que B n'est pas prête à les traiter, une partie des données seront perdues,
  . B doit donc pouvoir avertir A pour lui dire "attends" ou "continue",

- le temps de transit des informations de A vers B ne peut plus être considéré comme négligeable (de même de B vers A). Ainsi, par exemple, quand B veut dire à A "attends je n'ai plus de place", le temps que ce message, parvienne à A, ce dernier à encore envoyé d'autres données.

- les messages dans les deux sens peuvent être altérés par la transmis- sion. Ainsi, si B envoi un message "attends" qui est altéré, A ne va pas le comprendre et continuer à transmettre des données vers B. Inversement si B envoie un message "continue" qui est altéré, A ne va pas reprendre ses émissions.

Primitives de base

Nous allons tout d'abord décrire un certain nombre de primitives élémentaires, fournies par le système d'exploitation (avec une aide éventuelle du matériel).

Tout d'abord, les messages échangés par les applications peuvent être de longueur quelconque.

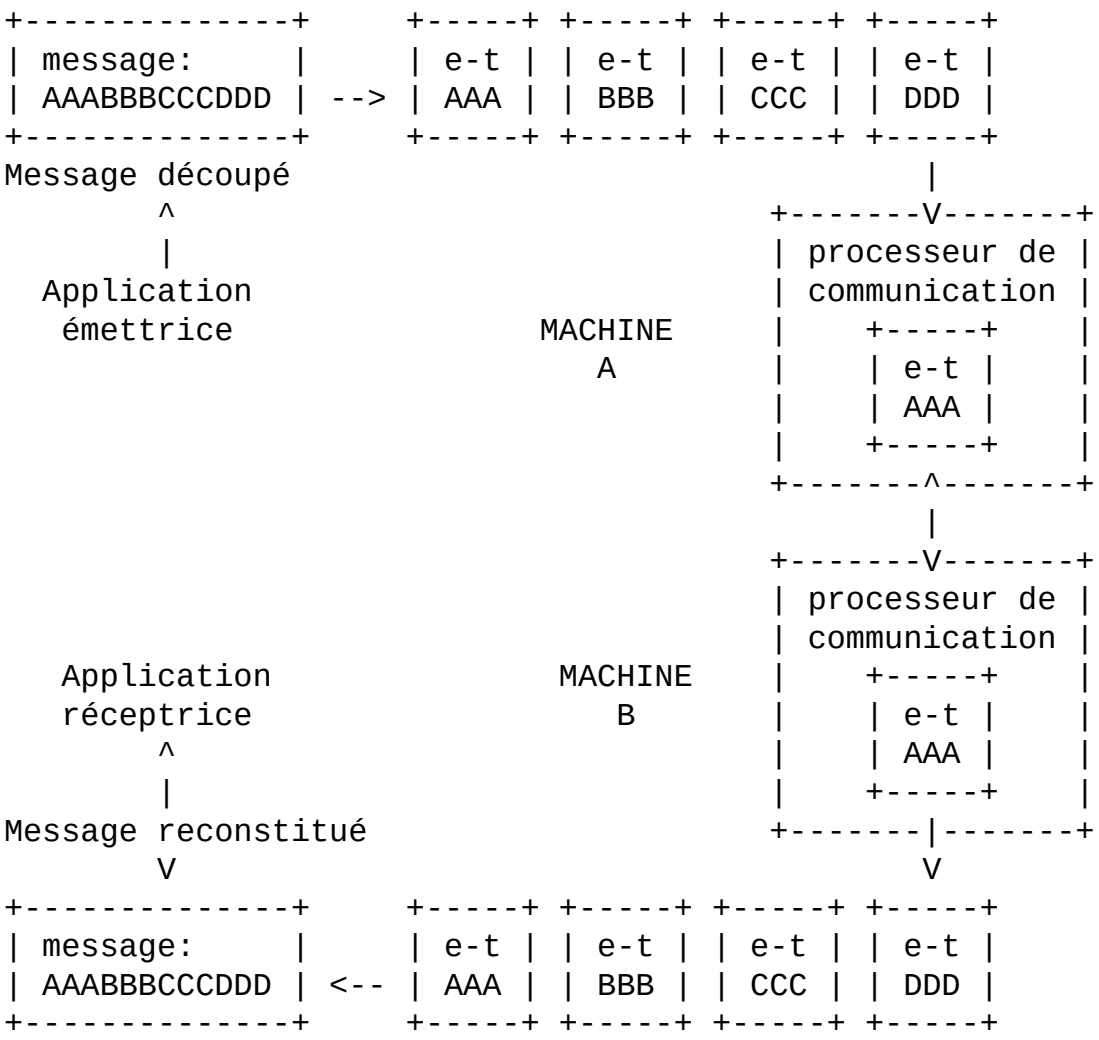
Ce n'est pas le cas des échanges qui sont effectués entre les matériels des processeurs de communication qui contrôlent la ligne reliant les deux machines.

Ceux-ci, et le système, vont donc découper les données à transmettre en blocs plus petits, et insérer ces blocs dans des

"paquets" appelés (dans le langage des réseaux) des UNITÉS DE DONNÉES.

Ces unités de données sont constituées comme un enregistrement:
Unité_de_donnée: en_tête
données
L'en-tête de chaque unité permet de reconstituer le message originel sur la machine destinataire.

Les processeurs de communication vont exécuter un protocole pour échanger les unités de données.



Pour écrire l'algorithme du protocole on va utiliser les primitives suivantes:

envoi (u_d)	envoi d'une unité de données vers l'autre processeur de communication
recoit (u_d)	recevoir une unité de données de l'autre processeur de communication
lire (mess)	lire un message depuis l'application

```
                        cliente de la communication
        écrit (mess)    écrire un message à destination de
                        l'application cliente
```

Nous pouvons maintenant écrire le protocole de communication le plus simple, en supposant une ligne de communication FIABLE (toutes les unités de données envoyées parviennent au destinataire sans erreur, aucune n'est perdue et aucune n'est dupliquée).

Nous allons supposer, de plus, que les processeurs de communication ne mémorisent qu'une seule unité de donnée et que c'est, à ce niveau, l'application qui découpe ses envois en messages assez petits pour tenir dans une seule unité de données.

```
protocole machine_A     /* procédure d'émission */
{       struct unité_de_données { int en-tête;
                                  char *info; }
        unité_de_données u_d , ack ;
        char *mess;
while ()                /* boucle infinie */
{
        lire (mess);    /* lire nouveau mess à transmettre */
        u_d.info= mess; /* le copier dans unité de données */
        envoi (u_d)     /* envoyer cette unité */
        recoit (ack)    /* attendre accusé de réception */
}}


protocole machine_B     /* procédure de réception */
{       struct unité_de_données { int en-tête;
                                  char *info; }
        unité_de_données u_d , ack ;
        char *mess;
while ()                /* boucle infinie */
{
        recoit (u_d);   /* lire une unité de données */
        ecrit (u_d.info); /* transmettre message au destin.*/
        envoi (ack);    /* envoi accusé de réception */
}}
```

On a donc un échange en alternance d'unités de données "utiles" et d'unités de données "de service" destinées à réguler la communication;

```
Machine A - émetteur    ------>    Machine B - récepteur
--------------------                ---------------------
```

```
        |                                       |
lire (mess);                                    |
u_d.info = mess;                    recoit (u_d) ! bloqué
        |                                       .    ! en attente
envoi (u_d);                                    .
           ----------+----+->----------->  .
recoit (ack)         |u_d |        recoit (u_d);
        .            |data|        ecrit (u_d.info);
bloqué  .            +----+                    |
attend  .                                envoi (ack);
        .  <-------<-+----+--------
recoit (ack)         |u_d |                    |
        |            |ack |                    |
        |            +----+                    |
        |                                       |
lire (mess);                                    |
u_d.info = mess;                    recoit (u_d) ! bloqué
        |                                       .    ! en attente
envoi (u_d);                                    .
           ----------+----+->----------->  .
recoit (ack)         |u_d |        recoit (u_d);
        .            |data|        ecrit (u_d.info);
bloqué  .            +----+                    |
attend  .                                envoi (ack);
        .  <-------<-+----+--------
recoit (ack)         |u_d |                    |
        |            |ack |                    |
        |            +----+
```

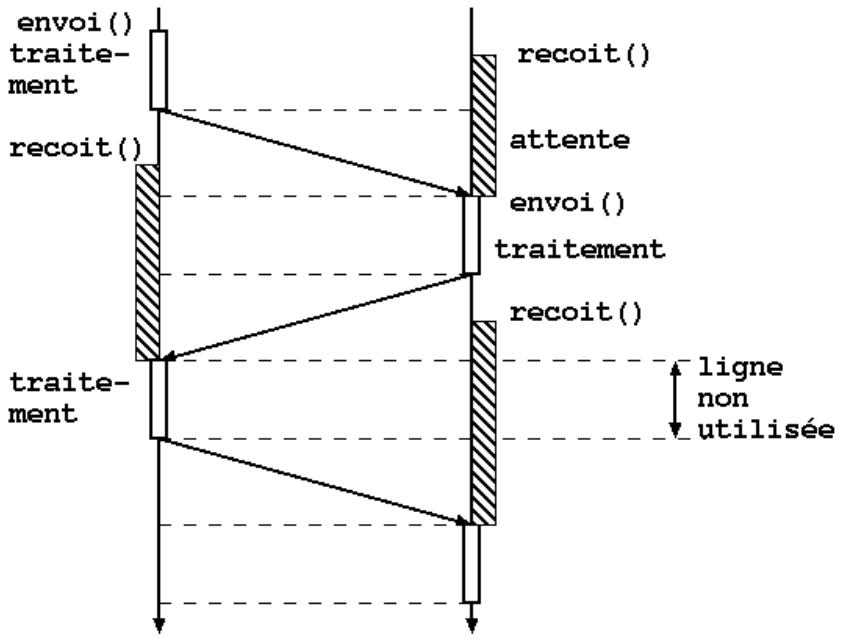


FIG. 35: Échanges simples sur une liaison

4.2 Transformation d'une ligne non fiable en un circuit fiable

Nous allons maintenant compliquer la situation en cessant de faire l'hypothèse que la ligne de transmission est fiable. Certaines unités de données peuvent donc parvenir modifiées au destinataire.

On va supposer que l'on dispose d'une primitive permettant de tester la validité d'une unité de données:

```
valid (u_d)     = 1 si u_d correcte, = 0 sinon
```

Ce test de validité se fait grâce à un calcul de bits de parité ajoutés par l'émetteur à la suite du message. Dans la pratique, on utilise des codes de Hamming permettant de détecter les erreurs avec une bonne fiabilité.

Le protocole que l'on va utiliser doit traiter un cas supplémentaire: l'arrivée d'une u_d que l'on détecte comme incorrecte. Le processeur de B doit alors prévenir le processeur de A de cette erreur afin que l'u_d erronée soit réémise.

Première tentative:

```
------------------
Sur B, récepteur:
     recoit (u_d);          /* lire une unité de données */
     si valid (u_d) = 1     /* u_d correcte */
         ecrit (u_d.info); /* transmettre message */
         ret = ack;        /* renvoi ok */
     else
         ret = err;        /* renvoi err: réémettre */
     envoi (ret);
```

Si A reçoit un accusé de réception "ok" il émet le message suivant, sinon il réémet la même unité de données que précedemment.

! MAIS ! Si la ligne n'est pas fiable, et peut altérer des u_d de l'émetteur, cette même ligne peut AUSSI altérer les u_d envoyées par le récepteur comme accusé de réception !

Ainsi si A reçoit un accusé de réception que la vérification déclare erroné, il ne sait plus si cet accusé de réception lui demandait d'émettre l'u_d suivante ou de réémettre la dernière !

Si A décide d'émettre la suivante et que l'ack avait la

```
  valeur "réémettre", une u_d est perdue !
Si A décide de réémettre la dernière u_d envoyée et que
  la valeur de l'ack était "ok", B va recevoir DEUX FOIS
  cette u_d ! Il y aura donc duplication d'un message.

Il faut donc à B un moyen de faire la distinction entre les u_d
reçues de A.



Deuxième tentative:
-------------------

Ce moyen de distinguer les u_d va être un numéro d'ordre inséré
dans chaque u_d:

Unité_de_donnée:  en_tête
                  numéro
                  données

Le processeur de communication du site A va numéroter les
unités de données envoyées vers B. Il augmente le numéro de 1,
pour chaque NOUVELLE u_d. Bien sûr, le numéro reste inchangé
lorsque A réémet une u_d, suite à l'arrivée d'un ack indiquant
une erreur de transmission, OÙ à la réception d'un ack LUI-MÊME
incorrect.


+--------------+     +-----+ +-----+ +-----+ +-----+
| message:     |     | e-t | | e-t | | e-t | | e-t |
|              |     |  1  | |  2  | |  3  | |  4  | ...
| AAABBBCCCDDD | --> | AAA | | BBB | | CCC | | DDD |
+--------------+     +-----+ +-----+ +-----+ +-----+
Message découpé


Nous aurons alors le protocole suivant:

protocole machine_A     /* procédure d'émission */
                        /* avec numérotation des u_d */
{  struct unité_de_données { int en-tête;
                             int numéro;
                             char *info; }
   unité_de_données u_d , ack ;
   int numero_ud;  /* numéro de l'u_d envoyée */
   char *mess;
   numero_ud = 1 ; /* initialiser numéro d'u_d */
   lire (mess);    /* lire premier mess à transmettre */
```

```
while ()           /* boucle infinie */
{
   u_d.info = mess;        /* copier mess dans u_d */
   u_d.numero = numero_ud; /* affecter un numéro à l'u_d */
   envoi (u_d)             /* envoyer cette unité */

   recoit (ack)            /* attendre accusé de réception */
   if( valid (ack) == 1 )  /* réception correcte de ack */
   { if(ack.info == "u_d bien reçue") /* u_d bien arrivée */
       { numero_ud = numero_ud + 1 ;  /* u_d suivante */
         lire (mess);                 /* lire mess suivant */
       }   /* si u_d mal arrivée : réémettre */
   }       /* si ack incorrecte : réémettre */
} /* fin while */
} /* fin */


protocole machine_B     /* procédure réception */
                        /* avec numérotation des u_d */
{  struct unité_de_données { int en-tête;
                             int numéro;
                             char *info; }
   unité_de_données u_d , ack ;
   int numero_ud;  /* numéro de l'u_d attendue */
   char *mess;
   numero_ud = 1 ; /* initialiser numéro d'u_d attendue */

while ()                /* boucle infinie */
{
   recoit (u_d)            /* attendre u_d */
   if( valid (u_d) == 1 )  /* réception correcte de u_d ? */
   { if(numero_ud == u_d.numero)  /* reçu la bonne ? */
      {  ecrit (u_d.info); /* transmettre mess contenu */
         numero_ud = numero_ud + 1 /* attendre suivant */
         ack.info = "u_d bien reçue"
      }
      else                       /* pas reçu la bonne */
           ack.info= "u_d bien reçue" /* mais ack ok pour */
      endif           /* éviter qu'elle soit réémise.  */
   }                   /* i.e. on décide de "l'oublier" */
   else
     ack.info = "u_d mal reçue"
   endif

   envoi (ack);        /* envoi accusé de réception */
} /* fin while */
} /* fin */
```

MAIS ! une ligne de communication peut non seulement altérer des u_d, et modifier leur ordre, MAIS AUSSI PERDRE des unités de données.

Or, dans le deuxième essai, ci-dessus, l'émetteur A attend une réponse de B (un ack), pour poursuivre ses émissions. Si l'ack de B est perdu par la ligne, les deux protocoles se bloquent indéfiniment.

De même, si une u_d de A se perd, B attend indéfiniment.

Ils ne sont donc pas assez robustes pour fonctionner dans cette situation nouvelle, avec des pertes possibles d'u_d.

Il faut donc trouver un moyen de limiter cette attente éventuelle de A ou B.

Troisième tentative:

Ce moyen va être un compteur de temps provoquant une interruption de l'attente à son expiration (time-out).

On va utiliser une primitive "reçoit" modifiée:

```
   is= recoit (u_d,délai)  recevoir une unité de données de
                           l'autre processeur de communication
                           avec un délai de garde
   Si réception avant fin de "délai", sortir de "recoit" avec
      is=ok et u_d contient unité reçue
   Si rien reçu pendant "délai", sortir de "recoit" avec
      is="hors délai".
```

Nous aurons alors le protocole suivant:
(les lignes ajoutées par rapport au deuxième cas sont marquées par "++>").

```
protocole machine_A     /* procédure d'émission */
                        /* avec numérotation des u_d */
{   struct unité_de_données { int en-tête;
                              int numéro;
                              char *info; }
    unité_de_données u_d , ack ;
    int numero_ud;  /* numéro de l'u_d envoyée */
++> int délai = VALEUR; /* valeur du délai */
    char *mess;
    numero_ud = 1 ; /* initialiser numéro d'u_d */
```

```
    lire (mess);    /* lire premier mess à transmettre */

while ()                /* boucle infinie */
{
    u_d.info = mess;        /* copier mess dans u_d */
    u_d.numero = numero_ud; /* affecter un numéro à l'u_d */
    envoi (u_d)             /* envoyer cette unité */

--> is = recoit (ack,délai) /* attendre accusé de réception */
++> if (is=="hors délai")   /* u_d OU ack perdu: réémettre */
++> {}      /* rien: retourner à while */
++> else    /* sinon reçu ack: cas précédent */

    if( valid (ack) == 1 )  /* réception correcte de ack */
    { if(ack.info == "u_d bien reçue") /* u_d bien arrivée */
        { numero_ud = numero_ud + 1;   /* u_d suivante */
          lire (mess);                 /* lire mess suivant */
        }       /* si u_d mal arrivée : réémettre */
    }           /* si ack incorrecte : réémettre */

++> endif   /* fin if hors délai */

} /* fin while */
} /* fin */


protocole machine_B     /* procédure réception */
                        /* avec numérotation des u_d */
.....

while ()                /* boucle infinie */
{
-->  is = recoit (u_d,délai) /* attendre u_d */
++>  if (is=="hors délai") /* u_d OU ack perdu: réémettre ack */
++>  {}      /* rien: retourner à envoi ack puis while */
++>  else    /* sinon reçu ack: cas précédent */

     if( valid (u_d) == 1 )  /* réception correcte de u_d ? */
     { if(numero_ud == u_d.numero)  /* reçu la bonne ? */
        {  ecrit (u_d.info); /* transmettre mess contenu */
           numero_ud = numero_ud + 1 /* attendre suivant */
           ack.info = "u_d bien reçue"
        }
        else                        /* pas reçu la bonne */
           ack.info = "u_d bien reçue" /* mais ack =ok pour */
        endif               /* éviter qu'elle soit réémise. */
     }                      /* i.e. on décide de "l'oublier" */
     else
       ack.info = "u_d mal reçue"
```

```
     endif
++>  endif   /* fin if hors délai */

     envoi (ack);    /* envoi accusé de réception */
} /* fin while */
} /* fin */
```

Quelle doit être la valeur du délai de garde ?

Quand A émet une u_d en direction de B, cette u_d "voyage" sur la ligne de communication à une certaine vitesse: il existe donc une borne supérieure au temps que va mettre une u_d pour aller de A à B. Le processeur B va prendre un certain temps pour traiter l'u_d puis va renvoyer vers A une u_d servant d'accusé de réception.

Vu de A on donc:
t1= temps de transfert de u_d vers B
t2= temps de traitement de u_d sur B
t3= temps de transfert de u_d vers A

Après un temps t= t1+t2+t3, A s'attend à recevoir l'accusé de réception de B.

Si les temps t1, t2 et t3 sont des bornes supérieures choisies largement, on peut régler le délai de garde de A à la valeur t1+t2+t3. Si après ce temps, A n'a pas reçu l'u_d en retour de B, il en conclut que l'une des u_d s'est perdue.

Remarque

Les deux processeurs A et B effectuent une boucle infinie. Donc le numéro affecté aux unités de données va devenir très grand et ne plus pouvoir être codé sur le champ prévu au départ. Il faudra donc le remettre à zéro lorsque le débordement se produira. On va alors réutiliser les numéros.

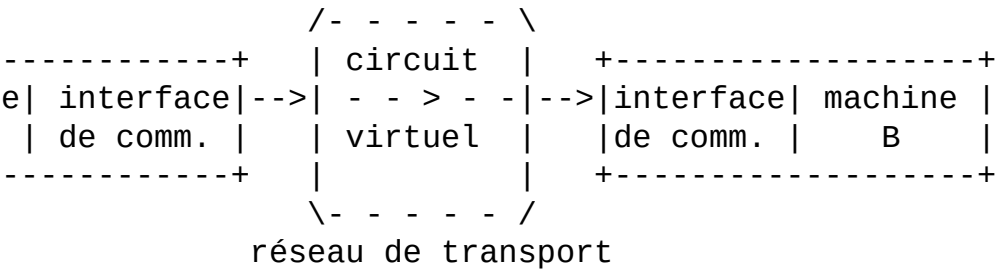
Pour qu'il n'y ait pas de confusion possible d'u_d, il faut que, au moment du passage au maximum, TOUTES LES U_D, ET TOUS LES ACKs correspondants, de numéros 1,2,3,... aient disparu du réseau.

Ainsi, sur un réseau où le délai de garde doit être réglé à 4 secondes, si on peut envoyer 5000 u_d par secondes, 4*5000 > 2**16, un champ de 16 bits pour le numéro d'u_d sera insuffisant.

Quatrième tentative:

On a pas envisagé le cas, dans les protocoles précédents, où des unités de données parviennent au destinataire dans un ordre différent de l'ordre d'émission.

Bien sûr, ceci ne peut pas se produire dans le cas d'une seule ligne de communication reliant deux machines. Mais il peut se produire si cette ligne de communication est, en fait, un circuit virtuel dans un réseau.



Le circuit virtuel, suivant le type de réseau utilisé, ne garanti pas nécéssairement le séquencement des unités de données (c'est le cas par exemple d'un datagramme dans le domaine internet). Ainsi deux u_d, peuvent emprunter des CHEMINS PHYSIQUES différents, de durée différentes, et donc parvenir à destination dans un ordre différent de l'ordre d'émission.

Supposons que le processeur de communication de B, puisse gérer un certain nombre de zones tampons où il va pouvoir STOCKER les u_d qui lui arrivent dans le désordre.

Le récepteur pourra adopter un comportement tel que:

```
recoit (u_d 3)
                écrit (u_d 3)
                acquitte (u_d 3)
recoit (u_d 5)
                stocke (u_d 5)
                acquitte (u_d 5)
recoit (u_d 6)
                stocke (u_d 6)
                acquitte (u_d 6)
recoit (u_d 4)
                écrit (u_d 4)
                acquitte (u_d 4)
                écrit (u_d 5)
                écrit (u_d 6)
```

B émet dès réception les acquittements de 5 et 6 afin d'éviter que le délai de garde associé à 5 et 6 ne se déclenche et que ces deux u_d ne soient réémisent inutilement par A.

Dans ce cas, le délai de transit d'une u_d sur le réseau est "long" par rapport à la vitesse d'émission des u_d par A. Comme B peut stocker quelques u_d, A peut en profiter pour émettre À L'AVANCE des u_d vers B, SANS ATTENDRE d'avoir reçu l'acquittement des précédentes.

On pourra avoir des séquences du type:

```
    A                       B
envoi (1)
envoi (2)
envoi (3)              recoit(1)
                       ack (1)
                              écrit (3)
                       recoit(3)
                       ack (3)
                       recoit(2)
recoit(ack 3)          ack (2)
                              écrit (2)
recoit(ack 1)                 écrit (3)
envoi (4)
recoit(ack 2)          recoit(4)
envoi (5)              ack (4)
recoit(ack 3)                 écrit (4)
...                    ...
```

Le nombre d'u_d envoyée par A "en avance" s'apelle LA FENÊTRE. La borne inférieure de la fenêtre est le numéro d'u_d attendue par B, la borne supérieure est plus élevée d'un nombre qui dépend du nombre de tampons disponibles dans B.

Comme B doit délivrer les u_d dans l'ordre, il doit garder une place disponible pour recevoir l'u_d de numéro "attendu". C'est seulement alors qu'il peut délivrer cette u_d et toutes les suivantes, en séquence, déja parvenues et stockées en attente.

De son côté, A ne doit pas "submerger" B d'u_d que celui-ci ne pourrait ni stocker (plus de tampons) ni délivrer (car l'u_d "attendue" ne serait pas arrivée). C'est pourquoi, dans l'exemple ci-dessus, A ne "redémarre" que lorsqu'il reçoit l'ack de la première u_d envoyée: A sait alors que B a libéré au moins un buffer.

Dans l'exemple ci-dessus, A attends de recevoir l'ack de 1 pour continuer ses émissions. Mais il aurait pu envoyer l'u_d suivante un peu plus tôt, en supposant qu'elle parviendrait sur B, APRÈS que celui-ci ait reçut, acquitté et délivré l'u_d 1. Ceci en supposant que l'ack de 1 est "en route". On parle alors pour A de stratégie d'émission optimiste par opposition à la stratégie pessimiste du schéma.

Si la tratégie de A est TROP optimiste et que B reçoit l'u_d suivante (no.4 dans le schéma) AVANT d'avoit pu libéré ses tampons, il peut soit:

- ignorer cette arrivée trop précoce, l'u_d 4 ne sera pas acquittée et au bout du délai de garde, A va la réémettre,

- envoyer à A un acquittement spécial "4 trop tôt", demandant à A la réémission, mais cette réémission pourra étre faite par A, avant l'expiration du délai de garde. Ceci accélère la "reprise" du cours normal des émissions.

SR03 2004 - Cours Architectures Internet - Réseau local ethernet
©Michel.Vayssade@utc.fr – Université de Technologie de Compiègne.

5 SR03 2004 - Cours Architectures Internet - Réseau local ethernet

5.1 Les réseaux locaux de type ethernet

ethernet : premiers travaux en 76 (Xerox Park, Palo Alto)
1ère norme industrielle : Dec + Xerox + Intel : sept 82
puis ISO 802.3 qui définit leformat des trames et les paramètres du réseau.

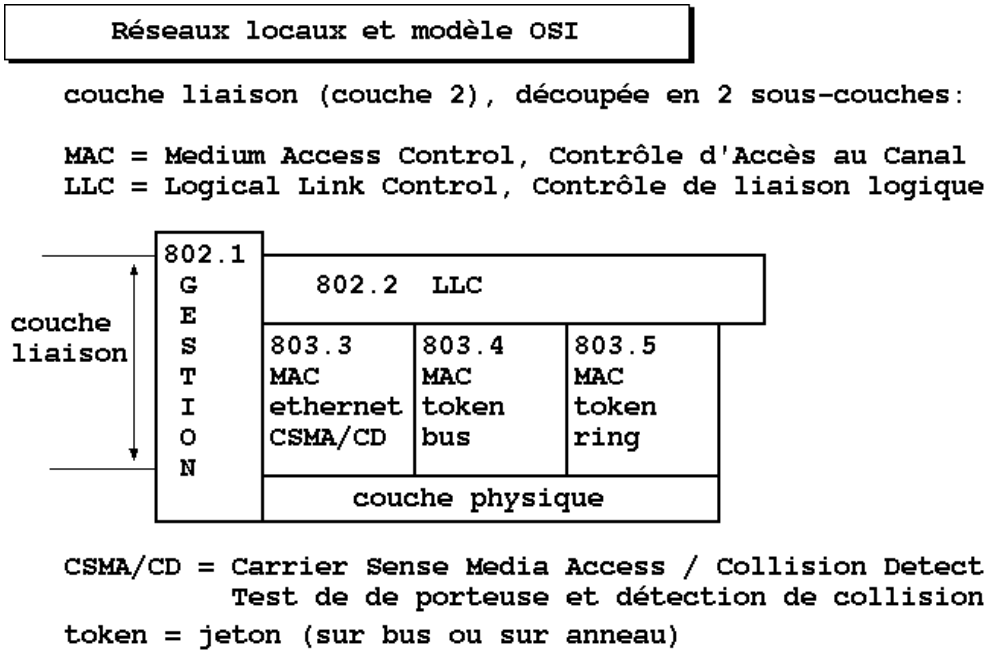


FIG. 36: Normes régissant ethernet

Adressage

L'émission d'un message est faite sur une **liaison** à laquelle **tous** les abonnés sont **raccordés**.

- le ou les **destinataires** doivent **reconnaître** les **trames** au moment ou elles passent "devant" eux ;
- chaque trame contient :
  - l'adresse du destinataire,
  - l'adresse de l'émetteur (source),
  - des informations de contrôle,
  - les data (contenu du message).

Chaque équipement branché sur un ethernet possède une adresse **unique** : l'adresse "MAC".

Le caractère unique de "l'adresse ethernet" est assuré par une norme internationale qui affecte un numéro unique à chaque frabriquant de matériel ethernet. L'adresse associe ce nuléro

de fabriquant à un numéro de série, l'ensemble (sur 48 bits, soit 6 octets) est stocké dans une mémoire non volatile sur l'équipement.

Trame ethernet

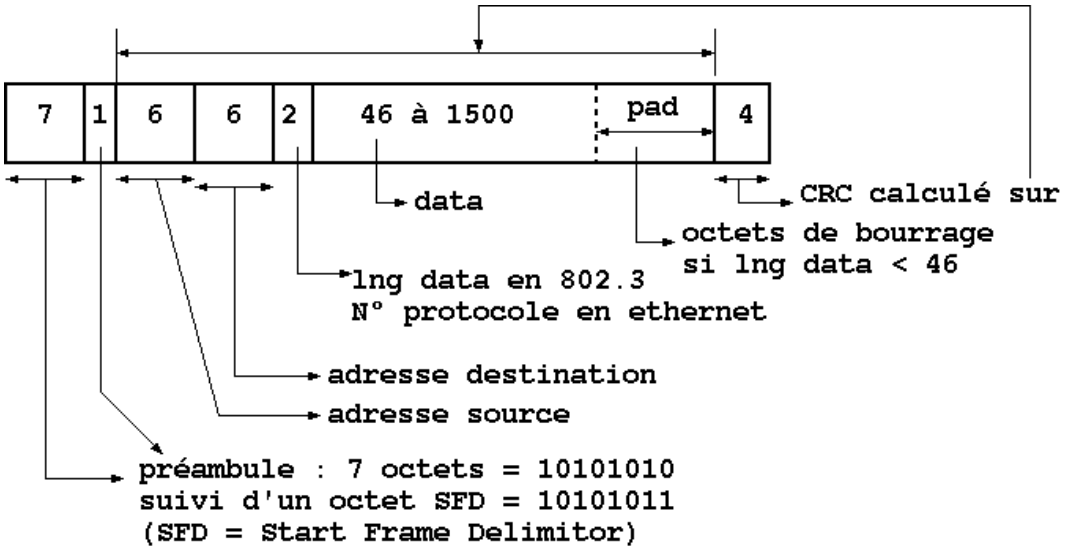


FIG. 37: La trame ethernet

ethernet et 802.3

```
différences entre trames ethernet et 802.3
        802.3           ethernet
        | dest |        [ dest |
        [ srce |        | scre |
        |ln LLC|        | data |
la trame 802.3 contient un champ longueur_data_LLC absent de la trame
ethernet. Cette dernière contient directement le paquet du protocole de
niveau 3.
donc la longueur des data utiles se trouve dans le paquet de niveau 3
il faut cette lng pour enlever les bits de bourage : avec la trame
ethernet, ne peut pas se faire au niveau de la couche 2, sans "regarder"
dans le paquet de niveau 3  => il y a violation de la notion de couche

comment distinguer une trame 802.3 d'une trame ethernet ?

         <- 6 octets-> <- 6 octets-> <2 octets>
802.3   | destination |  adr source | lng LLC |
ether   | destination |  adr source | num pro |

les 2 octets suivant l'adr scre contiennent :
        pour 802.3 : une longuer <=1500 (en octets)
        pour ether : un numéro de protocole >= 0x0800 (=2048 > 1500)

adresses MAC : début= 3 octets fabriquent fin= 3 octets numéro de série
        exemples d'adr. de fabriquants
                00:00:0C        cisco
                08:00:1D        cabletron
```

```
08:00:20        SUN
08:00:2B        DEC
08:00:51        IBM
```

Le protocole ethernet : CSMA/CD

CSMA/CD est un protocole "à compétition" : tous les équipements sont à l'écoute du média. Quand un équipement a besoin d'émettre :

– si le média est occupé, il ajourne l'émission,

– si le média est libre, il émet.

Mais, il est possible que deux équipements décident d'émettre au même instant : il y aura alors **collision**.

Les équipements sont construits de telle sorte qu'ils **écoutent le média en même temps qu'ils émettent**. Ainsi, s'ils sont seuls à émettre, le signal écouté doit être le même que le signal émis. Si les deux signaux sont différents, l'équipement en déduit qu'il y a collision.

En cas de collision, l'équipement cesse d'émettre et entame la séquence de **résolution de conflit**.

La détection de collision dépends du délai de propagation sur le cable et de la distance entre les deux équipements émetteurs. En effet, l'émission est faite par l'envoi sur le média d'un signal électromagnétique dont la vitesse de propagation est **finie**. Il faut donc un certain délai, **dt** pour que le signal émis par une station parvienne aux autres stations.

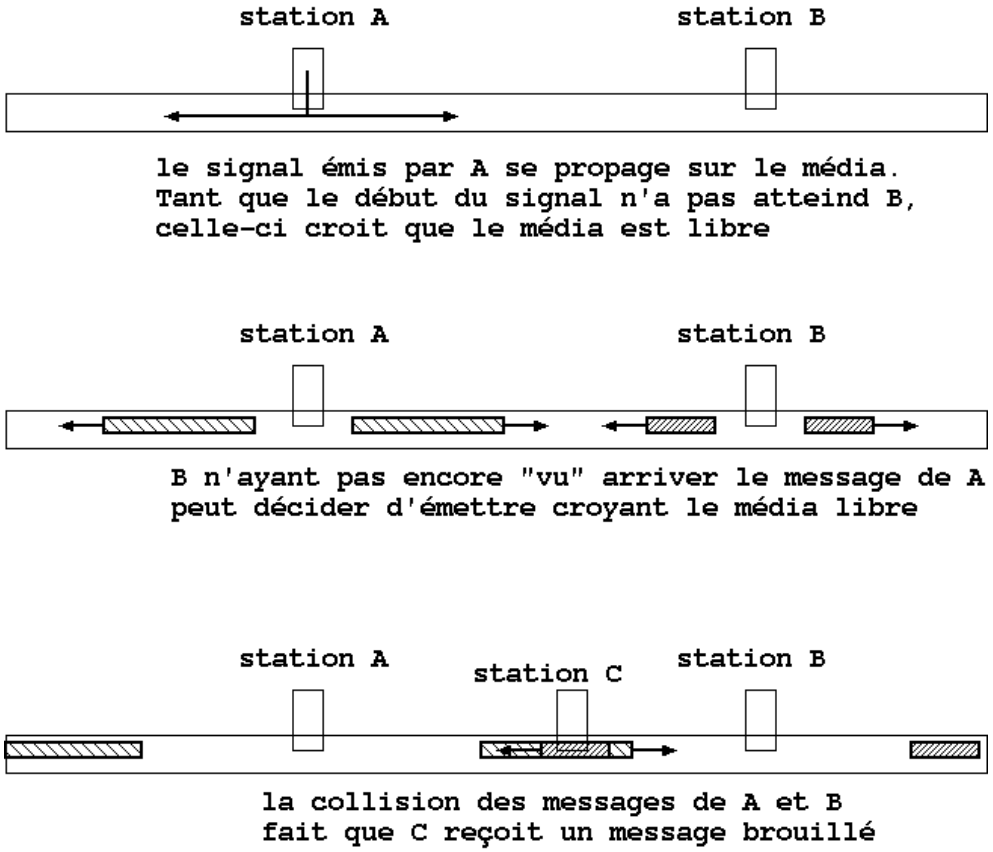


FIG. 38: Émission de trames ethernet et collision

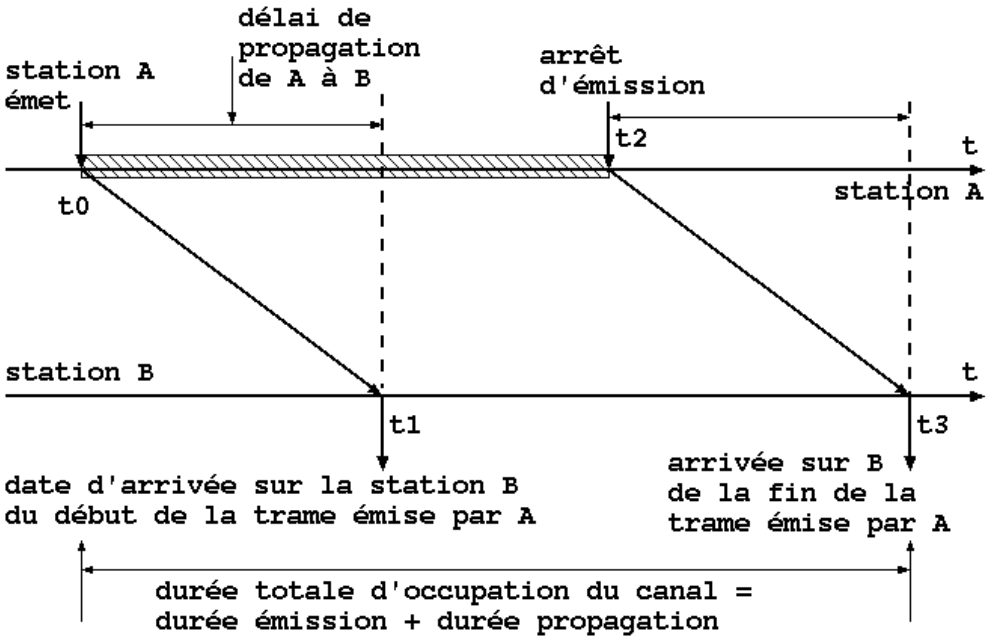


FIG. 39: Émission de trames ethernet et occupation du canal 1/2

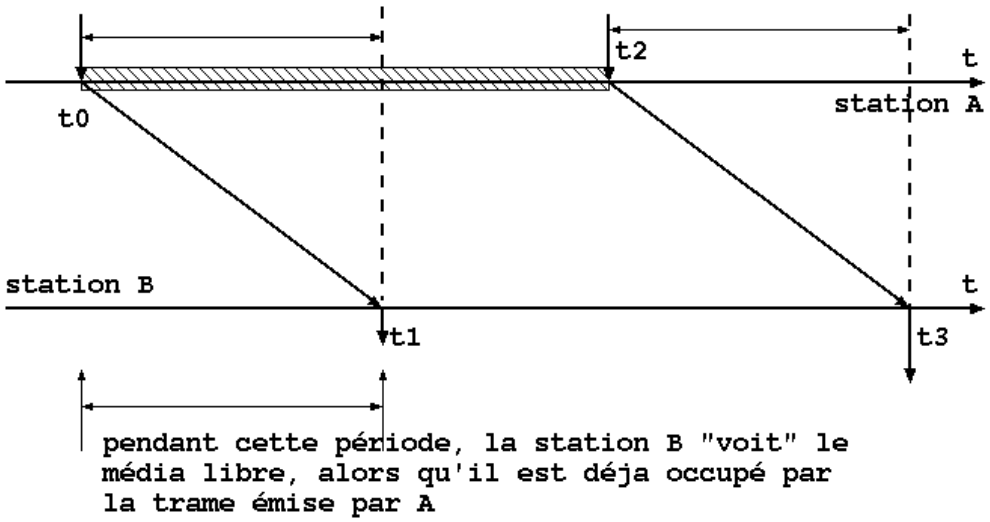


FIG. 40: Émission de trames ethernet et occupation du canal 2/2

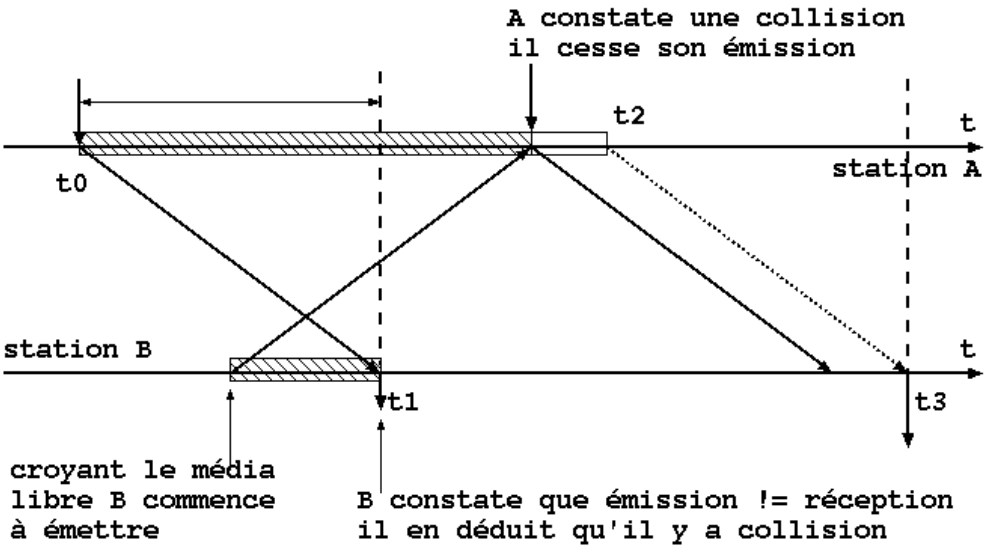


FIG. 41: Trames ethernet et condition de collision

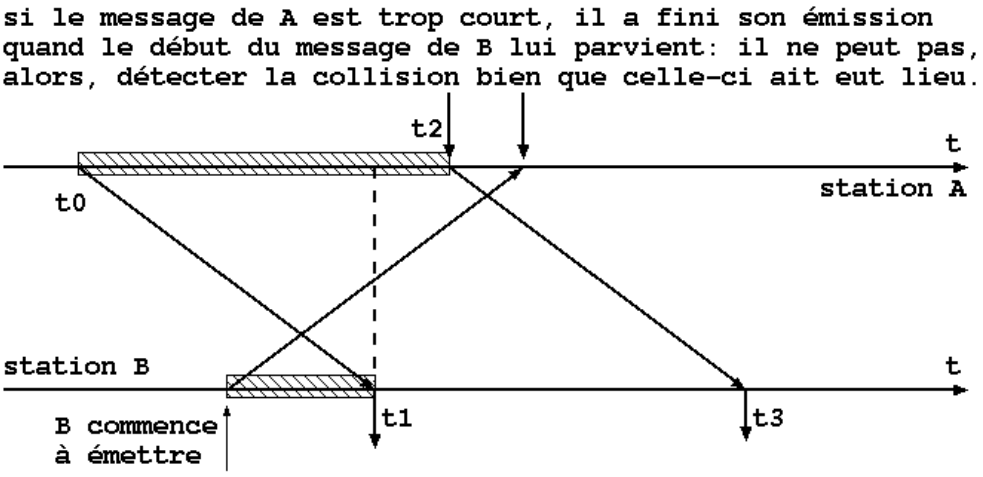


FIG. 42: Trames ethernet, collision et condition de détection 1/2

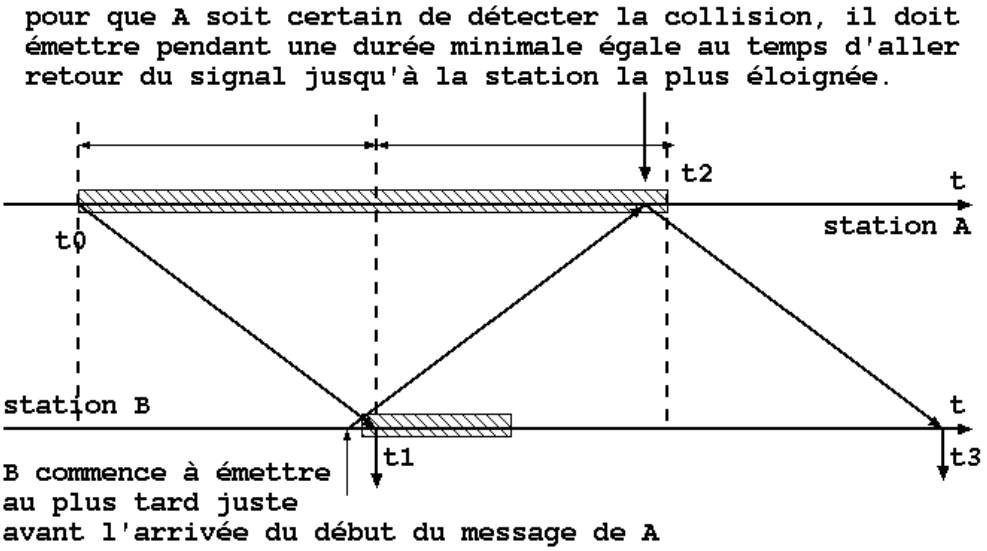


FIG. 43: Trames ethernet, collision et condition de détection 2/2

CSMA/CD : principe de fonctionnement

1. attente d'un silence minimum avant émission ("délai intertrame" égal à 9.6μs),
2. longueur minimale de trame (si data < 46 octets, utilisation de bits de bourrage (padding)), supérieure au temps du plus long aller-retour pour assurer qu'une collision soit détectée,
3. après détection d'une collision, **continuation** de l'émission de bits de brouillage (au moins 32) ; une trame ainsi tronquée (runt frame) aura au moins 96 bits (64 d'en-tête + 32 de brouillage) ;
4. à la réception, ignorer toute trame de longueur insuffisante (jabbers) ; normalement il n'y en a pas, mais toutes les implémentations ne sont pas forcément correctes, par exemple quand on connecte des équipements 10BASE-T sur des prises RJ45 100BASE-T ;
5. après une collision, exécuter l'**algorithme de ré-émission**.

Nota : la seule raison du délai inter-trames spécifié par la norme est de permettre aux stations de faire basculer ses circuits de connexion du mode émission au mode réception. Il ne faut pas oublier que la spécification date des années 1970. Certains vendeurs trichent et fabriquent des cartes avec un délai plus court entre deux trames. Ces cartes vont plus vite que la concurence, par contre elles peuvent ne pas interopérer correctement avec celles d'autres fabriquants.

algorithme de ré-émission

Après une collision, les deux stations émettrices des trames entrées en collision, vont devoir ré-émettre leur trame. Or il ne faudrait pas qu'elles fassent cette tentative après le même délai depuis la collision.

De plus ces deux stations ne peuvent pas "se concerter" pour décider qui va ré-émettre le premier, puisque pour se concerter elle devraient échanger des informations, donc émettre sur le média !

Il faut donc un algorithme **distribué**, qui exécuté indépendamment par les deux stations fournisse à chacune un délai différent avant de ré-émettre la trame entrée en collision.

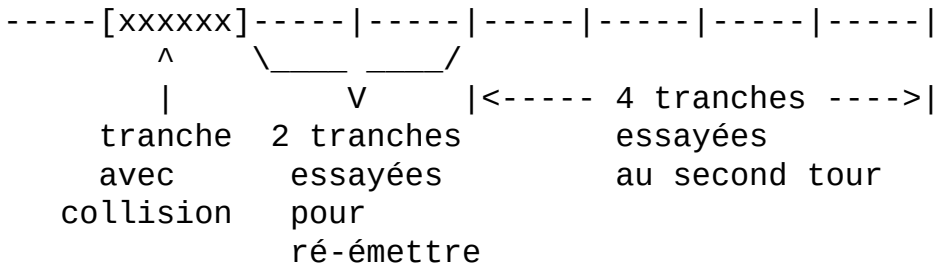
L'idée de base d'ethernet est de **ré-émettre la trame après un délai aléatoire** avec les caractéristiques suivantes pour le délai :

- l'écart entre deux valeurs possibles du délai est supérieur au temps de propagation sur le bus. Ainsi, en l'absence d'autre traffic on est certain que les deux stations ré-emettent sans collision.
- les délais sont de plus en plus longs à chaque nouvelle collision pour la même trame ;

algorithme de ré-émission : description plus formelle

L'alogithme BEB (Binary Exponential Backoff) est défini par :

- après le Nième collision (pour la même trame),
  - si N=16, abandon et signaler erreur,
  - i = min(N,10) (le délai double chaque fois, jusqu'à N=10),
  - tirer un entier aléatoire "na" dans l'intervalle [0, $2^i - 1$],
  - délai = na * ie (ie est l'intervalle élémentaire, 51,2μs).



```
après collision chaque station A et B impliquée tire {0,1} aléatoirement
pour ré-émettre dans une des 2 tranches suivantes ==> 4 cas
               A       B
               0       1       A réémet, puis B
               1       0       B réémet, puis A
               0       0       nouvelle collision
               1       1       nouvelle collision
si nouvelle collision on passe à 4 tranches => 1 chance sur 4
seulement de faire une troisième collision
```

D'où vient ce délai de 51,2μs ?

On a vu que pour être certain de détecter une collision, une trame devait avoir une longueur minimale égale au double du temps de propagation entre les 2 stations les plus éloignées (worst case).

Pour un ethernet à 10mégabits/sec, 51,2μs correspond à l'émission de 512 bits (on dit parfois 512 temps bits). Ce qui donne 64 octets (14 de préambule, 46 de data et 4 de CRC).

De plus, il ne fallait pas que ce délai soit trop long. Le choix du délai était donc aussi le choix de la distance maximum d'un réseau ethernet (environ 2500 m). Cette durée d'aller-retour maximum est appelée "tranche canal".

La relation entre la longueur du réseau et la durée d'une trame est bien sûr une conséquence de la vitesse de propagation du signal dans le câble de cuivre : environ 2/3 de la vitesse de la lumière.

- vitesse de la lumière dans le vide = 300 000 km/s,
- vitesse du signal électrique dans le câble = 200 000 km/s,
- longueur occupée par un bit dans un ethernet à 10 Mbits/s = (200,000,000 ms) / (10,000,000 bits / s) = 20 mètres,
- temps d'aller-retour sur un câble de 500 m = 2 * 500 / 200,000,000 = 5 10^{-6} s = 5 μs, soit le temps d'émettre 10 000 000 * 5 10^{-6} = 50 bits.

On a calculé 50 bits, et on a dit que la norme a choisit 512 bits ! Pourquoi ?

Si la longueur maximum d'un câble est de 500m, la norme précise aussi qu'il est possible d'étendre cette distance jusqu'à 2500m en utilisant 4 répéteurs.

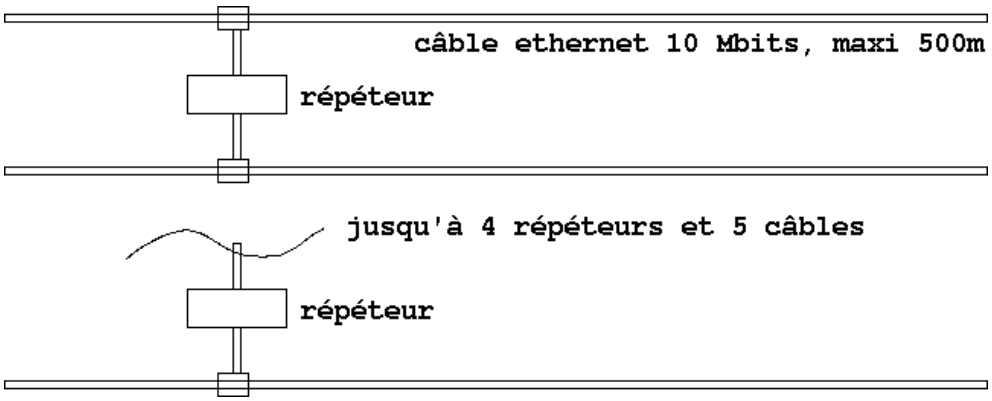


FIG. 44: ethernet : câbles et répéteurs

Il faut donc tenir compte, en plus de la propagation sur les câbles, de la durée de traversée des répéteurs.

De plus, les concepteurs d'ethernet, ont choisi des spécifications en général deux fois plus strictes que nécessaire pour pallier aux marges d'erreurs dans la construction des équipements à l'époque de la création de la norme.

Ainsi quand on construit un ethernet, on peut se permettre des "petites" violations de la norme (un câble un peu trop long ici, un répéteur de trop ailleurs, etc ...). Il fonctionnera quand même correctement.

Nombre maximum de trames sur un ethernet

La durée d'émission d'une trame étant directement liée à sa taille, il faut faire le calcul pour les plus petites trames possibles et pour les plus grosses :

- **trame mini** : 8+14+46+4 = 72 octets = 576 bits.
- **trame maxi** : 8+14+1500+4 = 1526 octets = 12 208 bits.

à chacune il faut ajouter le délai inter-trames de 96 bits, soit :

- **trame mini** : 576 + 96 = 672 bits.
- **trame maxi** : 12 208 + 96 = 12304 bits.

d'où le nombre de tramesmaxi :

- **plus petite trame** : $10^6/672$ = 14 880 trames/sec.
- **plus grosse trame** : $10^6/12304$ = 812 trames/sec.

On peut alors calculer la quantité de données LLC transportées en supposant qu'il n'y ait pas de collision :

- **plus petite trame** : 14 880 * 46 = 684 480 octets/sec.
- **plus grosse trame** : 812 * 1500 = 1 218 000 octets/sec.

Soit, avec une occupation du média à 60% de **0.5 à 0.7 Mo/s**.

Remarque : pour des données utiles de 512 octets, on trouve environ 2200 trames par seconde.

5.2 Avantages et dédauts d'ethernet

Avantages

- logique simple,
- réalisation décentralisée, symétrique,
- ne conduit pas à des blocages,
- il existe de nombreux fournisseurs indépendants,
- les délais de transmission sont courts sur un bus peu utilisé.

Défauts

- ne permets pas de garantir qu'une station pourra émettre une donnée dans un délai déterminé,
- le calcul du délai aléatoire repose sur la connaissance d'un majorant du délai maximal de propagation qui donne aussi le calcul de la longueur minimale de la trame :
  - si calculé trop large, l'efficacité est réduite,
  - aussi il est fixé par la norme, ce qui limite la dimension maximale du réseau ;

Depuis la publication de la norme, l'apparition de nouveaux équipements (ponts ("bridge") et routeurs, fibres optiques, ...) a permis d'étendre les réseaux ethernet pratiquement autant que souhaité.

5.3 Configuration de réseaux ethernet ou 802.3

câblage ethernet "traditionnel"
câble jaune, coaxial 50 ohms, 10 Mbits, maxi 500m

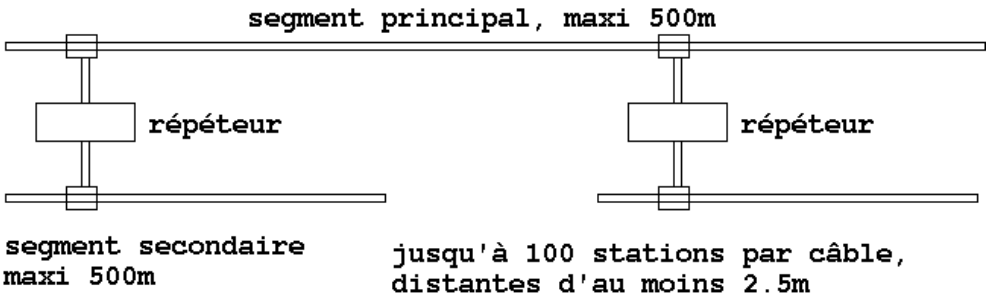


FIG. 45: l'ethernet d'origine : câbles et répéteurs

Câblage ethernet "moderne" : câble à paires torsadées,
10 Mbits, maxi 100m (10BaseT : T=Twisted)

L'ethernet 100 et l'ethernet 1000 sont aussi limités à
100m, à condition d'utiliser des câbles de catégorie >=5.

La topologie d'un réseau 10BaseT est un arbre de
concentrateurs (hubs) et de stations. Chaque station
est connectée directement au hub par un câble 10BaseT.

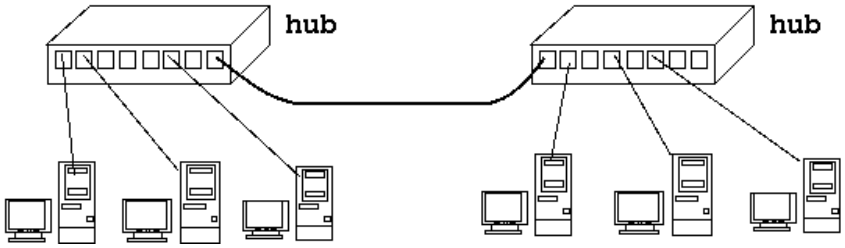


FIG. 46: l'ethernet aujourd'hui : 10BaseT, hubs et switchs

Un "hub" est un répéteur : quand un message arrive sur l'un de ses ports (matétialisé par une prise RJ45), ce message est répété sur tous les ports du hub. Le hub ce comporte exactement comme un câble unique. Toutes les stations peuvent se parler sans intermédiaire. Le hub constitue aussi un "domaine de collision".

Si on relie entre eux deux hubs, on étends le domaine de collision.

Extension d'un ethernet par un pont ("bridge")

Un pont est un équipement plus sophistiqué qu'un simple répéteur. Il possède une mémoire, une unité centrale et (au moins) deux interfaces ethernet.

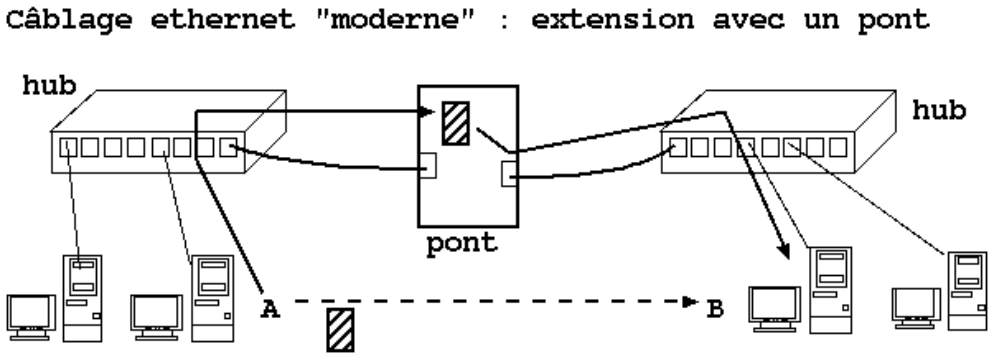


FIG. 47: Extension d'un ethernet par un pont ("bridge")

Fonctionnement : quand une station A veut envoyer un message à une station B, située "de l'autre côté" du pont, c'est le pont qui va récupérer **complètement** le message, puis le ẽ-émettre sur son autre interface, en faisant croire à B que le message à été émis directement par A.

Les stations A et B n'ont aucun moyen de savoir qu'il y a entre elles un poont qui a lu le message et l'a ré-émis. Ceci permet d'insérer un pont dans un réseau, sans rien modifier sur les stations déja connectées.

Le pont est suffisamment "intelligent" pour ne pas retransmettre les collisions, séparant ainsi le réseau en deux domaines de collision. De plus, comme le message en transit est intégralement copié dans la mémoire du pont, le calcul du délai de 51,2μs "repart à zéro" de l'autre côté du pont, permettant ainsi une grande extension du réseau. En particulier par l'utilisation de plusieurs ponts connectés entre eux par des fibres optiques.

Comment le pont "sait-il" de quel côté se situe une machine ? En fait il le sait par auto-apprentissage : comme toute trame émise contient l'adresse ethernet de la source, il peut très vite remplir une table contenant toutes les adresses sources ayant émis des trames et les associer à l'interface sur laquelle il a "vu" cette source.

Le pont se met dans un mode spécial dit "promiscuous" : il écoute **toutes** les trames qui circulent sur chacun des réseaux et ne retransmet de l'autre côté que celles qui y sont destinées.

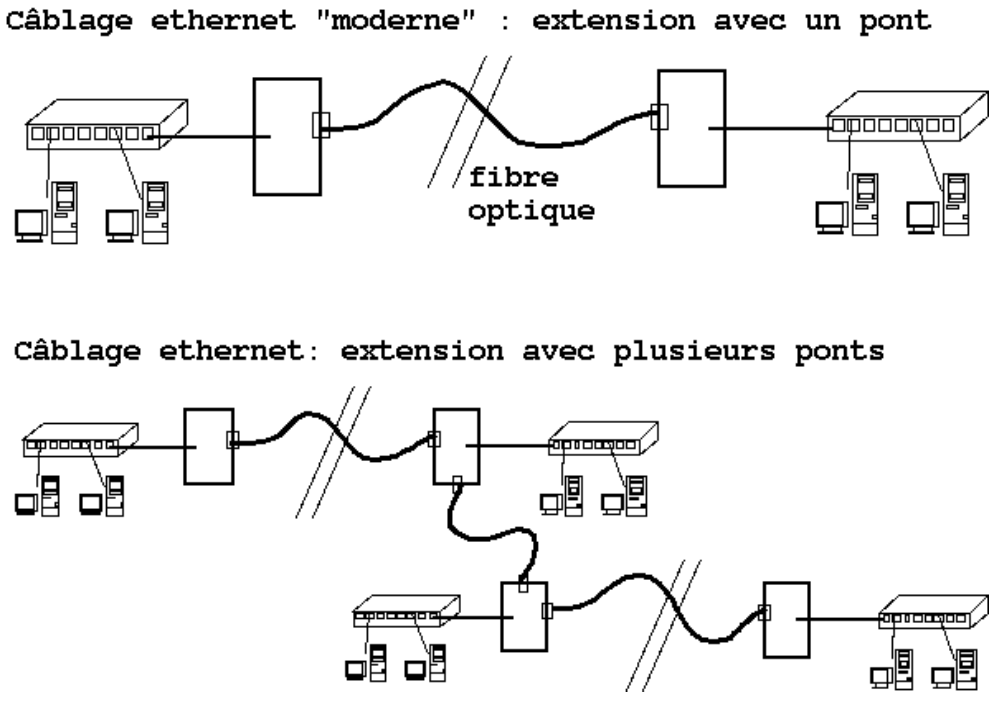


FIG. 48: Extension d'un ethernet par des ponts

Utilisation de commutateurs ethernet ("switch")

Un commutateurs a le même aspect qu'un hub : une boîte comportant des prises RJ45 (de 4 à 48 suivant les modéles). Il existe de plus gros commutateurs pouvant contenir plusieurs centaines de prises : ils sont constitués de chassis dans lesquels on insère des cartes de 24 ou 48 prises.

La différence entre un switch et un hub est bien sûr à l'intérieur : le commutateur est une sorte de pont généralisé : chacun des ports du commutateur se comporte comme un port de pont. Les paquets qui arrivent sur un port sont stockés dans la mémoire du commutateur et mis dans la file d'attente du port de destination en sortie.

5.4 Passage à l'ethernet 100

À 10 Mbps durée de trame mini est 51,2μs. À 100 Mbps il faudrait 640 octets pour remplir cette durée, or la plupart des trames sont petites, donc il faudrait beaucoup de bourrage entrainant une sous-utilisation du réseau.

Ceci implique de réduire la durée maxi de propagation pour compenser, donc la distance maxi du réseau.

Le choix de la norme est de passer la tranche canal à 5,12μs et de garder une taille de trames identique à ether-10 et de diviser les distances par 10.

En pratique, l'ethernet 100 n'existe que sur des câbles à paires torsadées de catégorie supérieure ou égale à 5 sur lesquelles la distance utilisable est de 100m.

Un autre avantage de ce choix de diminuer la tranche canal en gardant la même taille de trame et que les équipements peuvent traiter sans modification des trames émises à 10 et des trames "100". Ainsi il est possible de construire des commutateurs ou des hubs qui gèrent un mélange d'équipements "10" et d'équipements "100".

5.5 Différence entre débit nominal et débit utile, différence entre LAN et WAN

- **LAN** : temps propagation courts : on peut utiliser ce délai court dans la définition des protocoles ;
- **WAN** : temps propagation longs => en général, plusieurs paquets sont émis avant de recevoir une information en retour => protocoles différents.

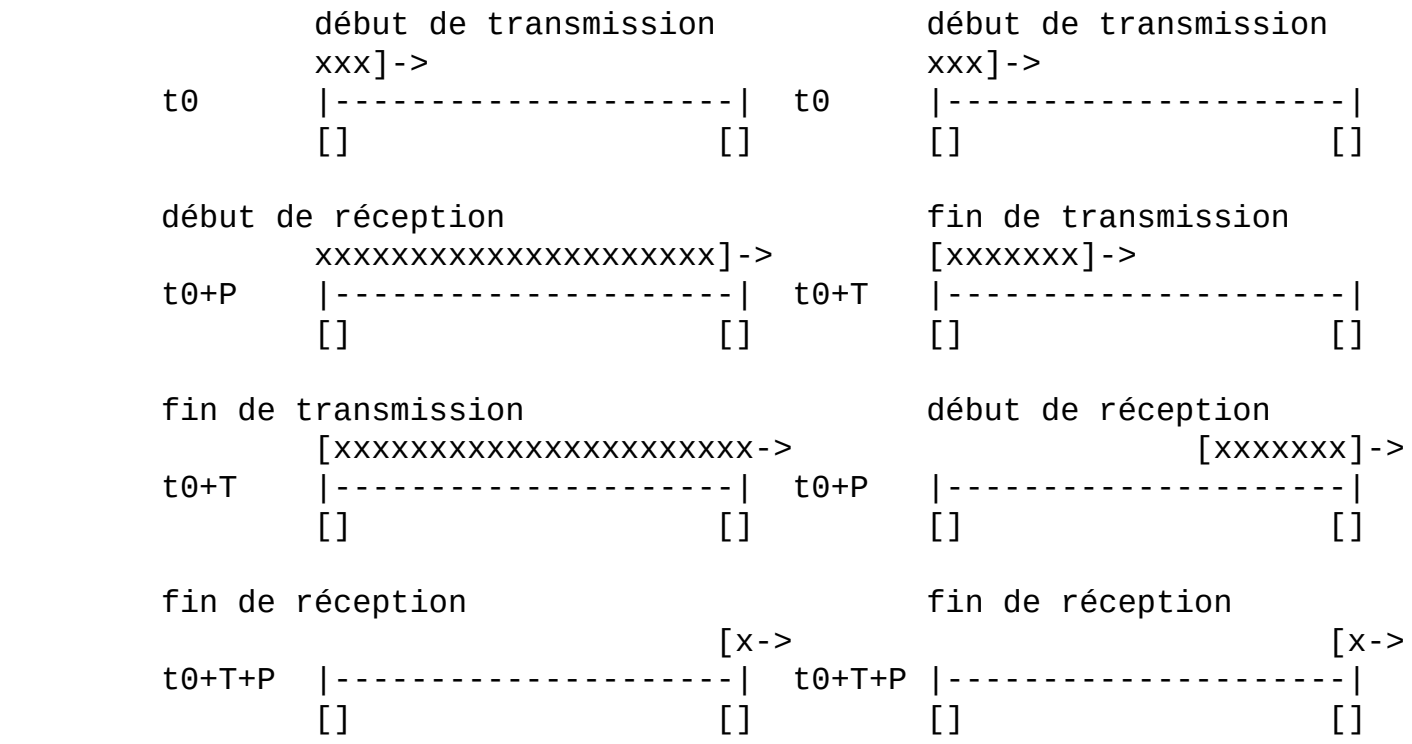
Soit "a = P / T" le rapport du temps de propagation (temps mis par le premier bit pour atteindre la destination) sur le temps de transmission (temps mis par la source pour envoyer le paquet).

Le média est occupé depuis le premier bit envoyé (posé sur le média) jusqu'à la réception du dernier bit par le destinataire, soit : occupation = transmission + propagation.

Le pourcentage d'utilisation du média est : U = T / (T+P) (rapport de la partie utile, la transmission, sur l'occupation).

Soit U = 1 / (1 + P/T) = 1 / (1+a)

Les scémas ci-dessous illustrent la formule dans les deux cas où le temps de propagation est » temps de transmission (cas le plus fréquent dans les WAN haut débit), et celui où le temps de propagation est « au temps de transmission (cas le plus fréquent dans les LAN).



Le média est libéré quand le dernier bit du message est retiré par le destinataire. Ce dernier bit entre sur le réseau à t0+T et en ressort "P" secondes plus tard.

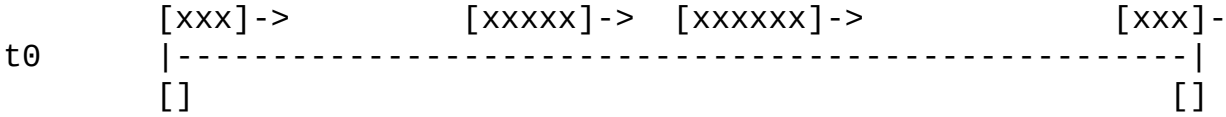
Application numérique

débit Mbps	taille trame (bits)	long réseau (km)	a	U
1	100	1	0.05	0.95
1	1000	10	0.05	0.95
1	100	10	0.5	0.67
10	1000	1	0.05	0.95
10	100	1	0.5	0.67
100	10000	1	0.05	0.95
100	1000	1	0.5	0.67
100	100	1	5.	0.17
100	10000	10	0.5	0.67
100	1000	10	5.	0.17
100	1000	50	25.	0.04

Exemple :
débit 10 Mbits, trame de 100 bits =>
T=100*10*10^-6=10^-5.
P=D/V= 1 / 200 000 = 0.5 10^-5
a = P/T = 0.5

Ainsi on voit que lorsque le débit et la distance augmentent, il devient très difficile d'utiliser correctement le réseau.

Sur un WAN il faut utiliser des protocoles qui permettent d'envoyer plusieurs trames en même temps sur le réseau :



Application numérique

En 1 nanoseconde, la lumière parcours $30010^6/10^9 = 0,3m$ ... dans le vide.

Dans une fibre optique d'indice 1,45, la vitesse est de 210 000 km/s. La distance parcourue en une nanoseconde est de 0,21m.

Pour parcourir une fibre optique de 6000 km, il faut à la lumière 30 ms, soit 60ms d'aller-retour.

Si dans une fibre optique gérée à 1 Gigabits/sec on envoie des paquets de 576 octets, soit 4608 bits, en 30ms, on a le temps de déposer 6510 paquets, soit plus de 3 Mo utiles !

Si, à la réception du premier paquet qui ressort de la fibre, le destinataire envoie un ordre "stop", cet ordre d'arrêt ne parvient à l'émetteur qu'une fois que celui-ci à déja envoyé 6 méga-octets !

Donc, on ne peut pas "remplir" une liaison de ce type avec une seule connexion gérée par un protocole qui implique des échanges entre émetteur et récepteur.

Pour "remplir" les liaisons à très haut débit, on va multiplexer un très grand nombre de liaisons à bas débit afin d'avoir toujours quelque chose d'utile à transférer. La gestion de ce type de liaison est le métier des opérateurs de télécom, assez différent de celui de la gestion de réseaux locaux.

Par exemple, pour "remplir" les fibres utilisées aujourd'hui au débit de 1 térabit/s, on multiplexe par équipement WDM (Wave Division Multiplexing) 40 circuits à 2.4 gigabits, eux-mêmes constitués du multiplexage de liaisons à plus bas débit.

Vous pourriez faire remarquer, que la distance étant toujours présente, la liaison bas débit tout en bas de la chaine n'est pas non plus "remplie". D'abord, du fait qu'elle est à plus bas débit, la perte est moindre, mais surtout, l'opérateur se moque qu'elle soit sous utilisée car il a vendu la totalité du débit.

Quand on achète 512kbits à un opératuer, on paye la même chose que l'on s'en serve ou pas !

Page blanche

SR03 2004 - Cours Architectures Internet - Bases d'un internet
©Michel.Vayssade@utc.fr – Université de Technologie de Compiègne.

6 SR03 2004 - Cours Architectures Internet - Bases d'un internet

6.1 Introduction au concept d'internet

On a vu que la gestion d'une communication à distance entre deux programmes pouvait devenir vite complexe.

Les concepteurs disposent de deux approches pour masquer cette complexité :

– utiliser les programmes de niveau "application",
– cacher les détails dans le système d'exploitation.

La première approche conduit à des communications peu efficaces (on a vu qu'il n'est pas si simple d'utiliser au mieux la bande passante d'un réseau à haut débit). De plus, l'ajout de fonctions nouvelles conduit à modifier les programmes sur chaque noeud du réseau.

Si on ne dispose pas de service de transport de type "réseau", les programmes applicatifs doivent se charger de la communication de proche en proche. Or, l'expérience a montré que cette méthode fonctionne très mal sur de grands réseaux (elle n'est pas "scalable", ne peut changer d'échelle).

L'alternative est de baser les communications entre programmes sur un système d'interconnexion qui opère au niveau du réseau.

C'est un mécanisme, inclus dans le S.E. qui assure en temps réel le contrôle de l'acheminement des paquets de données entre l'application émettrice et l'application destinataire.

Ce mécanisme va commuter de "petits" paquets de données plutôt que des messages "longs" (petits = 100 à 1000 octets).

Ceci présente plusieurs avantages :

– une correspondance directe avec les réseaux physique sous-jacents (ethernet, FDDI, X25, ATM, ...),
– séparer l'activité de communication des programmes applicatifs, ce qui permet aux machines intermédiaires de faire leur travail sans avoir besoin de comprendre les informations de niveau applicatif,
– conserver de la souplesse au système (modifier les applis ou ajouter de nouveaux protocoles).

Le concept de base d'un internet est celui d'un système de communication abstrait dissociant la notion de communication des détails techniques des réseaux et masquant ces détails de bas niveau aux programmes d'application. Ce système abstrait repose sur la technique d'interconnexion de réseaux (internetworking).

Cette notion d'interconnexion résulte de deux constats :

– un seul type de réseau ne peut satisfaire tous les besoins (différences entre LAN et WAN),
– on souhaite pouvoir interconnecter tout type de réseaux.

6.2 Architecture d'un internet

Les utilisateurs ou les programmes d'application ne doivent pas avoir à connaitre les détails des connexions physiques pour pourvoir utiliser un internet.

On doit pouvoir connecter un nouveau réseau à un internet existant sans avoir besoin de le connecter à un ordinateur central.

Tous les ordinateurs de l'internet doivent partager un ensemble d'identificateurs universels.

Comment des réseaux différents sont-ils connectés pour former un internet ?

> Un internet viable nécessite des équipements spécialisés capables de faire passer des paquets de données d'un réseau à un autre.
> ==> Ces équipements sont appelés "passerelles IP" ou "routeurs IP" (Internet gateway ou Internet router).

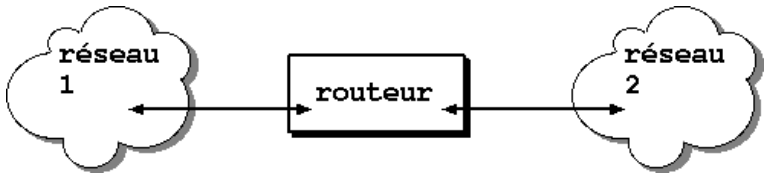


FIG. 49: Réseaux interconnectés par un routeur

Ici le routeur [R] prends des blocs de données sur le réseau 1 qui sont destinés à des machines du réseau 2, et les transmet sur le réseau 2 (en faisant si nécessaire les conversions de trames si les réseaux ne sont pas physiquement de même type) (et symétriquement dans l'autre sens).

Dans un internet plus complexe que celui de la fig. précédente, chaque routeur doit connaître des données relatives à la topologie générale du réseau bien au-delà des réseaux qui lui sont immédiatement raccodés.

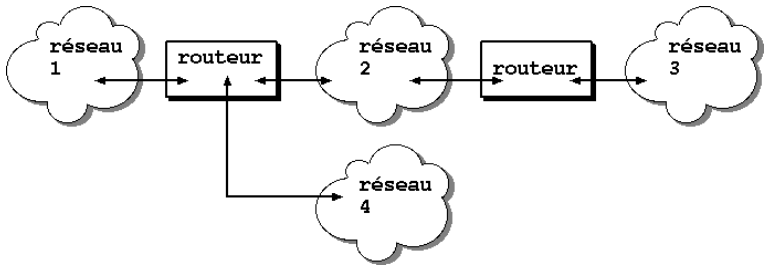


FIG. 50: Réseaux interconnectés par des routeurs

Dans ce deuxième exemple, R1 doit savoir que les données à destination du réseau "3" doivent être envoyées vers le réseau 2 ; de même, R2 doit savoir que les données qu'il reçoit pour les réseaux 1 et 4 doivent être envoyées sur le réseau 2.

Quand l'internet grandit, la tâche du routeur qui doit choisir la route que vont prendre les paquets qui transitent par lui devient plus complexe.

> Un internet est un ensemble de réseaux interconnectés par des équipements spécialisés appelés routeurs qui assurent le service d'acheminement des données d'une source quelconque vers une destination quelconque de façon transparente pour la source et la destination.

On pourrait craindre que dans un internet contenant des millions de machines chaque routeur doivent connaître la lccalisation de l'ensemble des machines. En fait non, l'adressage hiérarchique de IP permet à chaque routeur de n'avoir à connaître que des numéros de réseaux (et même dans les versions récentes des ensembles de réseaux que l'on a rendus "contigus" au sens de la topologie du réseau). Ainsi les routeurs peuvent être réalisés avec de petits ordinateurs.

La façon dont les routeurs apprennent et mettent à jour leurs connaissances des routes vers les autres routeurs de l'internet (protocoles RIP, BGP, EGP) fait partie des cours sur les réseaux.

Les protocoles TCP/IP ont été conçus pour que les utilisateurs puissent se représenter l'internet comme un simple moyen de communication entre toutes les machines participantes, quel que soit le sous-réseau sur lequel elles sont effectivement branchées.

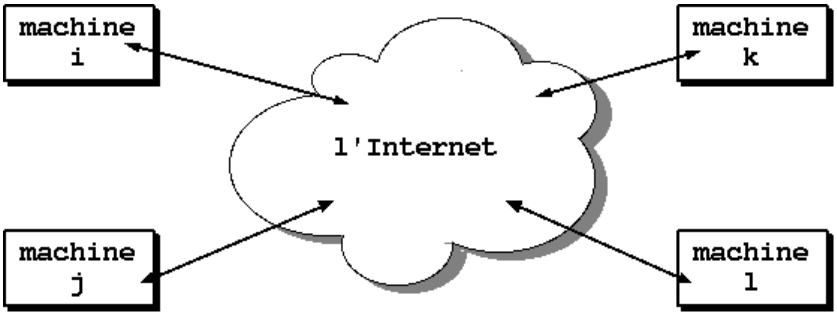


FIG. 51: L'Internet permet d'interconnecter des machines

Il est possible en utilisant correctement les protocoles TCP/IP d'écrire des programmes d'application qui fonctionneront entre l'ordinateur [j] et l'ordinateur [k] sans avoir à connaître le détail des sous-réseaux (éventuellement nombreux) qui les séparent.

Il est même possible de modifier la structure du réseau sous-jacent et de continuer à utiliser les programmes d'application sans avoir besoin de les recompiler.

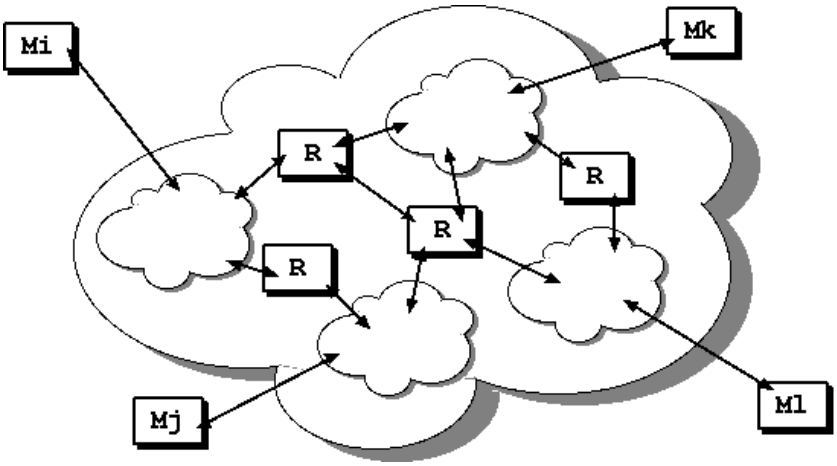


FIG. 52: L'Internet est un réseau de réseaux

Les routeurs d'un internet n'assurent pas une connectivité directe d'un réseau quelconque vers un autre réseau quelconque. Le traffic entre deux réseaux peut emprunter un ou plusieurs réseaux intermédiaires. Les réseaux qui participent à un internet doivent accepter du traffic "de transit" en plus de leur traffic propre. En contrepartie il peut écouler son propre traffic en utilisant les autres réseaux de l'internet.

Nota : en pratique l'Internet est constitué de réseaux de transport gérés par des opérateurs qui passent entre eux des accords de traffic croisé (accord de "peering") tenant compte des quantités de traffics échangés et donnant lieu éventuellement à des compensations financières (les plus gros font payer les petits, comme d'habitude). Les réseaux des fournisseurs d'accès (ISP : Internet Service Provider, FAI en français) sont connectés en bordure de ces réseaux de transport (moyennant finances évidemment).

6.3 Stratification en couches de TCP/IP

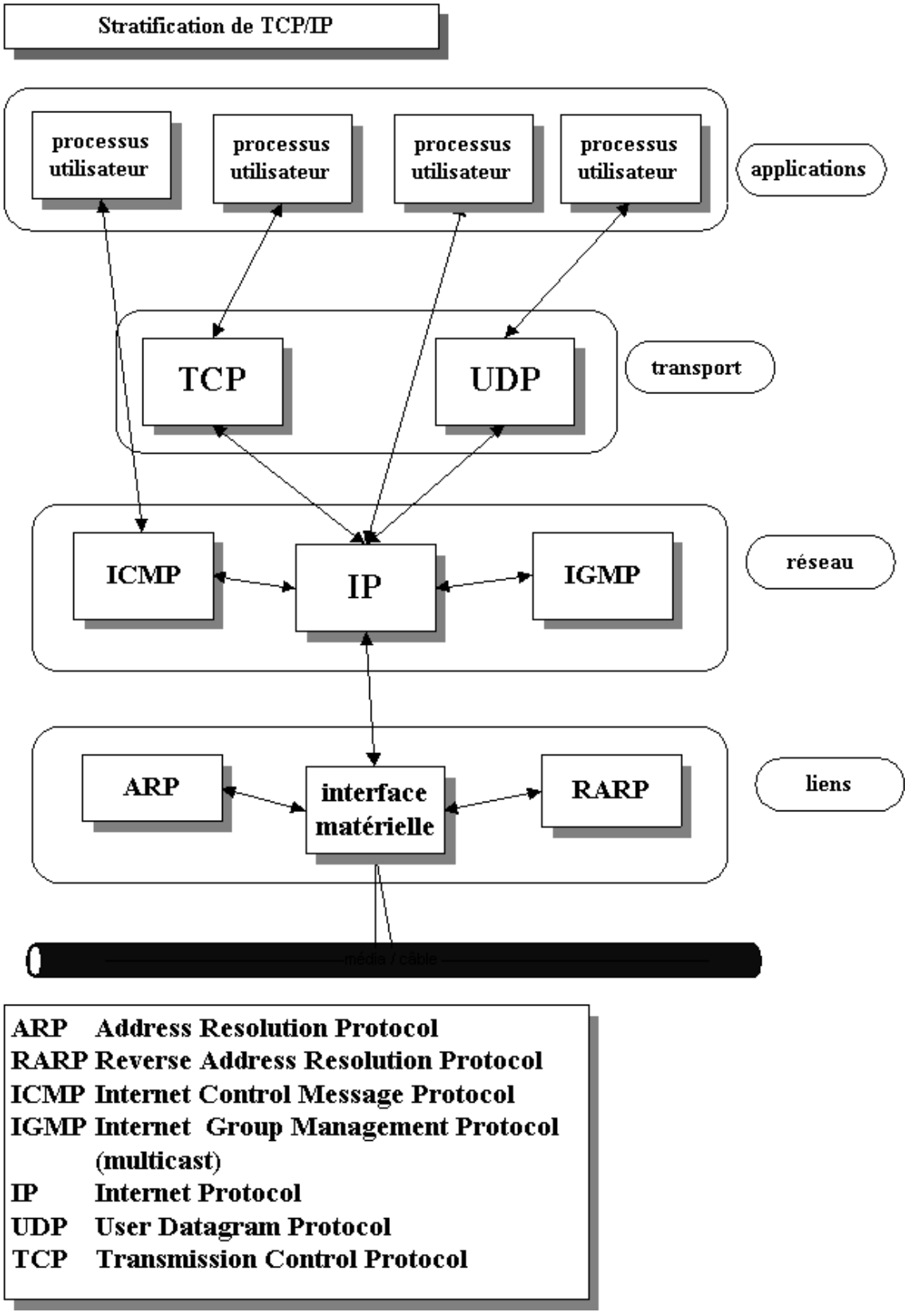


FIG. 53: Les couches de TCP/IP

– **Couche lien :** carte réseau + driver gèrent interface avec le média (LLC+MAC).

- **Couche réseau : IP :** gère les échanges de paquets, envoie les paquets directement au destinataire ou à une passerelle (gateway) en fonction des informations de routage (netstat -r) dont elle dispose,
- **Couche transport : TCP :** gère le flux de données
  - cette couche envoie à IP des paquets de taille ad-hoc,
  - fait la gestion des acquittements, établit des temporisations pour s'assurer que tous les paquets sont acquittés (garantie de transmission fiable),
  - UDP, elle, fournit un service plus simple, non fiable, sans ACK.

Ces 3 couches fonctionnent en **mode système** et gèrent les détails de la communication.

Couche applicative : elle fonctionne en mode utilisateur. Exemples : Telnet, rlogin, FTP, SMTP (mail), SNMP, HTTP, ...

Beaucoup d'applications sont non-symétriques : le côté client et le côté serveur sont différents.

6.4 Couches liens, IP et routage

La couche IP et la couche liens ne font pas seulement la liaison entre deux machines, mais aussi entre deux réseaux en utilisant un équipement équipé de deux cartes d'interface (de type éventuellement différent : par exemple une carte ethernet et une carte token-ring). Cet équipement est appelé un **routeur**.

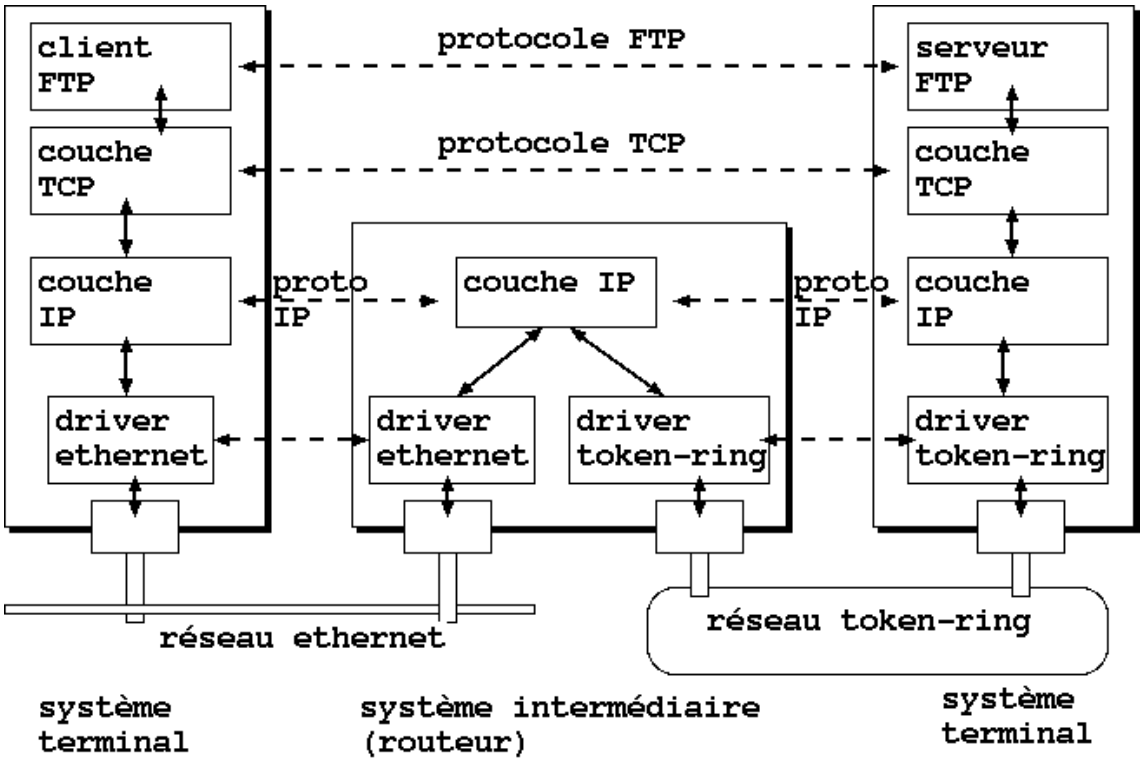
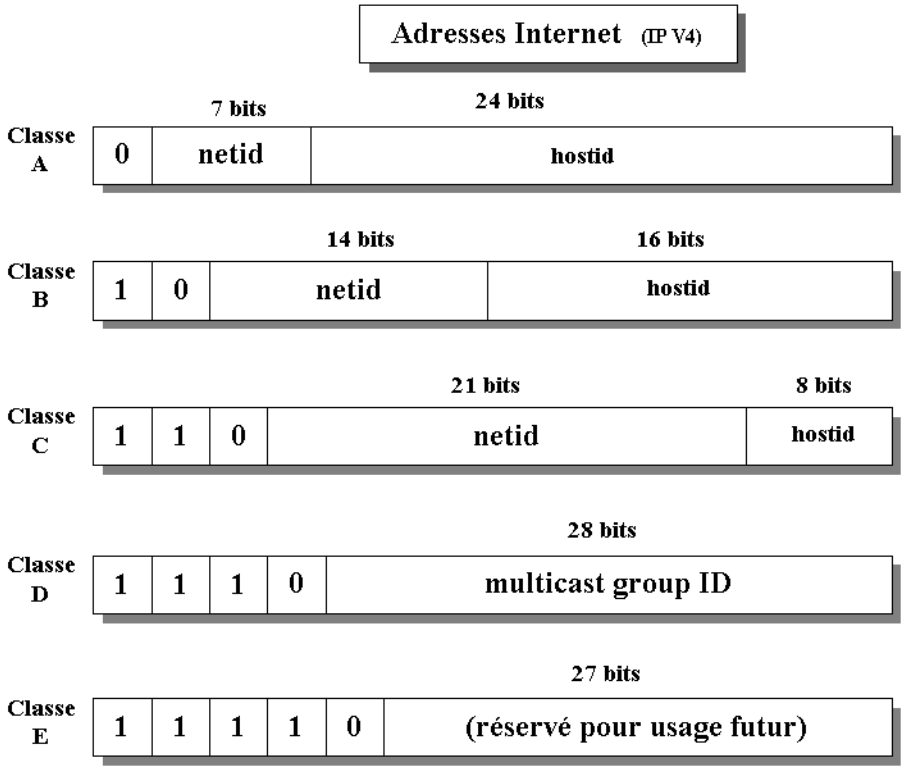


FIG. 54: Un routeur fonctionne sur la couche IP

Le protocole IP est dit protocole de saut en saut (hop by hop), alors que les protocles TCP (ou UDP) sont dits "protocoles de bout en bout" (end to end).

6.5 Les adresses IP

Dans un internet les adresses de toutes les machines participantes doivent être **uniques**.
==> ceci implique une cohérence dans les attributions.
==> autorité globale : Inter-NIC (Network Information Center).
cette autorité **délègue** la gestion de zones géographiques à des autorité locales, en leur confiant la gestion d'un **groupe d'adresses**.
Les adresses IP sont de trois types :
– unicast : concerne **une** machine,
– broadcast : concerne **toutes** les machines **d'un** réseau,
– multicast : concerne **un ensemble** de machines (les membres d'un groupe multicast).



A	0.0.0.0	**127.**255.255.255
B	**128.**0.0.0	**191.**255.255.255
C	**192.**0.0.0	**223.**255.255.255
D	**224.**0.0.0	**239.**255.255.255
E	**240.**0.0.0	**247.**255.255.255

FIG. 55: Les adresses internet

6.6 Les noms de domaines et de machines : le DNS

Toute machine a donc une adresse unique. Toutefois les machines sont utilisées par des humains qui manipulent plus facilement des noms que des numéros.

Dans le monde TCP/IP, le **service de noms de domaines** (**DNS**) est une application qui gère une base de donnée **distribuée** qui associe un nom unique à chaque adresse.

Des fonctions du système d'exploitation (**gethostbyname**) permettent de passer des noms aux adresses et inversement.

L'espace de noms du DNS est une structure arborescente. Chaque noeud de l'arbre possède une étiquette (un nom), la liste des étiquettes depuis la racine désigne un sous-domaine.

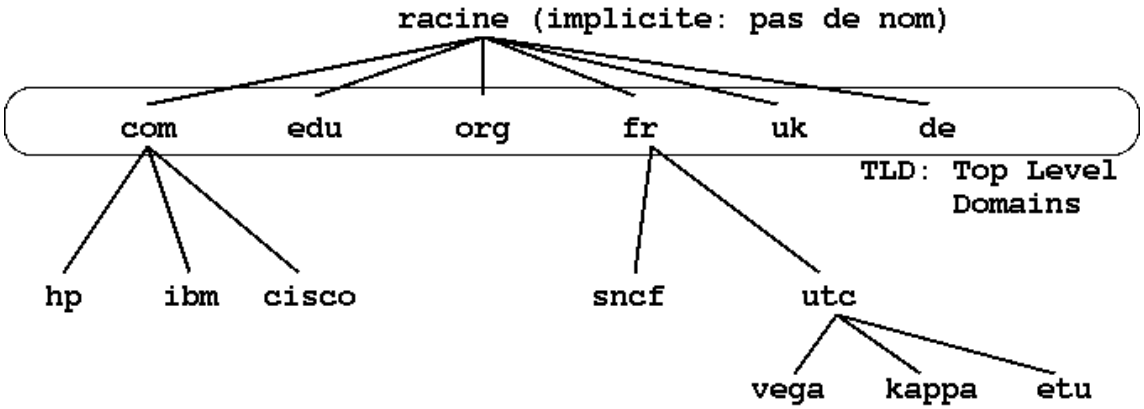


FIG. 56: L'arbre du domaine de noms

À chaque noeud de l'arbre est associé un ensemble d'informations constituant le "Ressource Record" (RR). Ce peut être une adresse IP (pour une machine) ou la désignation d'un serveur de courrier (exemple du noeud "etu" dans le schéma ci-dessus).

Le DNS est maintenu par un ensemble de serveurs de noms. Chacun d'eux se voit attribuer une autorité pour des parties de l'espace des noms. Une requête de type "SOA" (Source Of Authority) donne le serveur qui a ctte délégation pour un domaine :

```
$ dig -tSOA kappa.utc.fr
...
;; AUTHORITY SECTION:
utc.fr.                 299     IN      SOA     orion.utc.fr.
```

Quand un programme a besoin d'une information DNS (traduire un nom en adresse ou l'inverse), il interroge un programmme résolvant (le "resolver") qui lui-même interroge un serveur DNS.

Le résolver trouve l'adresse du serveur à interroger dans le fichier **/etc/resolv.conf** (voir man resolv.conf).

Si ce serveur DNS ne connait pas la réponse, il interroge le serveur DNS de niveau supérieur qui se chargera de transmettre la question. La réponse sera, soit trouvée dans un cache, soit atteindra un serveur DNS ayant autorité pour le domaine concerné par la question.

Les programmes gérant le DNS (serveurs et résolveurs) utilisent les ports **TCP 53** et **UDP 53**. Il faut donc "laisser passer" ces 2 ports dans la configuration d'un routeur pare-feu.

Le DNS est en général implémenté par le programme BIND, issu de la distribution unix de Berkeley. Ce programme BIND s'incarne dans le démon **named**.

Les étiquettes (noms des noeuds) sont définies sur au plus 63 caractères, dont le premier désigne la longueur (donc 62 car. utiles). **Mais** la longueur totale d'un nom de domaine (par

exemple kappa.utc.fr) ne doit pas dépasser 255 caractères. D'où la tendance des administrateurs à choisir des noms courts, sinon on limite implicitement la profondeur de l'arbre.

Majuscules et miniscules sont **interchangeables** dans un nom DNS : KAPPA.utc.fr est la même chose que Kappa.UTC.fr.

Exemple :

```
$ host kappa.utc.fr
kappa.utc.fr has address 193.55.117.100

$ host -a kappa.utc.fr
Trying "kappa.utc.fr"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 42370
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 5, ADDITIONAL: 4

;; QUESTION SECTION:
;kappa.utc.fr.                  IN      ANY

;; ANSWER SECTION:
kappa.utc.fr.           7194    IN      A       193.55.117.100

;; AUTHORITY SECTION:
utc.fr.                 7194    IN      NS      ns0.pasteur.fr.
utc.fr.                 7194    IN      NS      ns1.pasteur.fr.
utc.fr.                 7194    IN      NS      orion.utc.fr.
utc.fr.                 7194    IN      NS      epsilon.utc.fr.
utc.fr.                 7194    IN      NS      ns.crihan.fr.

;; ADDITIONAL SECTION:
ns.crihan.fr.           86015   IN      A       195.221.20.10
ns.crihan.fr.           78556   IN      AAAA    2001:660:7401:201::10
orion.utc.fr.           6815    IN      A       195.83.155.16
orion.utc.fr.           7194    IN      AAAA    2001:660:4201:1::2

Received 244 bytes from 193.252.19.3#53 in 121 ms
```

Fonction inverse

Pour retrouver un nom à partir d'une adresse, il a été créé un pseudo-domaine **IN-ADDR.ARPA** dans lequel les noeuds sont étiquettés par leur adresse IP, lue à l'envers (pour avoir le numéro de réseau près de la racine). Ainsi, kappa.utc.fr est repésentée par :

```
$ host 193.55.117.100
100.117.55.193.in-addr.arpa domain name pointer kappa.utc.fr.
```

6.7 Encapsulation dans TCP/IP

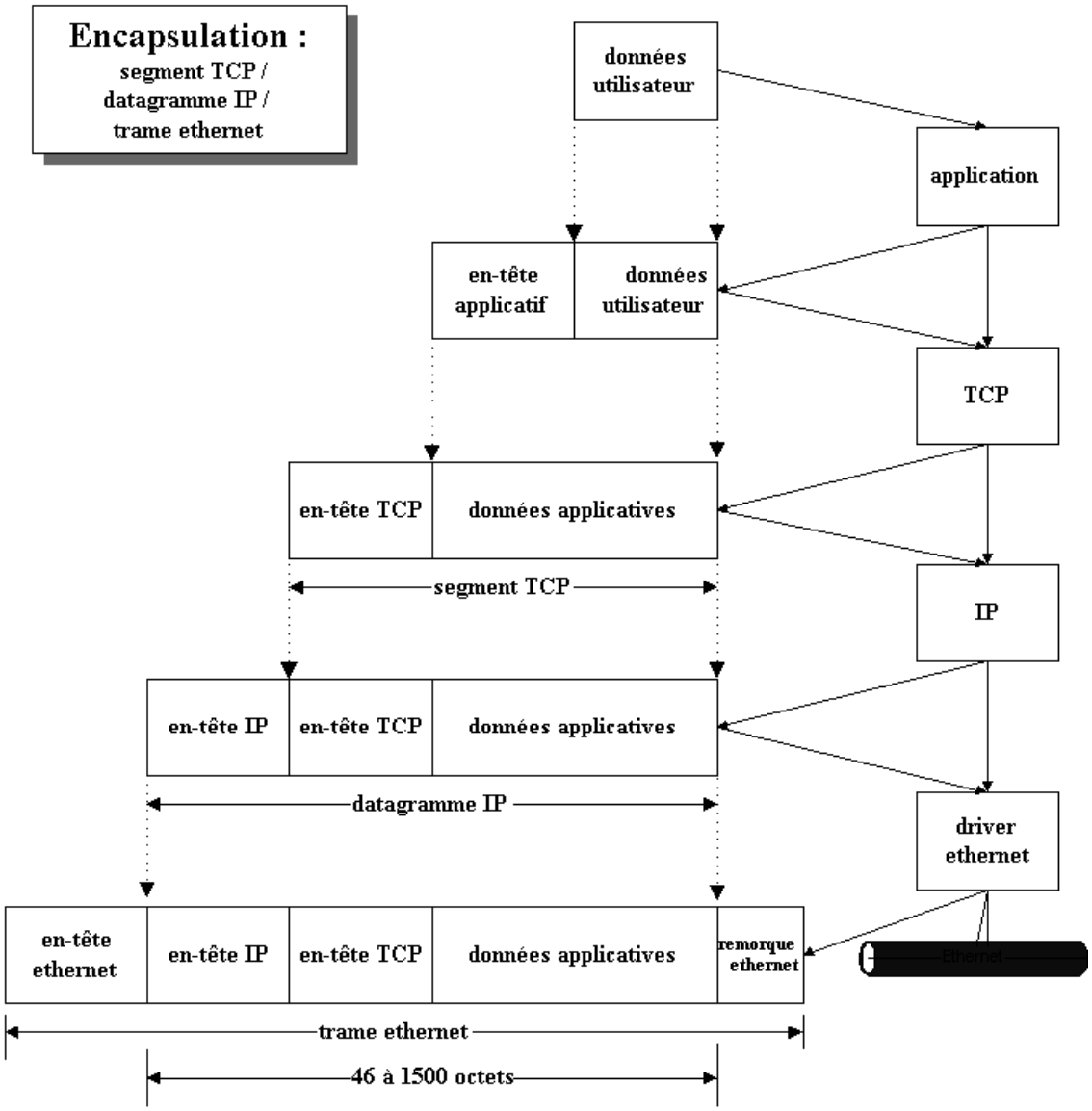


FIG. 57: Encapsulation dans TCP/IP

Au niveau de la couche liens, IP transmet à l'interface réseau un **paquet** qui peut être :

– soit un datagramme complet (cas le plus fréquent),

– soit un fragment de datagramme (si le MTU du réseau est trop petit).

TCP, UDP, ICMP et IGMP envoient tous des données à IP.

=> à l'arrivée il faudra rendre à chaque service les données qui sont les siennes.

==> pour cela, IP distingue l'origine de chaque paquet par le champ **proctocole** de l'en-tête IP : ICMP=1, IGMP=2, TCP=6 et UDP=17.

De façon analogue, TCP (ou UDP) peuvent recevoir des données de plusieurs applications différentes.

=> ils utilisent pour distinguer chacune des applications utilisatrices la notion de **numéro de port** (un entier sur 16 bits).

Un certain nombre de ports sont préaffectés à certaines applications : ce sont les **ports bien connus** (well known ports) (liste dans **/etc/services**), par exemple :

– FTP serveur = TCP / 21

– Telnet serveur= TCP / 23
– TFTP serveur = UDP / 69

Les ports bien connus sont en général entre 1 et 1023. Ils sont gérés par l'**IANA** (Internet Assigned Numbers Authority).

Sur unix, les ports 1 à 1023 sont réservés aux process systèmes (en "su").

6.8 Démultiplexage des trames IP

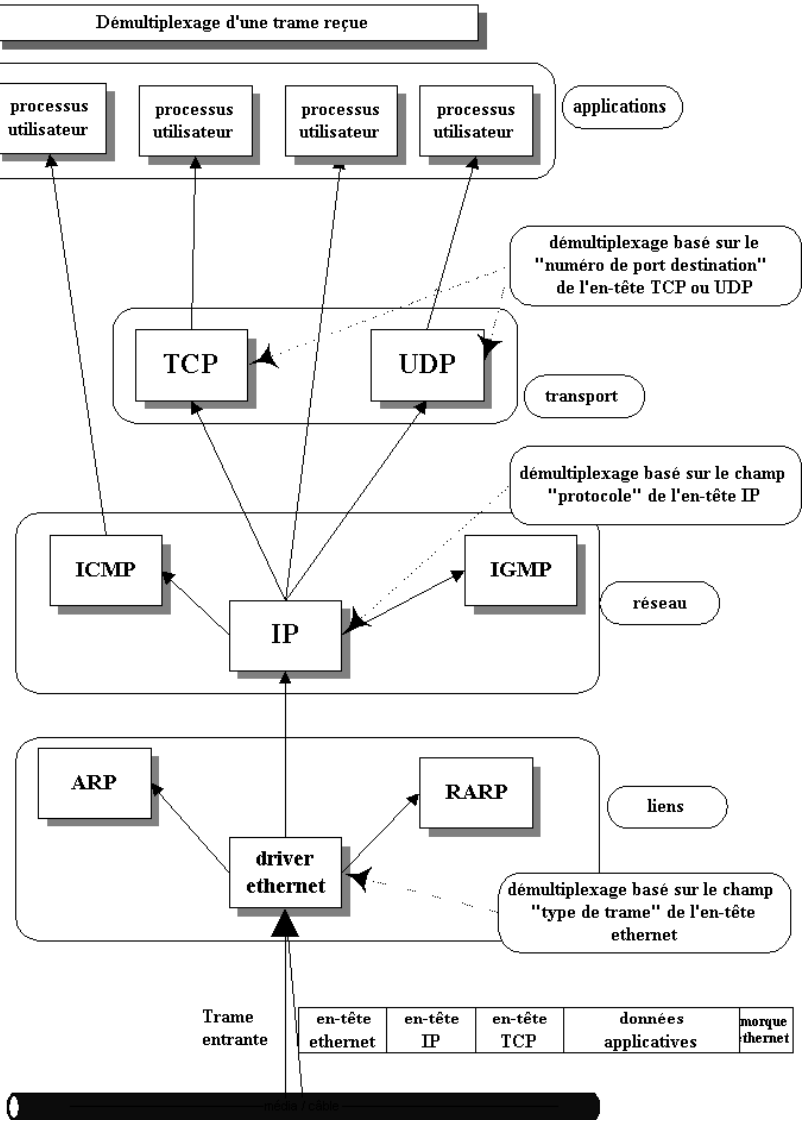
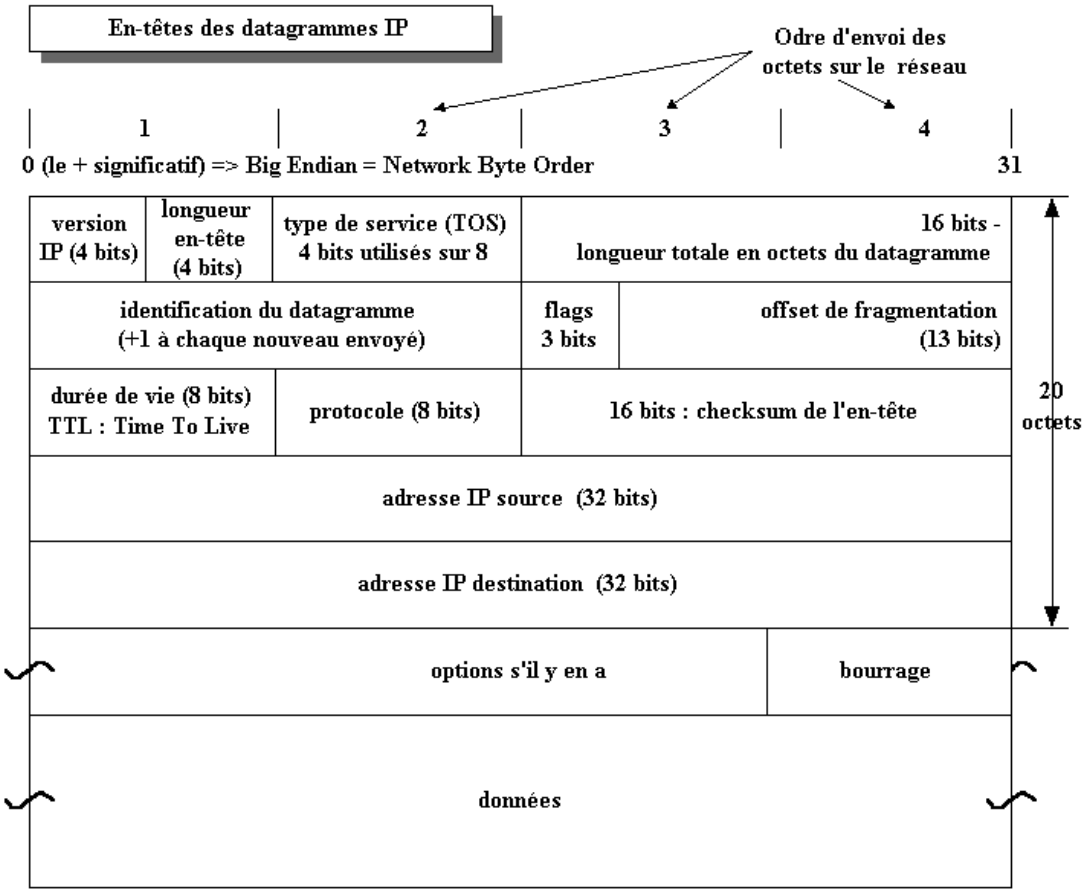


FIG. 58: Démultiplexage des trames dans TCP/IP



flags | bit : "more fragments" = 1 si fragment, = 0 si dernier frag.
| bit : "don't fragment"

fragment offset : permet remettre fragments à leur place si arrivent dans le désordre.
longueur totale : oblig. car couche liens peut faire une trame plus longue que le datagramme (ex: ethernet lng mini 46 octets).

FIG. 59: Entêtes des datagrammes IP

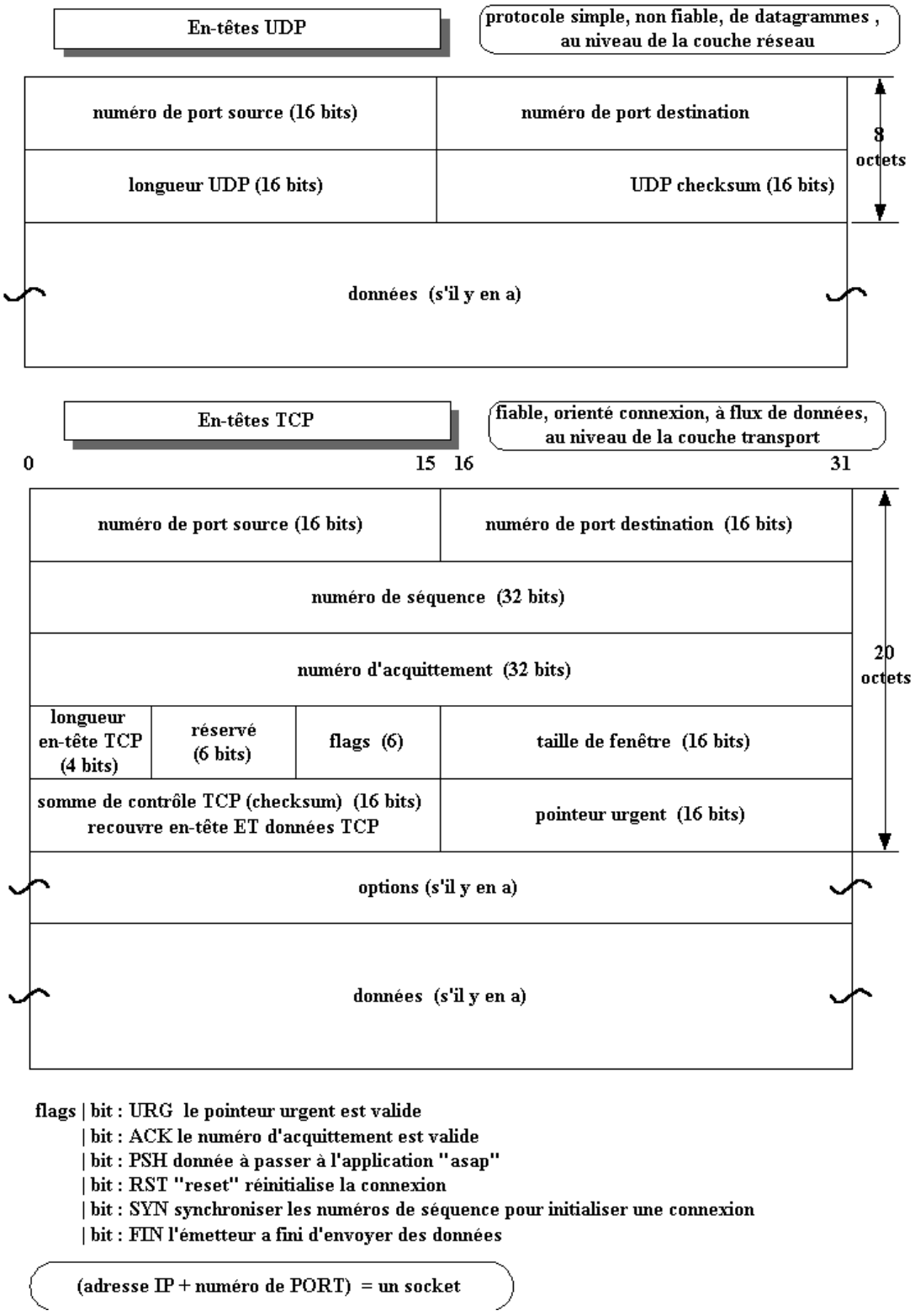


FIG. 60: Entêtes UDP et TCP

6.9 Encapsulation ethernet

Rappel : sur un réseau ethernet peuvent cohabiter (et interopérer) 2 types de trames :
- ethernet - RFC 894,
- 802.3 - RFC 1042.

```
[root@localhost /root]# arp -a
brx (192.168.130.1) at 40:40:40:40:30:C6 [ether] on eth0
```

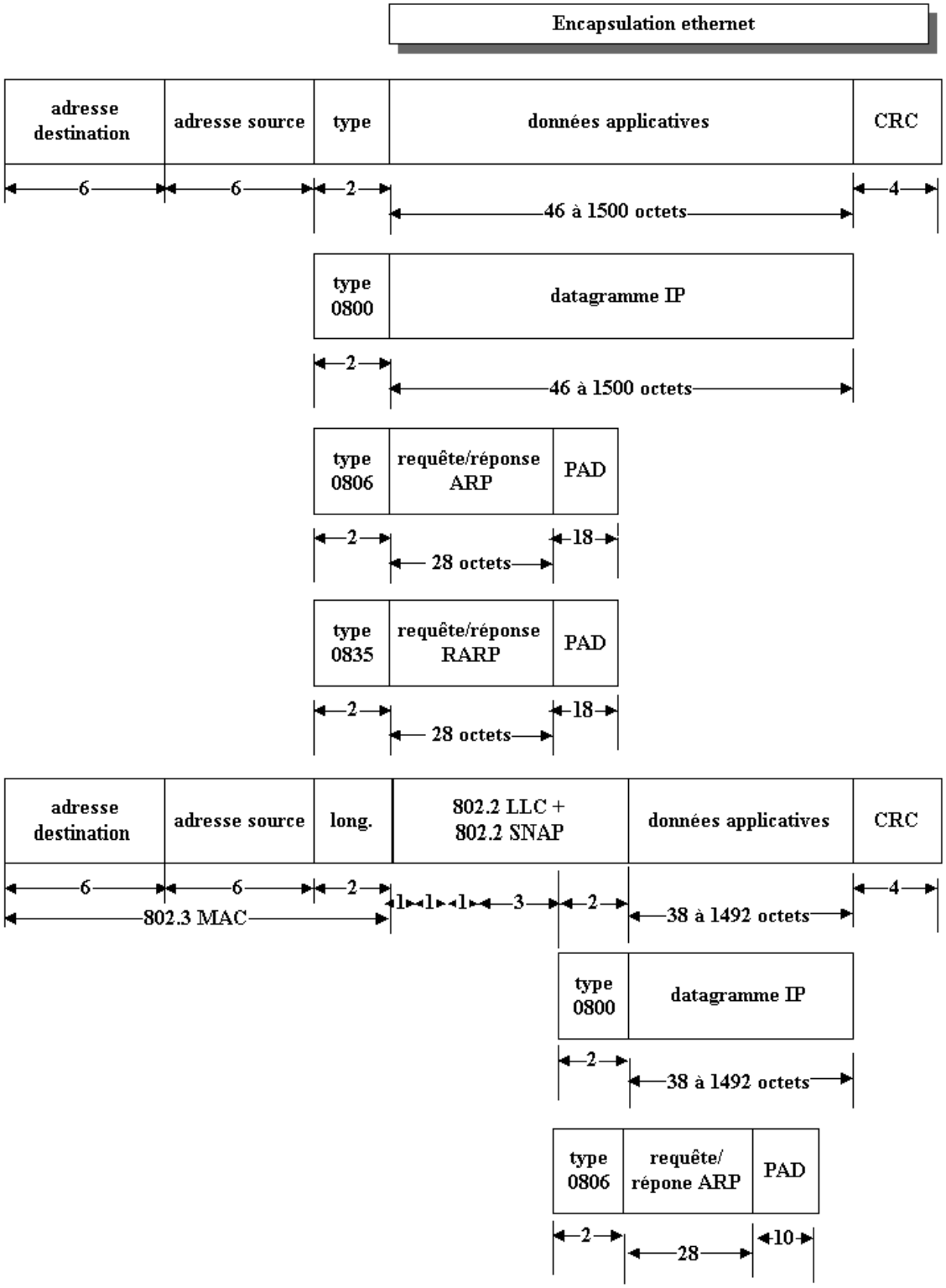


FIG. 61: Encapsulation ethernet

6.10 Protocole PPP

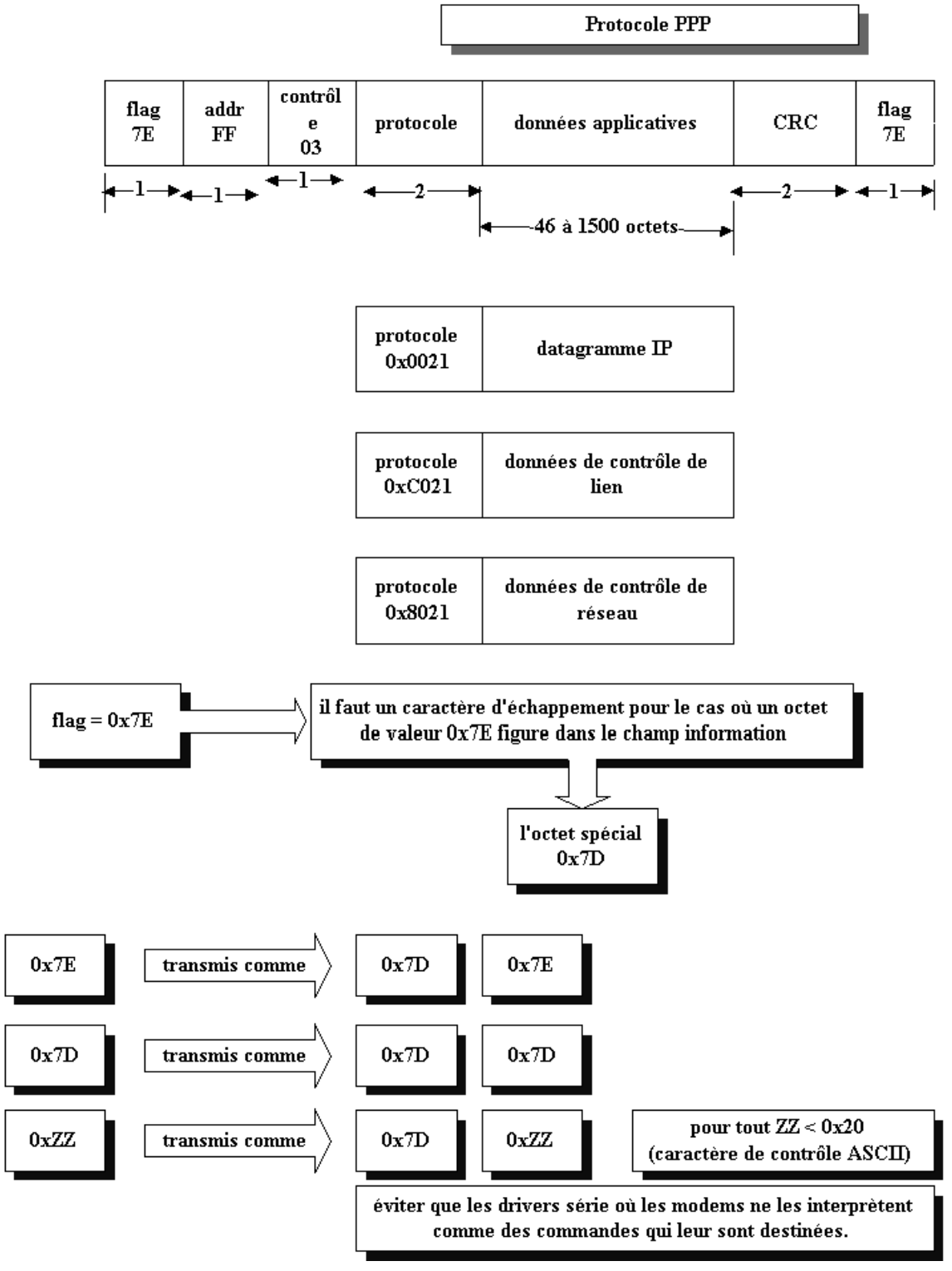


FIG. 62: Protocole PPP

6.11 IP : le Protocole Internet

IP est le protocole de base de la pile "TCP/IP". Les données de TCP, d'UDP, d'ICMP et d'IGMP sont transmises comme des datagrammes IP.

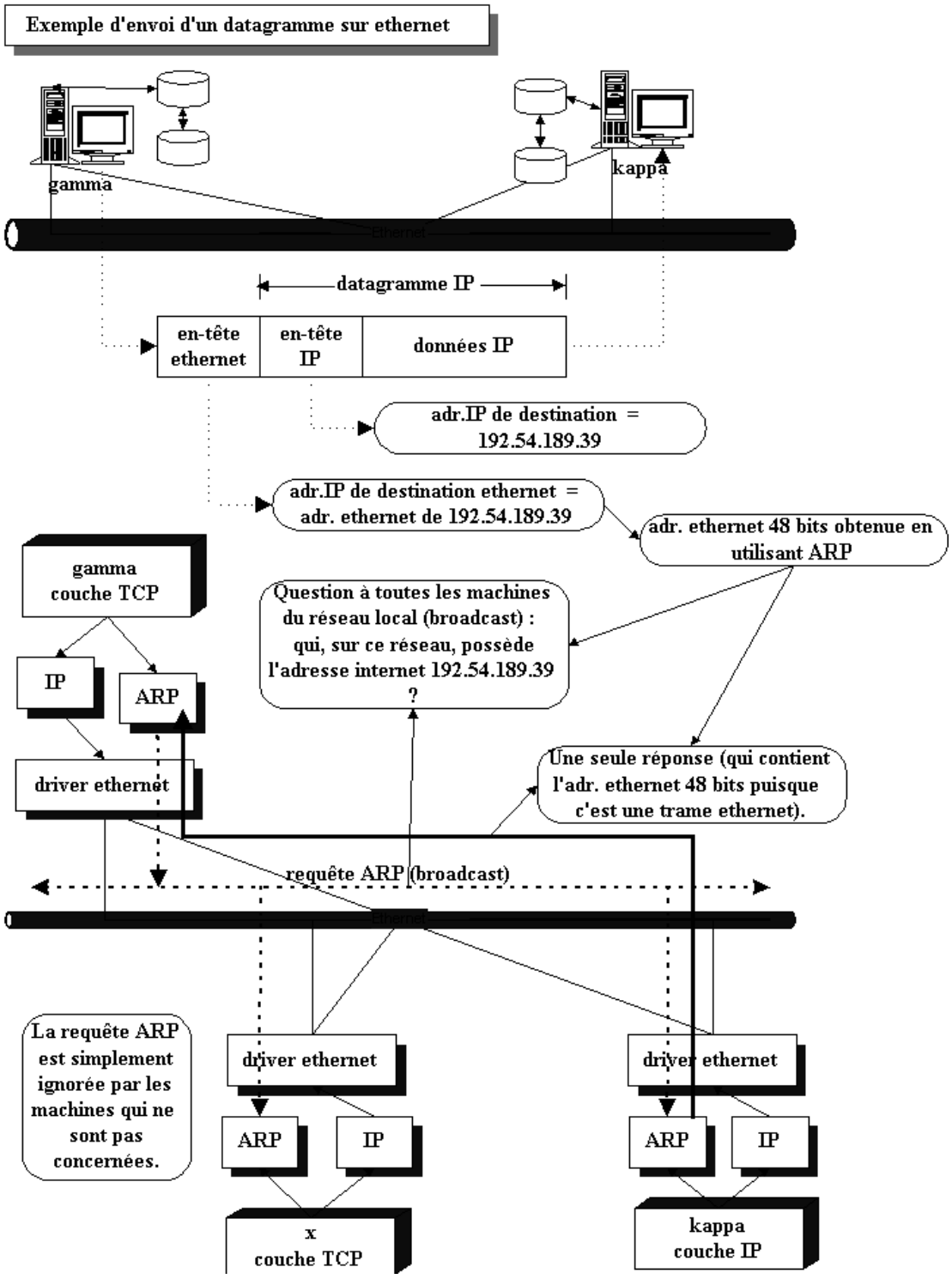


FIG. 63: Envoi d'un datagramme sur ethernet

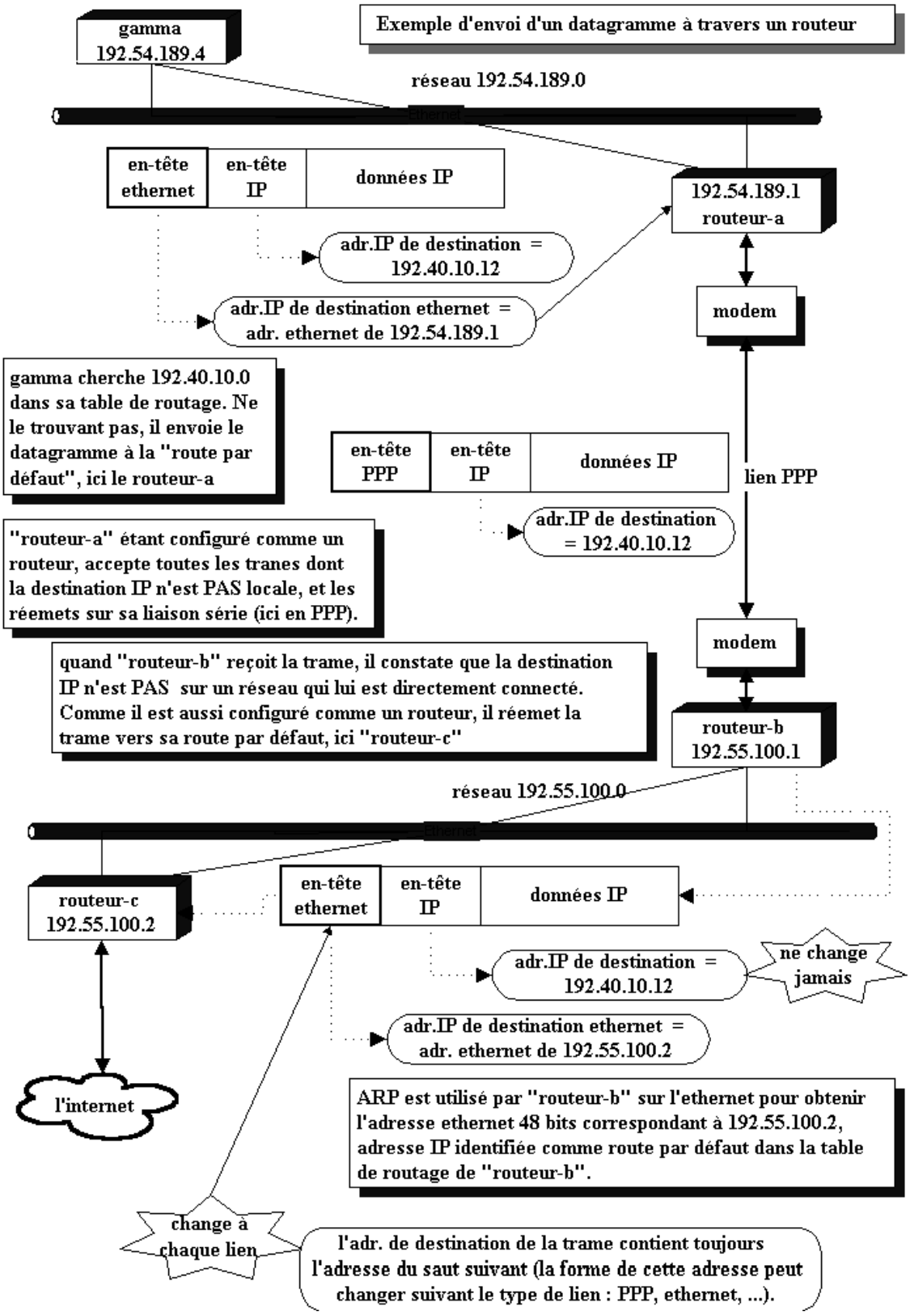


FIG. 64: Envoi d'un datagramme à travers un routeur

IP délivre un service **non fiable, sans connexion**.

– **non fiable :** pas de garantie qu'un datagramme IP atteigne sa destination avec succès. IP fournit un service "de moindre effort". Toute la fiabilité requise doit être assurée par les couches supérieures (e.g. TCP).

– **sans connexion :** IP ne maintient aucune information d'état concernant les datagrammes successifs. Chacun est géré indépendamment des autres. => ils peuvent être délivrés en désordre (ils peuvent suivre des chemins différents dans le réseau).

Le routage IP se fait sur une base de saut à saut : IP ne connaît la route complète (de bout en bout) d'aucune destination. Chaque machine connaît seulement les destinations qui lui sont directement connectées. La table de routage d'une machine fournit seulement l'adresse IP du routeur de saut suivant, vers lequel le datagramme doit être envoyé.

Deux commandes unix permettent d'obtenir des informations sur l'état des interfaces réseau (**ifconfig**) et l'état des liens IP (TCP ou UDP) (**netstat**).

```
[mv@localhost mv]$ ifconfig
bash: ifconfig: command not found
[mv@localhost mv]$ su -
Password:
[root@localhost /root]# ifconfig
eth0   Link encap:Ethernet  HWaddr 00:00:B4:95:40:CF
       inet addr:192.168.130.5  Bcast:192.168.130.255  Mask:255.255.255.0
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:15806 errors:0 dropped:0 overruns:0 frame:1
       TX packets:16511 errors:0 dropped:0 overruns:0 carrier:0
       collisions:2 txqueuelen:100
       Interrupt:3 Base address:0x300

lo     Link encap:Local Loopback
       inet addr:127.0.0.1  Mask:255.0.0.0
       UP LOOPBACK RUNNING  MTU:3924  Metric:1
       RX packets:62 errors:0 dropped:0 overruns:0 frame:0
       TX packets:62 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:0

[root@localhost /root]#
```

```
277 lo33 b116serv:~> ifconfig -a
lo0: flags=849<UP,LOOPBACK,RUNNING,MULTICAST> mtu 8232
        inet 127.0.0.1 netmask ff000000
hme0: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
        inet 172.19.2.90 netmask ffff0000 broadcast 172.19.255.255

273 lo33 b116serv:~> netstat -r     (sur solaris)

Routing Table:
  Destination           Gateway       Flags Ref  Use  Interface
-------------------- --------------- ----- --- ------ --------
r4.utc.fr              b116serv         U     4  30337  hme0
r4b.utc                b116serv         U     4 147828  hme0
BASE-ADDRESS.MCAST.net b116serv         U     4      0  hme0
default                c6k-msfc-bf.utc  UG    0 225581
localhost              localhost        UH    0  36687  lo0

[root@localhost /root]# netstat -r     (sur linux)
```

```
Kernel IP routing table
Destination     Gateway  Genmask       Flags  MSS Window irtt Iface
192.168.130.0   *        255.255.255.0  U       0 0          0 eth0
127.0.0.0       *        255.0.0.0      U       0 0          0 lo
default         brx      0.0.0.0        UG      0 0          0 eth0

[root@localhost /root]# netstat -t -p -a    (sur linux)
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (servers and established)
Proto Rcv-Q Snd-Q Local Addr  Foreign Addre State       PID/Prog.name
tcp        0      0 tn5:1238    sirius:telnet ESTABLISHED 26084/telnet
tcp        0      0 tn5:1221    brx:telnet    ESTABLISHED 25736/telnet
tcp        0      0 *:X         *:*           LISTEN      25413/X
tcp        0      0 *:587       *:*           LISTEN      629/sendmail:accep
tcp        0      0 *:smtp      *:*           LISTEN      629/sendmail:accep
tcp        0      0 *:printer   *:*           LISTEN      585/
tcp        0      0 *:ssh       *:*           LISTEN      564/sshd
tcp        0      0 *:login     *:*           LISTEN      491/
tcp        0      0 *:shell     *:*           LISTEN      491/
tcp        0      0 *:telnet    *:*           LISTEN      491/
tcp        0      0 *:finger    *:*           LISTEN      491/
tcp        0      0 *:auth      *:*           LISTEN      431/identd
tcp        0      0 *:1024      *:*           LISTEN      362/
tcp        0      0 *:sunrpc    *:*           LISTEN      335/

274 lo33 b116serv:~> netstat -P tcp   (sur solaris)

TCP                                     Send     Recv
 Local Address    Remote Address      Swind -Q Rwind -Q  State
---------------- -------------------- ----- -- ----- -- -------
b116serv.32784   b116serv.32780       32768  0 32768  0 ESTABLISHED
b116serv.32780   b116serv.32784       32768  0 32768  0 ESTABLISHED
b116serv.32778   lima.34301            8760  0  8760  0 ESTABLISHED
b116serv.55128   sigma.utc.fr.ftp     32736  0  8760  0 CLOSE_WAIT
b116serv.55148   sigma.utc.fr.ftp     32736  0  8760  0 CLOSE_WAIT
localhost.31418  localhost.37370      32768  0 32768  0 CLOSE_WAIT
b116serv.32778   lima.37423           8760  0  8760  0 ESTABLISHED
b116serv.telnet  satanas.gi.utc.57607 24820  0  8760  0 ESTABLISHED
b116serv.telnet  ncd10.utc.fr.3673     3911  0  9112  0 ESTABLISHED
b116serv.40228   xsme15.utc.6000       4096  0  9216  0 ESTABLISHED
```

Les commandes **ping** et **traceroute** permettent de tester la connectivité : si ping et traceroute indiquent que la connectivité est acquise et que les applications (telnet, mozilla) ne fonctionnent pas, il faut chercher la raison du côté du DNS (vérifier resolv.conf).

```
1001 vayssade kappa:~> ping vega
PING vega.utc.fr (195.83.156.40): 56 data bytes
64 bytes from 195.83.156.40: icmp_seq=0 ttl=63 time=1 ms
64 bytes from 195.83.156.40: icmp_seq=1 ttl=63 time=0 ms
```

```
64 bytes from 195.83.156.40: icmp_seq=2 ttl=63 time=0 ms

----vega.utc.fr PING Statistics----
3 packets transmitted, 3 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 0/0/1 ms

1004 vayssade kappa:~> ping -R vega
PING vega.utc.fr (195.83.156.40): 56 data bytes
64 bytes from 195.83.156.40: icmp_seq=0 ttl=63 time=1 ms
RR:   c6k-msfc-bf.utc.fr (195.83.156.1)
      vega.utc.fr (195.83.156.40)
      c6k-msfc-si.utc.fr (195.83.155.1)
      kappa.utc.fr (195.83.155.100)
64 bytes from 195.83.156.40: icmp_seq=1 ttl=63 time=0 ms (same route)

1006 vayssade kappa:~> traceroute vega
traceroute to vega.utc.fr (195.83.156.40), 30 hops max, 40 byte packets
 1  c6k-msfc-si (195.83.155.1)  1 ms  0 ms  1 ms
 2  vega (195.83.156.40)  1 ms  1 ms  0 ms

[root@localhost /root]# traceroute vega
traceroute to vega (195.83.156.40), 30 hops max, 38 byte packets
 1  brx (192.168.130.1)  3.515 ms  3.154 ms  2.997 ms
 2  brxutc (195.83.155.35)  45.499 ms  48.789 ms  46.050 ms
 3  195.83.155.1 (195.83.155.1)  51.317 ms  47.926 ms  50.609 ms
 4  vega (195.83.156.40)  49.084 ms  49.604 ms  49.719 ms
```

Page blanche

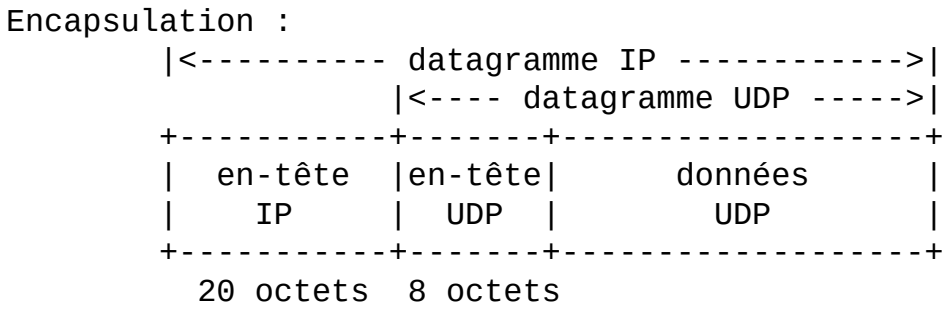
SR03 2004 - Cours Architectures Internet - Les protocoles UDP et TCP
©Michel.Vayssade@utc.fr – Université de Technologie de Compiègne.

7 SR03 2004 - Cours Architectures Internet - Les protocoles UDP et TCP

7.1 Le protocole UDP

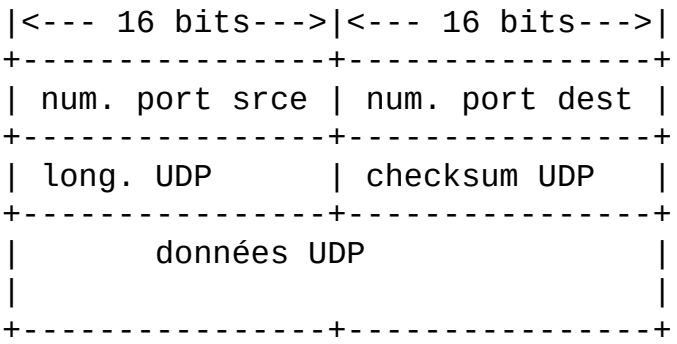
UDP est un protocole de transfert de messages (appelés datagrammes), sans garantie de fiabilité, ni de remise dans l'ordre d'envoi.

Un datagramme UDP est généré pour chaque opération (sendto) faite par un processus applicatif.



L'application doit se préoccuper de la taille des datagrammes qu'elle émet : si cette taille dépasse le MTU du réseau, le datagramme IP fait l'objet d'une fragmentation (vrai pour chaque réseau traversé => MTU du chemin).

En-tête UDP :



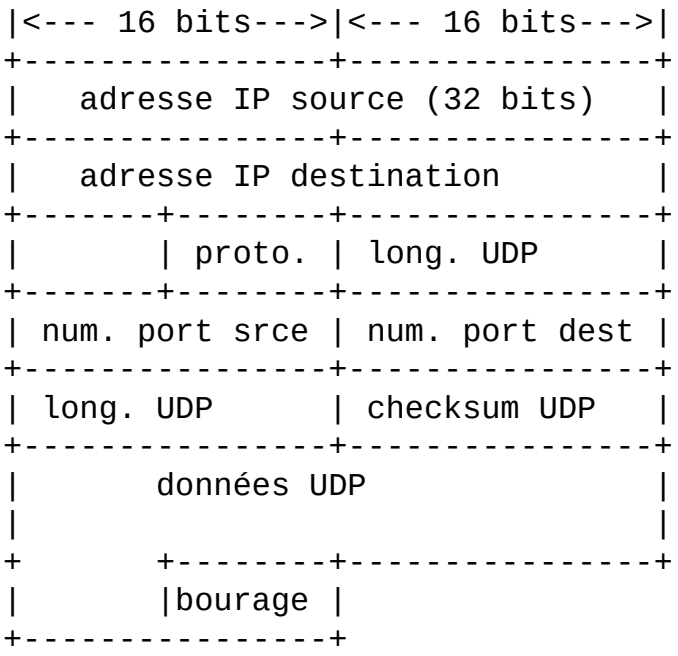
Les numéros de ports identifient le processus émetteur et le processus destinataire.
"long UDP" = longueur en-tête + long. data

Somme de contrôle UDP (checksum)

Rem : l'en-tête UDP ne contient pas l'adr. IP des machines srces et dest. Cette info. doit être récupérée dans le datagramme IP englobant.

Pour calculer la somme de contrôle (somme des compléments à 1 de mots de 16 bits), UDP (mais aussi TCP) ajoutent au datagramme un **pseudo-en-tête** de 12 octets. Cet en-tête ne fait pas partie du voyage : il est généré au départ et regénéré à l'arrivée et ne sert qu'au calcul de la somme de contrôle.

Le pseudo en-tête contient entre autre, l'adr.IP de la machine émettrice et l'adr.IP de la machine destinataire. En vérifiant la somme de contrôle, on vérifie ainsi que le datagramme est parvenu sur la bonne machine.



Si une erreur est détectée dans le calcul de la somme de contrôle, le datagramme est détruit en silence, sans émission de message d'erreur.

Fragmentation IP :

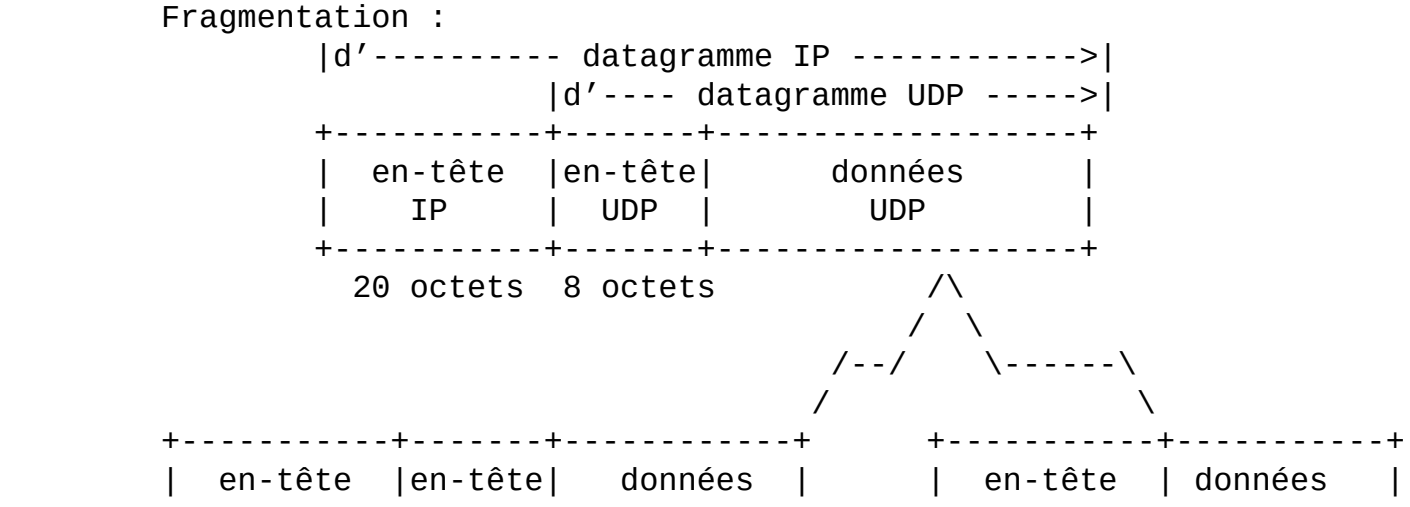
Si un datagramme a une longueur plus grande que le MTU de l'un des réseaux traversés, la ccuche IP fragmente le datagramme avant de l'émettre.

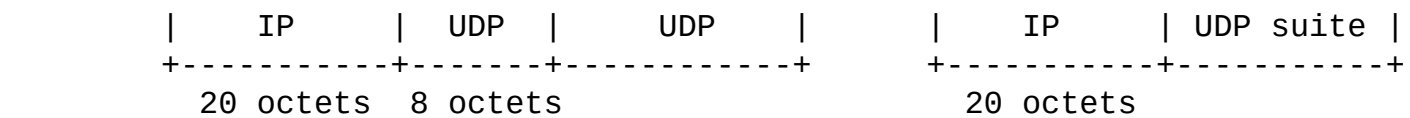
Quand un datagramme est fragmenté, il n'est réassemblé **que** sur la machine de destination finale. C'est la couche IP qui réassemble. Ainsi la fragmentation reste invisible à UDP ou à TCP.

L'en-tête contient un champ **identification** qui contient une valeur unique pour chaque datagramme IP envoyé par l'émetteur. Ce champ est **copié** dans chaque fragment.

Les fragments, sauf le dernier, ont dans le champ **flags** le bit **more fragments** mis à 1.

La façon dont les fragments sont construits fait que si **un** fragment est perdu, tout le datagramme doit être réémis.





Conception d'un serveur UDP

La conception et l'implémentation des clients est généralement plus facile que celle des serveurs : typiquement, un client démarre, envoie un datagramme à un serveur, lit la réponse et se termine.

Un serveur, lui, démarre, se met en sommeil en attendant une requête d'un client, répondent à **ce** client, puis se remettent en sommeil.

Le serveur doit demander au système d'exploitation l'origine du message (adr.IP source et numéro de port source).

Souvent les serveurs UDP sont des serveurs **itératifs**. Il n'y a donc qu'un seul process serveur pour supporter les requêtes de tous les clients qui arrivent, toutes sur le même port UDP (celui du serveur).

Pendant le traitement d'une requête, s'il en arrive une autre, elle doit être mise en attente. Il y a une limite à la longueur de cette file d'attente. Si la file déborde, les datagrammes arrivants sont rejetés par la couche UDP de la machine du serveur. Les clients ne sont pas prévenus de ce rejet. Le serveur n'est pas prévenu que sa file d'attente est surchargée.

7.2 Le protocole TCP

Services de TCP et en-tête TCP

TCP est un protocole de transfert de flus d'octets, <str>avec</str> garantie de fiabilité, et de remise des octets dans l'ordre d'envoi, sans duplication.

```
Encapsulation :
        |<---------- datagramme IP -------------->|
                    |<-------- segment TCP ------>|
        +-----------+---------+-------------------+
        |  en-tête  | en-tête |      données      |
        |    IP     |   TCP   |        TCP        |
        +-----------+---------+-------------------+
          20 octets  20 octets
```

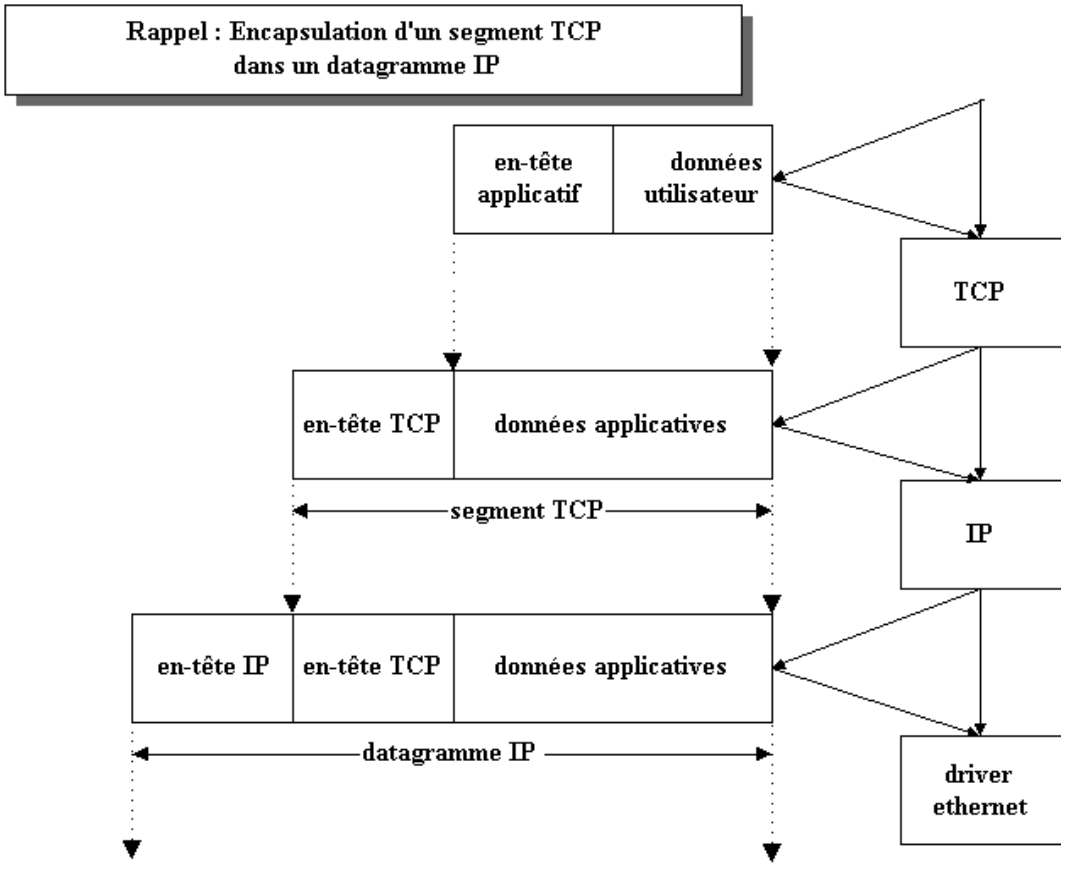


FIG. 66: Encapsulation TCP

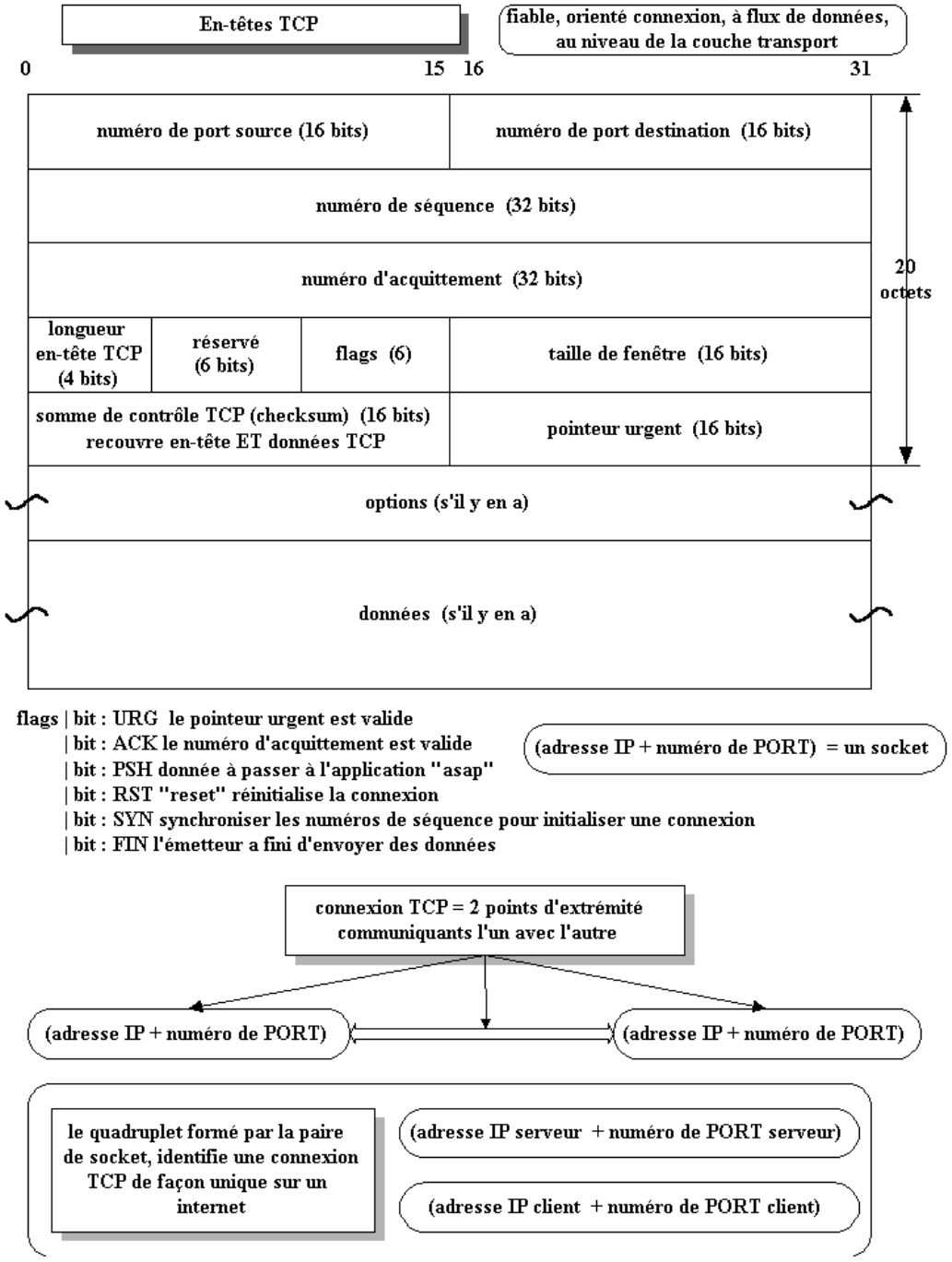


FIG. 67: Entêtes TCP

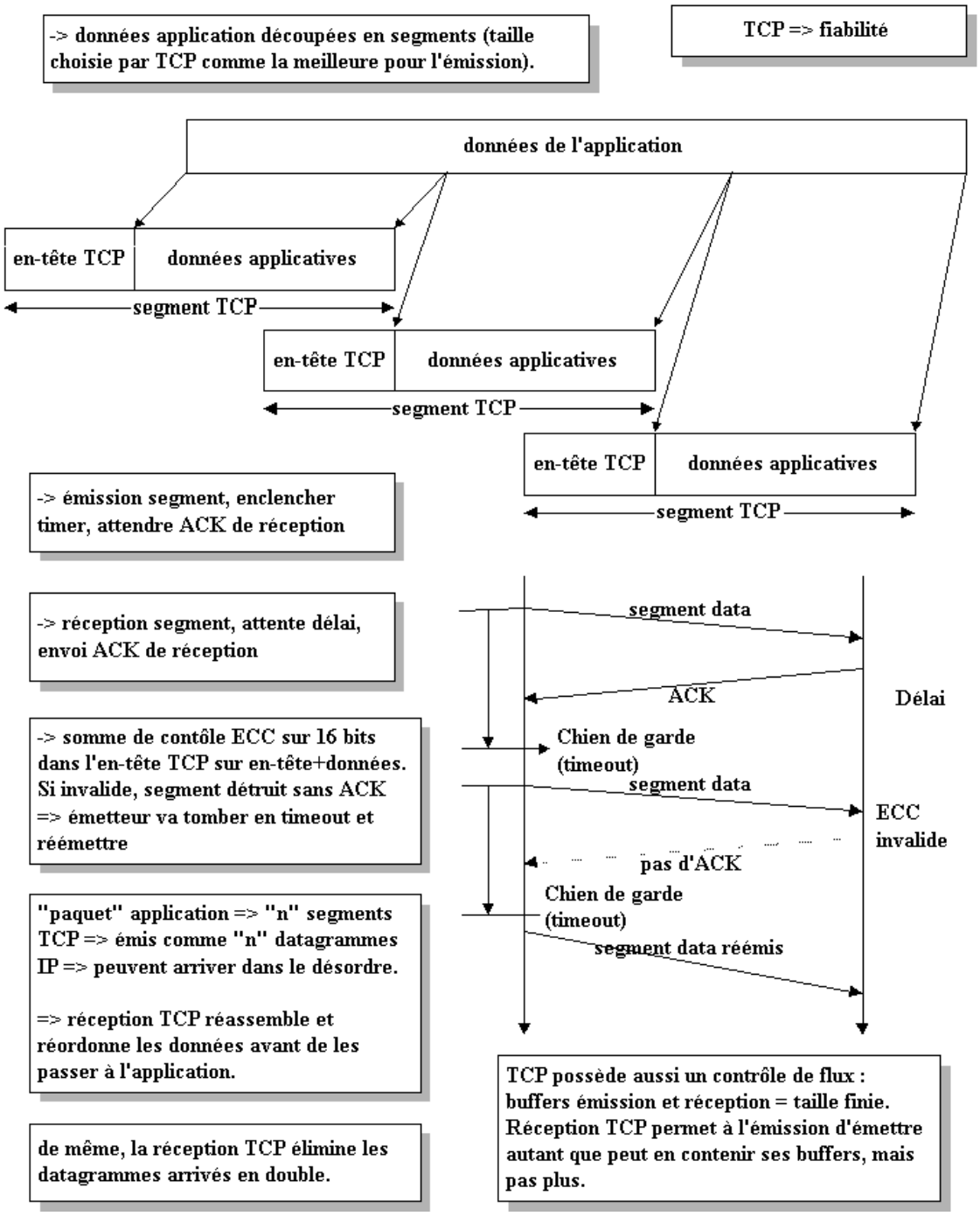


FIG. 68: TCP : fiabilité

TCP fournit un **service orienté connexion** et apporte la fiabilité par les actions suivantes :
- les données application sont fragmentées en **segments** dont la longueur est celle jugée la meilleure par TCP ;
- à chaque segment émis, TCP maintient un délai de garde ("timer") en attendant l'arrivée d'un acquittement de la part du destinataire du segment ;
- à la réception de données, un acquittement est émis ;
- à la réception TCP réordonne les données si nécessaire (elles ont voyagé dans des datagrammes IP) ;
- TCP maintient dans chaque segment une somme de contrôle d'extrémité à extrémité ;
- TCP rejette les données dupliquées ;
- TCP fournit un contrôle de flux pour ne pas surcharger un destinataire.

TCP fournit un **flux d'octets** (byte stream service) indifférenciés sans marqueurs de fin d'enregistrement (c'est éventuellement à l'application de les mettre).

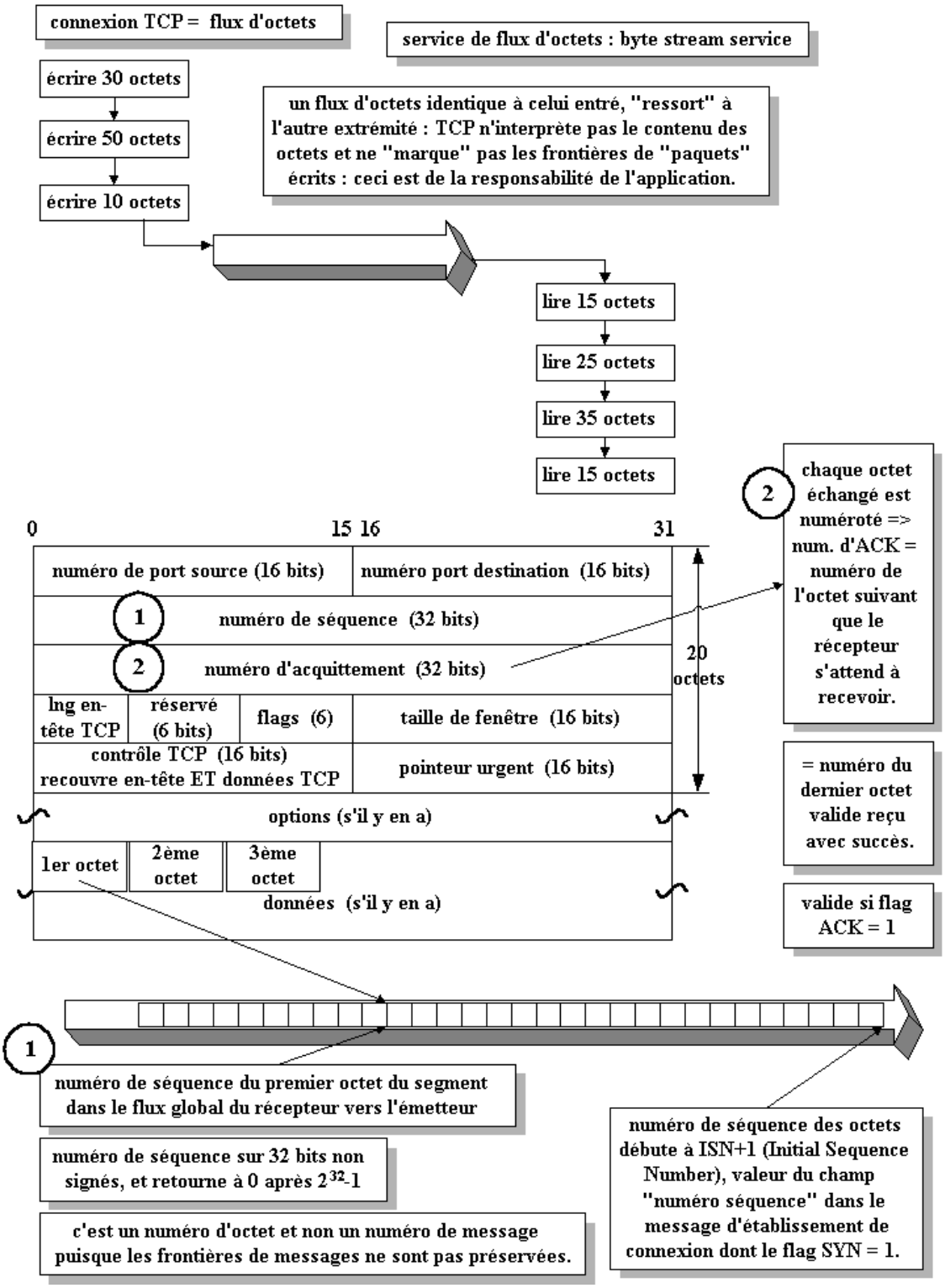


FIG. 69: TCP => flux d'octets 1/3

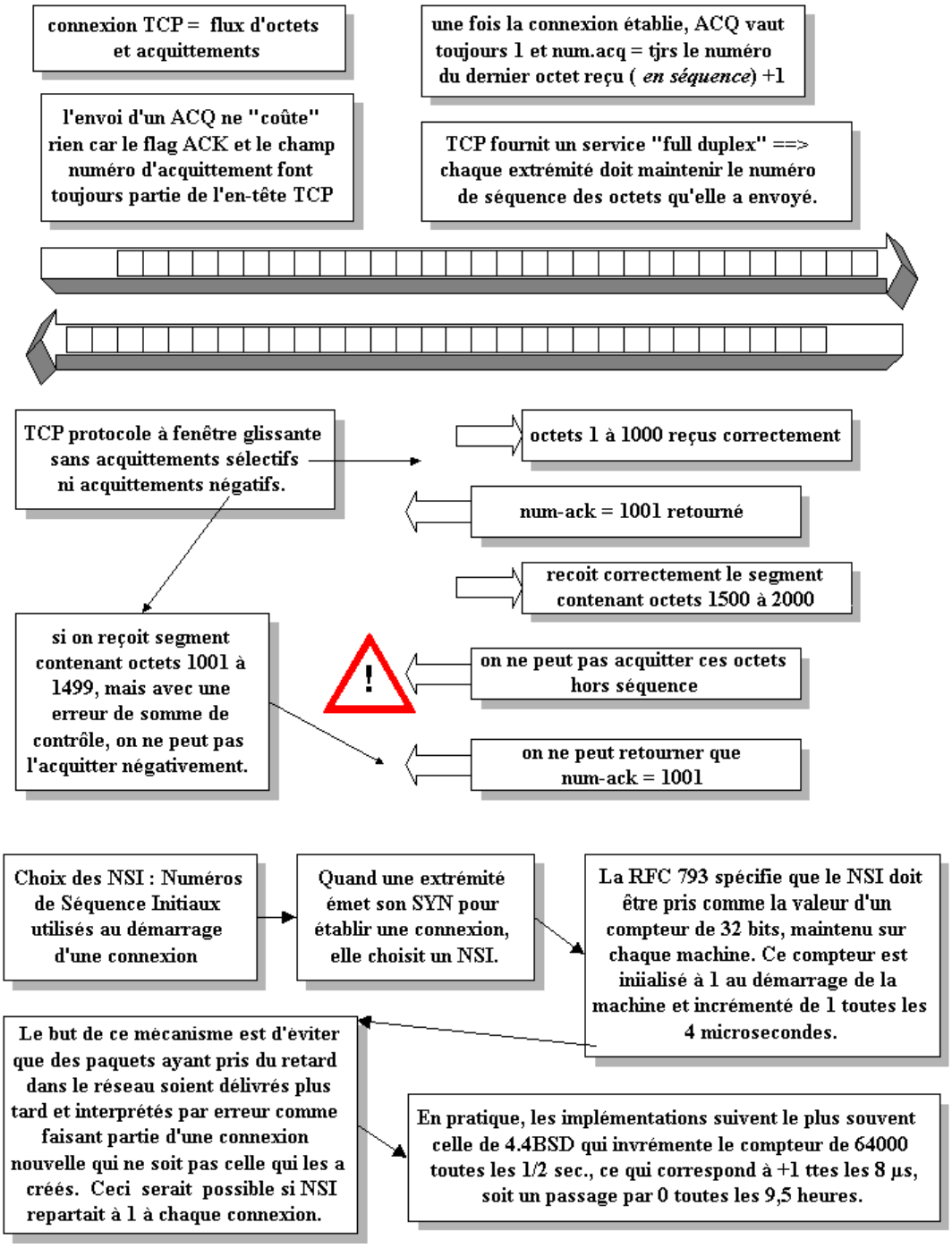


FIG. 70: TCP => flux d'octets 2/3

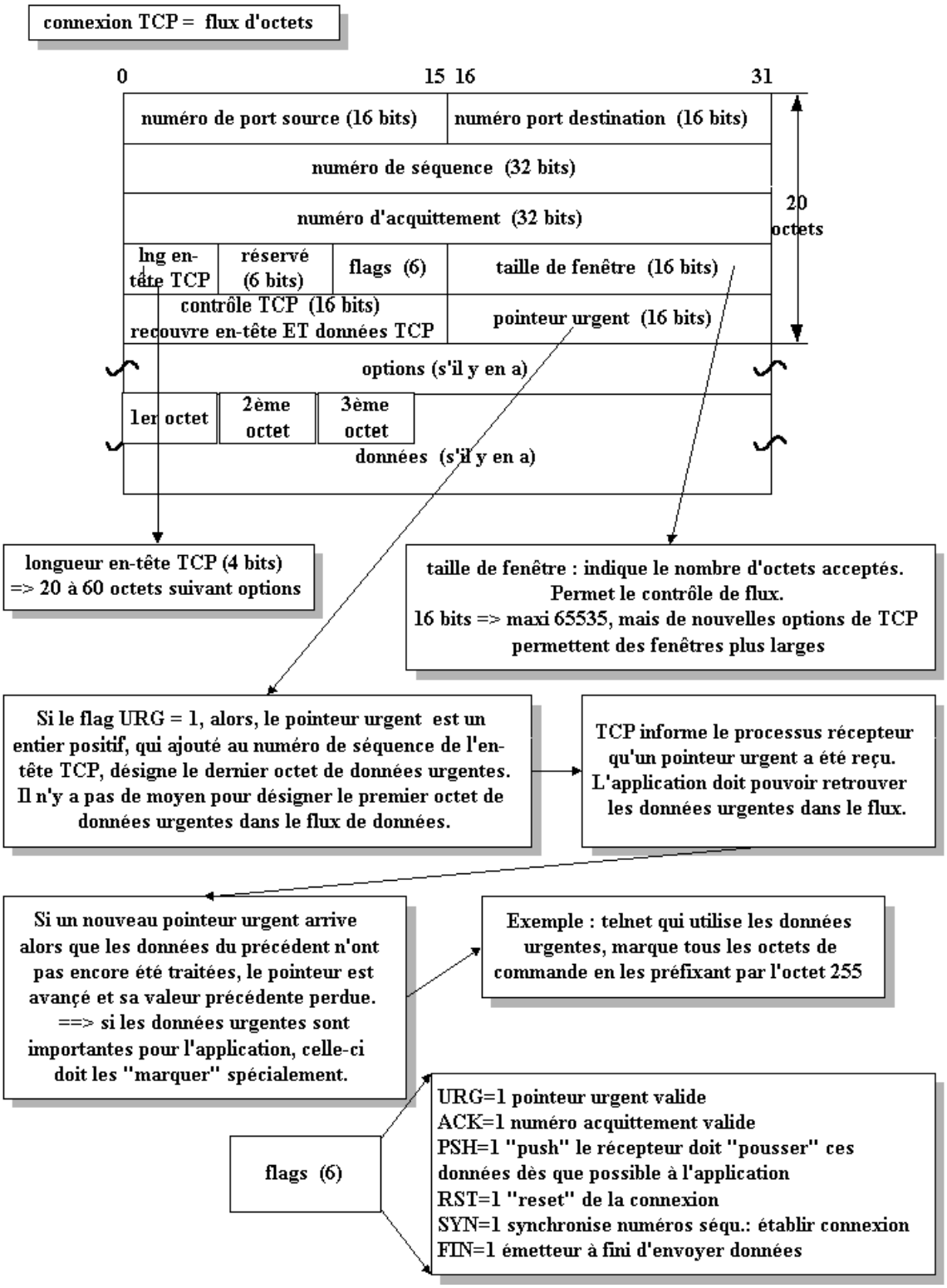


FIG. 71: TCP => flux d'octets 3/3

7.3 TCP messages urgents (bit URG)

envoi par : **send (descript_sock, buf, 1, MSG_OOB) ;** (OOB = Out_Of_Band).

Attention : cette appellation "hors bande" de l'API des sockets est **incorrecte** : de vraies données hors bande devraient passer par un canal logique séparé (le plus simple serait une seconde connexion TCP)

Si le message envoyé contient plusieurs caractères, seul le dernier est désigné par le pointeur urgent.

Le récepteur doit demander la notification par :
ioctl (descript_sock, SIOCSPGRP, &pid) ;
il recevra alors un signal SIGURG.

Le caractère urgent est inséré dans le buffer de lecture du récepteur mais il n'est pas accessible (par défaut) par la lecture normale :
read (descript_sock, buf, 1024) ;
va butter sur le caractère urgent (s'il y a 500 car. avant le car.urg, ce read lit 500 car ;)

Il faut lire le car.urg par :
read (descript_sock, buf, 1, MSG_OOB) ;

On peut tester si la position de lecture est parvenue au car. urgent par :
ioctl (descript_sock, SIOCATMARK,0) ;
qui renvoie -1 si oui.

Attention : il y a eu des interprétation différentes de la première RFC sur le sujet. Normalement, cela a été précisé dans une autre RFC (Host requirements) qui indique que le pointeur doit désigner le **dernier octet urgent**, MAIS, certaines implémentations héritées de BSD font pointer le pointeur sur l'octet **QUI SUIT** le dernier octet urgent.

Une implémentation correcte pourrait ne pas communiquer correctement avec ces implantations erronées mais répandues.

Socket et signaux

1. **SIGIO** données à lire sur un descripteur
2. **SIGURG** car. urgent arrivé
3. **SIGPIPE**n essaie d'écrire sur un descripteur fermé en écriture ou n'ayant plus de lecteur

Les primitives d'E/S bloquantes sont interrompues par l'arrivée d'un signal. Après traitement du signal, elles ressortent de l'appel système avec le **code statut -1 et errno=EINTR**.

Ceci concerne : read, write, accept, open, fcntl, wait, connect, semop, msgsnd, msgrcv, sendto, recvfrom, send, recv, ...

7.4 Établissement et terminaison d'une connexion TCP

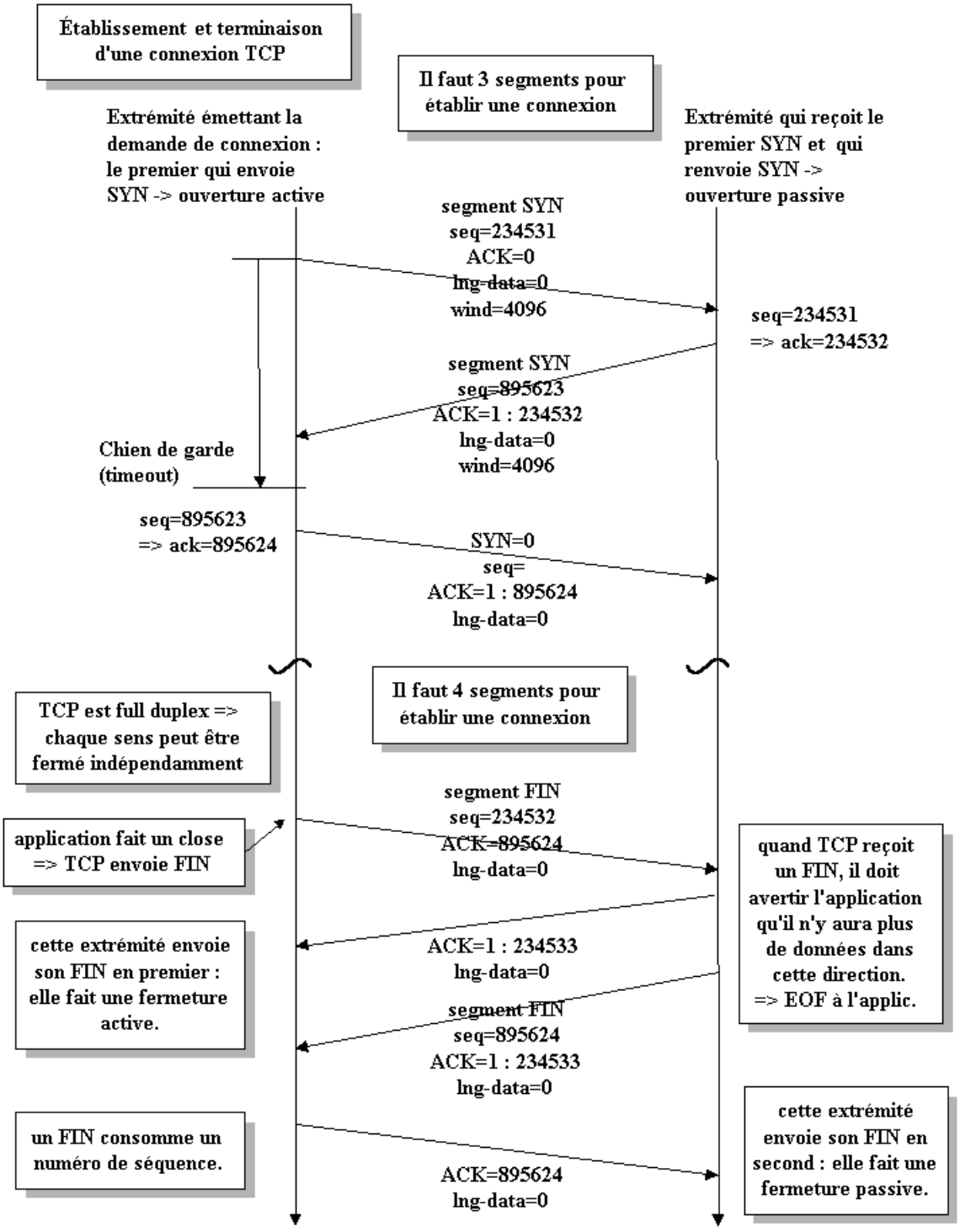


FIG. 72: TCP : établissement et terminaison d'une connexion

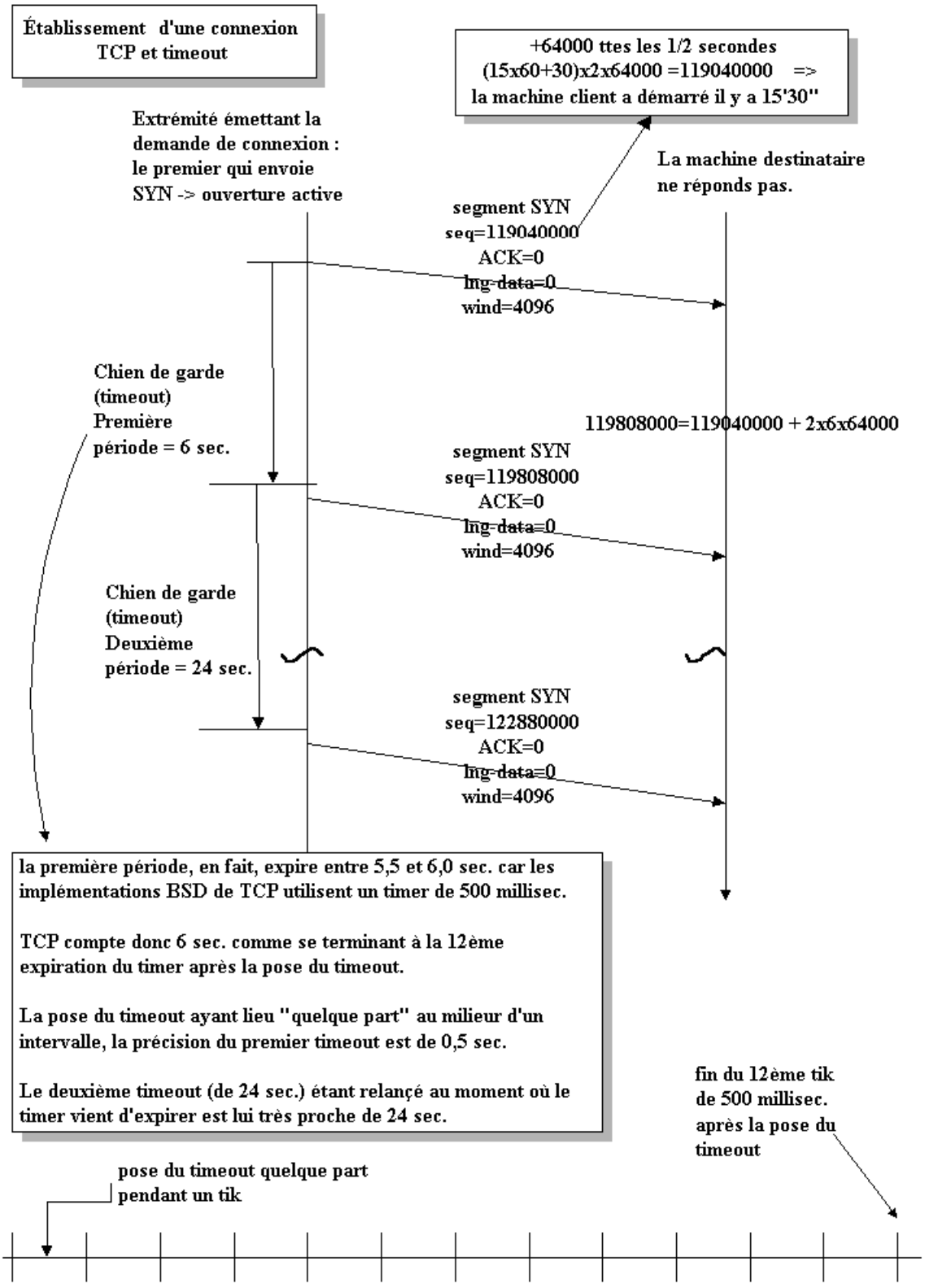


FIG. 73: TCP : établissement d'une connexion et time-out

TCP : MSS et MTU
taille maximum de segment

MSS : maximum segment size
MTU : maximum transfert unit

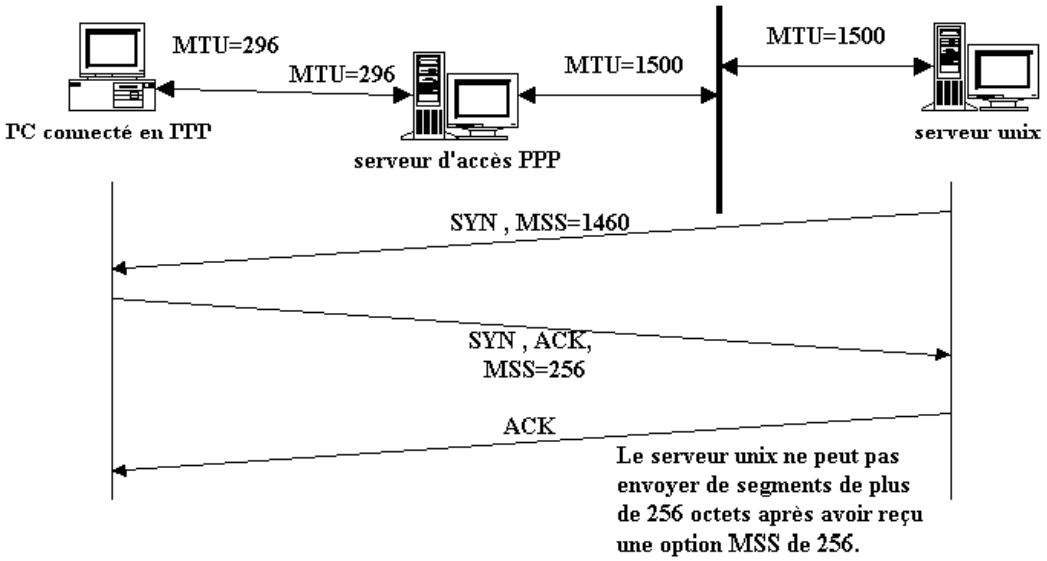
MSS : maximum segment size
= le + grand morceau de
données que TCP enverra vers
l'autre extrémité.

datagramme = MSS +20 pour en-tête TCP +20 pour en-tête IP
Si pas de MSS annonçée dans les segments SYN => val. par
défaut = 536 => datagramme de 576 octets

lors de l'établissement d'une connexion (envoi du segment SYN)
TCP envoie une valeur de MSS <= au MTU de l'interface sortante.

Sur un ethernet : MTU = 1500 => MSS = 1460. C'est ce que font
certaines implémentations (ex. SunOS 4.1.3, Solaris 2.2).
Certaines implémentations annoncent MSS=1024 pour que les
données soient un multiple de 512 (ex. BSD/386, SVR4).

Le MSS permet à une machine
d'éviter la fragmentation sur les
connexions qui empruntent des
liaisons avec un petit MTU.



Une machine ne peut choisir un MSS inférieur à la valeur par
défaut que si son interface est connectée à une liaison ayant un
MTU de moins de 576. Si 2 machines sont connectées sur un
ethernet et que leur connexion traverse une liaison dont le MTU
est de 296 (256+20+20), la fragmentation se produira. SAUF à
utiliser le mécanisme de découverte de MTU du chemin.

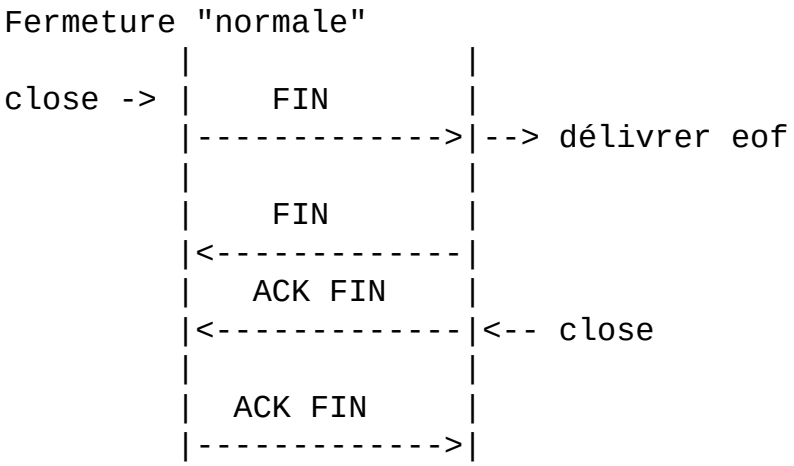
FIG. 74: TCP : MSS et MTU

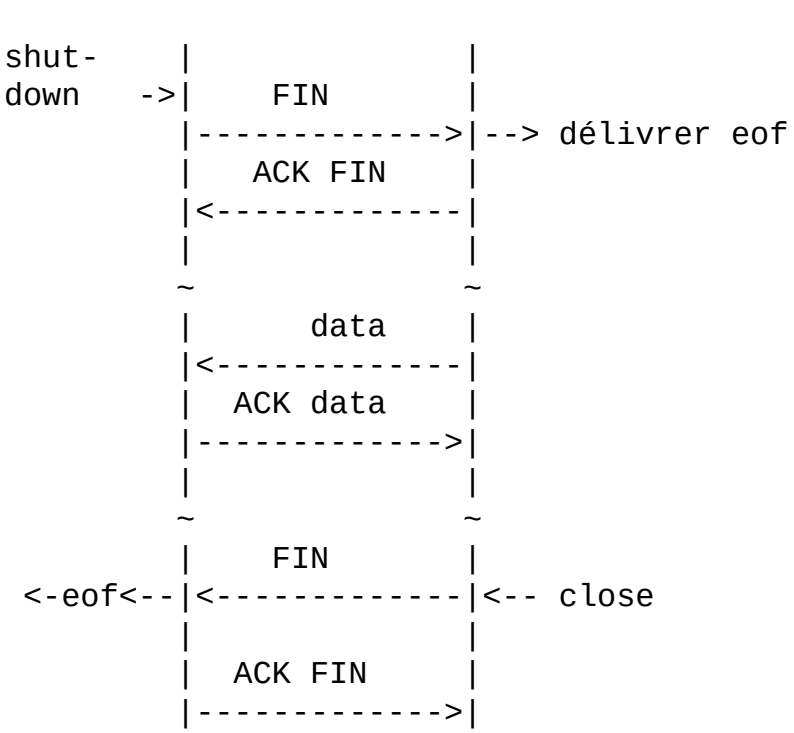
7.5 Semi-fermeture de TCP

TCP permet à une extrémité de fermer sa sortie (envoyer le segment FIN) tout en continuant à recevoir des données sur son entrées (jusqu'à l'arrivée du segment FIN).

L'API des sockets BSD supporte cette fonction par l"appel système **shutdown** avec **1** en deuxième argument (alors que l'appel de **close** ferme les deux sens de la connexion).

Cette possibilité est appelée **semi-fermeture** de TCP.





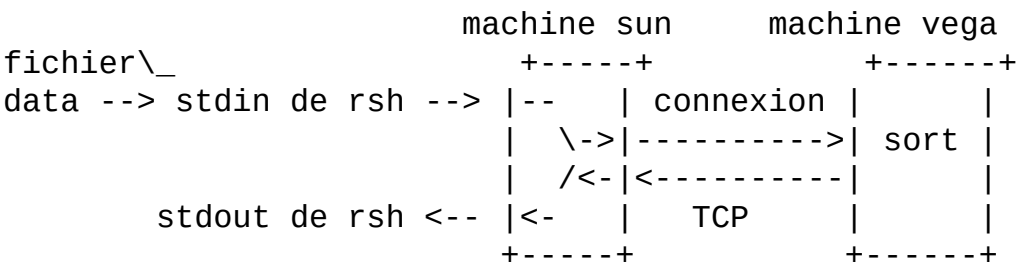
À quoi sert la Semi-fermeture de TCP ?

Exemple d'une commande **rsh** sur unix :

sun> **rsh vega sort < fichier_data**

la commande sort est exécutée sur vega avec comme données d'entrée le contenu du fichier "fichier_data" **situé sur sun**.

rsh crée une connexion TCP entre lui-même et le programme exécuté sur l'autre machine :



ensuite rsh copie l'entrée standard sur la connexion, et la sortie de la connexion sur sa sortie standard.

Le programme **sort** ne peut générer aucune sortie tant qu'il n'a pas reçu **toutes** les données à trier ! Comment sait-il qu'il a reçu toutes les données ? Seulement quand il reçoit une fin de fichier (eof) !

Donc rsh doit utiliser la semi-fermeture pour faire parvenir cet **eof** à sort qui peut alors commençer à trier et ensuite envoyer le résultat sur sa sortie standard qui peut encore accepter des données car elle n'est pas fermée (semi-fermeture).

Sans la semi-fermeture il aurait fallut utiliser une autre technique pour faire parvenir l'indication de fin de fichier.

7.6 État d'attente 2MSL

Après la fermeture de la connexion, TCP pase dans l'état **TIME_WAIT** aussi appelé état d'attente 2MSL (2MSL timeout).

MSL = Maximum Segment Lifetime : durée de vie maximum d'un segment

Les segments TCP sont transmis dans des datagrammes IP qui possèdent un champ TTL (Time To Live) après lequel ils sont silencieusement détruits.

Ceci pour éviter que des datagrammes tournent à l'infini dans le réseau suite à des problèmes de routage.

La RFC 793 spécifie MSL à 2 minutes mais les implémentations les plus courantes sont de 30", 1 minute ou 2 minutes selon les cas.

Le but de cette attente de 2 x MSL est de permettre à TCP de renvoyer l'ACK de FIN au cas ou celui-ci se serait perdu et que l'autre extrémité ne l'ayant pas reçu et étant tombée en timeout ait renvoyé son segment FIN.

******* un effet de bord de cette attente est que la paire de socket (IP serv, PORT serv, IP client, PORT cli) ne peut pas être réutilisée immédiatement.

C'est l'extrémité qui fait la fermeture active qui passe dans l'état TIME_WAIT. Le plus souvent c'est le client. Si on relance aussitôt le client, il ne peut pas réutiliser le même port. Ce n'est pas grave puisque le client utilise le plus souvent un port éphémère attribué dynamiquement par le système.

Par contre si on arrête brutalement le serveur, en provoquant donc une fermeture active, et si on relance aussitôt le serveur, on obtient l'erreur "**address already in use**".

Il n'y a pas d'autre solution alors que d'attendre la fin du TIME_WAIT.

7.7 Conception d'un serveur TCP

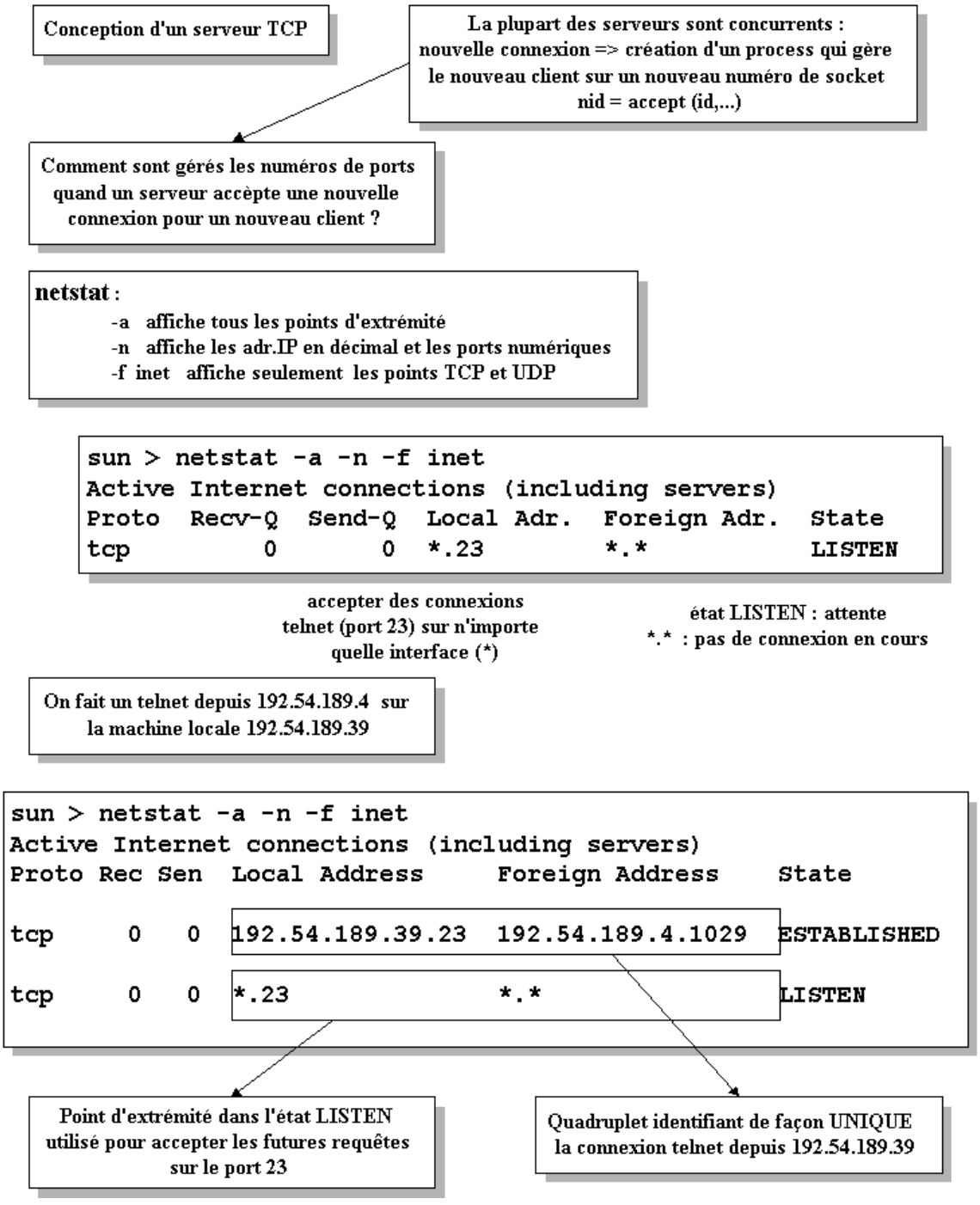


FIG. 75: Conception d'un serveur TCP 1/2

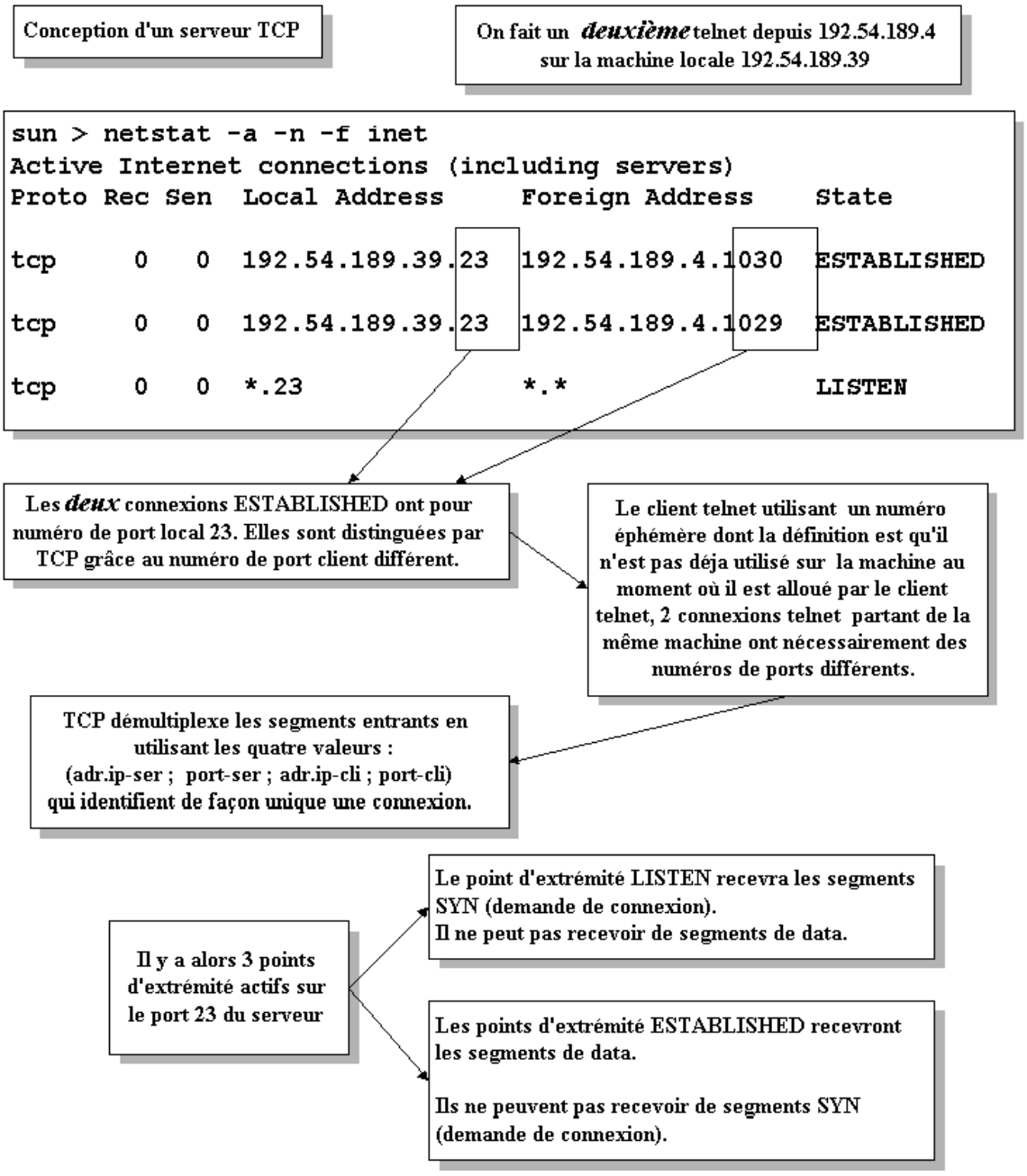


FIG. 76: Conception d'un serveur TCP 2/2

7.8 TCP : transfert de données

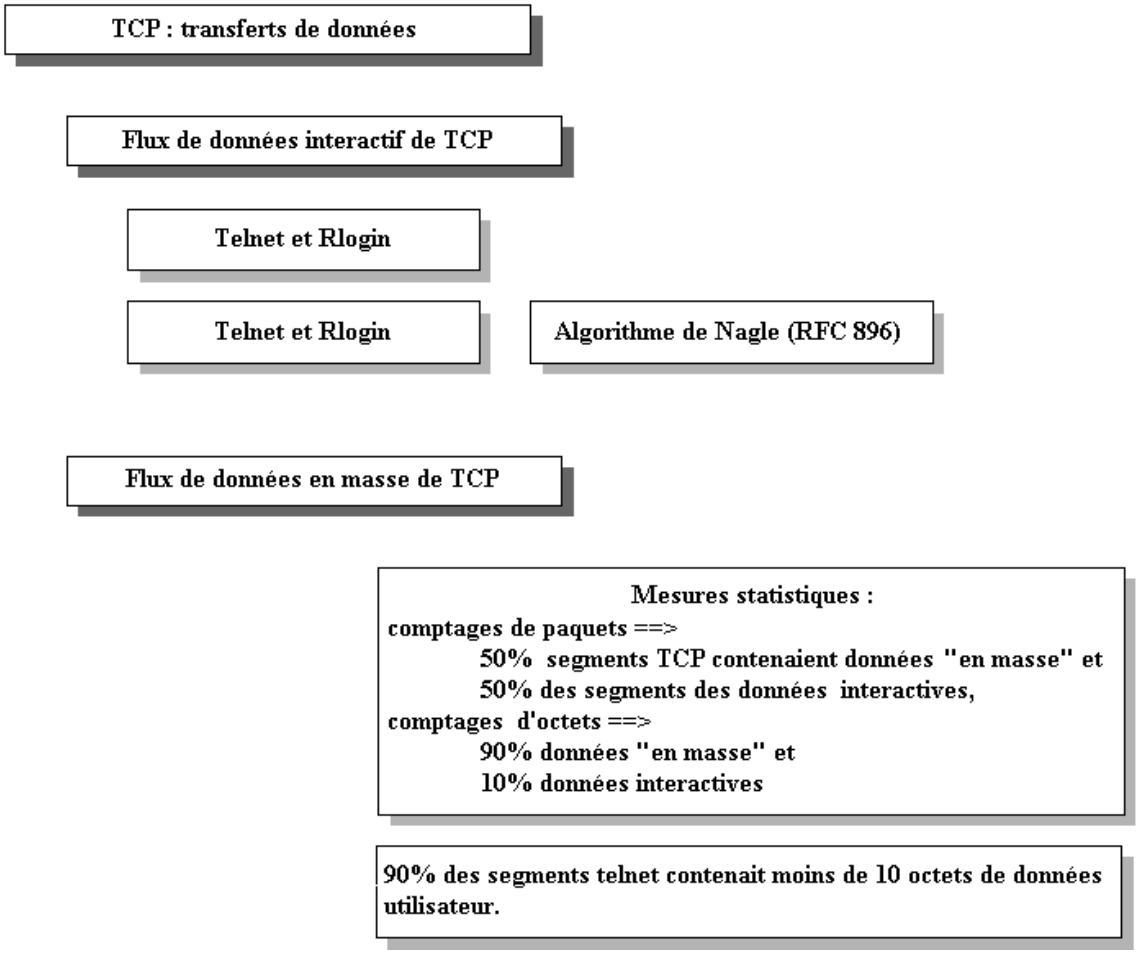


FIG. 77: TCP : transfert de donnée

7.9 TCP : flux de données interactif

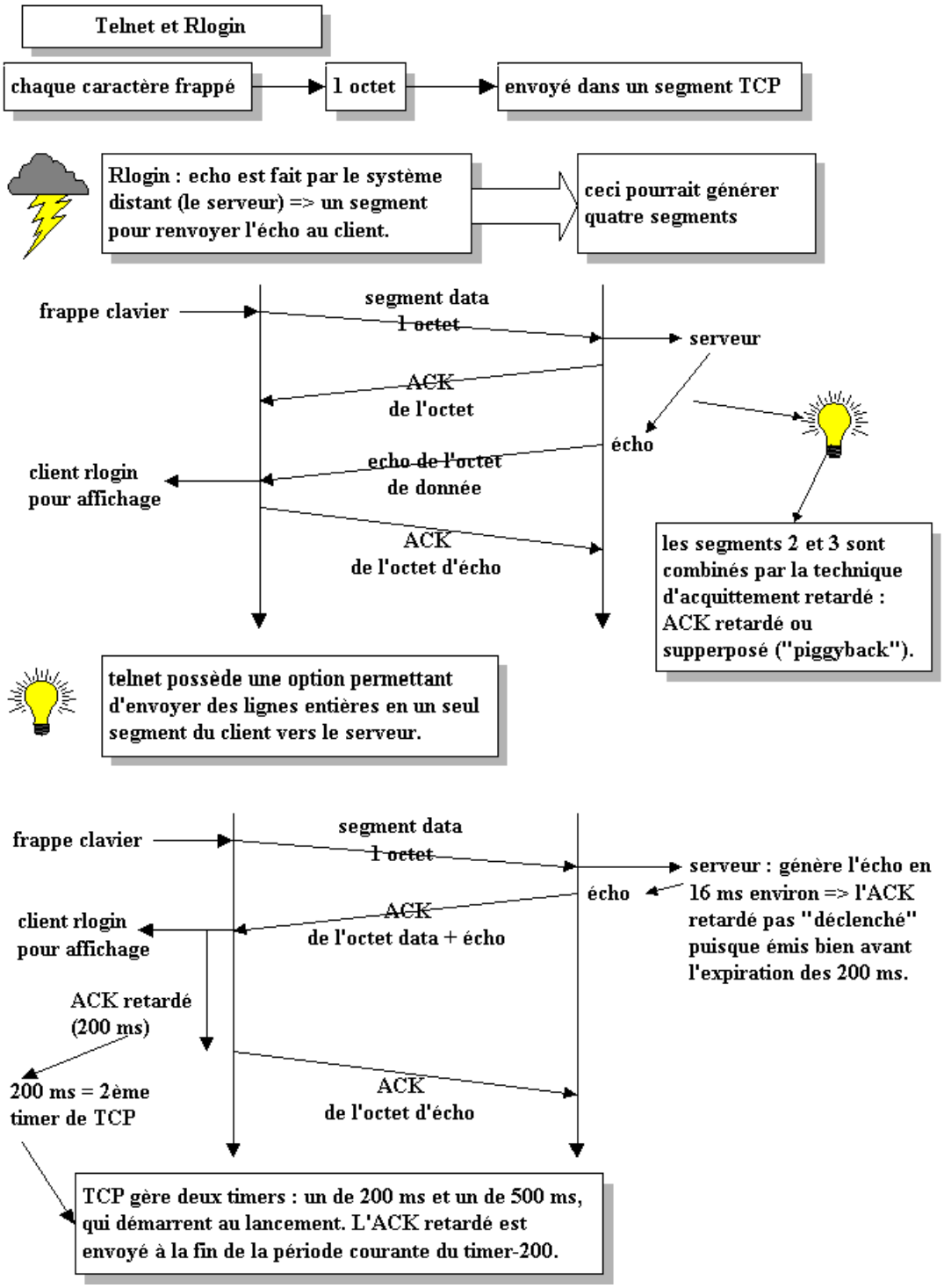


FIG. 78: TCP : Telnet et rlogin

Telnet et Rlogin

Algorithme de Nagle (RFC 896)

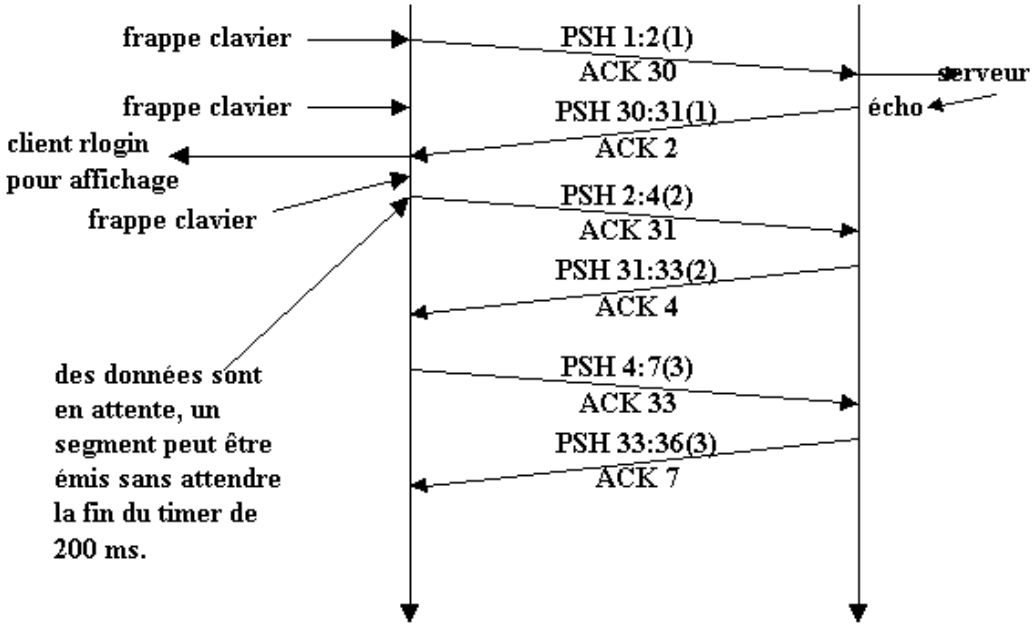
Problème : dans rlogin, 1 octet à la fois circule du client vers le serveur. Ceci génère des datagrammes de 41 octets (20 en-tête TCP + 20 en-tête IP + 1 Data) pour 1 utile.

Très nombreux "tinnygrams" génants sur les réseauc longue distance (WAN).

Algorithme de Nagle (RFC 896) : une connexion TCP ne peut avoir qu'un petit segment qui n'a pas encore été acquitté. Pas de petit segment envoyé tant que ACK pas reçu. Au lieu de cela, TCP collecte les données pour les envoyer en un seul segment.
Algo. élégant, car autorégulé : si les ACK reviennent vite, les données sont envoyées vite par petits paquets. Si les ACK tardent (congestion) les données sont envoyées par paquets plus gros (diminution de la charge).

Sur un ethernet, avec un rlogin, le temps moyen de retour des échos+ACK est de moins de 20 ms => il faudrait frapper au clavier à plus de 50 caractères par seconde pour déclencher l'algorithme de Nagle.

Sur un WAN par contre, le temps d'aller-retour est bien plus grand.



L'option TCP_NODELAY de l'API des sockets, permet de dévalider l'algorithme de Nagle dans certaines applications interactives (ex. X11) (problème avec les touches de fonction générant des séquences <escape> si une séquence est "tronquée" par TCP, le serveur devra attendre la fin de la séquence pour renvoyer l'écho, donc la fin du timeout de 200ms).

FIG. 79: TCP : Telnet et rlogin - Algorithme de Nagle

7.10 TCP : flux de données en masse

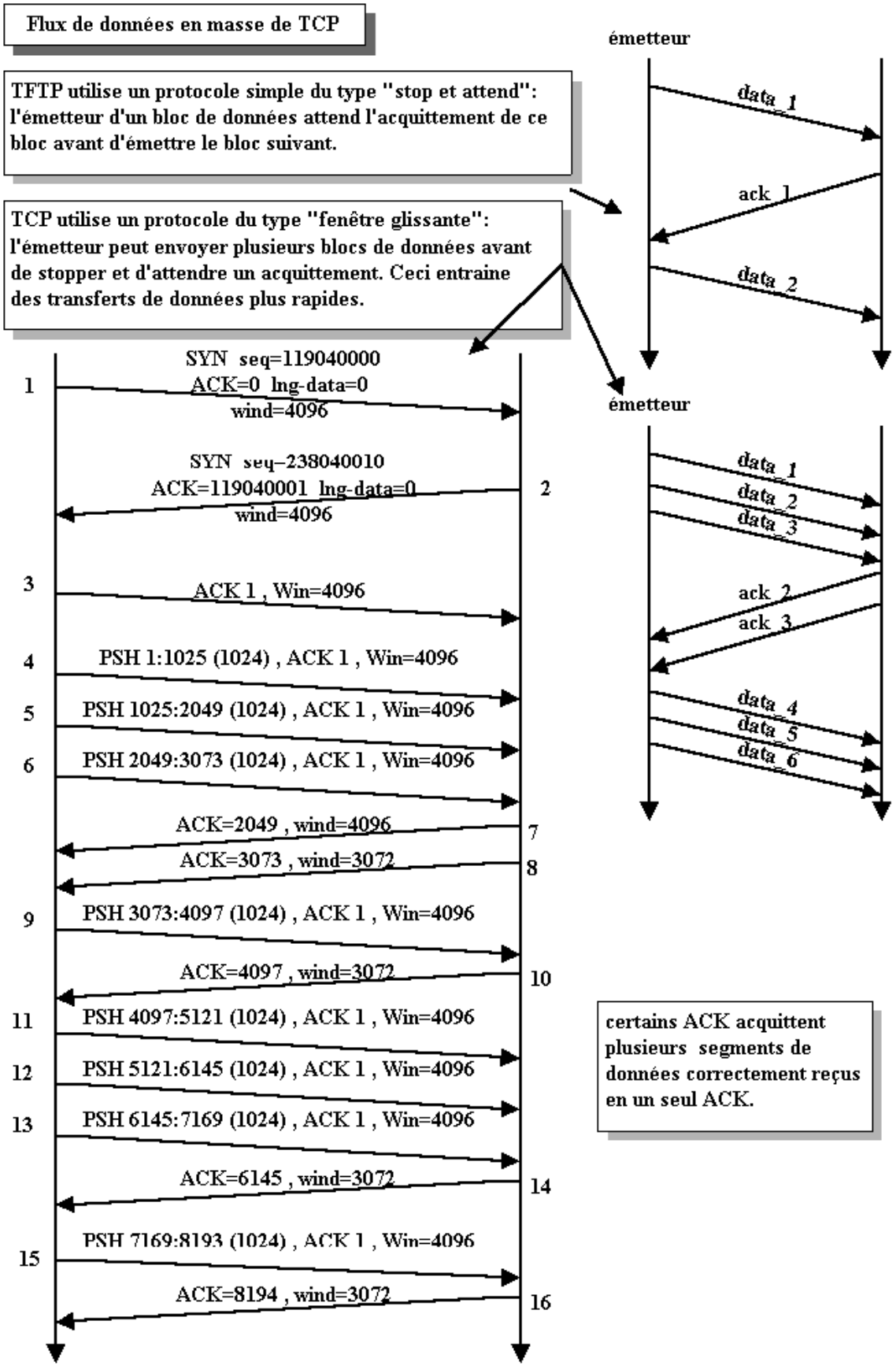


FIG. 80: Flux de données en masse de TCP 1/2

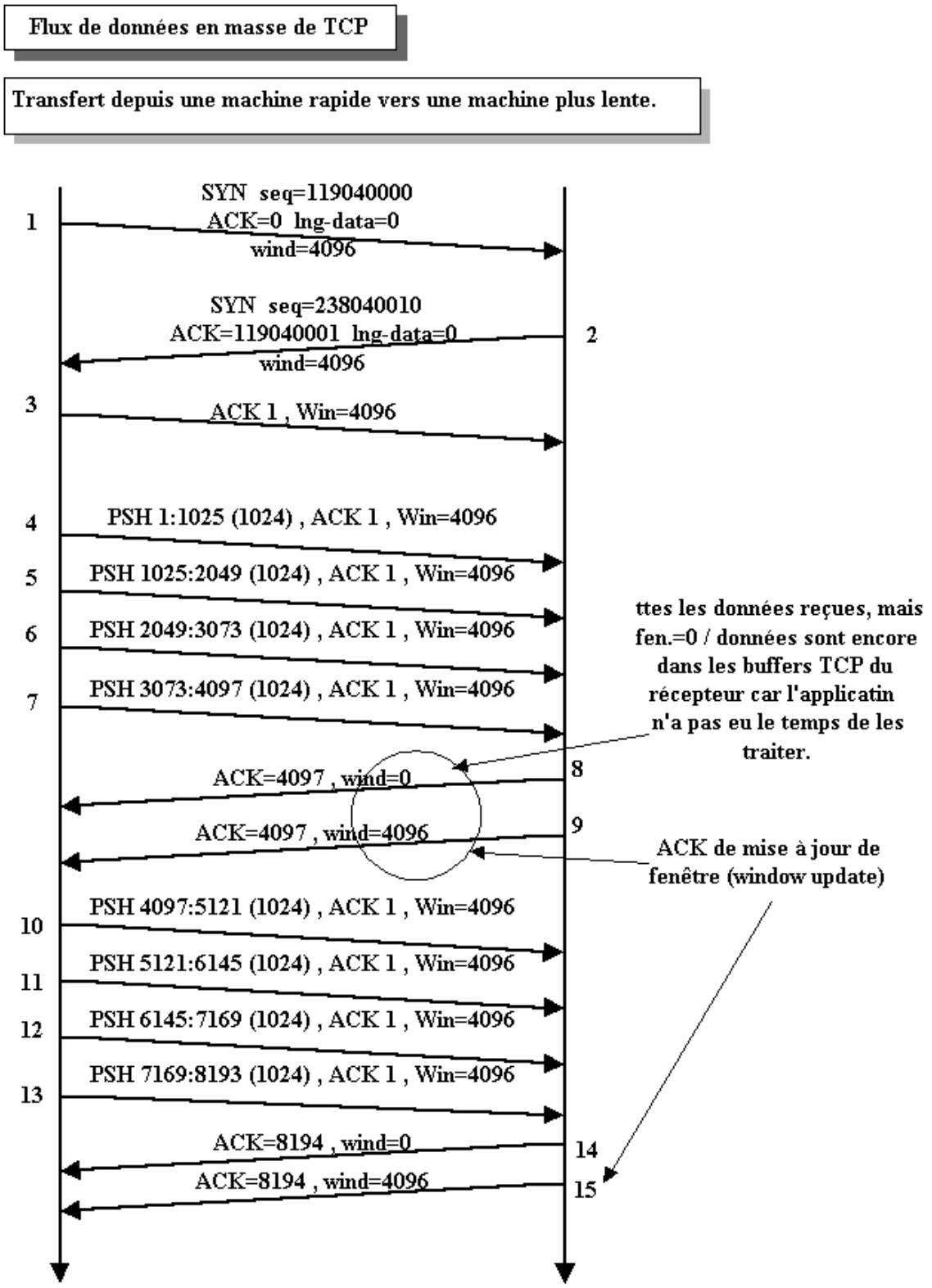


FIG. 81: Flux de données en masse de TCP 2/2

SR03 2004 - Cours Architectures Internet - Le RPC

8 SR03 2004 - Cours Architectures Internet - Le RPC

8.1 Les outils de communication en environnement réparti

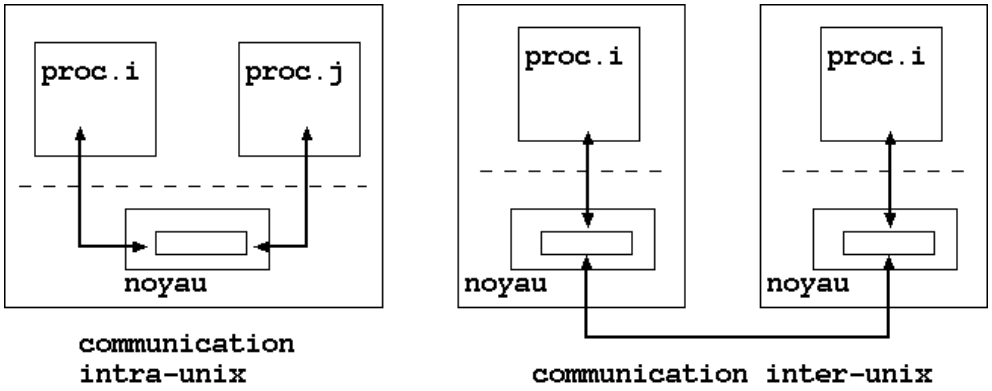


FIG. 82: Communication locale ou distante

On a vu l'outil de base des comunications entre process situés sur des machines distantes : **le socket**, qui crèe un **canal de communication bidirectionnel**.

Sur cet outil de base on a construit un certain nombre d'applications, par exemple :

- le transfert de fichiers (FTP),
- le courier (mail SMTP),
- la connexion à distance (telnet, rlogin, ssh),
- le service de nommage DNS,
- le graphique à distance (X11, VNC,...),
- ..

8.2 Le principe du RPC

Pour construire plus facilement des aplications plus sophistiquées, il est apparût le besoin pour un "service réseau" de plus haut niveau.

C'est au moment de la conception de NFS que SUN a conçu le service de **RPC** ("Remote Procedue Call"), appel de procédure à distance.

Ce mode de communication vient s'ajouter à ceux déja disponibles, "au-dessus" des couches TCP et UDP car il est d'un niveau conceptuel supérieur au simple échange de données.

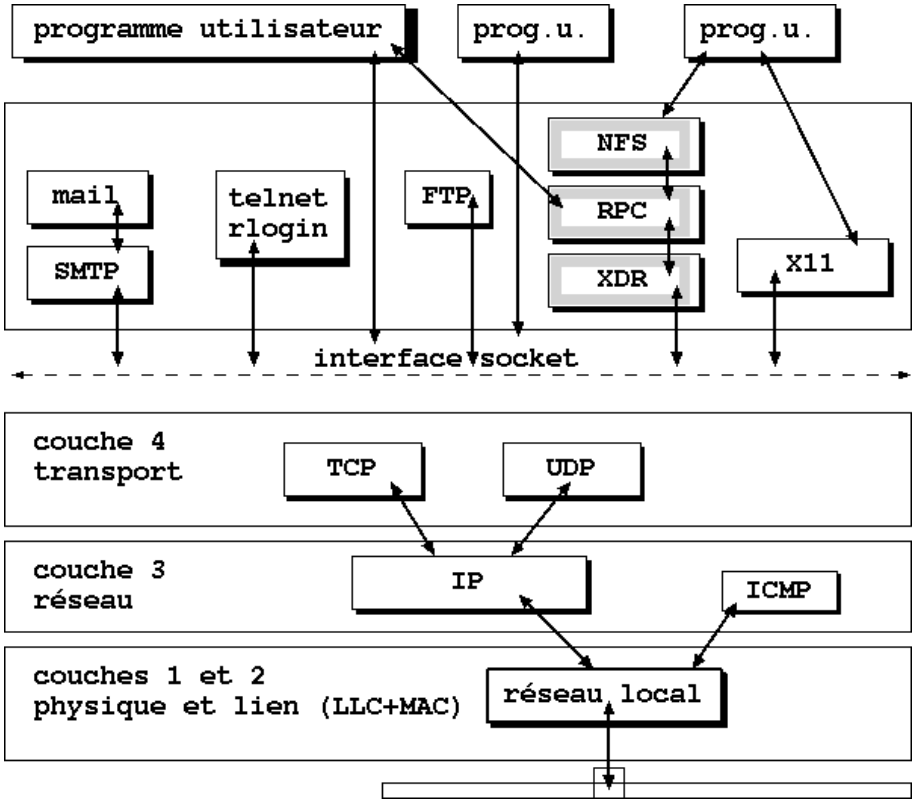


FIG. 83: Positionnement du RPC

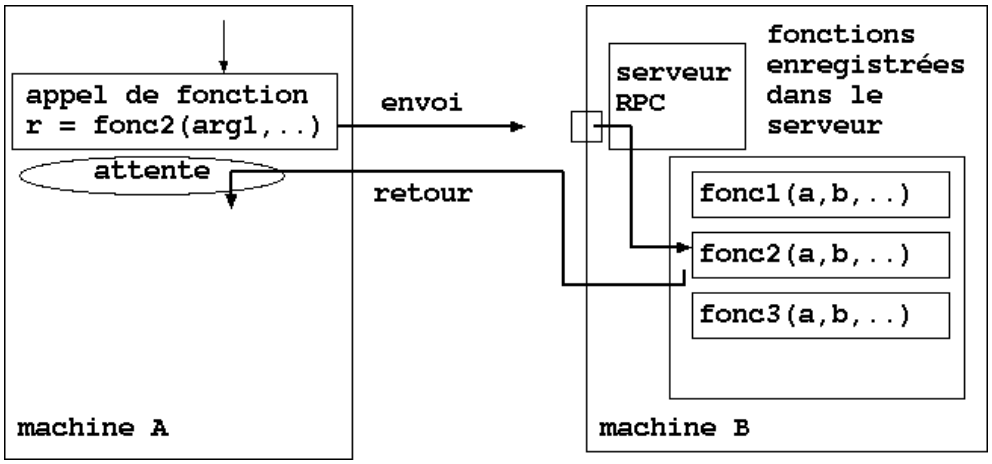


FIG. 84: Le principe du RPC

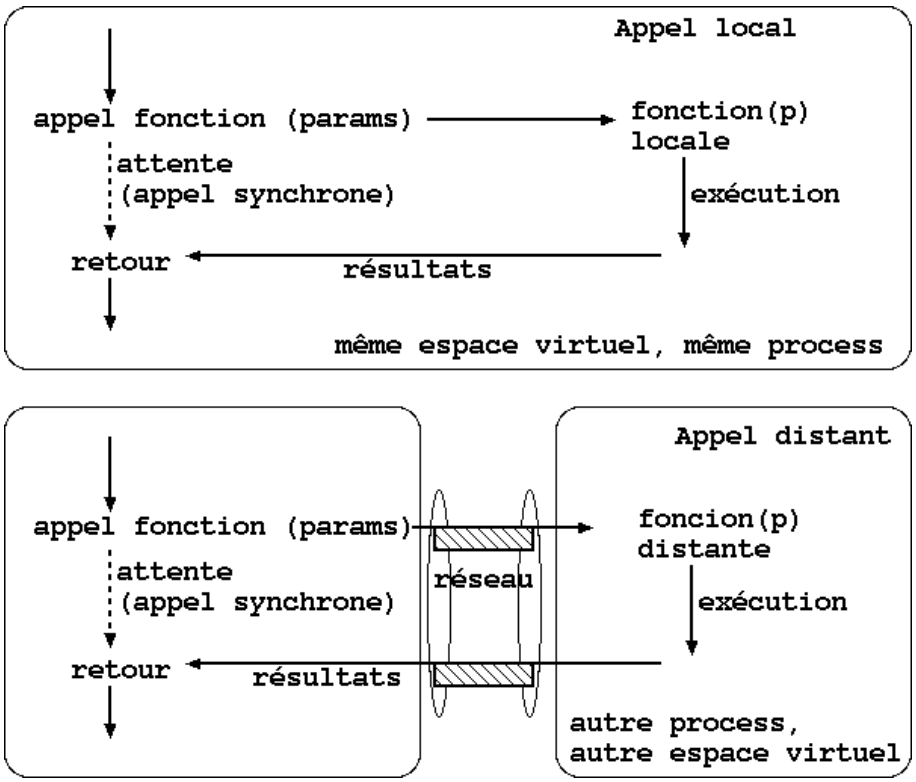


FIG. 85: Appel de procédure local versus appel distant

Ainsi, le RPC réalise un appel de sous-programme à travers le réseau. Le RPC est construit au-dessus des sockets. C'est une interface de plus haut niveau.

Les arguments sont encapsulés (empaquetés : "argument marshalling"), puis envoyés au process qui "propose" les services des fonctions qui sont enregistrées.

Le serveur RPC, décode le contenu du paquet et appelle la bonne fonction avec les arguments envoyés par l'appelant. La fonction s'exécute et renvoie un ou des résultats. Ces résultats sont eux aussi empaquetés ("result marshalling") et envoyés sur le réseau au process appelant.

Le service RPC est indépendant du protocole sous jacent utilisé pour transportter les messages, car il n'a besoin d'aucune information sur la façon dont le message est transmis.

Si on utilise le RPC au-dessus de UDP, c'est l'application qui doit se préocuper de la fiabilité des échanges et mettre en place ses propres mécanismes de chiens de garde (time-out) et de retransmission.

Ainsi, si l'application, utilisant le RPC au-dessus de UDP, reçoit une réponse après avoir fait une réémission après un time-out, elle peut seulement conclure que la procédure a été excutée **au moins une fois**.

Si l'application, utilisant le RPC au-dessus de TCP, reçoit une réponse, elle peut conclure que la procédure a été excutée **exactement une fois**.

ONC RPC est disponible à la fois sur TCP et sur UDP. Le choix de UDP comme transport, plus léger, ne doit être fait que si l'application a certaines caractéristiques :

- les procédures sont idempotentes (la procédure peut être exécutée plusieurs fois sans effet de bord) ; par exemple, lire un bloc de données est idempotent, céer un fichier ne l'est pas ;
- la taille des arguments aussi bien que des résultats est inférieure à la taille maximum d'un paquet UDP (8 Koctets) ;
- le serveur doit gérer plusieurs centaines de clients ; un serveur UDP peut faire cela facilement car il n'a pas à garder d'information d'état pour chacun de ses clients ; Par contre un serveur TCP qui garde une information d'état pour chaque client connecté va consommer

beaucoup de ressources (mémoire, descripteurs, ports socket) ;

Les éléments ci-dessus expliquent, que NFS étant un protocole sans état (par exemple il n'y a pas de fonction "open" dans NFS), puisse sans problème utiliser RPC sur UDP.

8.3 Le RPC de SUN (ONC RPC)

Le mécanisme de RPC a été spécifié par SUN et est connu sous le nom de SUN/RPC ou ONC/RPC (défini dans la RFC 1831) (ONC = Open-Network Computing).

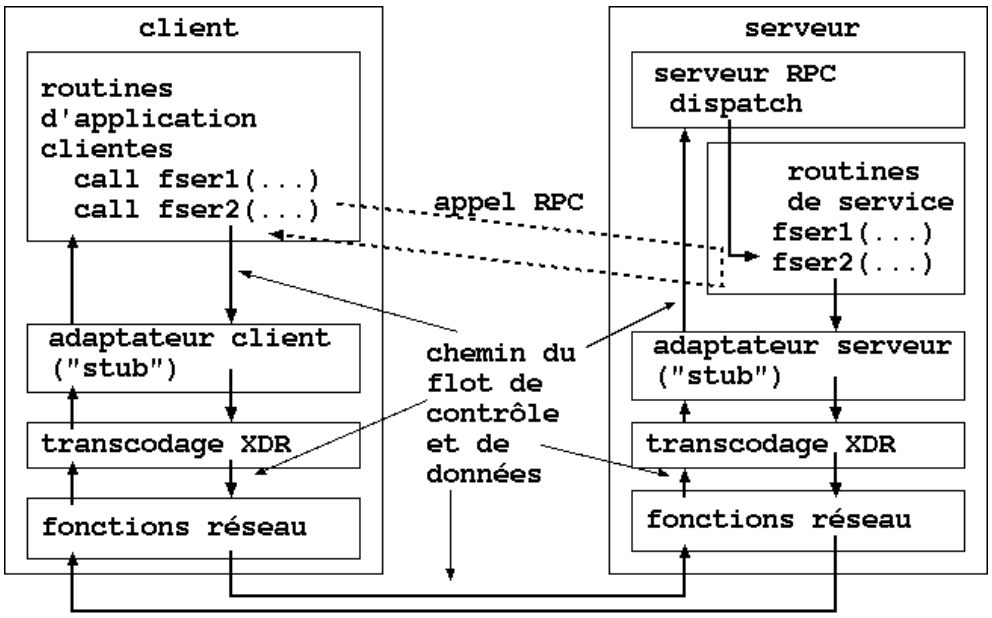


FIG. 86: Le mécanisme du RPC : notion d'adaptateur

L'adaptateur "simule" la procédure appelée. En effet, le client et le serveur sont deux programmes différents qui ont tous deux été compilés et linkés. Si le client fait appel à une fonction "fser1", le linker exige de trouver une fonction de ce nom lors de la création de l'exécutable.

Donc, une fonction ayant le nom "fser1" est fournie par le "stub". Mais, au lieu de faire le travail de la vraie fonction "fser1", cette pseudo fonction va servir de mandataire ("proxy") et déléguer le travail à la vraie fonction, située dans le serveur RPC.

La fonction de l'adaptateur va utiliser les fonctions de la bibliothèque XDR pour encapsuler les paramètres et les données d'appel de "fser1", puis fabriquer un paquet qu'elle va transmettre sur le réseau, en utilisant un socket UDP ou un socket TCP.

Ce paquet est envoyé au serveur RPC qui, en fonction du contenu du paquet, va envoyer ("dispatch") cet appel à la bonne fonction de service. Pour cela il utilise aussi les fonctions XDR pour "déballer" les données et les passer à la fonction à travers ses arguments.

Celle-ci s'exécute normalement, et fabrique une structure de données contenant les résultats. Cette structure sera traduite par des fonctions XDR avant d'être incluse dans un paquet "réseau" à destination du client.

Le "stub" côté client va réceptionner ce paquet, déballer la réponse avec les fonctions XDR, écrire les différentes parties de la réponse dans les arguments de la fonction appelée.

La fonction appelante du client "croit" faire un appel de procédure ordinaire. La complexité est masquée par le "stub".

Les stubs client et serveur utilisent un protocole définissant le format des messages qu'ils échangent entre eux pour réaliser un appel distant.

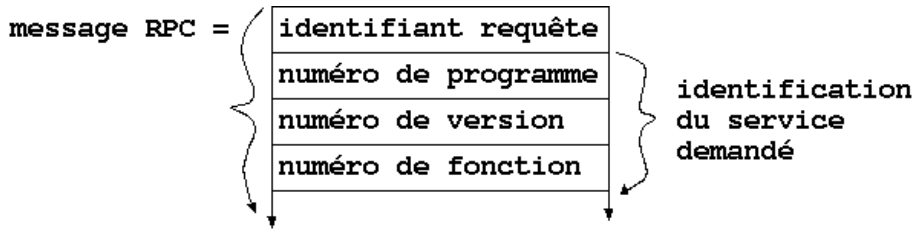


FIG. 87: Messages RPC

8.4 Mécanisme du RPC

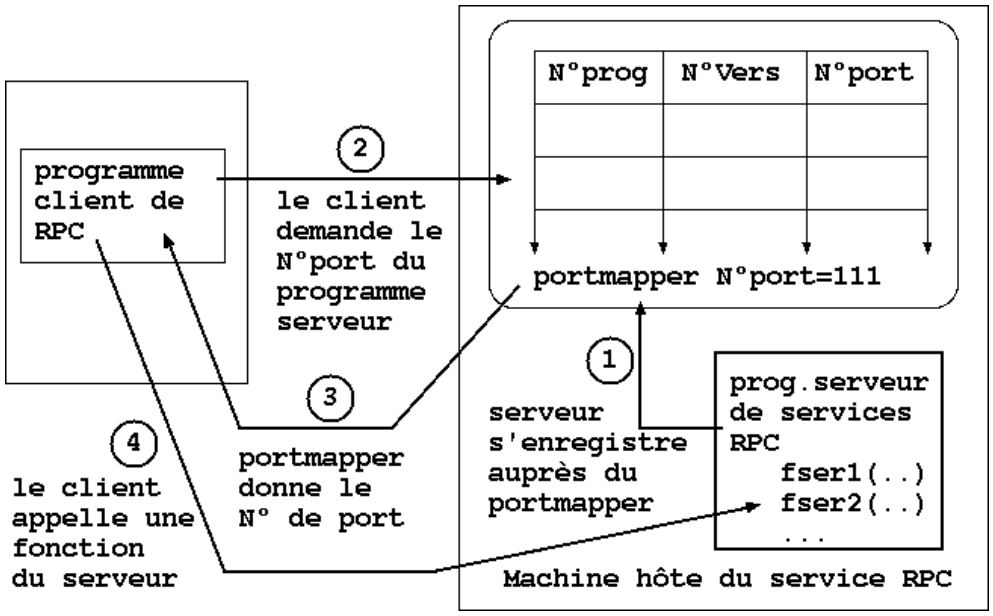


FIG. 88: Le mécanisme du RPC : le portmapper

Un "service RPC" est une liste de fonctions appelables à distance, avec une définition des arguments d'appel et de retour.

Un "service RPC" est repéré par un **numéro de programme** plus un **numéro de version**.

Un service RPC est **localisé** par un appel à un programme particulier, le **portmapper** (démon portmap) qui est en écoute sur un **port "bien connu"** (rappel : les numéros de port bien connus sont attribués par l'IANA).

Le port bien connu du portmapper est le **port 111**, aussi bien en UDP qu'en TCP :

```
[root@linux]# rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
.....
1 sr03 sunserv:~> rpcinfo -p
   program vers proto   port  service
    100000    4   tcp    111  rpcbind
    100000    4   udp    111  rpcbind
```

Il existe un certain nombre de programmes systèmes qui utilisent le RPC et qui sont déclarés auprès du portmapper avec des numéros conventionnels. Ces services sont listés dans le fichier "/etc/rpc".

```
2 sr03 sunserv:~> more /etc/rpc
#ident  "@(#)rpc        1.11    95/07/14 SMI"  /* SVr4.0 1.2 */
#       rpc
rpcbind         100000  portmap sunrpc rpcbind
rstatd          100001  rstat rup perfmeter
rusersd         100002  rusers
nfs             100003  nfsprog
.....
```

La fonction système **getrpcbyname** permet de récupérer le numéro d'un service en donnant son nom (défini dans /etc/rpc).

Identification des procédures.

Comme vu sur le schéma, les procédures sont regroupées en **programmes** ou services RPC. Chaque procédure est identifiée, parmi celles d'un programme, par un entier défini symboliquement.

Un programme est identifié par un entier long de 32 bits. Par exemple NFS a comme identifiant le numéro de programme 100003 (nombre décimal).

Les valeurs possibles sont découpées en plages. Une plage est réservée aux programmes "connus" (par exemple NFS). Les numéros y sont attribués par SUN.

Les utilisateurs peuvent choisir un numéro dans la plage 0x20000000 :0x3fffffff.

8.5 Programmation d'un service RPC

Il y a trois façons d'écrire un service RPC :

- **L'interface "de haut niveau".** Elle est plus facile à programmer que la suivante, moyennant quelques limites : elle impose UDP, et impose aussi le nombre de tentatives et la valeur du time-out en cas de non-réponse.
- **L'interface "de la couche basse".** Elle lève toutes les limitations précédentes, mais est beaucoup moins simple à utiliser.
- **Le langage RPCL et le compilateur RPCGEN.** À partir de la spécification de l'interface, **rpcgen** va générer les fichiers sources des stubs, des fonctions interfaces et des fonctions XDR. Le programmeur n'a plus qu'à écrire le serveur, le client et l'implémentation des fonctions de service.

8.6 Premier exemple : programmation sur l'interface de haut niveau

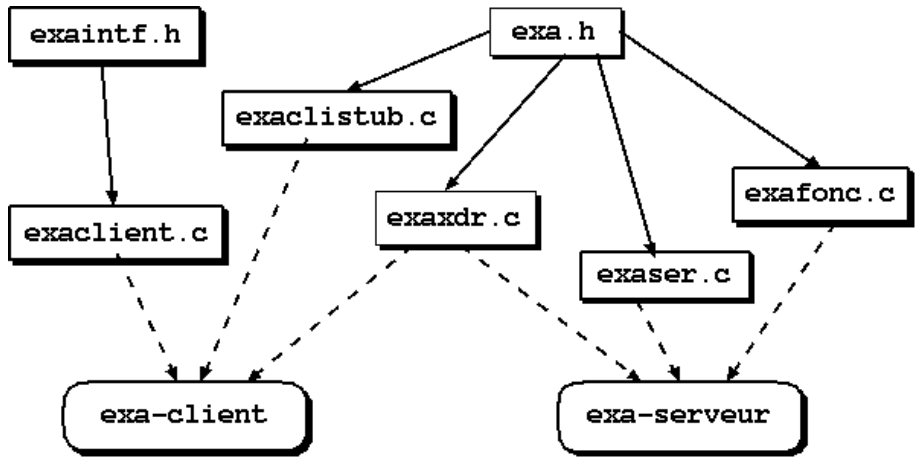


FIG. 89: Exemple RPC avec l'interface de haut niveau

Le figure 89 montre les fichiers sources et les deux exécutables produits : le serveur qui contient les fonctions de service RPC et le client qui les appelle.

On va voir dans l'ordre :

- **exa.h** définitions symboliques et définitions de structures,
- **exaxdr.c** fonctions de conversion XDR des structures échangées,
- **exafonc.c** implémentation des fonctions de service RPC,
- **exaser.c** programme principal du serveur qui enregistre le service et attends les requêtes,
- **exaintf.h** définition de l'interface des fonctions distantes,
- **exaclistub.c** l'adaptateur client ("stub"),
- **exaclient.c** le code du client qui appelle les fonctions distantes.

Les ordres de compilation et link sont les suivants :

```
$ gcc -c exaxdr.c
$ gcc -c exafonc.c
$ gcc -o exa-serveur exaser.c exafonc.o  exaxdr.o

$ gcc -c exaclistub.c
$ gcc -o exa-client exaclient.c exaclistub.o exaxdr.o
```

exa.h

```
 1  /* exa.h : fichier de définition de structures */
 2  /* inspiré de : Unix, Programmation et Communication,
 3                  Rifflet, Yunès, Dunod, Paris 2003 */
 4  /* UTC UV SR03 - (c) Michel.Vayssade@utc.fr
 5   */
 6
 7  #include <stdio.h>
 8  #include <stdlib.h>
 9  #include <rpc/types.h>
```

exa.h (suite)

```
10 #include <rpc/xdr.h>
11
12 /*0x20000001= 536870913 ds plage 0x20000000 :0x3fffffff */
13 /* numéros de prog. ou de service RPC et version */
14 #define EXA_PROG 0x20000001
15 #define EXA_V1 1       /* version 1 */
16 #define EXA_READ  1    /* procedure read */
17 #define EXA_WRITE 2    /* procedure write */
18 #define TAILLEMAX 1024 /* taille maxi d'un read ou write */
19
20 typedef struct {        // structure requête write
21   char  *fichier;       // nom fichier
22   long  place;          // place (deplacement depuis début)
23   char  *buf;           // data
24   int   nbre;           // nbre de car. à écrire
25 } ecriture;
26
27 typedef struct {        // structure requête read
28   char  *fichier;       // nom fichier
29   long  place;          // place (deplacement depuis début)
30   int   nbre;           // nbre de car. à lire
31 } lecture;
32
33 typedef struct {        // structure réponse read
34   int   nbrelu;         // nbre de car. effectivement lus
35   char  *buf;           // data
36 } replecture;
37
38 /* déclaration des filtres XDR définis dans exaxdr.c*/
39 bool_t xdr_ecriture(XDR *, ecriture *);
40 bool_t xdr_lecture(XDR *, lecture *);
41 bool_t xdr_replecture(XDR *, replecture *);
42
43 void *exawrite(ecriture *); // prototypes fonctions serveur
44 void *exaread (lecture *);  // renvoient un pter non typé
45
```

exaxdr.c

```
1 /* exaxdr.c : fichier de définition fonctions XDR */
2 /* inspiré de : Unix, Programmation et Communication,
3                 Rifflet, Yunès, Dunod, Paris 2003 */
4 /* UTC UV SR03 - (c) Michel.Vayssade@utc.fr
5  */
6 #include "exa.h"
```

exaxdr.c (suite)

```
 7
 8  /* voir "man xdr" pour les fonctions XDR dispo. */
 9  bool_t xdr_ecriture(XDR *xdr, ecriture *ecrire) {
10  return (   xdr_string(xdr, &ecrire->fichier, 64)
11          && xdr_long  (xdr, &ecrire->place)
12          && xdr_bytes (xdr, &ecrire->buf,
13                             &ecrire->nbre, TAILLEMAX)
14          && xdr_int   (xdr, &ecrire->nbre) );
15  }
16
17  bool_t xdr_lecture(XDR *xdr, lecture *lire) {
18  return (   xdr_string(xdr, &lire->fichier, 64)
19          && xdr_long  (xdr, &lire->place)
20          && xdr_int   (xdr, &lire->nbre) );
21  }
22
23  bool_t xdr_replecture(XDR *xdr, replecture *lire) {
24  int n;
25  n = lire->nbrelu; if (n<0) n=0; /* si<0, statut err */
26  return (   xdr_int   (xdr, &lire->nbrelu)
27          && xdr_bytes (xdr, &lire->buf, &n, TAILLEMAX) );
28  }
```

exafonc.c

```
 1  /* exafonc.c : définition des fonctions de service RPC */
 2  /* inspiré de : Unix, Programmation et Communication,
 3                  Rifflet, Yunès, Dunod, Paris 2003 */
 4  /* UTC UV SR03 - (c) Michel.Vayssade@utc.fr
 5   */
 6  #include "exa.h"
 7  #include <unistd.h>
 8  #include <fcntl.h>
 9  #include <errno.h>
10  extern int errno;
11
12  /* les fonctions de service RPC DOIVENT renvoyer
13     un pointeur void * sur une zone de mémoire
14     obligatoirement allouée en static */
15  static replecture repread;
16  static int repwrite;  // write renvoie nbre car.écrits
17  static int bufalloc=0; // allouer buffer une fois
18
19  void *exawrite (ecriture *ecrire) {
20    int id;
```

exafonc.c (suite)

```
21    id = open(ecrire->fichier, O_WRONLY);
22    if(id==-1) { repwrite=-1; return(void *)&repwrite;}
23    if(lseek(id,ecrire->place, SEEK_SET)==-1) {
24       repwrite=-2; return(void *)&repwrite;}
25    repwrite = write(id, ecrire->buf, ecrire->nbre);
26    close(id);
27    return(void *)&repwrite;
28  }
29
30  void *exaread (lecture *lire) {
31    int id;
32    if (bufalloc==0) { bufalloc=1;
33        repread.buf = (char *)malloc(TAILLEMAX); }
34    id = open(lire->fichier, O_RDONLY);
35    if(id==-1) { repread.nbrelu=-1; return(void *)&repread;}
36    if(lseek(id,lire->place, SEEK_SET)==-1) {
37       repread.nbrelu=-2; return (void *)&repread;}
38    repread.nbrelu = read(id, repread.buf, lire->nbre);
39    close(id);
40    return(void *)&repread;
41  }
```

exaser.c

```
 1  /* exaser.c : le serveur RPC */
 2  /* inspiré de : Unix, Programmation et Communication,
 3                  Rifflet, Yunès, Dunod, Paris 2003 */
 4  /* UTC UV SR03 - (c) Michel.Vayssade@utc.fr
 5   */
 6  #include "exa.h"
 7
 8  main() {
 9    // commencer par le désenregistrer par sécurité */
10    pmap_unset(EXA_PROG, EXA_V1);
11
12    // enregistrer chacune des fonctions de service
13
14    if (registerrpc(EXA_PROG, EXA_V1, EXA_READ,
15                    exaread, xdr_lecture,
16                             xdr_replecture) ==-1) {
17       fprintf(stderr,"Echec register EXA_READ\n");exit(2);}
18
19    if (registerrpc(EXA_PROG, EXA_V1, EXA_WRITE,
20                    exawrite, xdr_ecriture, xdr_int) ==-1){
21       fprintf(stderr,"Échec register EXA_READ\n");exit(2);}
```

exaser.c (suite)

```
22
23    // le serveur se met en attente des requêtes
24    svc_run();  // jusqu'à arrêt du serveur ; never return
25    fprintf(stderr,"Échec de svc_run\n");
26  }
```

exaintf.h

```
 1  /* exaintf.h   */
 2  /* définition d'interface des fonctions de service RPC */
 3  /* inspiré de : Unix, Programmation et Communication,
 4                  Rifflet, Yunès, Dunod, Paris 2003 */
 5  /* UTC UV SR03 - (c) Michel.Vayssade@utc.fr
 6     fichier à inclure dans les programmes clients
 7   */
 8  /*            machine  nom fi  place  buf  taille */
 9  int distread  (char *, char *, long, char *, int);
10  int distwrite (char *, char *, long, char *, int);
11
```

exaclistub.c

```
 1  /* exaclistub.c : fonctions "talon" des services RPC */
 2  /* inspiré de : Unix, Programmation et Communication,
 3                  Rifflet, Yunès, Dunod, Paris 2003 */
 4  /* UTC UV SR03 - (c) Michel.Vayssade@utc.fr
 5     ces fonctions présentent aux programmes clients
 6     l'interface des fonctions distantes et effectuent
 7     l'emballage/déballage des arguments et l'appel
 8     distant effectif vers le serveur
 9   */
10  #include "exa.h"
11  #include <rpc/rpc.h>
12  #include <rpc/clnt.h>
13
14  int distread  ( char *host, char *file,
15                  long place, char *buffer, int size){
16    // variables locales pour emballage/déballage
17    replecture repread ; int rep ; lecture lire ;
18    repread.buf = buffer ;
19    lire.fichier = file ;
20    lire.place = place ;
21    lire.nbre = size ;
```

exaclistub.c (suite)

```
22    rep = callrpc(host, EXA_PROG, EXA_V1, EXA_READ,
23          (xdrproc_t)xdr_lecture, (char *)&lire,
24          (xdrproc_t) xdr_replecture, (char *)&repread);
25    if (rep!=RPC_SUCCESS) {
26        clnt_perrno(rep); return -1; }
27    return repread.nbrelu;
28  }
29
30
31  int distwrite ( char *host, char *file,
32                  long place, char *buffer, int size){
33    // variables locales pour emballage/déballage
34    int reponse, rep; ecriture ecrire;
35    ecrire.fichier = file;
36    ecrire.place = place;
37    ecrire.buf = buffer;
38    ecrire.nbre = size;
39    rep = callrpc(host, EXA_PROG, EXA_V1, EXA_WRITE,
40          (xdrproc_t)xdr_ecriture, (char *)&ecrire,
41          (xdrproc_t)xdr_int, (char *)&reponse);
42    if (rep!=RPC_SUCCESS) {
43        clnt_perrno(rep); return -1; }
44    return reponse;
45  }
```

exaclient.c

```
 1  /* exaclient.c : client des services RPC */
 2  /* inspiré de : Unix, Programmation et Communication,
 3                  Rifflet, Yunès, Dunod, Paris 2003 */
 4  /* UTC UV SR03 - (c) Michel.Vayssade@utc.fr
 5     appelle les fonctions présentées aux programmes clients
 6     dans l'interface exaintf.h
 7   */
 8  #include "exaintf.h"
 9
10  char buf[32];  int rep;
11  char buffer[] = "1234567890-";
12  char file[] = "exatest.txt";
13  char host[] = "localhost";
14
15  main() {
16
17    printf("lire portion de %s sur %s\n",file, host);
18    rep = distread(host, file, 8, buf, 12);
```

exaclient.c (suite)

```
 19    printf("lu %d car. = %s \n",rep,buf);
 20
 21    printf("ecrire portion de %s sur %s\n",file, host);
 22    rep = distwrite(host, file, 8, buffer, 12);
 23    printf("ecrit %d car. \n",rep);
 24
 25    printf("relire même portion de %s sur %s\n",file, host);
 26    rep = distread(host, file, 8, buf, 12);
 27    printf("lu %d car. = %s \n",rep,buf);
 28
 29  }
```

L'exécution donne :

```
./exa-serveur       ! lancer le serveur

                    ! dans une autre fenêtre :
[root@linux]# rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
.........

 536870913    1   udp  32805
[root@linux]# rpcinfo -u localhost 536870913 1
program 536870913 version 1 ready and waiting

$ more exatest.dat
-abc-def-hij-klm-nop-qrs-tuv-wxy-z01
$ cp exatest.dat exatest.txt

$ ./exa-client
lire portion de exatest.txt sur localhost
lu 12 car. = -hij-klm-nop
ecrire portion de exatest.txt sur localhost
ecrit 12 car.
relire même portion de exatest.txt sur localhost
lu 12 car. = 1234567890--


ctrl-C pour arrêter le serveur

[root@linux]# rpcinfo -u localhost 536870913 1
rpcinfo: RPC: Timed out
program 536870913 version 1 is not available
```

8.7 Deuxième exemple : programmation avec rpcgen et rpcl

Prenons l'exemple d'une application locale dans laquelle un progamme principal "main.c" appelle des fonctions situées dans un fichier source "sp.c" (cf fig ; 90).

On va transformer cette application en deux parties : une application "client" qui contiendra "main.c" et une application serveur qui contiendra "sp.c" et qui rendra les fonctions de "sp.c" appelables à distance par le mécanisme de RPC.

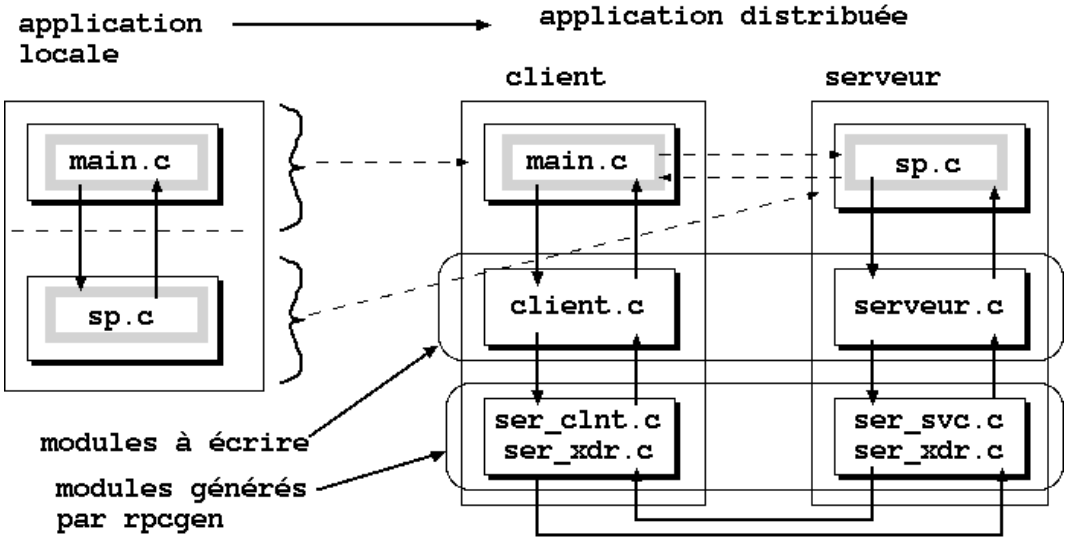


FIG. 90: RPC : utilisation de rpcgen 1/2

Pour chaque fonction de "sp.c" appelée par "main.c", on écrit dans les fichiers de service "client.c" et "serveur.c" des fonctions d'encapsulation qui vont masquer aux yeux du client d'une part, à ceux des fonctions de "sp.c" d'autre part, la complexité des appels à distance.

Pour réaliser les fonctions auxiliaires d'emballage et déballage des arguments d'appel et de réponse par les fonctions de la bibliothèque XDR, on va utiliser l'utilitaire **rpcgen**.

C'est un compilateur qui prend en entrée un fichier de **description d'interface** et qui génère des fichiers **sources** de type **.h** et **.c** qui seront inclus dans les commandes de compilation et édition de liens des programmes serveur et client.

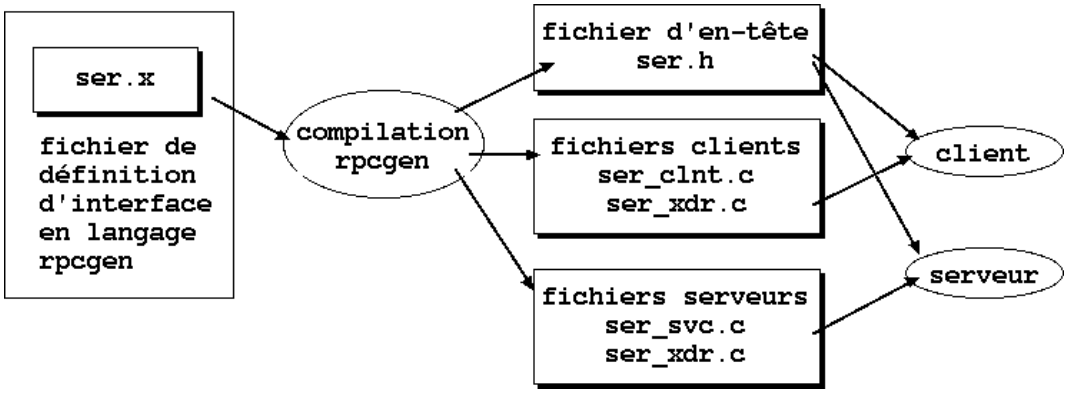


FIG. 91: RPC : utilisation de rpcgen 2/2

Exemple de fichier rpcgen

Pour une fonction dont le prototype serait : `int proc (int i1, int i2);`.

On passe obligatoirement par l'intermédiaire d'une structure. C'est cette structure qui sera convertie par les fonctions XDR.

```
struct argproc {
int x1;
int x2; };
```

Le compilateur rpcgen va construire les filtres XDR qui vont permettre de transférer cette structure. Il va aussi construire les fonctions d'appel RPC qui vont prendre ces structures en argument et en valeur de retour pour passer les données dans les deux sens.

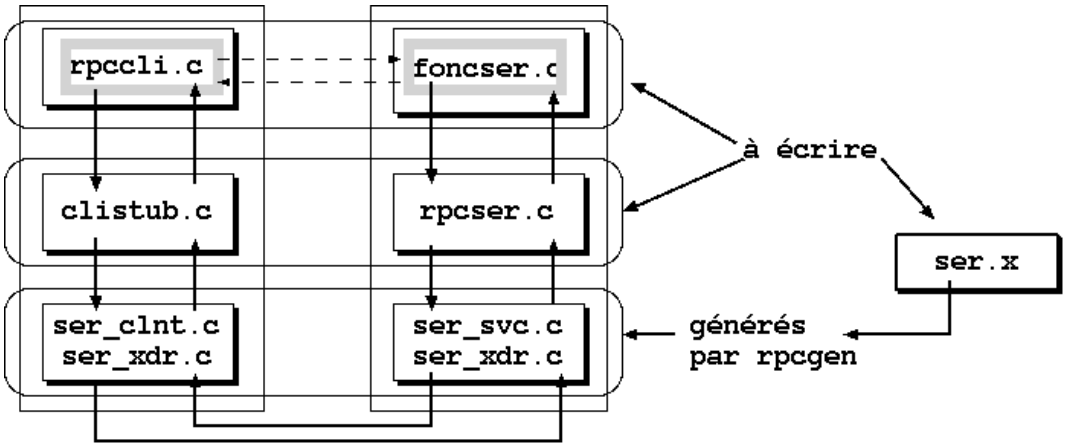


FIG. 92: Fichiers de l'exemple rpcgen

Fichier rpcgen de description du service :

ser.x

```
 1  /* ser.x : fichier rpcgen*/
 2  /* UTC UV SR03 - (c) Michel.Vayssade@utc.fr */
 3  /* version de ser.x avec FONC02 renvoyant un double
 4     et simplification déclaration param. fonctions   */
 5
 6  struct entrees{ int i1;
 7                  int i2;};
 8
 9  struct entrees2{ double i3;
10                   double i4;};
11
12  program PROGSERVICE
13  {  version VERSERVICE
14     {
15         int FONC01(entfonc) =1;      /* fonction numero 1*/
16         double FONC02(entrees2) =2;  /* fonction numero 2*/
17
18     }=2;    /*version numero 2 de PROGSERVICE */
19  } = 839909901;  /* 0x32100005; /* numero de programme */
20                  /* =839909381 en decimal */
21
```

Compilation :

```
rpcgen ser.x

gcc -o cli rpccli.c clistub.c ser_clnt.c ser_xdr.c

gcc -o ser rpcser.c foncser.c ser_svc.c ser_xdr.c
```

Le fonctionnement des quatre fichiers principaux est donné sur la figure 93.

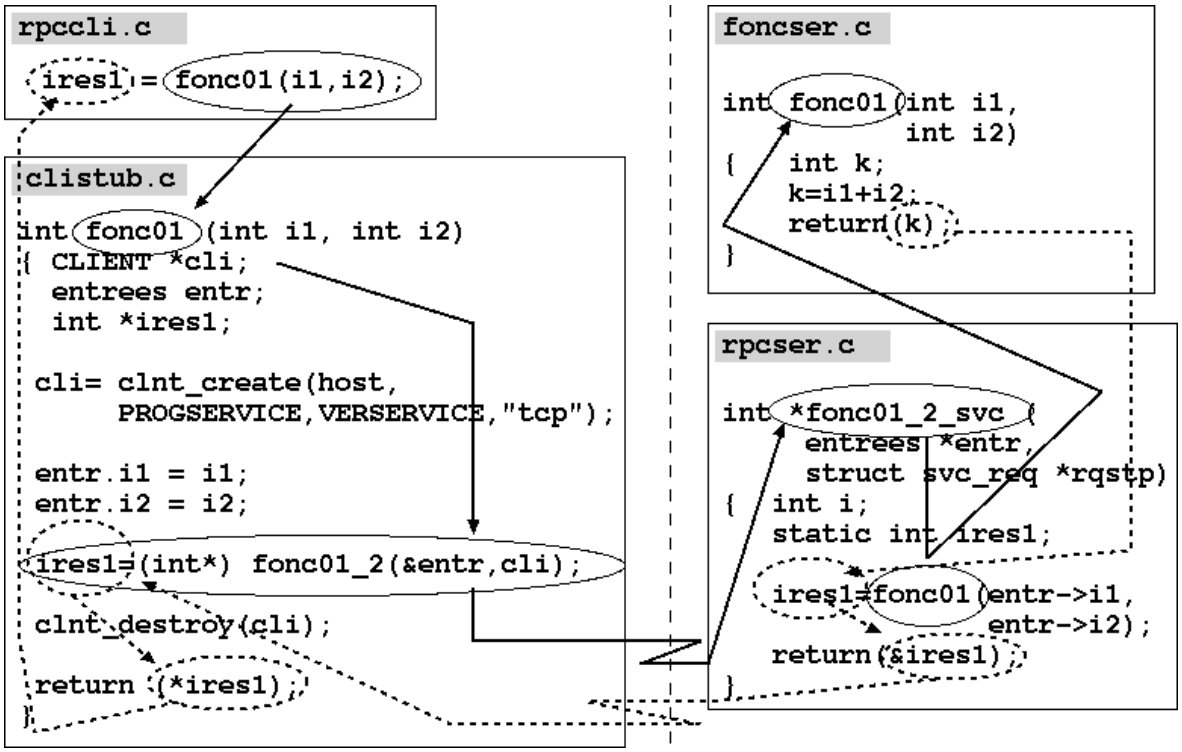


FIG. 93: Trace des appels et retours de résultats

rpccli.c

```
 1  /* rpccli.c  - exemple rpc - client */
 2  /* UTC UV SR03 - (c) Michel.Vayssade@utc.fr */
 3  #include <stdio.h>
 4  #include <rpc/rpc.h>
 5  #include <stdlib.h>
 6  #include <time.h>
 7
 8  int fonc01(int , int);
 9  double fonc02(double, double);
10
11  /* fonctions équivalentes à fonc01,. mais locales */
12  int loc01 (int i1, int i2)
13  { int ires1;
14  ires1=i1+i2;
```

rpccli.c (suite)

```
15  return(ires1);
16  }
17  double loc02 (double xx, double xy)
18  { double ires2;
19  ires2=xx*xy;
20  return(ires2);
21  }
22
23  main(int argn, char **argv) {
24  int i1,i2,ires,ires1,k,i , t0,t1,t2;
25  double z,i3,i4;
26  if(argn!=5){
27    printf("syntaxe : > client n1 n2 n3 n4 -\
28    Avec n1,n2= int et n3,n4=double\n");
29    exit (0);}
30  i1 = atoi (argv[1]); i2 = atoi (argv[2]);
31  i3 = atof (argv[3]); i4 = atof (argv[4]);
32
33  printf("Appel au serveur avec i1= \
34  %5d, i2=%5d,i3=%f,i4=%f\n",i1,i2,i3,i4);
35  t0= clock();
36  printf("Temps 1=%d\n",t0);
37
38  /*Appel a distance, comme si les procédures étaient locales*/
39    for(i=0;i<50;i++) {
40        ires1 = fonc01(i1,i2);
41        z = fonc02(i3,i4);
42    }
43  t1 = clock();
44  printf("resultat fonc01= %7d\n", ires1);
45  printf("resultat fonc02= %f\n", z);
46  printf("Temps 2=%d %d\n",t1,t1-t0);
47
48
49  /* Appel aux procedures locales*/
50    for(i=0;i<50000;i++) {
51        k = loc01 (i1,i2);
52        z = loc02 (i3,i4);
53    }
54  t2 = clock();
55  printf("resultat loc01= %7d\n", k);
56  printf("resultat loc02= %7f\n", z);
57  printf("Temps 3=%d %d \n",t2,t2-t1);
58
59  }
```

clistub.c

```
 1  /* clistub.c - exemple rpc - talon d'appel du client */
 2  #include "ser.h"
 3  #include <rpc/rpc.h>
 4  #define SERVEUR "localhost"
 5
 6  double fonc02(double,double);
 7  int fonc01(int, int);
 8
 9  int fonc01 (int i1, int i2)
10  {   CLIENT *cli;  /* handle client*/
11      entrees entr; /* parametres d'appel */
12      int *ires1;   /* resultat*/
13      char *host;   /* nom du serveur distant*/
14
15  /*connexion au serveur*/
16  host=SERVEUR;
17  cli= clnt_create(host,PROGSERVICE,VERSERVICE,"tcp");
18  if (cli==NULL) {clnt_pcreateerror(host);return (-1);}
19
20      entr.i1 = i1; /* prépare structure argument d'appel */
21      entr.i2 = i2;
22
23  /* faire l'appel rpc a fonc01_1 car dans ser.x on declare
24     un appel a FONC01 version =1 */
25      ires1=(int*) fonc01_2(&entr,cli);
26
27  if (ires1 ==NULL) {
28      clnt_perror(cli,"err appel rpc"); return(-1);}
29      clnt_destroy(cli);  /* libérer handle connexion */
30
31      return (*ires1);  /* paramètre de retour */
32  }
33
34
35  double fonc02 (double i3,double i4)
36  {   CLIENT *cli;    /* handle client */
37      entrees2 entr;  /* parametres d'appel */
38      double *ires2; /* resultat */
39      char *host;     /* nom du serveur distant */
40
41  host=SERVEUR;   /* connexion au serveur */
42  cli= clnt_create(host,PROGSERVICE,VERSERVICE,"tcp");
43  if (cli==NULL) {clnt_pcreateerror(host);return (-1);}
44
45      entr.i3 = i3;  /* prépare structure argument d'appel */
46      entr.i4 = i4;
```

clistub.c (suite)

```
47
48  /* faire l'appel rpc a foncser_2 car dans ser.x on declare
49     un appel a FONCSER version =2 */
50      ires2 = fonc02_2(&entr,cli);
51      printf("clistub %f\n",*ires2);
52
53  if (ires2 ==NULL) {
54      clnt_perror(cli,"err appel rpc"); return(-1);}
55      clnt_destroy(cli);      /*liberer handle*/
56
57      return (*ires2);  /* parametre de retour */
58  }
59
```

rpcser.c

```
 1  /* rpcser.c - exemple rpc - serveur.c talon d'appel */
 2
 3  #include "ser.h"
 4  #include <stdio.h>
 5  FILE *iout;
 6
 7  int fonc01(int , int);
 8  double fonc02(double, double);
 9
10  /*appel a fonc01_2 car declare version=2 dans ser.x*/
11  int *fonc01_2_svc (entrees *entr, struct svc_req *rqstp)
12  {  int i;
13     static int ires1;
14
15     ires1=fonc01(entr->i1,entr->i2);
16     return(&ires1);
17  }
18
19
20  /*appel a fonc02_2 car declare version=2 dans ser.x*/
21  double *fonc02_2_svc (entrees2 *entr, struct svc_req *rqstp)
22  {  int i;
23     static double dble;
24
25     dble = fonc02(entr->i3,entr->i4);
26
27   iout = fopen ("log.log","a+b");
28   i = fprintf (iout,"fonc02 i3 i4= %f %f dd=%f \n",
29                  entr->i3,entr->i4,dble);
```

rpcser.c (suite)

```
30   fflush(iout);
31   fclose(iout);
32
33     return(&dble);
34  }
35
```

foncser.c

```
 1  /* foncser.c - exemple rpc  */
 2  /* fonc01.c et fonc02.c fonctions de service */
 3  /* fonctions de service appelees par rpc */
 4
 5  int fonc01(int i1, int i2) {
 6      int k;
 7      k=i1+i2;
 8      return(k);
 9  }
10
11  double fonc02(double i3, double i4) {
12      double k;
13      k=i3*i4;
14      return k;
15  }
```

Exécution de l'exemple précédent :

```
! dans une première fenêtre
$ ./ser

! dans une deuxième fenêtre
$ ./cli 3 4 1.2 2.
Appel au serveur avec i1=  3, i2=  4, i3=1.200000,i4=2.000000
Temps 1=0
clistub 2.400000
........
clistub 2.400000
resultat fonc01=       7
resultat fonc02= 2.400000
Temps 2=20000 20000
resultat loc01=       7
resultat loc02= 2.400000
Temps 3=30000 10000
```

Variante de l'exemple précédent :

On modifie le paramètre de retour de la fonction "fonc2" de façon à lui faire renvoyer un réel "double", à la place de la structure contenant un double.

Ce qui est à noter ici ce sont les différences dans la manière de traiter correctement ce paramètre de retour. Les diférences entre les codes sources des deux exemples sont données ci-dessous :

diff-ex1-ex2.txt

```
 1 [vayssade@linux ex2]$ diff ser.x ../ex1/ser.x
 2 3,4d2
 3 < /* version de ser.x avec FONC02 renvoyant un double
 4 <    et simplification déclaration param. fonctions   */
 5 7a6
 6 > typedef struct entrees entfonc;
 7 10a10,12
 8 > typedef struct entrees2 entfonc2;
 9 >
10 > struct sortie2{ double dd;};
11 16c18
12 <        double FONC02(entrees2) =2;  /* fonction numero 2*/
13 --
14 >        sortie2 FONC02(entfonc2) =2; /* fonction numero 2*/
15 19c21
16 < } = 839909901;  /* 0x32100005; /* numero de programme */
17 --
18 > } = 839909900;  /* 0x32100005; /* numero de programme */
19
20
21 [vayssade@linux ex2]$ diff rpccli.c ../ex1/rpccli.c
22 [vayssade@linux ex2]$ diff foncser.c ../ex1/foncser.c
23
24 [vayssade@linux ex2]$ diff clistub.c ../ex1/clistub.c
25 38c38
26 <     double *ires2; /* resultat */
27 --
28 >     sortie2 *ires2; /* resultat */
29 51c51
30 <     printf("clistub %f\n",*ires2);
31 --
32 >     printf("clistub %f\n",ires2->dd);
33 57c57
34 <     return (*ires2);  /* parametre de retour */
35 --
36 >     return (ires2->dd);  /* parametre de retour */
37
38
39 [vayssade@linux ex2]$ diff rpcser.c ../ex1/rpcser.c
40 21c21
41 < double *fonc02_2_svc (entrees2 *entr, struct svc_req *rqstp)
```

diff-ex1-ex2.txt (suite)

```
42  --
43  > sortie2 *fonc02_2_svc (entrees2 *entr, struct svc_req *rqstp)
44  23c23
45  <    static double dble;
46  --
47  >    static sortie2 dble;
48  25c25
49  <    dble = fonc02(entr->i3,entr->i4);
50  --
51  >    dble.dd = fonc02(entr->i3,entr->i4);
52  29c29
53  <                 entr->i3,entr->i4,dble);
54  --
55  >                 entr->i3,entr->i4,dble.dd);
56
```

Programme auxiliaire : unreg.c : Le petit programme suivant permet de désenregistrer un serveur, pour le cas où on a besoin de l'arrêter et de le relancer souvent.

unreg.c

```
 1  /* unreg.c : unregister prog/fonc from portmapper */
 2
 3  #include "ser.h"
 4  #include <rpc/rpc.h>
 5
 6  main ()
 7  {
 8  printf("unregister prog=%d service=%d\n",PROGSERVICE, VERSERVICE);
 9
10  svc_unregister(PROGSERVICE, VERSERVICE);
11
12  }
```

On peut aussi, comme dans l'exemple 1 faire débuter le serveur par un essai de désenregistrement (voir code source de exaser.c).

8.8 Troisième exemple : reprogrammation avec rpcgen et rpcl du premier exemple

Il faut commencer par construire un fichier descriptif dans lequel sont définies les structures d'entrée et de sortie et les foncions appelables par RPC :

exb.x

```
 1 /* exb.x  définition d'interface distante pour rpcgen */
 2 /* inspiré de : Unix, Programmation et Communication,
 3                 Rifflet, Yunès, Dunod, Paris 2003 */
 4 /* UTC UV SR03 - (c) Michel.Vayssade@utc.fr       */
 5 /*---------------------------------------*/
 6
 7 #define TAILLEMAX 1024 /* taille maxi d'un read ou write */
 8
 9 /* définition strutures échangées, recopiées de exa.h */
10 /* les commentaires par // pas acceptés par rpcgen    */
11
12 struct ecriture {     /* structure requête write */
13   string fichier<64>; /* nom fichier, maxi 64 car. */
14   long   place;       /* place (deplacement depuis début)*/
15   char   buf<TAILLEMAX>;           /* data */
16   int    nbre;         /* nbre de car. à écrire */
17 };
18
19 struct lecture {      /* structure requête read */
20   string fichier<64>; /* nom fichier, maxi 64 car. */
21   long   place;       /* place (deplacement depuis début)*/
22   int    nbre;        /* nbre de car. à lire */
23 };
24
25 struct replecture {   /* structure réponse read */
26   int   nbrelu;       /* nbre de car. effectivement lus */
27   char  buf<TAILLEMAX>;           /* data */
28 };
29
30 /* noms de programme (service RPC), version et fonctions
31    distantes, copiées de exa.h pour montrer la similitude */
32
33 program EXB_PROG {
34   version EXB_V1 {
35       /* les 2 fonctions disponibles dans ce service */
36       replecture EXAREAD (lecture) =1; /* fonc 1 */
37       int EXAWRITE(ecriture) =2;   /* fonction 2 */
38   } = 2;  /* version 2 */
39 } = 0x20000002; /* numéro de prog. ou de service RPC */
40
```

Ensuite on écrit les sources des programmes client, serveur et des adaptateurs, en s'aidant des compilations rpcgen, puis on crèe les exécutables :

```
        ! préparer fichier data de test
cp ../exa/exatest.dat ./exatest.txt

        ! dans une première fenêtre
./serveur
        ! dans une deuxième fenêtre
./client

[root@linux root]# rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper

 536870914    2   udp  32771
 536870914    2   tcp  32777

[root@linux root]# rpcinfo -u localhost  536870914 2
program 536870914 version 2 ready and waiting

[vayssade@linux exb]$ cp ../exa/exatest.dat ./exatest.txt
[vayssade@linux exb]$ ./client
lire portion de exatest.txt sur localhost
lu 12 car. = -hij-klm-nop
ecrire portion de exatest.txt sur localhost
ecrit 12 car.
relire même portion de exatest.txt sur localhost
lu 12 car. = 1234567890--
```

L'exécution donne :

```
[vayssade@linux exb]$ ls
exb.x
[vayssade@linux exb]$ rpcgen exb.x ; ls
exb_clnt.c  exb.h  exb_svc.c  exb.x  exb_xdr.c

! demander à rpcgen de faire des squelettes pour les
! adaptateurs client et serveur
[vayssade@linux exb]$ rpcgen -Ss -C exb.x > exbfonc.c
[vayssade@linux exb]$ rpcgen -Sc -C exb.x > temp_cli.c


nedit exbfonc.c
! reprendre code des fonctions de ../exa/exafonc.c
```

```
nedit exbclistub.c
! inspiré de temp_cli.c et ../exa/exaclistub.c

cp ../exa/exaclient.c ./exbclient.c
nedit exbclient.c

gcc -o client exbclient.c exbclistub.c exb_clnt.c exb_xdr.c
gcc -o serveur exb_svc.c exbfonc.c exb_xdr.c
```

Les codes sources à écrire (les autres sont générés par rpcgen) :

exbclient.c

```
 1  /* exbclient.c : client des services RPC */
 2  /* inspiré de : Unix, Programmation et Communication,
 3                  Rifflet, Yunès, Dunod, Paris 2003 */
 4  /* UTC UV SR03 - (c) Michel.Vayssade@utc.fr
 5     appelle les fonctions présentées aux programmes clients
 6     dans l'interface exaintf.h
 7   */
 8  #include "exaintf.h"
 9
10  char buf[32];  int rep;
11  char buffer[] = "1234567890-";
12  char file[] = "exatest.txt";
13  char host[] = "localhost";
14
15  main() {
16
17    printf("lire portion de %s sur %s\n",file, host);
18    rep = distread(host, file, 8, buf, 12);
19    printf("lu %d car. = %s \n",rep,buf);
20
21    printf("ecrire portion de %s sur %s\n",file, host);
22    rep = distwrite(host, file, 8, buffer, 12);
23    printf("ecrit %d car. \n",rep);
24
25    printf("relire même portion de %s sur %s\n",file, host);
26    rep = distread(host, file, 8, buf, 12);
27    printf("lu %d car. = %s \n",rep,buf);
28
29  }
```

exbclistub.c

```
 1  /* exbclistub.c : stub client des services RPC */
```

exbclistub.c (suite)

```
 2  /* inspiré de : Unix, Programmation et Communication,
 3                  Rifflet, Yunès, Dunod, Paris 2003 */
 4  /* UTC UV SR03 - (c) Michel.Vayssade@utc.fr
 5   */
 6
 7  #include "exb.h"                /* from template */
 8
 9  int distread  ( char *host, char *file,
10                  long place, char *buffer, int size){
11    // variables locales pour emballage/déballage
12
13                               /* from template */
14      CLIENT *clnt;
15      replecture  *result_1;
16      lecture  exaread_2_arg;
17      int  *result_2;
18      ecriture  exawrite_2_arg;
19
20      /* connexion au serveur RPC */
21      clnt= clnt_create(host, EXB_PROG, EXB_V1,"udp");
22      if(clnt==NULL) {clnt_pcreateerror(host);exit(1);}
23
24      /* préparer la structure d'appel */
25      exaread_2_arg.fichier = file;
26      exaread_2_arg.place = place;
27      exaread_2_arg.nbre = size;
28
29      /* faire l'appel RPC */
30      result_1 = exaread_2(&exaread_2_arg, clnt);
31      if (result_1 == (replecture *) NULL) {
32              clnt_perror (clnt, "call failed"); }
33
34      clnt_destroy(clnt); /* liberer handle connexion */
35
36    /* récupérer résultat, le ranger dans buffer client */
37    /* contenu de result_1 n'est défini qu'après exaread_2()
38       un tableau "buf" déclaré dans exb.x est traduit par
39       rpcgen en une struture contenant :
40            int buf_len;    la longueur du tableau
41            void * buf_val; l'adresse du tableau       */
42
43    /* copier data reçues dans buffer fourni par appelant */
44      strncpy (buffer,result_1->buf.buf_val,
45                      result_1->buf.buf_len);
46      return result_1->buf.buf_len;
47  }
48
```

exbclistub.c (suite)

```
49  int distwrite ( char *host, char *file,
50                  long place, char *buffer, int size){
51    // variables locales pour emballage/déballage
52
53                                /* from template */
54       CLIENT *clnt;
55       replecture  *result_1;
56       lecture  exaread_2_arg;
57       int  *result_2;
58       ecriture  exawrite_2_arg;
59
60       /* connexion au serveur RPC */
61       clnt= clnt_create (host, EXB_PROG, EXB_V1,"udp");
62       if(clnt==NULL) {clnt_pcreateerror(host);exit(1);}
63
64       /* préparer la structure d'appel */
65       exawrite_2_arg.fichier = file;
66       exawrite_2_arg.place = place;
67       exawrite_2_arg.nbre = size;
68       exawrite_2_arg.buf.buf_len = size;
69       exawrite_2_arg.buf.buf_val = buffer;
70
71       /* faire l'appel RPC */
72       result_2 = exawrite_2(&exawrite_2_arg, clnt);
73       if (result_2 == (int *) NULL) {
74               clnt_perror (clnt, "call failed"); }
75
76       clnt_destroy(clnt);  /* liberer handle connexion */
77
78       /* récupérer le status renvoyé */
79       return *result_2;
80  }
```

exbfonc.c

```
 1  /* exbfonc.c : stub client des services RPC */
 2  /* inspiré de : Unix, Programmation et Communication,
 3                  Rifflet, Yunès, Dunod, Paris 2003 */
 4  /* UTC UV SR03 - (c) Michel.Vayssade@utc.fr
 5   * implémentation des fonctions de service
 6   */
 7
 8  #include "exb.h"
 9  #define TAILLEMAX 1024 /* taille maxi d'un read ou write */
10  /* devrait être dans exb.h. Bug rpcgen sur linux? */
```

exbfonc.c (suite)

```
11  #include <fcntl.h>
12
13  static int repwrite; // write renvoie nbre de car. écrits
14  static int bufalloc=0; // allouer buffer une fois
15
16  replecture *
17  exaread_2_svc(lecture *lire, struct svc_req *rqstp)
18  {
19          static replecture  repread;
20    int id;
21    if (bufalloc==0) { bufalloc=1;
22        repread.buf.buf_val = (char *)malloc(TAILLEMAX);
23        repread.buf.buf_len = TAILLEMAX; }
24    id = open(lire->fichier, O_RDONLY);
25    if(id==-1){repread.nbrelu=-1; return(void *)&repread;}
26    if(lseek(id,lire->place, SEEK_SET)==-1) {
27       repread.nbrelu=-2; return (void *)&repread;}
28    repread.nbrelu = read(id, repread.buf.buf_val,
29                              lire->nbre);
30    repread.buf.buf_len = repread.nbrelu;
31    close(id);
32
33          return &repread;
34  }
35
36  int *
37  exawrite_2_svc(ecriture *ecrire, struct svc_req *rqstp)
38  {
39    int id;
40    id = open(ecrire->fichier, O_WRONLY);
41    if(id==-1) { repwrite=-1; return(void *)&repwrite;}
42    if(lseek(id,ecrire->place, SEEK_SET)==-1) {
43       repwrite=-2; return(void *)&repwrite;}
44    repwrite= write(id, ecrire->buf.buf_val,ecrire->nbre);
45    close(id);
46          return &repwrite;
47  }
```

8.9 Autres possibilités des RPCs

Diffusion : Il est possible de diffuser ("broadcast") une requête avec le transport UDP par la primitive **clnt_broadcast()**.

Démarrage et arrêt du serveur par le client :

Le démon **inetd** peut lancer d'autres démons mais aussi des services RPC. Le paragraphe suivant est extrait de la documentation BSD de inetd.

Le programme inetd est lancé lors du démarrage par /etc/rc (voir rc(8)). Il écoute le réseau pour détecter les connexions sur certaines sockets internet. Lorsqu'une connexion est trouvée, il détermine à quel service correspond cette socket et appelle le programme qui peut satisfaire la requête. Ce dernier est appelé avec le service de sockets et ses description d'entrée, sortie et erreur standard. Une fois le programme terminé, inetd poursuit son écoute (sauf dans quelques cas décrits plus loin). Essentiellement, inetd permet de faire tourner un démon pour en appeler plusieurs autres, réduisant ainsi la charge du système.

Lors de son exécution, inetd détermine sa configuration à partir d'un fichier qui, par défaut, est /etc/inetd.conf. Pour les services ONC RPC, le nom est celui qui apparait dans le fichier /etc/rpc.

Pour utiliser le mécanisme, le client fait un rexec() du serveur avec le numéro du programme obtenu en consultant le portmapper. Le serveur est lancé par inetd. Le client attends quelques secondes et appelle le serveur.

On arrête le serveur en invoquant une fonction de celui-ci qui se termine par un exit().

RPC et "callback" RPC

On peut se trouver dans une situation ou un serveur RPC fait lui-même appel par RPC à un autre serveur. Dans ce cas, il peut être plus efficace que le serveur "final" renvoie la réponse directement à l'appelant de départ plutôt que de remonter successivement tous les appels.

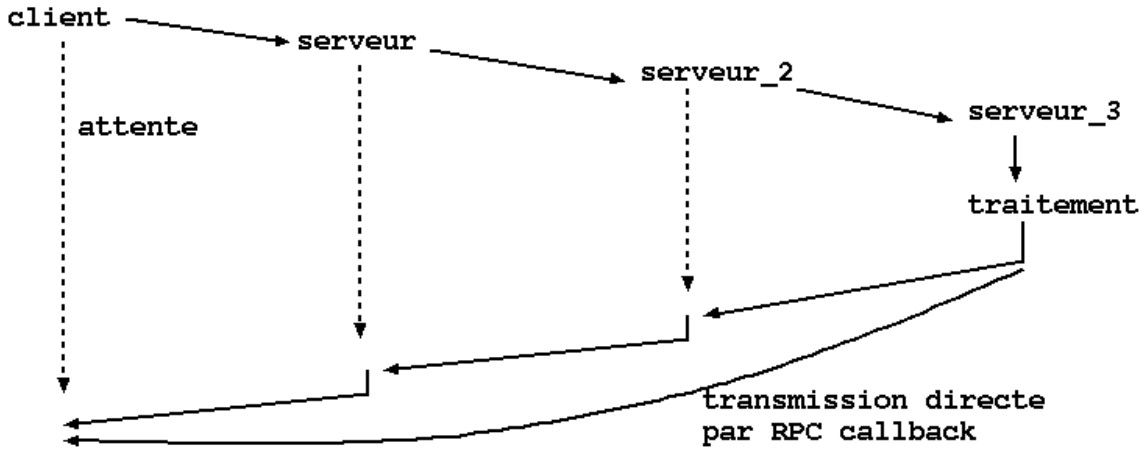


FIG. 94: RPC callback

On prépare l'appel en callback en créant deux interfaces. Par exemple si on utilise rpcgen :

- **ser.x** : pour l'interface du serveur,
- **serb.x** : pour l'interface "callback" du client.

Dans ser.x on ajoute une fonction info(adr_callback) qui va permettre au client de donner au serveur le nom de la machine du client (host) et l'identité du seveur callback qu'il aura au préalable enregistré auprès du portmapper de la machine cliente.

Puis, le client envoie sa requête au serveur. Le serveur renvoie une réponse vide qui sert d'acquitement, et le serveur fork un fils qui va traiter la requête et faire un appel RPC vers le serveur callback du client pour transmettre au client les résultats de sa requête.

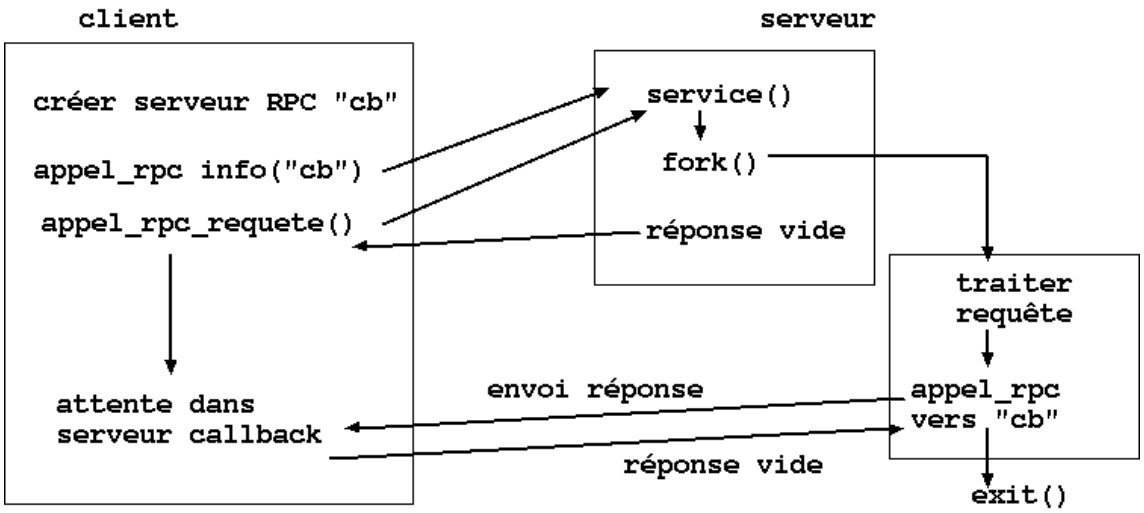


FIG. 95: RPC callback : mécanisme

SR03 2004 - Cours Architectures Internet - Le Web et http

9 SR03 2004 - Cours Architectures Internet - Le Web et http

9.1 Le Web et http : fonctionnement du protocole

Le "oueb" est fondé sur le protocle "http" (Hyper Text Transfer Protocole). Comme souvent il a une origine "accidentelle". Une technique un peu "bricolée" pour des usages internes s'est répandue partout pour une raison unique : elle est d'utilisation **facile**.

L'objectif des concepteurs était qu'un grand nombre de clients puissent accéder à des données réparties sur une collection de serveurs, sans avoir à se soucier de la localisation de ces serveurs.

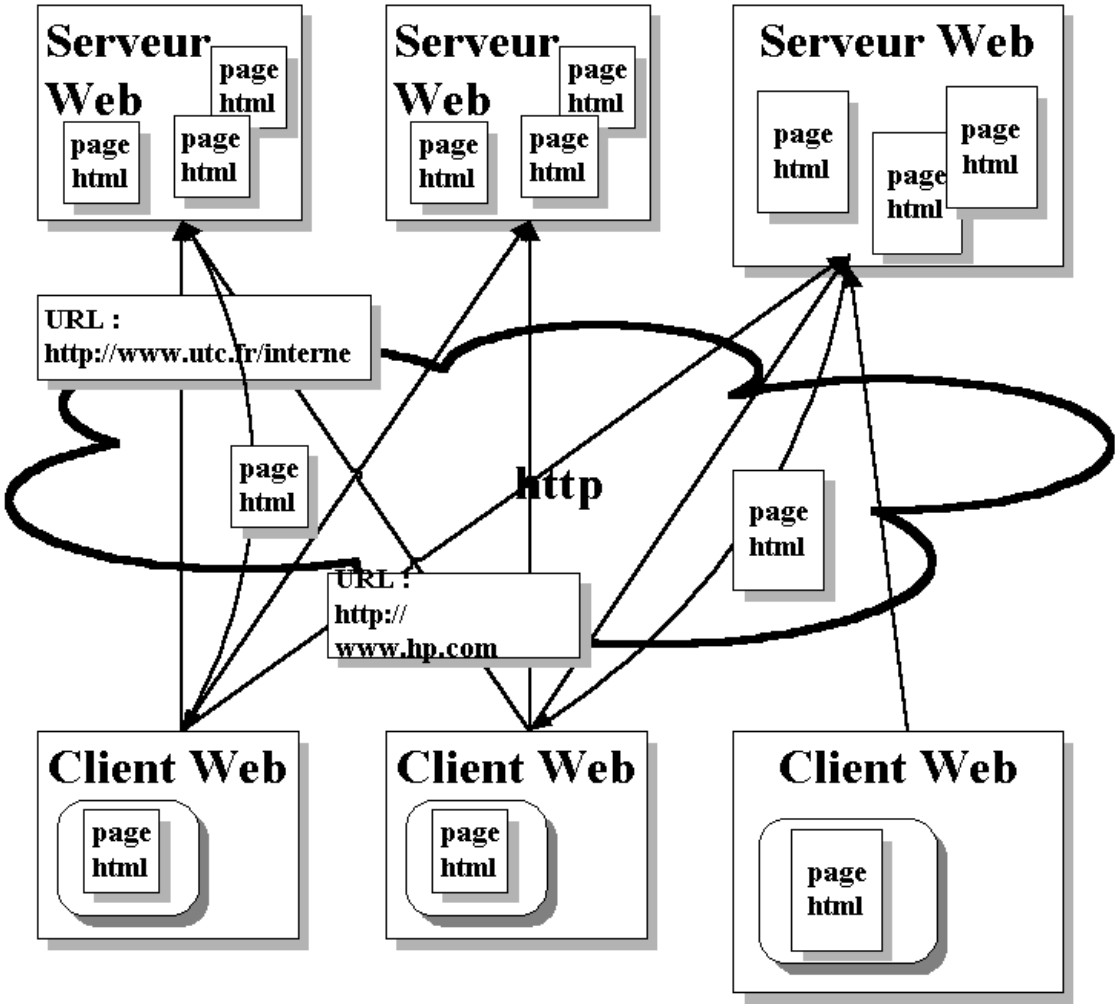


FIG. 96: Web et http

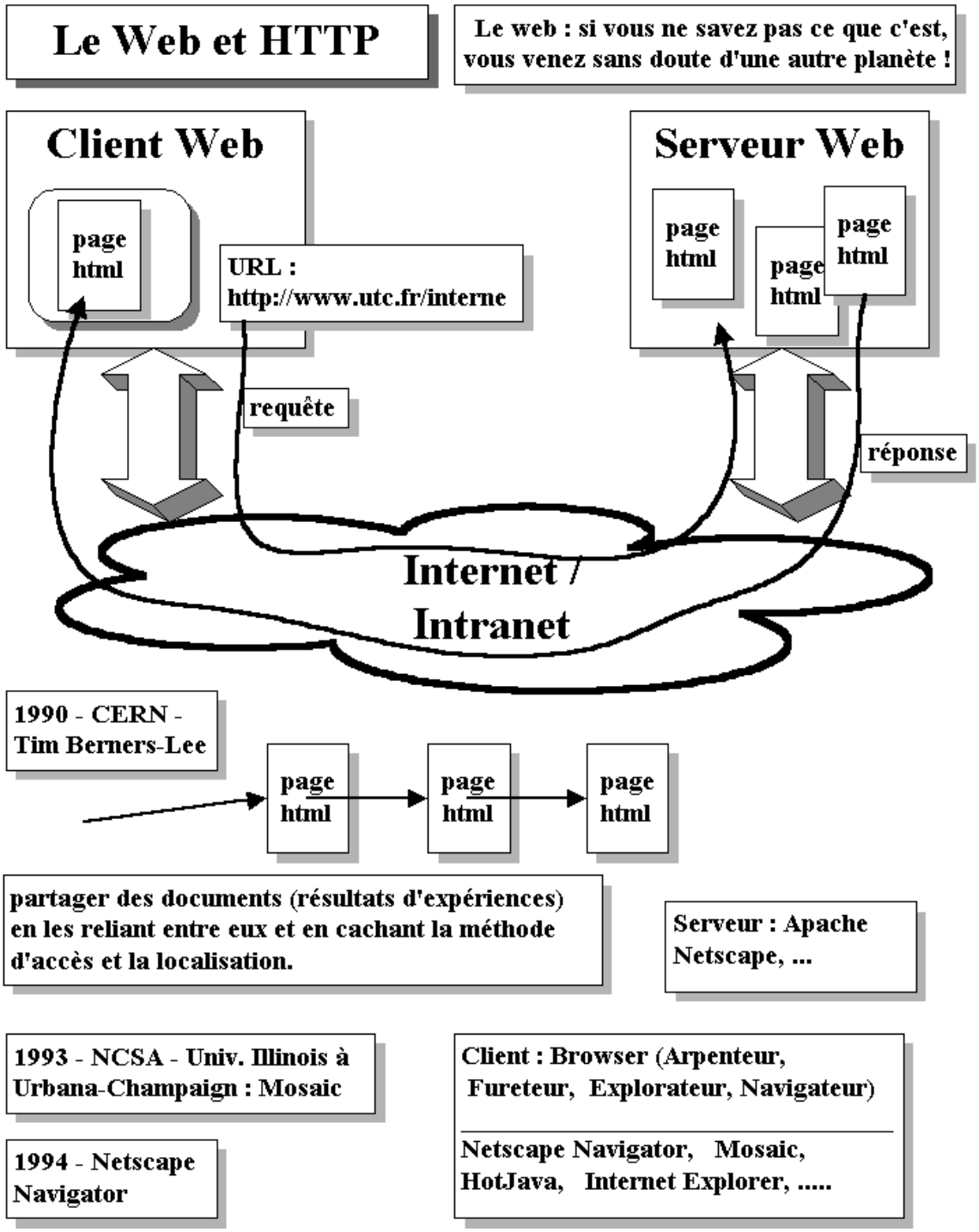


FIG. 97: Web et http

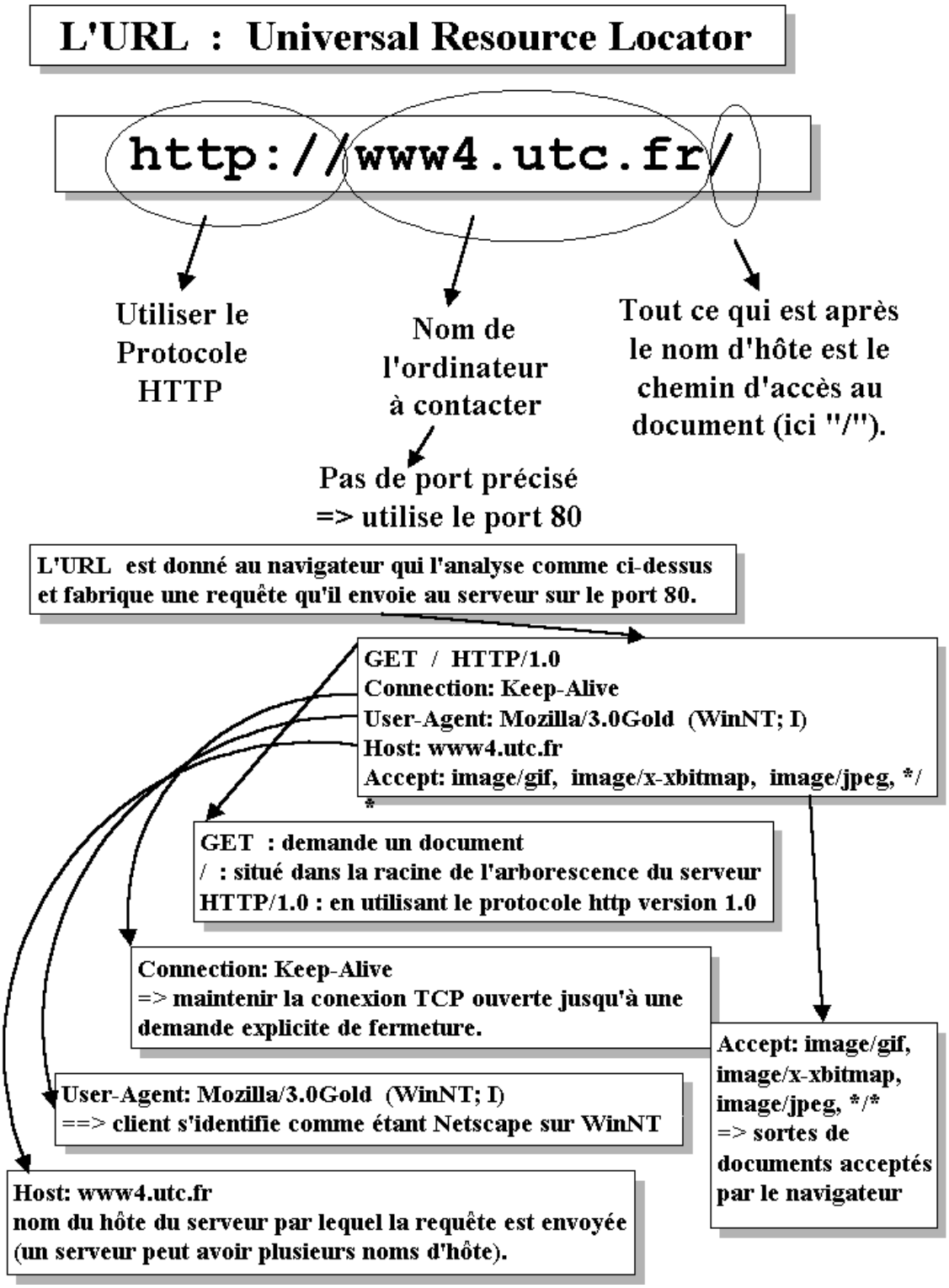


FIG. 98: Web et http

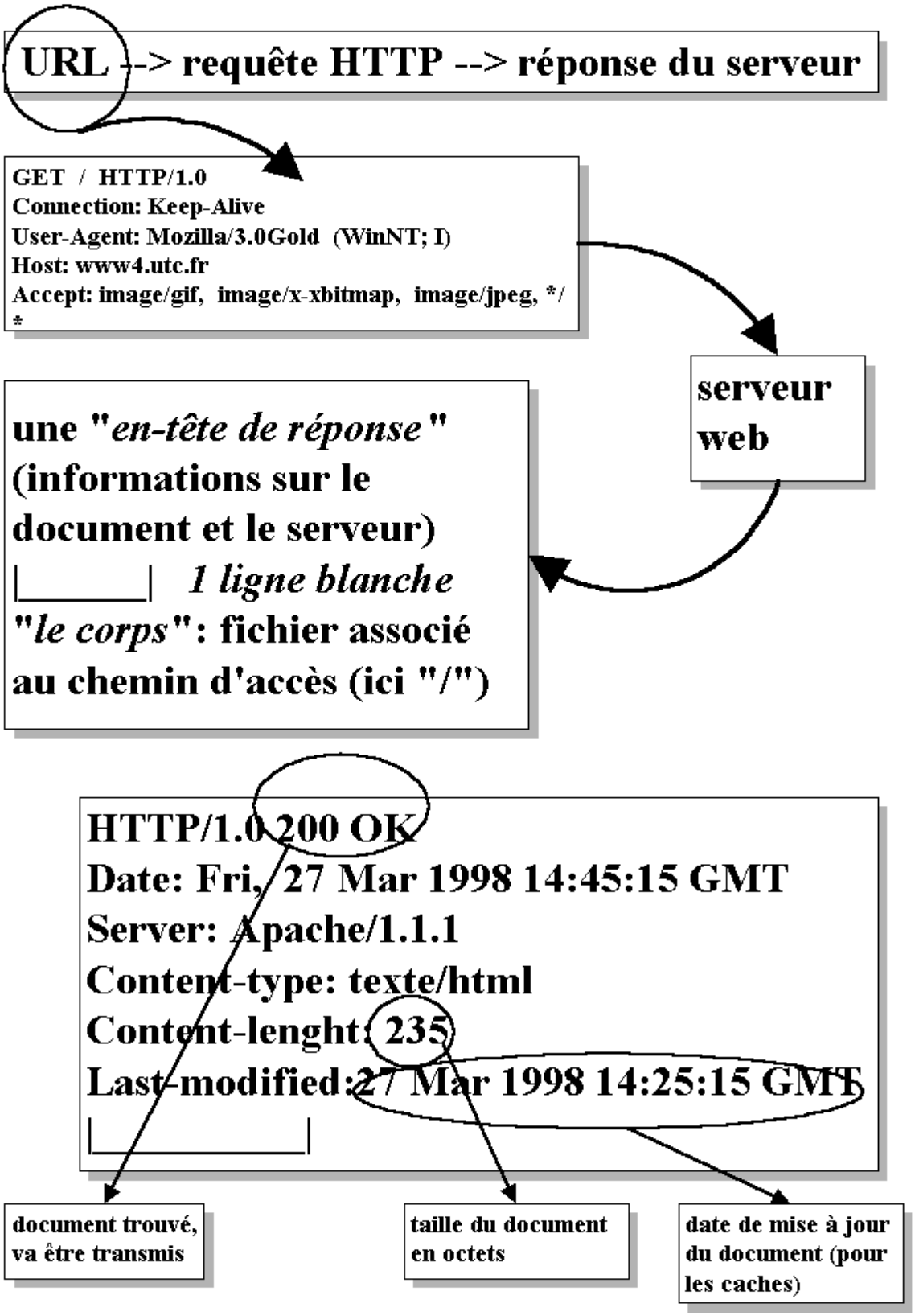


FIG. 99: Web et http

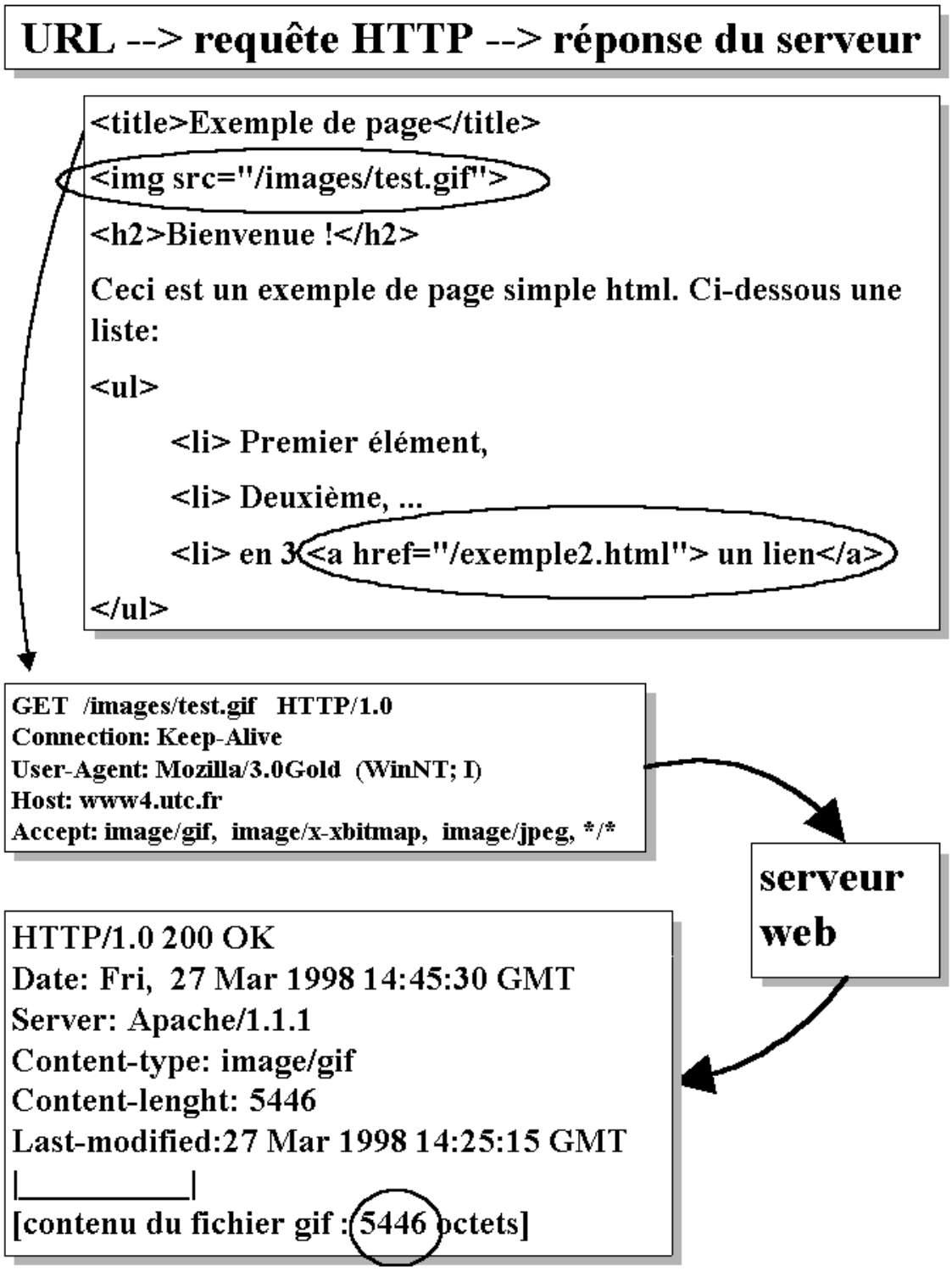


FIG. 100: Web et http

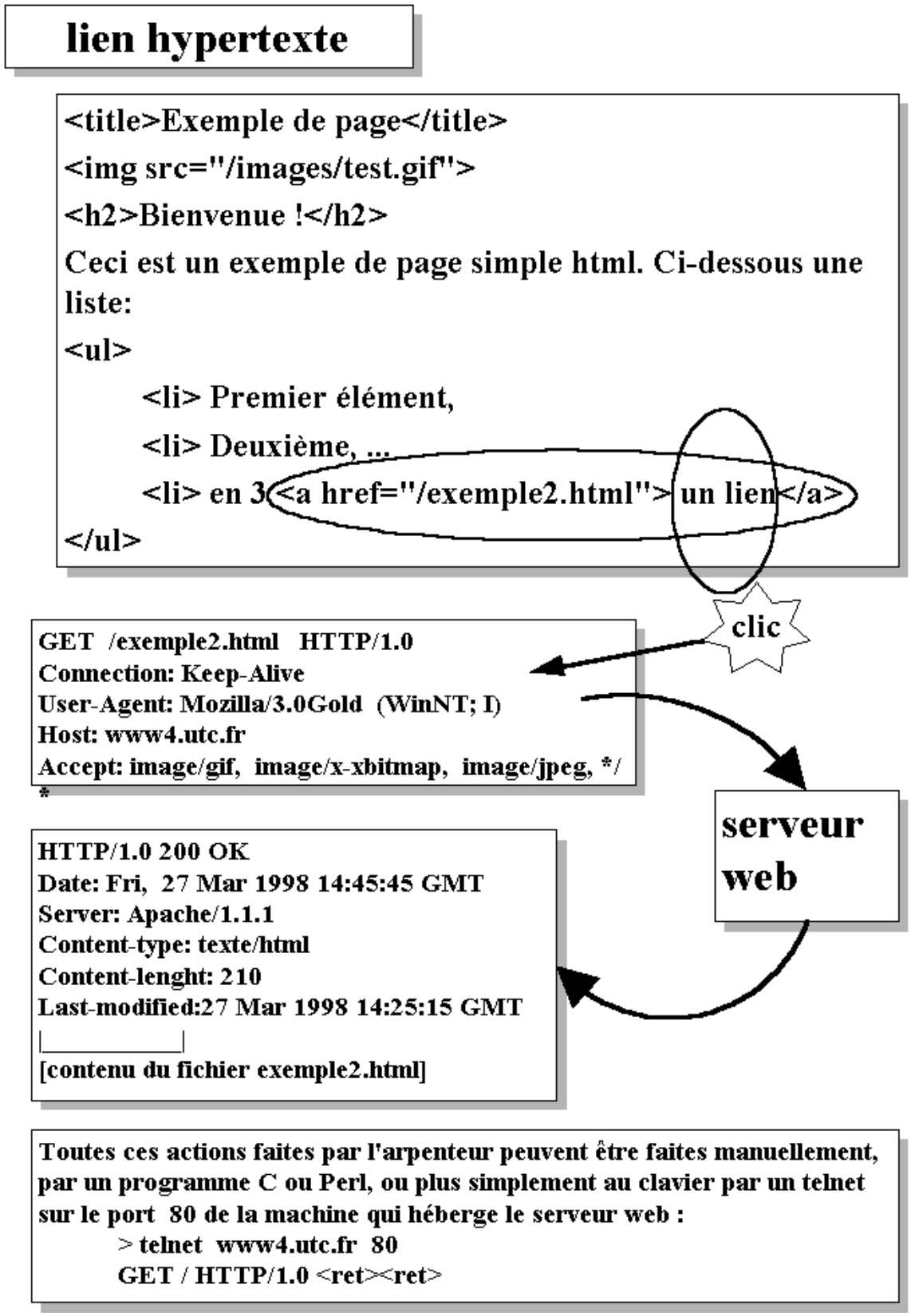


FIG. 101: Web et http

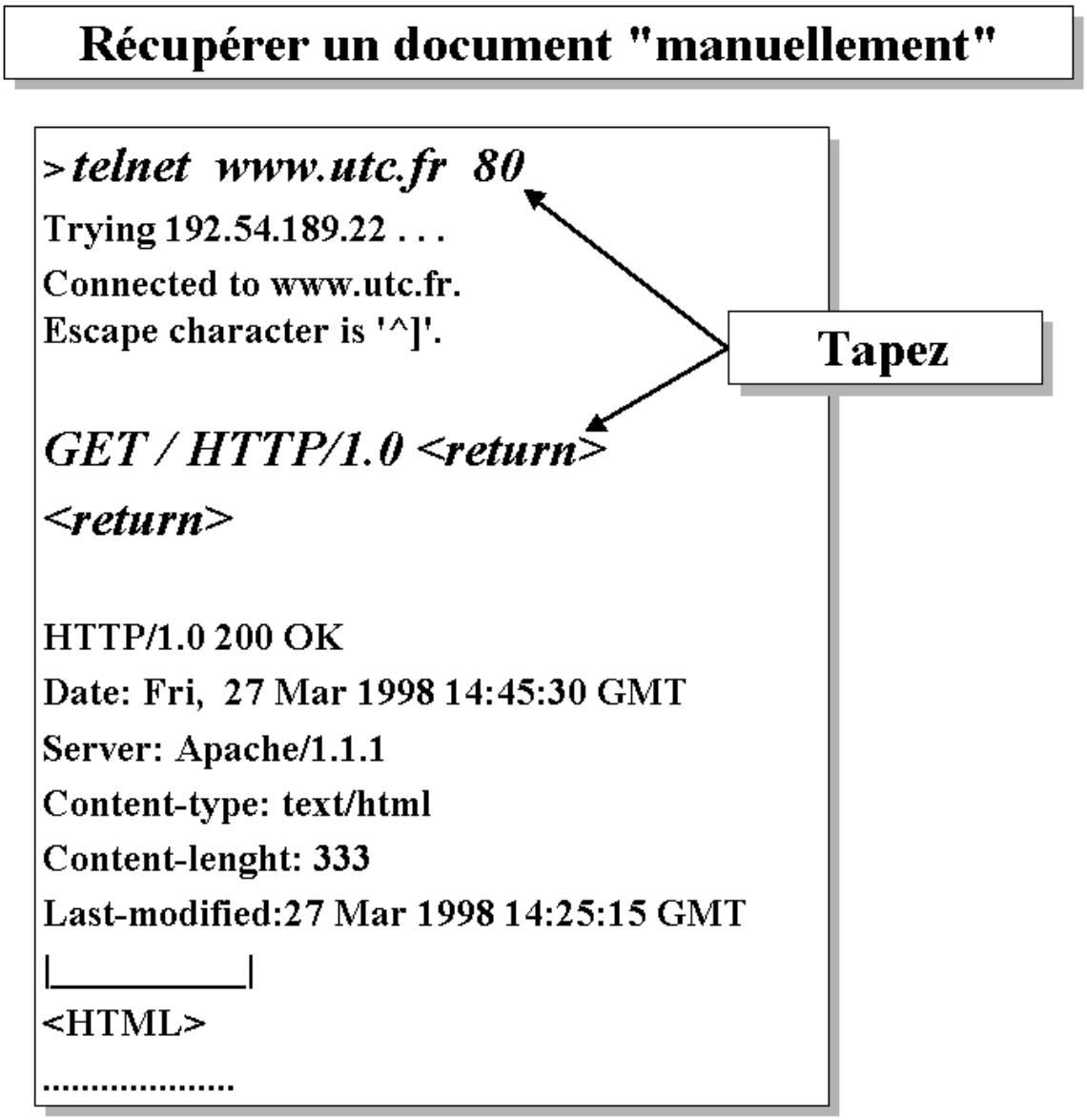


FIG. 102: Web et http

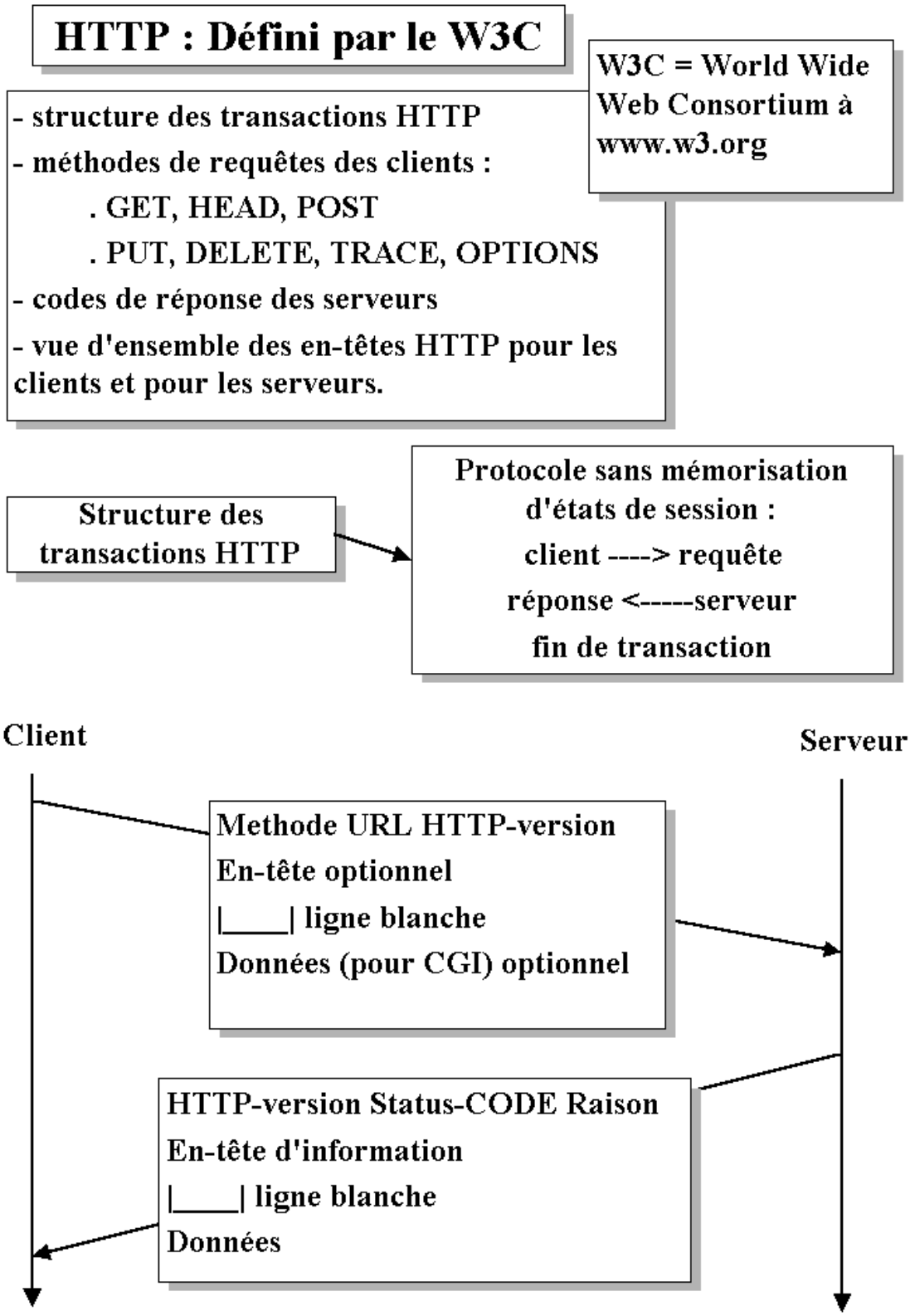


FIG. 103: Web et http

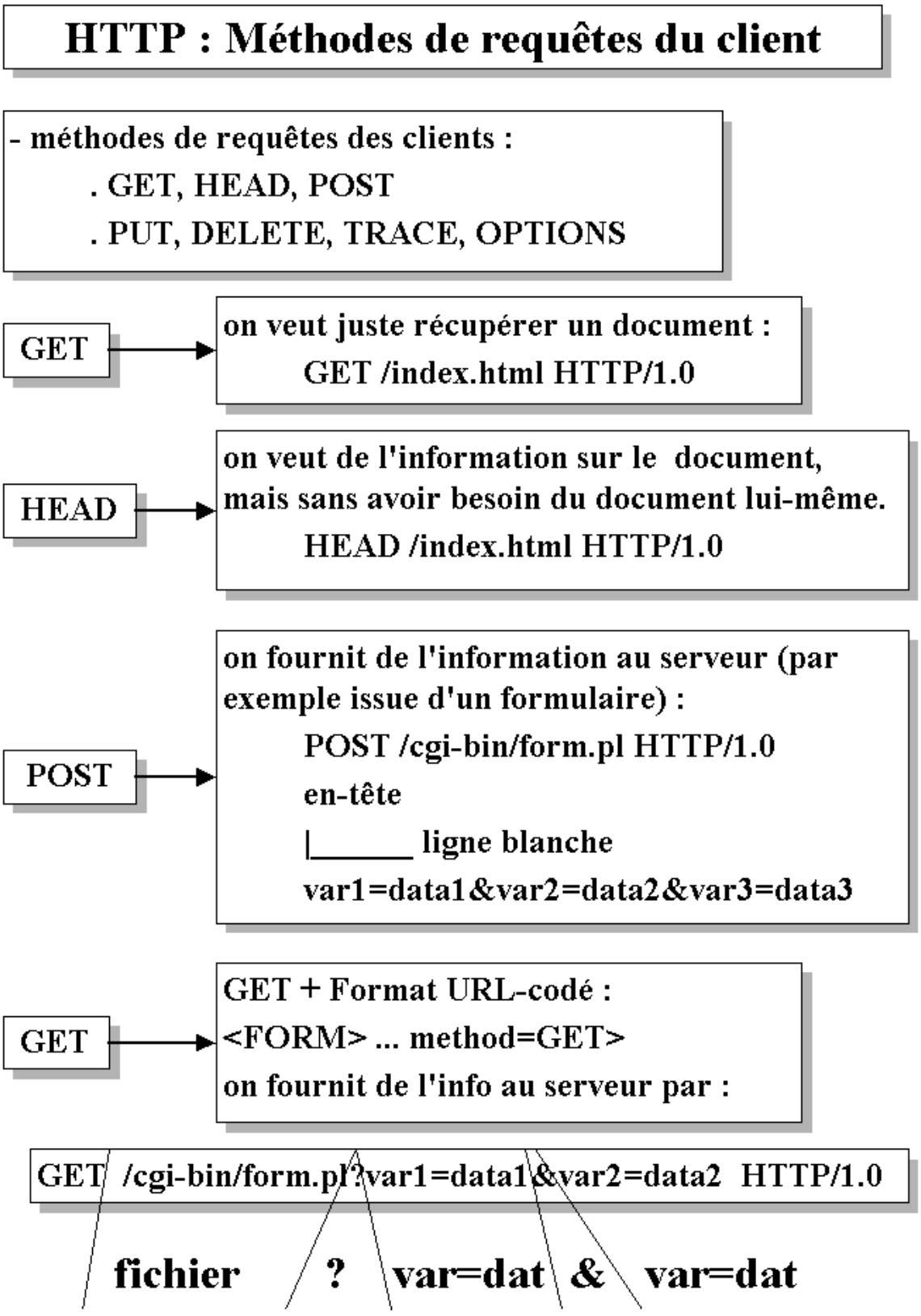


FIG. 104: Web et http

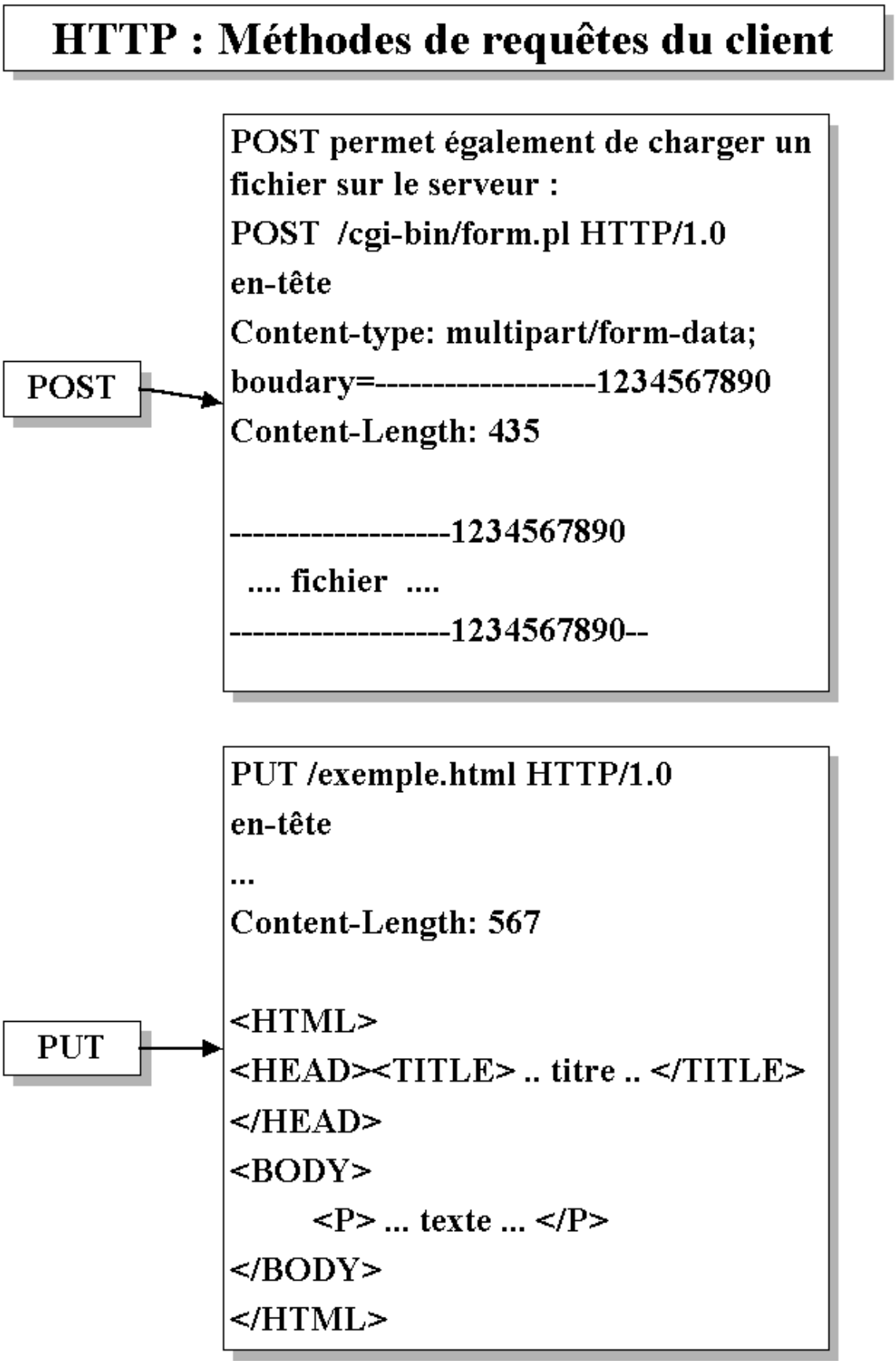


FIG. 105: Web et http

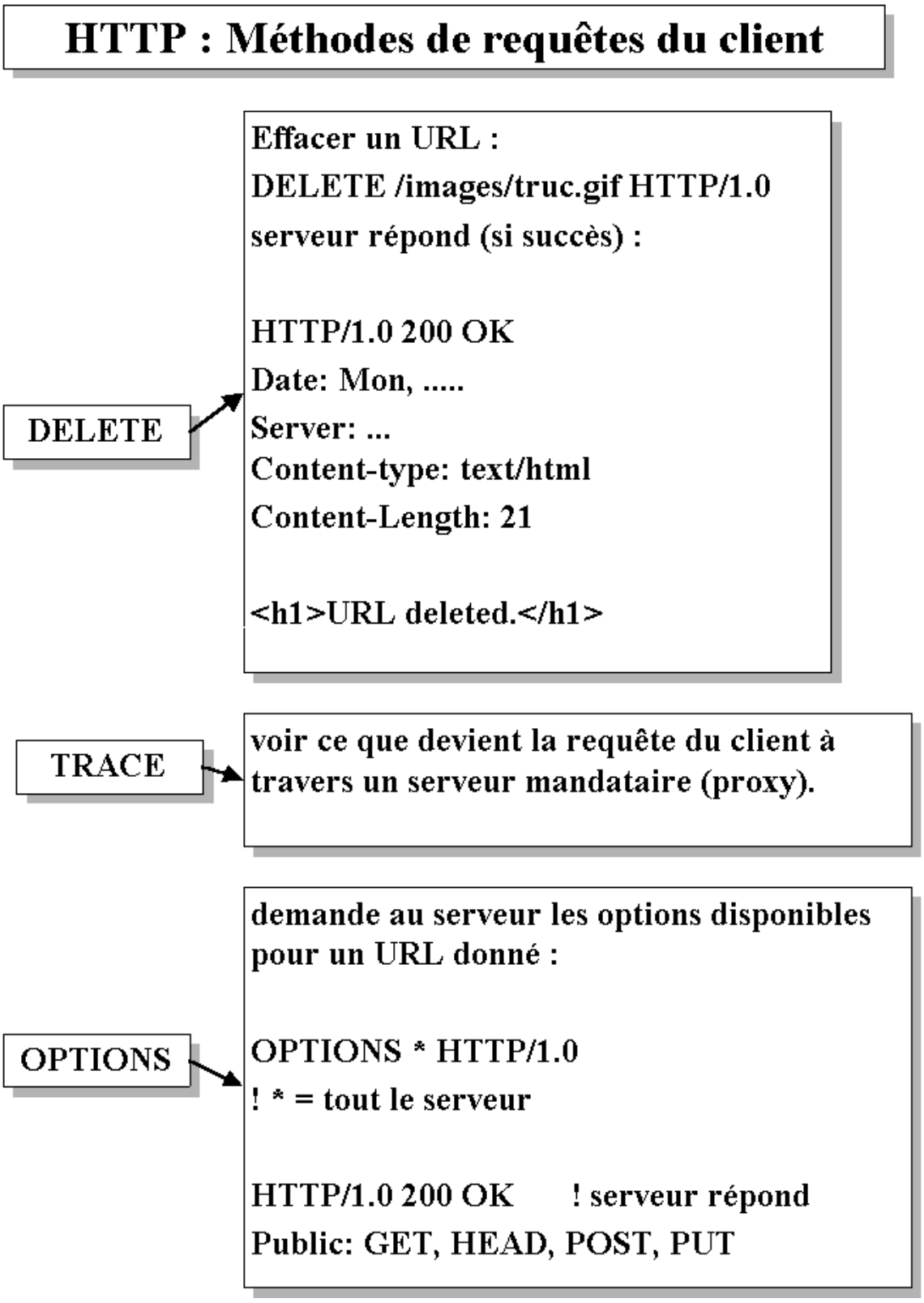


FIG. 106: Web et http

HTTP : Codes de réponse du serveur

Série	Signification
100-199	Information
200-299	Requête client réussie
300-399	Requête client redirigée, autre action nécessaire pour récupérer le document
400-499	Requête client incomplète ou erronée
500-599	Erreur dans le serveur

HTTP : en-têtes

en-têtes généraux (clients et serveurs) :
information: date, keep-alive, ...

en-tête de requête (clients) :
config du client et format des documents souhaités

en-tête de réponse (serveurs) :
config du serveur et infos sur le document envoyé

en-tête d'entité (serveurs et clients (POST ou PUT)) :
décrivent le format des données envoyées

FIG. 107: Web et http

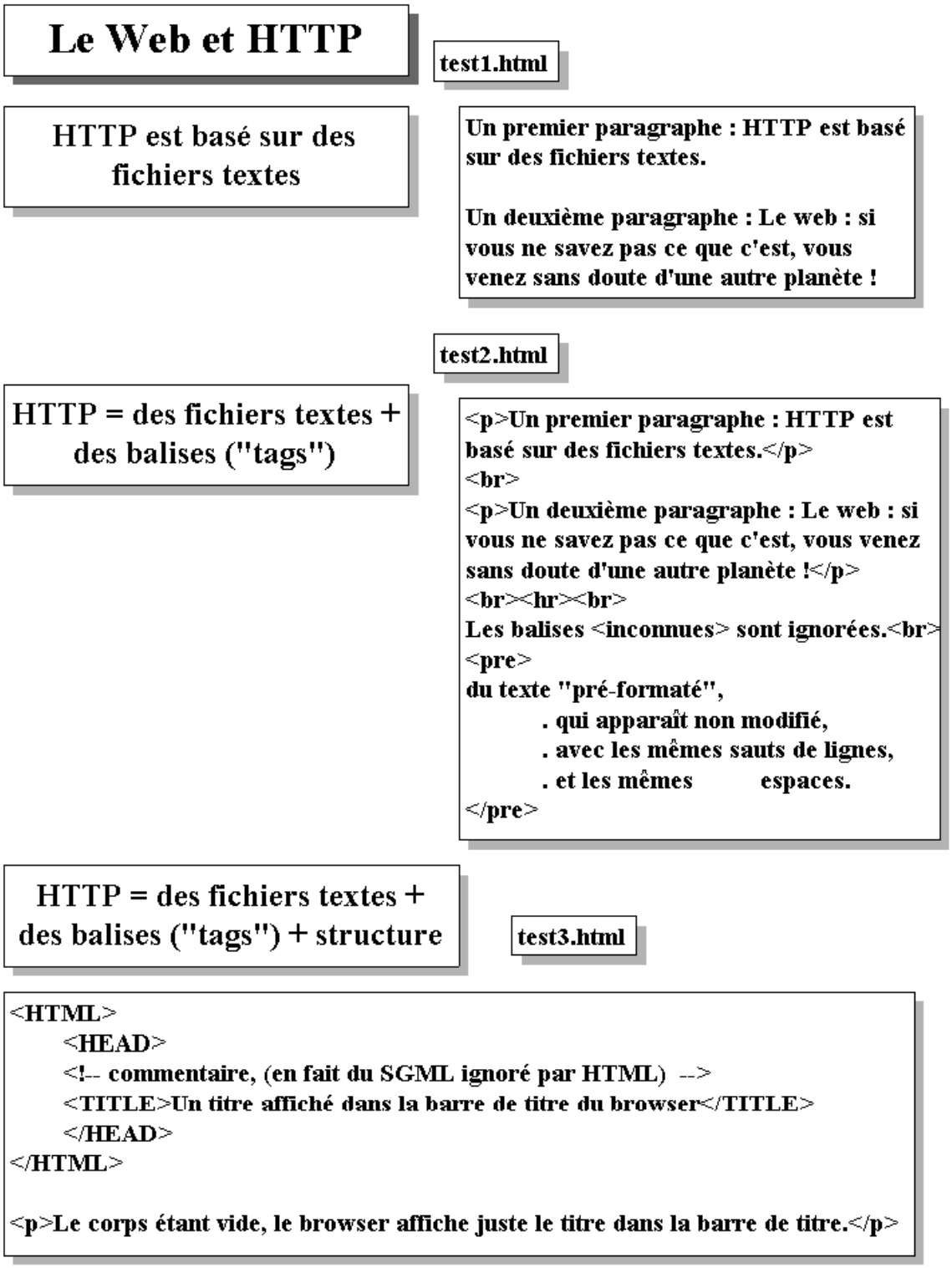


FIG. 108: Web et http

HTTP = des fichiers textes + des balises ("tags") + structure

test4.html

```
<HTML>
      <HEAD>
      <!-- commentaire, (en fait du SGML ignoré par HTML)  -->
      <TITLE>Un titre affiché dans la barre de titre du browser</TITLE>
      </HEAD>

      <BODY>
      <H1>Ceci est un titre de niveau 1 (le plus gros)</H1>
<p><b>Un premier paragraphe :</b> HTTP est basé sur des fichiers
textes.</p>
<br>
<p><i>Un deuxième paragraphe :</i> Le web : si vous ne savez pas ce que
c'est, vous venez sans doute d'une autre planète !</p>
<p><b><i>Un troisième :</i></b></p> <u>exemple souligné.</u>
<br><hr><br>
Les balises <inconnues> sont ignorées.<br>
<pre>
du texte "pré-formaté",
        . qui apparaît non modifié,
        . avec les mêmes sauts de lignes,
        . et les mêmes        espaces.
</pre>
      </BODY>
</HTML>
```

Attention ! ---> HTML décrit la structure du document, mais pas son format : celui-ci dépend de la configuration du browser au moment de la lecture.

FIG. 109: Web et http

HTTP = des fichiers textes + des balises ("tags") + structure + liens hypertextes

test5.html

```
<HTML>
    <HEAD>    <TITLE>titre</TITLE>    </HEAD>
    <BODY>
    <H3>Titre de niveau 3 </H3>
<p><b>Un premier paragraphe :</b> Exemple de liens HTTP.</p>
<br>
Un lien "absolu" :
<A HREF="http://www.utc.fr/interne">Aller à la page interne de l'UTC</A>
<br>
Un lien "relatif" : <A HREF="test3.html">voir test3.html</A>
<br><hr><br>

<pre> Pour "dessiner" avec du texte "pré-formaté", il faut utiliser une fonte fixe
telle que "Courier" :
<font face="Courier New" size=+2 color="#ffff00">
      +-------------+
      | un "dessin" |
      +-------------+
</font></pre>
    </BODY>
</HTML>
```

Couleur : color="#RRGGBB"

RR = 00 à FF valeurs 0 à 256 du rouge,
GG = idem pour le vert et BB pour le bleu.
Donc : "#FFFF00" = rouge + vert + 0 = jaune.

FIG. 110: Web et http

HTTP = des fichiers textes + des balises ("tags") + structure + liens hypertextes

test6.html

```
<HTML>
<HEAD><TITLE>Exemple de document avec renvois internes</TITLE></
HEAD>
<BODY>
<A NAME="debut"></A><!-- définition d'un étiquette -->
<H3>Exemple de document avec renvois internes : début.</H3>
<UL>
<LI><A HREF="#label1">Texte 1</A></LI>
<LI><A HREF="#label2">Texte 2</A></LI>
<LI><A HREF="#label3">Texte 3</A></LI>
</UL>
<HR>
<A NAME="label1"></A>
<H3>Texte 1</H3>
<P>Texte 1 ......</p>
<A HREF="#debut">debut</A></P>

<HR>
<A NAME="label2"></A>
<H3>Texte 2</H3>
<P>Texte 2 ......</p>
<A HREF="#debut">debut</A></P>

<HR>
<A NAME="label3"></A>
<H3>Texte 3</H3>
<P>Texte 3 ..... On peut aussi faire référence à une étiquette
dans un autre fichier :
<A HREF="test7.html#debut">aller à "debut" dans test7.html.</p>
<A HREF="#debut">debut</A></P>

<br><hr>
</BODY>
</HTML>
```

FIG. 111: Web et http

HTTP : Utilisation des tableaux pour formatter une page

test7.html

```
<HTML>
<HEAD>
<TITLE>Exemple de page construite avec une table</TITLE>
</HEAD>
<BODY>
<H3>Exemple de page construite avec une table.</H3>
<table cellspacing=10 cellpadding=5 width=100% border=1>
<tr>
        <td width=10%> cellule de gauche </td>
        <td width=80%> cellule du milieu </td>
        <td width=10%> cellule de droite </td>
</tr>
<tr>
        <td width=10%> </td>
        <td width=80%>
<p>cellspacing=10  => espace entre deux cellules</p>
<p>cellpadding=5  => espace entre la bordure d'une cellule et le texte où
l'image à l'intérieur de la cellule.</p>
<p>width=100%  => utiliser toute la largeur de la page.</p>
<p>border=1  => un pixel de large pour la bordure : trace un cadre.</p>
                                </td>
        <td width=80%> </td>
</tr>
</table>
<br><hr>
</BODY>
</HTML>
```

FIG. 112: Web et http

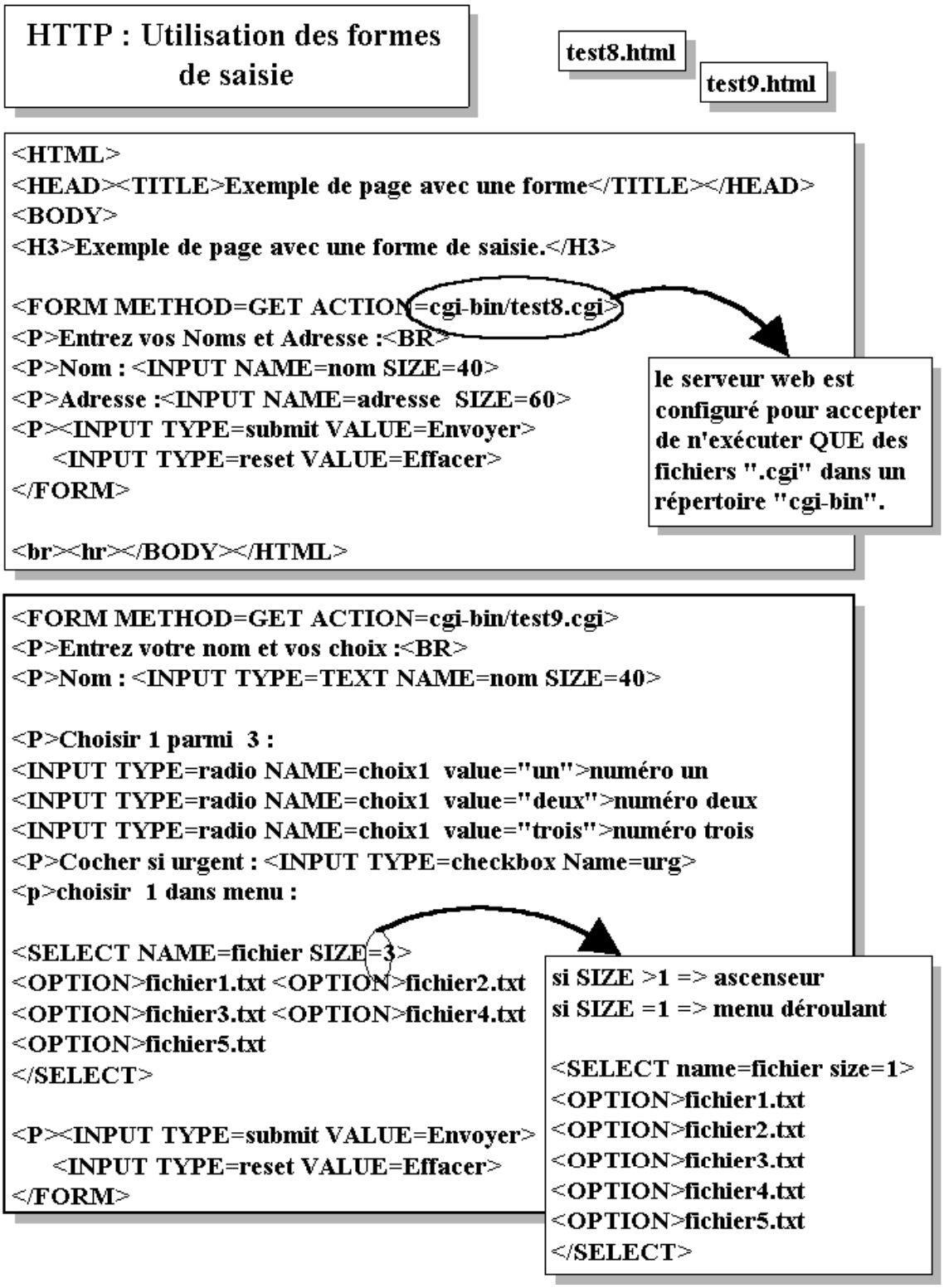


FIG. 113: Web et http

9.2 Client "pull"

Il est possible, de faire demander un rafraichissement automatique de la page par le navigateur. C'est alors le navigateur qui redemande la page : le serveur n'a pas besoin de mémoriser l'identité du client.

doc.html

```
 1  <HTML>
 2  <HEAD>
 3  <!- demo client pull : HTML 3.0 (Netscape Navigator >= 1.1) ->
 4  <META HTTP-EQUIV="Refresh" CONTENT=5>
 5  <TITLE>Utilisation client pull
 6  pour appeler un autre document</TITLE>
 7  </HEAD><BODY>
 8  <P>Insérer du texte ou des images ici</P>
 9  <img src="penguin.png">
10
11  </BODY></HTML>
```

doc.php3

```
 1  <HTML>
 2  <HEAD>
 3  <!- demo client pull : HTML 3.0 (Netscape Navigator >= 1.1) ->
 4  <META HTTP-EQUIV="Refresh" CONTENT=5>
 5  <TITLE>Utilisation client pull
 6  pour appeler un autre document</TITLE>
 7  </HEAD><BODY>
 8  <P>Insérer du texte ou des images ici</P>
 9  <P>Date, Heure =
10  <? setlocale ('LC_TIME', 'fr');
11  print(strftime("%A %d %B %Y %H :%M :%S"));?>
12
13  </BODY></HTML>
```

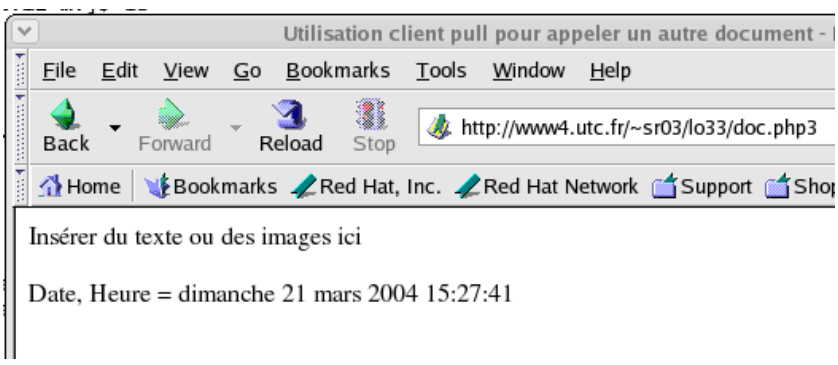


FIG. 114: doc.php3

Selon ce printcipe, on peut mettre en place des chaines de pages qui s'appellent les unes les autres.

doc1.html

```
 1  <HTML><HEAD>
 2  <META HTTP-EQUIV="Refresh" CONTENT="5; URL=doc2.html">
 3
 4  <TITLE>Utilisation client pull
 5  pour appeler un autre document</TITLE>
 6  </HEAD><BODY>
 7  <font size='+2'>
 8  <P>Insérer du texte ou des images ici</P>
 9  <P>ici doc1.html qui va appeler doc2.html</P>
10  </font>
11  </BODY></HTML>
```

doc2.html

```
 1  <HTML><HEAD>
 2  <META HTTP-EQUIV="Refresh" CONTENT="5; URL=doc1.html">
 3
 4  <TITLE>Utilisation client pull
 5  pour appeler un autre document</TITLE>
 6  </HEAD><BODY>
 7  <font size='+2'>
 8  <P>Insérer du texte ou des images ici</P>
 9  <P>ici doc2.html qui appelle doc1.html</P>
10  </font>
11  </BODY></HTML>
```

Un autre exemple, avec une interaction avec le client est donné ci-dessous.

sdoc1.html

```
 1  <HTML><HEAD>
 2  <META HTTP-EQUIV="Refresh" CONTENT="5; URL=sdoc2.html">
 3
 4  <TITLE>Utilisation client pull
 5  pour appeler un autre document</TITLE>
 6  </HEAD><BODY>
 7  <font size='+2'>
 8  <P>Insérer du texte ou des images ici</P>
 9  <P>ici sdoc1.html :  clic sur stop ou appelle sdoc2.html</p>
10  <CENTER>
11  <A HREF="stop.html"><IMG SRC="stop.gif" BORDER=0></A>
12  </CENTER>
```

sdoc1.html (suite)

```
13 </font>
14 </BODY></HTML>
```

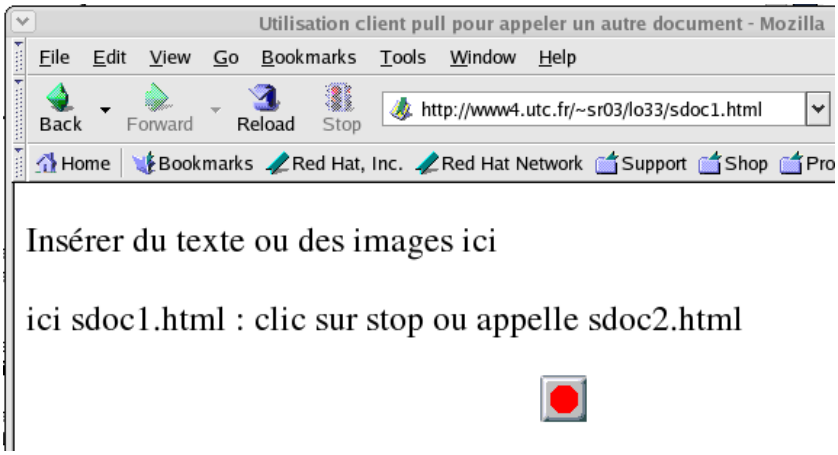


FIG. 115: sdoc1.html

sdoc2.html

```
 1 <HTML><HEAD>
 2 <META HTTP-EQUIV="Refresh" CONTENT="5; URL=sdoc3.html">
 3
 4 <TITLE>Utilisation client pull
 5 pour appeler un autre document</TITLE>
 6 </HEAD><BODY>
 7 <font size='+2'>
 8 <P>Insérer du texte ou des images ici</P>
 9 <P>ici sdoc2.html qui appelle sdoc3.html</P>
10 </font>
11 </BODY></HTML>
```

FIG. 116: sdoc2.html

sdoc3.html

```
 1  <HTML><HEAD>
 2  <META HTTP-EQUIV="Refresh" CONTENT="5; URL=sdoc1.html">
 3
 4  <TITLE>Utilisation client pull
 5  pour appeler un autre document</TITLE>
 6  </HEAD><BODY>
 7  <font size='+2'>
 8  <P>Insérer du texte ou des images ici</P>
 9  <P>ici sdoc3.html qui appelle sdoc1.html</P>
10  </font>
11  </BODY></HTML>
```

stop.html

```
 1  <HTML><HEAD>
 2  <TITLE>Utilisation client pull
 3  pour appeler un autre document</TITLE>
 4
 5  <script language="JavaScript">
 6          function pushgo() {
 7                  alert("on continue");
 8                  location = 'sdoc1.html';
 9          }
10          function pushstop() {
11                  alert("on arrete");
12                  location = 'arret.html';
13          }
14  </script>
15
16  </HEAD><BODY>
17  <P>Insérer du texte ou des images ici</P>
18  <P>
19  <form>
20  <input  type="button" name="Button1" value="continuer?"
21          onclick="pushgo()">
22  <input  type="button" name="Button2" value="arreter?"
23          onclick="pushstop()">
24  </form>
25  </BODY></HTML>
```

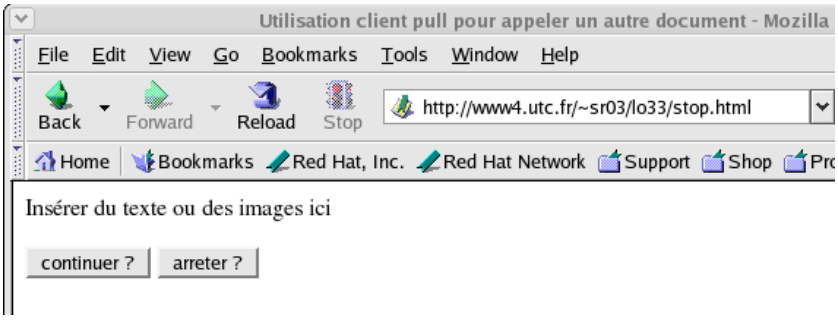


FIG. 117: stop.html

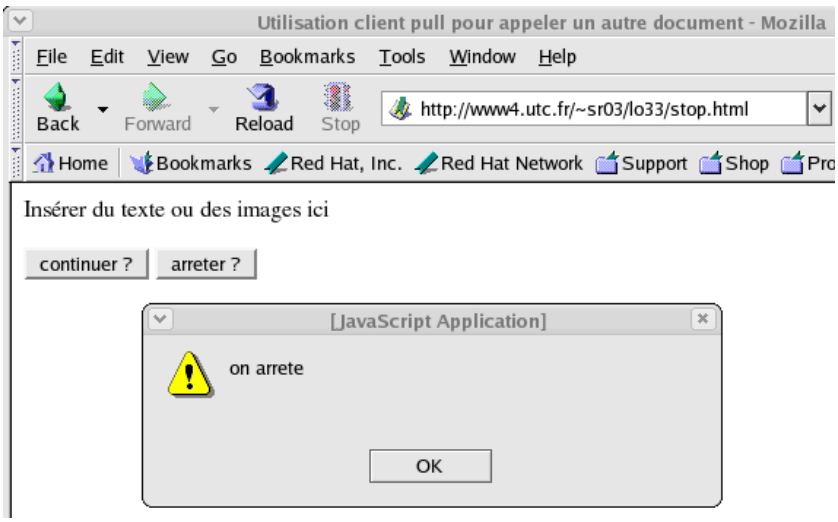


FIG. 118: stop.html javascript confirm

arret.html

```
 1 <HTML><HEAD>
 2 <TITLE>Utilisation client pull
 3 pour appeler un autre document</TITLE>
 4 </HEAD>
 5 <BODY>
 6 <P>Insérer du texte ou des images ici</P>
 7 <font size='+2'>
 8 <P>ici arr&ecirc;t.html</p>
 9 </font>
10 <hr>
11 </BODY></HTML>
```

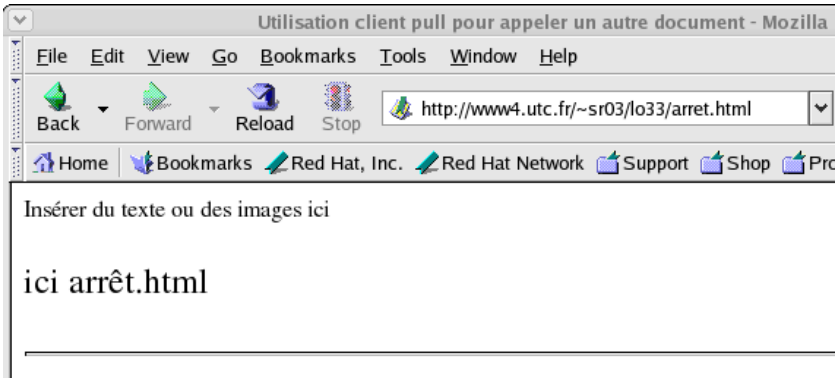


FIG. 119: arret.html

SR03 2004 - Cours Architectures Internet - Le Web et http
Fin du chapitre.

SR03 2004 - Cours Architectures Internet - Les CSS : feuilles de style en cascade

10 SR03 2004 - Cours Architectures Internet - CSS

10.1 Première feuille CSS : css01.htm

css01.htm

```
 1  <HTML><HEAD><TITLE>fichier css01.htm</TITLE>
 2  <STYLE TYPE="text/css">
 3  <!- /* Hide content from old browsers */
 4  P {
 5  font-size : 10pt;
 6  font-family : Geneva, sans-serif;
 7  color : #000000;
 8  }
 9  UL {
10  font-size : 12pt;
11  font-family : Geneva, sans-serif;
12  color : #FF0000;
13  }
14  /* end hiding content from old browsers */ ->
15  </STYLE>
16  </HEAD>
17  <BODY BGCOLOR="#FFFFFF" TEXT="#000000"
18          LINK="#008080" VLINK="#000080">
19
20  <font size="+2"> Début du fichier css01.htm <br>
21  <hr>
22  <P>Paragraphe  ... bla bla bla
23  </P>
24  <hr><br>
25  <UL><LI>Paragraphe PA... bla bla bla</LI>
26  </UL>
27  <hr><br>
28  Fin du fichier css01.htm <br>
29  </font>
30  </BODY></HTML>
```

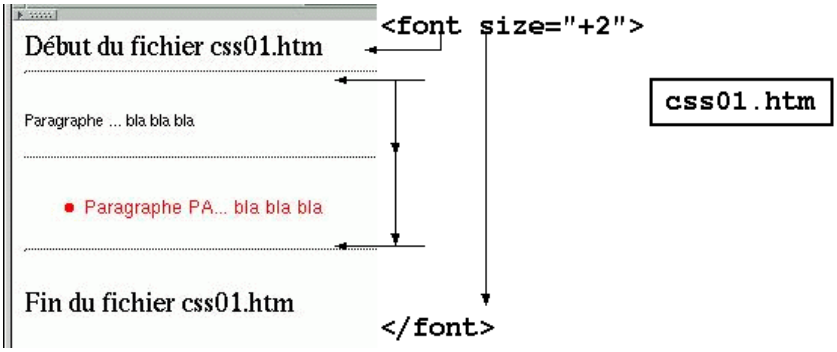


FIG. 120: css01.htm

Explications :

Le but des CSS est de séparer les informations de **"style"** du texte. Ainsi, dans l'exemple, on indique à un seul endroit comment doit apparaitre le matériel contenu dans une balise <P>...</p>. De même un autre formatage est indiqué pour les balises <UL>. On remarquera au passage que ceci implique en pratique l'obligation de **refermer** toutes les balises. On remarque aussi que la commande <font size="+2"> située au début du texte, qui est fermée seulement à la fin, devrait s'appliquer à tout le texte, mais qu'elle est temporairement annulée par le formatage imposé dans le style de la balise <P>.

Les types de CSS :

Il y a quatres types de CSS :

– inline,
– ID,
– class,
– html ;

Ils s'appliquent **dans cet ordre de priorité** : d'abord les "inline", puis, s'il y en a ceux définis par **ID**, puis ceux définis par **class**, et enfin le html classique. D'où le nom de styles "en cascade".

Attention : le mot clé "ID" étant aussi utilisé par JavaScript, il y a un conflit potentiel. Dans ce cas utiliser la méthode "class".

Exemple des types de CSS : css02.htm

css02.htm

```
 1  <HTML><HEAD><TITLE>fichier css02.htm</TITLE>
 2  <STYLE TYPE="text/css">
 3  P { font-size : 10pt; font-family : Geneva, sans-serif;
 4      color : #000000; }
 5  #pa { Background-Color : #00EEFF; color : #FF0000;
 6   font-size :12pt;}
 7  P.marge { font-size : 11pt; margin-left :2.5cm;
 8  color : #0000FF; }
 9  </STYLE>
10  </HEAD>
11  <BODY BGCOLOR="#FFFFFF" TEXT="#000000"
12          LINK="#008080" VLINK="#000080">
```

css02.htm (suite)

```
13 <font size="+2"> Début du fichier css02.htm <br>
14 Formatage "html" font size="+2".<br>
15 <P id="pa">Paragraphe formatage ID="pa" ...
16 bla bla bla bla bla bla.
17 </P>
18 <P>Paragraphe formatage de la classe P ...
19 bla bla bla bla bla bla.
20 </P>
21 <P>Paragraphe formatage de la classe P ...
22 bla bla bla bla bla bla<br>
23 mais <span id="pa">avec du texte en format
24 ID="pa"</span> au milieu du paragraphe.
25 </P>
26 <P class="marge">Paragraphe formatage de la classe
27 P.marge ... <br>
28 bla bla bla bla bla bla. Bla bla bla bla bla bla.
29 </P>
30 <P style="Background-Color :#FFFF00;
31    font-family :Helvetica; font-size :18pt;">
32 Paragraphe formatage inline
33 <span id="pa">avec du texte <br>en format ID="pa"</span>
34 au milieu du paragraphe.
35 </P>
36 Fin du fichier css02.htm <br>
37 Retour au formattage par défaut html.
38 </font>
39 </BODY></HTML>
```

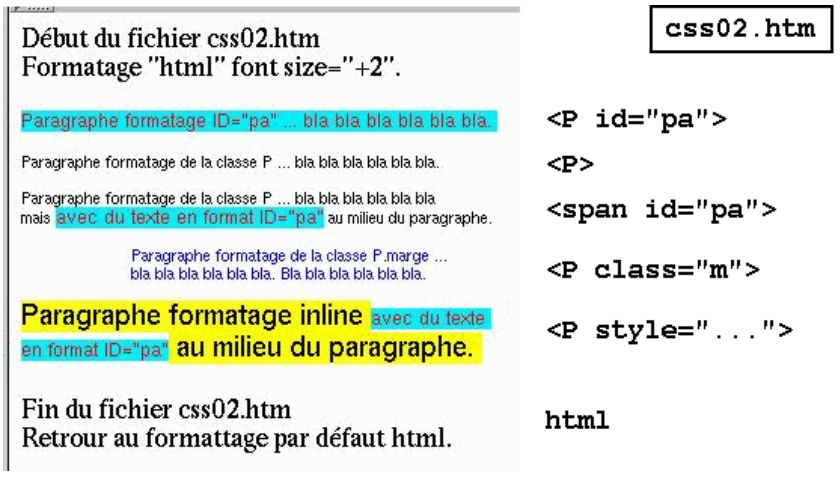


FIG. 121: css02.htm

10.2 Différentes façons d'introduire les CSS dans les pages html

Il y a quatres façon d'introduire les CSS dans les pages html :

- inline Styles : <P style="..."
- Enbeded Style Sheet (feuilles incorporées) : <style type="text/css">...</style>
- Linked Style Sheet (feuilles liées, dans un fichier externe) :
  <link rel="stylesheet" type="text/css" href="monstyle.css" />
- Imported Style Sheet (feuilles de style importées) :

```
<style type="text/css">
   @import url(mon_serveur/mon_style.css);
</style>
```

Exemple de "Linked Style" CSS : css03.htm et css03.css

css03.htm

```
 1  <HTML><HEAD><TITLE>css03.htm</TITLE>
 2
 3  <LINK HREF='css03.css' REL='stylesheet' TYPE='text/css'>
 4
 5  </HEAD>
 6  <BODY BGCOLOR="#FFFFFF" TEXT="#000000"
 7          LINK="#008080" VLINK="#000080">
 8  <font size="+2"> Formatage "html" font size="+2".<br>
 9  <hr>
10  <P id="pa">Paragraphe formatage ID="pa" ...
11  bla bla bla bla bla bla.
12  </P>
13  <P>Paragraphe formatage de la classe P  ...
14  bla bla bla
15  </P>
16  <P class="marge">Paragraphe formatage de la classe
17  P.marge ... <br>
18  bla bla bla bla bla bla. Bla bla bla bla bla bla.
19  </P>
20  <hr>
21  Retour au formattage par défaut html.
22  </font>
23  <br>html : après fermeture &lt;/font>.
24  </BODY></HTML>
```

css03.css

```
1  <STYLE TYPE="text/css">
2  <!- /* Hide content from old browsers */
3
4  P { font-size : 10pt; font-family : Geneva, sans-serif;
5      color : #000000; }
```

css03.css (suite)

```
 6  #pa { Background-Color : #00EEFF; color : #FF0000;
 7        font-size :12pt;}
 8  P.marge { font-size : 11pt; margin-left :2.5cm;
 9            color : #0000FF; }
10
11  /* end hiding content from old browsers */ ->
12  </STYLE>
```

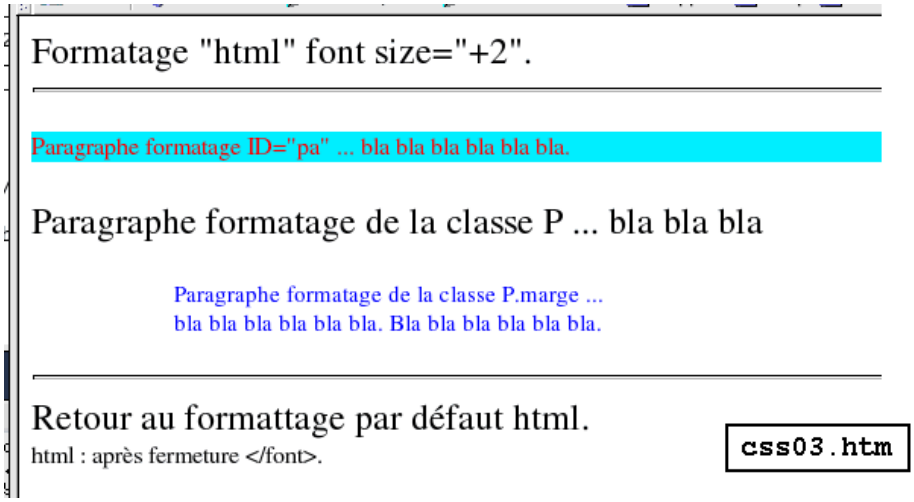


FIG. 122: css03.htm

Exemple de style CSS plus complexe : css04.htm et css04.css

css04.htm

```
 1  <HTML><HEAD><TITLE>css04.htm</TITLE>
 2
 3  <LINK HREF='css04.css' REL='stylesheet' TYPE='text/css'>
 4
 5  </HEAD>
 6  <BODY BGCOLOR="#FFFFFF" TEXT="#000000"
 7          LINK="#008080" VLINK="#000080">
 8
 9  <hr>
10  <P>Paragraphe ... bla bla bla
11   bla bla bla bla bla bla bla bla
12  </P>
13
14  <P id="P2">
15  Paragraphe ... bla bla bla bla bla bla bla bla bla bla
16   bla bla bla bla bla bla bla bla bla bla bla bla bla
17   bla bla bla bla bla bla bla bla bla bla bla
18  </P>
```

css04.htm (suite)

```
19
20  <P id="P3">
21  Paragraphe ... blu blu blu blu blu blu blu blu blu blu blu
22   blu blu blu blu blu blu blu blu blu blu blu blu blu blu
23   blu blu
24   <SPAN id="HI">blo blo blo</SPAN>
25   blu blu blu blu blu blu blu blu
26  </P>
27  <hr>
28  </BODY></HTML>
```

css04.css

```
 1  <STYLE TYPE="text/css">
 2  <!- /* Hide from old browsers - fichier css04.css - */
 3
 4  P { color : #000000;
 5  font-size : 10pt; font-family : Geneva, sans-serif; }
 6
 7  #P2 { color : #0000FF;
 8  position :absolute;top :50;left :80; width :110;
 9  font-size : 12pt; font-family : Geneva, sans-serif; }
10
11  #P3 {color :green; color :rgb(0,255,0);
12  background-color :rgb(222,222,222);
13  position :absolute;top :100;left :120;
14  font-size :14pt;width :150;
15  font-family : Geneva, sans-serif; }
16
17  #HI {color :green; background-color :rgb(255,0,0);
18  font-size :14pt; }
19
20  /* end hiding content from old browsers */ ->
21  </STYLE>
```

FIG. 123: css04.htm

Autre xemple de style CSS : css05.htm

css05.htm

```
 1  <html><head><title>css05.htm</title>
 2  <STYLE type="text/css">
 3  #retrait {
 4          text-align :justify;
 5          margin-left :10%;
 6          margin-right :10%;
 7          background-color :yellow;
 8          }
 9  .cadre{
10          font-weight :bold; text-align :center;
11          padding :5; marging :5;  border-color :black;
12          border-width :2; border-style :solid;
13          }
14  .cadreg{        float :left;
15          font-weight :bold; text-align :left;
16          width :300; padding :10; marging :10;
17          border-color :black; border-width :2;
18          border-style :solid;
19          }
20  .cadred{        float :right;
21          font-weight :bold; text-align :left;
22          width :300; background-color :yellow;
23          padding :10; marging :10; border-color :black;
24          border-width :2; border-style :solid;
25          }
26  </style>
27  </head>
28  <body>
29  <LINK HREF='css03.css' REL='stylesheet' TYPE='text/css'>
30  <P>
31     NEdit is a GUI style editor for plain text files.  It
```

css05.htm (suite)

```
32     provides mouse based editing and a streamlined editing
33     style, based  on  popular  Macintosh  and MS Windows
34     editors, for users of X workstations and X terminals.
35  </P>
36  <div id="retrait">
37  <hr>
38  <P>
39     NEdit is a GUI style editor for plain text files.  It
40     provides mouse based editing and a streamlined editing
41     style, based  on  popular  Macintosh  and MS Windows
42     editors, for users of X workstations and X terminals.
43  </P>
44  <IMG src="penguin.png" align="left">
45  <P>
46     NEdit is a GUI style editor for plain text files.  It
47     provides mouse based editing and a streamlined editing
48     style, based  on  popular  Macintosh  and MS Windows
49     editors, for users of X workstations and X terminals.
50     NEdit is a GUI style editor for plain text files.  It
51     provides mouse based editing and a streamlined editing
52     style, based  on  popular  Macintosh  and MS Windows
53     editors, for users of X workstations and X terminals.
54  </P>
55  <br clear="all">
56  <hr>
57  </div>
58
59  <P>
60     NEdit is a GUI style editor for plain text files. It
61     provides mouse based editing and a streamlined editing
62     style, based  on  popular  Macintosh  and MS Windows
63     editors, for users of X workstations and X terminals.
64  </P>
65  <hr>
66  <P class="cadreg">
67      NEdit is a GUI style editor for plain text files. It
68      pro■ vides mouse based editing and a streamlined
69      editing style.
70  </p>
71  <P>
72   Ce paragraphe se place à droite du "cadre gauche".
73     NEdit is a GUI style editor for plain text files. It
74     provides mouse based editing and a streamlined editing
75     style, based on popular Macintosh and MS Windows
76     editors, for users of X workstations and X terminals.
77  </P>
78  <P style="background-color :lightgreen;">
```

css05.htm (suite)

```
79      NEdit is a GUI style editor for plain text files. It
80      provides mouse based editing and a streamlined editing
81      style, based on popular Macintosh and MS Windows
82      editors, for users of X workstations and X terminals.
83  </P>
84  <P class="cadred">
85     NEdit is a GUI style editor for plain text files. It
86     provides mouse based editing and a streamlined editing
87     style.
88  </p>
89  <P>     Ce paragraphe se place à gauche du "cadre droit".
90     NEdit is a GUI style editor for plain text files. It
91     provides mouse based editing and a streamlined editing
92     style, based on popular Macintosh and MS Windows
93     editors, for users of X workstations and X terminals.
94  </P>
95  <hr>
96
97  </body></html>
```

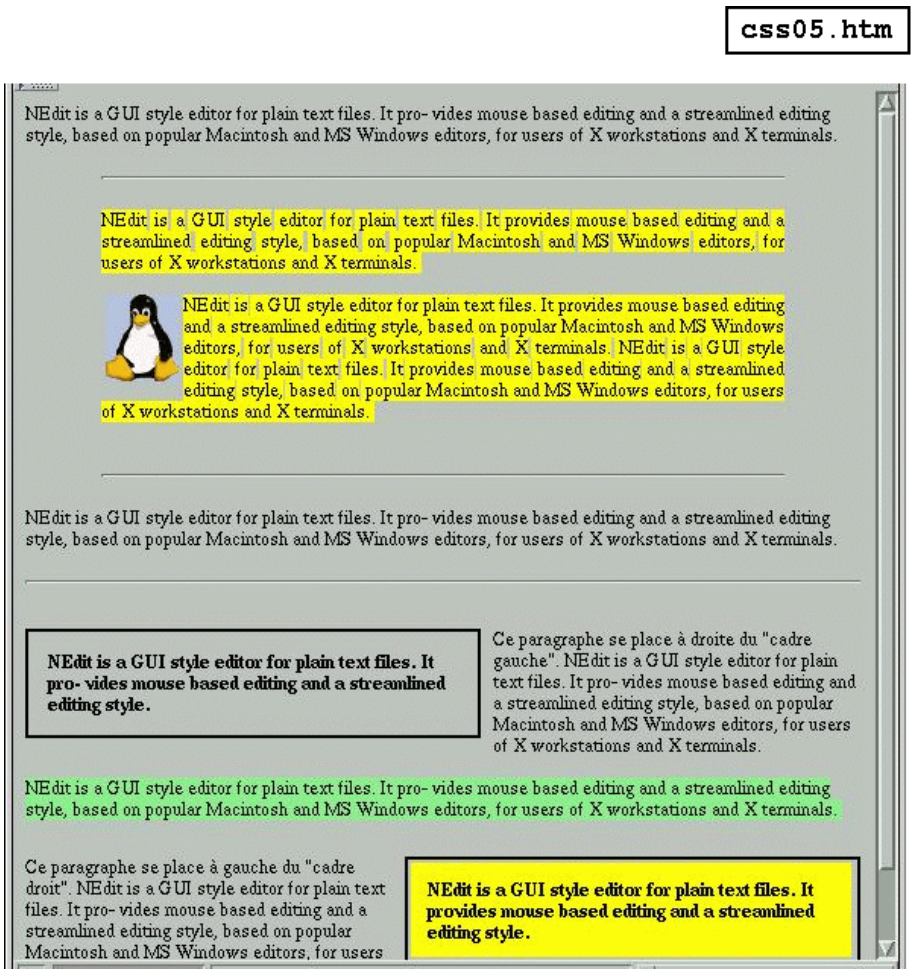


FIG. 124: css05.htm

SR03 2004 - Cours Architectures Internet - CSS
Fin du chapitre.

SR03 2004 - Cours Architectures Internet - JavaScript

11 SR03 2004 - Cours Architectures Internet - JavaScript

11.1 Le langage Javascript : intégrer JavaScript aux pages HTML

Premier script : js01.htm

js01.htm

```
 1  <html> <!---->
 2  <head>
 3  </head>
 4  Ceci est du html.<br>
 5    <script language="JavaScript">
 6  <!--  hide script from old browsers
 7       document.write("Ceci est écrit par \
 8                      JavaScript !<br><hr>")
 9  // end hiding contents -->
10    </script>
11  <br>
12  La suite est du html.<br>
13  </body>
14  </html> <!---->
```

L'exécution donne :
Ceci est du html.
Ceci est écrit par JavaScript !

La suite est du html.

Premier appel d'une fonction : js02.htm

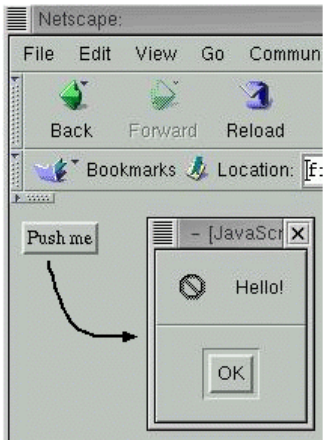
js02.htm

```
 1  <html> <!---->
 2  <head><!-- fichier js02.htm -->
 3    <script language="JavaScript">
 4       function pushbutton() {
 5         alert("Hello !");
 6    }
 7   </script>
 8  </head>
 9  <body>
10  <form>
11    <input type="button" name="Button1"
```

js02.htm (suite)

```
12    value="Push me" onclick="pushbutton()">
13   </form>
14  </body>
15  </html> <!---->
```

L'exécution donne :



js02.htm

FIG. 125: js02.htm

Appel d'une fonction : js021.htm et js022.js

Il est possible, avec les navigateurs de version supérieure ou égale à 4, de stocker le code javascript dans des fichiers séparés.

js021.htm

```
 1  <html> <!---->
 2  <head>
 3    <script language="JavaScript" src="./js022.js">
 4    </script>
 5  </head>
 6  <body>
 7  <form>
 8    <input type="button" name="Button1"
 9    value="Push me" onclick="pushbutton()">
10   </form>
11  </body>
12  </html> <!---->
```

js022.js

```
1  function pushbutton() {
2         alert("Hello !");
3  }
```

Gestion évènements souris : js03.htm

js03.htm

```
 1  <html> <!---->
 2  <head><!-- fichier js03.htm -->
 3  <script language="JavaScript">
 4  <!--  hide script from old browsers
 5    function getname(str) {
 6    document.write("ici dans getname<br>");
 7    document.write(document.URL);
 8      alert("Hi, "+ str+"!");
 9    location.replace(document.URL);
10    }
11  // end hiding contents -->
12  </script>
13  </head>
14  <body>
15  Please enter your name :
16  <form>
17    <input type="text" name="nom"
18    onBlur="getname(this.value)" value="">
19  </form>
20  </body>
21  </html> <!---->
```

Accès aux propriétés des objets : js04.htm

js04.htm

```
 1  <html> <!---->
 2  <body>
 3
 4  This is a simple HTML- page.
 5  <br>
 6
 7  Last changes :
 8
 9    <script language="JavaScript">
10    <!--  hide script from old browsers
11
12      document.write(document.lastModified)
13
14    // end hiding contents -->
15    </script>
16
17  </body>
18  </html> <!---->
```

Gestion évènements et boîte de dialogue : js05a.htm

js05a.htm

```
 1  <html> <!---->
 2  <head><!-- fichier js05a.htm -->
 3  <script language="JavaScript">
 4       function info1() {
 5         alert("mot en gras = tag <b>");
 6       }
 7       function info2() {
 8         alert("infos : explications");
 9       }
10  </script>
11
12  </head>
13  <body>
14  Cliquer sur les mots
15  <b onClick="info1()"><u>en gras soulignés</u></b> ou
16  passer la souris sur les mots en gras pour un complément
17  <b onMouseOver="info2()">d'infos</b>.<br>
18  </body>
19  </html> <!---->
```

Attention : la prise en compte des "events handler" "onClick, onMouseOver, ... n'est pas assurée par Netscape lorsque ces évènements sont déclarés dans des balises autres que <a href=...>, <input...>, <area> et <textarea>.

Le fichier js05a.htm a été testé avec succès sur Linux dans Mozilla/Galeon et Opera 6.0.

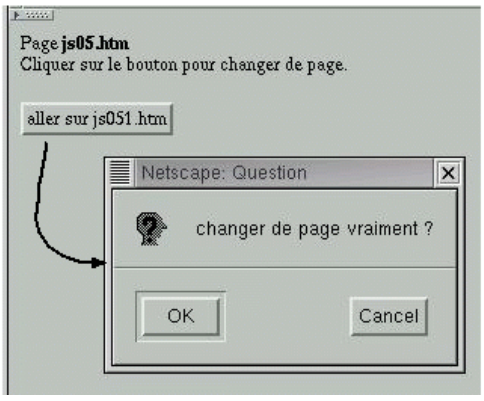
Gestion évènements et boîte de dialogue : js05.htm + js051.htm

js05.htm

```
 1  <html> <!---->
 2  <head>
 3  <script language="JavaScript">
 4       function info1() {
 5         choix = confirm("changer de page vraiment?");
 6         if (choix) {
 7           parent.location.href="js051.htm"
 8         } else {
 9           parent.location.href="js05.htm"
10         }
11       }
12  </script>
13  </head>
14  <body>
15  Page <b>js05.htm</b><br>
16  Cliquer sur le bouton pour changer de page.<br>
17  <form>
18   <INPUT TYPE="button" VALUE="aller sur js051.htm"
19          onClick="info1()">
20  </form>
21  </body>
22  </html> <!---->
```

js051.htm

```
 1  <html> <!---->
 2  <head>
 3  <script language="JavaScript">
 4       function info1() {
 5         choix = confirm("changer de page vraiment?");
 6         if (choix) {
 7           parent.location.href="js05.htm"
 8         } else {
 9           parent.location.href="js051.htm"
10         }
11       }
12  </script>
13  </head>
14  <body>
15  Page <b>js051.htm</b><br>
16  Cliquer sur le bouton pour changer de page.<br>
17  <form>
18   <INPUT TYPE="button" VALUE="aller sur js05.htm"
19          onClick="info1()">
20  </form>
21  </body></html> <!---->
```



js05.htm

FIG. 126: js05.htm

Suivant la réponse donnée à la boite de dialogue, on passe à js051 ou on reste sur js05.

Accès à des infos système : js06.htm

js06.htm

```
1  <script language="JavaScript">  // fichier js06.htm <!---->
2    today = new Date()
3    document.write("L'heure est : ",today.getHours()," :",
4                   today.getMinutes())
5    document.write("<br>")
6    document.write("La date est : ", today.getDate(),"/")
7    document.write(today.getMonth()+1,"/",today.getYear())
8  </script> <!---->
```

Accès à des fonctions système : js07.htm

js07.htm

```
 1  <html> <!---->
 2  <head><!-- fichier js07.htm -->
 3  <script language="JavaScript">
 4
 5  function RandomNumber() {
 6    today = new Date();
 7    num= Math.abs(Math.sin(today.getTime()));
 8    //num = Math.round(Math.abs(Math.sin (num)*1000000)) % limits;
 9    return num;
10  }
11  </script>
12  </head>
13
14  <body>
15  <script language="JavaScript">
16     document.write("Un nombre aléatoire : ", RandomNumber());
17  </script>
18  </body>
19  </html> <!---->
```

Créer une nouvelle fenêtre du navigateur : js08.htm

js08.htm

```
 1  <html> <!---->
 2  <head><!-- fichier js08.htm -->
 3  <script language="JavaScript">
 4
 5  function WinOpen() {
 6
 7     msg=open("frtest1.htm","DisplayWindow",
 8          "toolbar=no,directories=no,menubar=yes");
 9
10     msg.document.write("<HEAD><TITLE>Yo !</TITLE></HEAD>");
11
12     msg.document.write(
13     "<CENTER><h1><B>This is really cool !</B></h1></CENTER>");
14
15  }
16  </script>
17  </head>
18  <body>
19  Ouvrir une nouvelle fenêtre du navigateur :
20  <form>
21  <input type="button" name="Button1" value="Open new window"
22          onclick="WinOpen()">
```

js08.htm (suite)

```
23  </form>
24  <p>msg=open("frtest1.htm", "DisplayWindow", "toolbar=no,
25  directories=no,menubar=yes");</p>
26  <p>On peut associer à la nouvelle fenêtre les propriétés
27  suivantes :</p>
28  <ul>
29  <li>toolbar, menubar
30  <li>location, directories, status, copyhistory
31  <li>scrollbars, resizable, width=pixels, height=pixels
32  </ul>
33  </body>
34  </html> <!---->
```

11.2 JavaScript et les cadres ("frames")

Création de cadres dans le navigateur : js09.htm

js09.htm

```
 1  <HTML> <!---->
 2  <HEAD><!-- fichier js09.htm -->
 3  <title>Frames</title>
 4  </HEAD>
 5
 6    <FRAMESET ROWS="50%,50%">
 7        <FRAME SRC="frtest1.htm" name="fr1">
 8        <FRAME SRC="frtest2.htm" name="fr2">
 9    </FRAMESET>
10  </HTML>   <!---->
```

Création de cadres dans le navigateur : js091.htm

js091.htm

```
 1  <HTML> <!---->
 2  <HEAD><!-- fichier js091.htm -->
 3  <title>Frames</title>
 4  </HEAD>
 5
 6  <FRAMESET COLS="50%,50%">
 7
 8    <FRAMESET ROWS="50%,50%">
 9      <FRAME SRC="frtest1.htm">
10      <FRAME SRC="frtest2.htm">
11    </FRAMESET>
12
```

js091.htm (suite)

```
13    <FRAMESET ROWS="33%,33%,33%">
14      <FRAME SRC="cell.htm">
15      <FRAME SRC="cell.htm">
16      <FRAME SRC="cell.htm">
17    </FRAMESET>
18
19  </FRAMESET>
20  </HTML>  <!---->
```

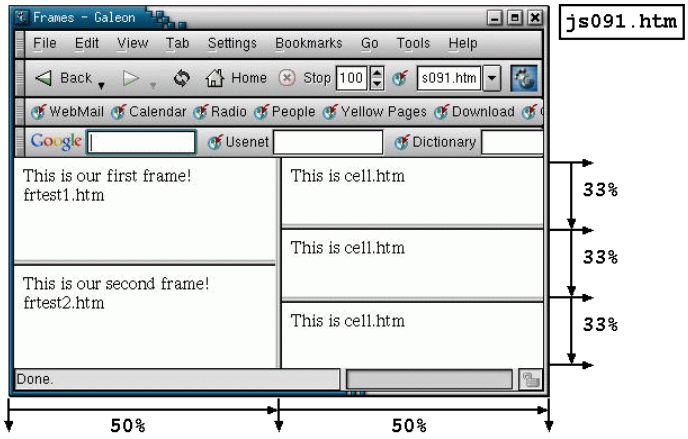


FIG. 127: js091.htm

Gestion de cadres dans le navigateur : js10.htm + js101.htm + js102.htm

js10.htm

```
1  <HTML> <!---->
2  <HEAD><!-- fichier js10.htm -->
3  <title>Frames</title>
4  </HEAD>
5
6    <FRAMESET ROWS="50%,50%">
7      <FRAME SRC="js101.htm" name="fr1" noresize>
8      <FRAME SRC="js102.htm" name="fr2">
9    </FRAMESET>
10 </HTML> <!---->
```

js101.htm

```
1  <HTML> <!---->
2  <!-- fichier js101.htm -->
3  <HEAD>
4  <script language="JavaScript">
5  <!-- Hiding
6       function hi() {
```

js101.htm (suite)

```
 7          document.write("Hi !<br>");
 8        }
 9        function yo() {
10          document.write("Yo !<br>");
11        }
12        function bla() {
13          document.write("bla bla bla<br>");
14        }
15  // -->
16  </script>
17  </HEAD><BODY>
18
19  Ceci est notre frame 1.
20
21  </BODY></HTML> <!---->
```

js102.htm

```
 1  <HTML> <!---->
 2  <!-- fichier js102.htm : -->
 3  <body>
 4  <p>
 5  Ceci est notre "frame" 2.</p>
 6
 7  <FORM NAME="buttonbar">
 8     <INPUT TYPE="button" VALUE="Hi" onClick="parent.fr1.hi()">
 9     <INPUT TYPE="button" VALUE="Yo" onClick="parent.fr1.yo()">
10     <INPUT TYPE="button" VALUE="Bla" onCLick="parent.fr1.bla()">
11  </FORM>
12  </BODY></HTML> <!---->
```

Question : Pourquoi cela "ne marche" qu'une fois ?

Gestion de cadres dans le navigateur : js11a.htm + js11fr1.htm + js11fr2.htm

js11a.htm

```
 1  <HTML> <!---->
 2  <HEAD><!-- fichier js11a.htm -->
 3  </HEAD>
 4
 5    <FRAMESET ROWS="50%,50%">
 6      <FRAME SRC="js11fr1.htm" name="fr1">
 7      <FRAME SRC="js11fr2.htm" name="fr2">
 8    </FRAMESET>
 9
10  </html> <!---->
```

js11fr1.htm

```
 1  <HTML> <!---->
 2  <HEAD><!-- fichier js11fr1.htm -->
 3  </HEAD>
 4
 5    <FRAMESET COLS="40%,60%">
 6      <FRAME SRC="jsgchild11.htm" name="g11">
 7      <FRAME SRC="jsgchild12.htm" name="g12">
 8    </FRAMESET>
 9
10  </html> <!---->
```

js11fr2.htm

```
 1  <HTML> <!---->
 2  <HEAD><!-- fichier js11fr2.htm -->
 3  </HEAD>
 4  <BODY>
 5  Ici js11fr2.htm <br>
 6  <font size="+1"><tt><pre>
 7
 8                 js11a.htm                 parent
 9                 /      \
10                /        \
11               /          \
12    fr1 (js11fr1.htm)  fr2 (js11fr2.htm)   children
13             /  \
14            /    \
15           /      \
16      gchild11  gchild12               'grandchildren'
17  ------------------------------------------------------
18  </pre></tt></font>
19
20  </body></html> <!---->
```

jsgchild11.htm

```
 1  <HTML> <!---->
 2  <HEAD><!-- fichier jsgchild11.htm -->
 3  </HEAD>
 4  <BODY>
 5  <font size="+1"><tt><pre>
 6
 7  --------------------------
 8
 9  Ici jsgchild11.htm
10
11  --------------------------
12
```

jsgchild11.htm (suite)

```
13 </pre></tt></font>
14
15 </body></html> <!---->
```

js11a.htm

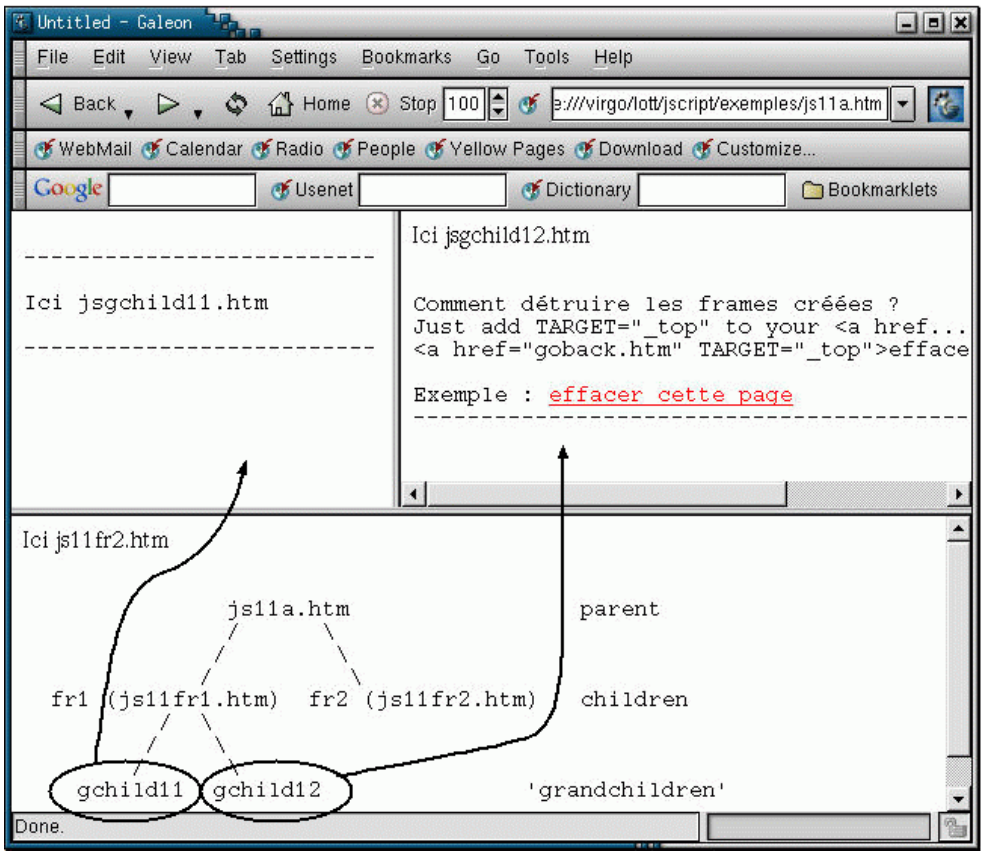


FIG. 128: js11a.htm

jsgchild12.htm

```
 1 <HTML> <!---->
 2 <HEAD><!-- fichier jsgchild12.htm -->
 3 </HEAD>
 4 <BODY>
 5 Ici jsgchild12.htm<br>
 6
 7 <font size="+1"><tt><pre>
 8
 9 Comment détruire les frames créées?
10 Just add TARGET="_top" to your &lt;a href...> tag :
11 &lt;a href="goback.htm" TARGET="_top">effacer cette page &lt;/a>
12
13 Exemple : <a href="jsgoback.htm" TARGET="_top">effacer cette page</a>
14 ------------------------------------------------
```

jsgchild12.htm (suite)

```
15  </pre></tt></font>
16
17  </body></html> <!---->
```

Action sur le navigateur : js12.htm

js12.htm

```
 1  <html> <!---->
 2  <head><!-- fichier js12.htm -->
 3  <script language="JavaScript">
 4  function statbar(txt) {
 5     window.status = txt;
 6  }
 7  </script>
 8  </head>
 9
10  <body>
11  Ici js12.htm : démo écriture dans barre d'état (statusbar).<br>
12
13  <form>
14  <input type="button" name="look" value="Write!"
15          onclick="statbar('Hi! This is the statusbar!');">
16
17  <input type="button" name="erase" value="Erase!" onclick="statbar('');">
18
19  </form>
20  </body></html> <!---->
```

Action sur le navigateur : js13.htm

js13.htm

```
 1  <html> <!---->
 2  <head><!-- fichier js13.htm -->
 3
 4  <script language="JavaScript">
 5  function statbar(txt) {
 6     window.status = txt;
 7     setTimeout("erase()",2000);
 8  }
 9  function erase() {
10     window.status="";
11  }
12  </script>
13  </head>
14  <body>
15
16  <a href="js13.htm"
```

js13.htm (suite)

```
 onMouseOver="statbar('Disappearing in 2 sec.!');return true;">
link</a>

</body>
</html> <!---->
```

Action sur le navigateur : js14.htm

js14.htm

```
<html> <!---->
<head><!-- fichier js14.htm -->

<script language="JavaScript">

var scrtxt="Ici un message "+
" et la suite du message ...";
var lentxt=scrtxt.length;
var width=100;
var pos=1-width;

function scroll() {
  pos++;
  var scroller="";

  if (pos==lentxt) {
    pos=1-width;
  }
  if (pos<0) {
    for (var i=1; i<=Math.abs(pos); i++) {
       scroller=scroller+" ";}
    scroller=scroller+scrtxt.substring(0,width-i+1);
  }
  else {
    scroller=scroller+scrtxt.substring(pos,width+pos);
  }
  window.status = scroller;

  setTimeout("scroll()",150);

  }
</script>
</head>

<body onLoad="scroll();return true;">

Ici la page js14.htm.<br>

Un texte défile dans la barre de status.

</body>
</html> <!---->
```

Action sur le navigateur : js15.htm

js15.htm

```
 1  <html><head><!-- fichier js15.htm --> <!---->
 2  <script language="JavaScript">
 3  <!-- Hide
 4  function writedoc() {
 5  var aa = "abc def";
 6  var mytab= new Array(3)
 7  mytab[0]=123
 8  mytab[1]="yes/no"
 9  mytab[2]=102030
10     document.write("<P>");
11     document.write("éléments du tableau :"+mytab.length+"<br>");
12     document.write("élément 0 : "+mytab[0]+"<br>");
13     document.write("élément 1 : "+mytab[1]+"<br>");
14     document.write("élément 2 : "+mytab[2]+"<br>");
15     document.write("élément 3 : "+mytab[3]+"<br>");
16  }
17  // -->
18  </script>
19  </head>
20  <body>
21  <FORM NAME="buttonbar">
22   <INPUT TYPE="button" VALUE="Back" onClick="history.back()">
23   <INPUT TYPE="button" VALUE="Home" onClick="location='js01.htm'">
24   <INPUT TYPE="button" VALUE="Next" onCLick="history.forward()">
25  <br>
26  <INPUT TYPE="button" VALUE="write document" onClick="writedoc()">
27  </FORM>
28  <p>On peut aussi utiliser : history.go(-1) et
29  history.go(1) ...</p>
30  </body></html> <!---->
```

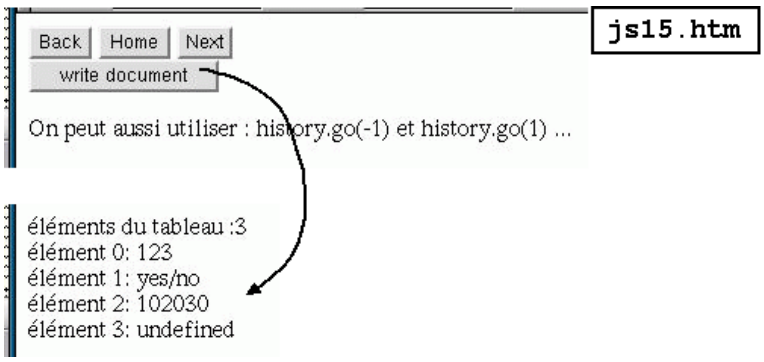


FIG. 129: js15.htm

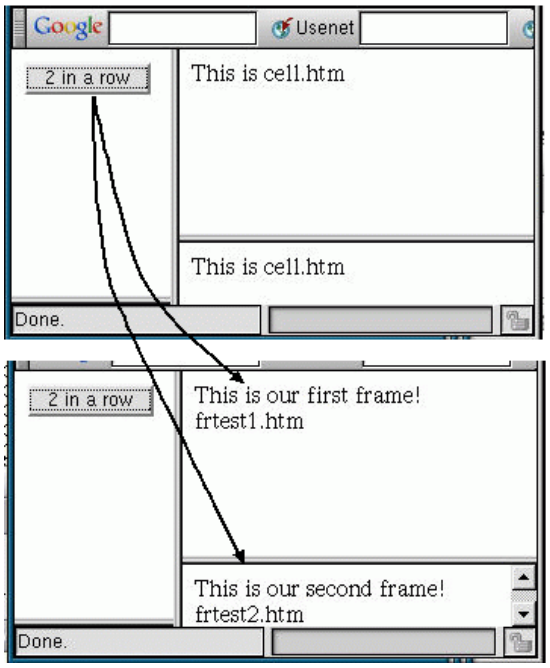
Action et frames : js16.htm et load2.htm

js16.htm

```
 1  <HTML><HEAD><!-- fichier js16.htm --> <!---->
 2  <title>Frames</title>
 3  </HEAD>
 4  <FRAMESET COLS="105,*">
 5    <FRAMESET ROWS="100%,*">
 6       <FRAME SRC="load2.htm" NAME="fr1">
 7    </FRAMESET>
 8
 9    <FRAMESET ROWS="75%,25%">
10       <FRAME SRC="cell.htm" NAME="fr2">
11       <FRAME SRC="cell.htm" NAME="fr3">
12    </FRAMESET>
13
14  </FRAMESET>
15  </HTML> <!---->
```

load2.htm

```
 1  <HTML><HEAD><!-- fichier load2.htm --> <!---->
 2  <script language="JavaScript">
 3    function loadtwo(page2, page3) {
 4       parent.fr2.location.href=page2;
 5       parent.fr3.location.href=page3;
 6  }
 7  </script>
 8  </HEAD><BODY>
 9  <FORM NAME="buttons">
10       <INPUT TYPE="button" VALUE="2 in a row"
11              onClick="loadtwo('frtest1.htm','frtest2.htm')">
12  </FORM>
13  </BODY></HTML> <!---->
```



js16.htm

FIG. 130: js16.htm et load2.htm

Action et frames : js17.htm et js171.htm

js17.htm

```
 1  <HTML> <!---->
 2  <HEAD><!-- fichier js17.htm -->
 3  </HEAD>
 4  <FRAMESET COLS="395,*">
 5    <FRAMESET ROWS="100%,*">
 6       <FRAME SRC="js171.htm" NAME="fr1">
 7    </FRAMESET>
 8
 9    <FRAMESET ROWS="75%,25%">
10       <FRAME SRC="cell.htm" NAME="fr2">
11       <FRAME SRC="cell.htm" NAME="fr3">
12    </FRAMESET>
13
14  </FRAMESET>
15  </html> <!---->
```

js171.htm

```
1  <HTML><HEAD><!-- fichier js171.htm --> <!---->
2  <script language="JavaScript">
3    function mafonction () {
4       parent.fr2.location.href="frtest1.htm";
5       parent.fr3.location.href="frtest2.htm";  }
6  </script>
7  </HEAD><BODY>
```

js171.htm (suite)

```
 8  Autres techniques de renvois vers d'autres pages.<br>
 9
10  <h3>Lien avec appel de fonction javascript</h3>
11  <b>&lt;a href="cell.htm onCLick="mafonction()">
12  cliquer ici&lt;/a> : </b>
13  <a href="cell.htm" onCLick="mafonction()">cliquer ici</a>
14  <br>
15  Autre syntaxe : <br>
16  <b>&lt;a href="javascript :mafonction()">Mon lien.&lt;/a>
17  <a href="javascript :mafonction()">Mon lien.</a>
18  <br><br>
19  Autre technique : <br>
20  &lt;a href="frtest1.htm" target="fr2">
21  charger frtest1 dans fr2&lt;/a><br>
22
23  <a href="frtest1.htm" target="fr2">charger frtest1 dans fr2</a>
24  <br>
25  <hr>
26  <br>
27  Une nouvelle page chargée dans une nouvelle fenêtre
28  du browser :<br>
29  &lt;a href="frtest1.htm" target="New_Window">charger&lt;/a>
30  <a href="frtest1.htm" target="New_Window">charger</a>
31
32  </BODY></HTML> <!---->
```

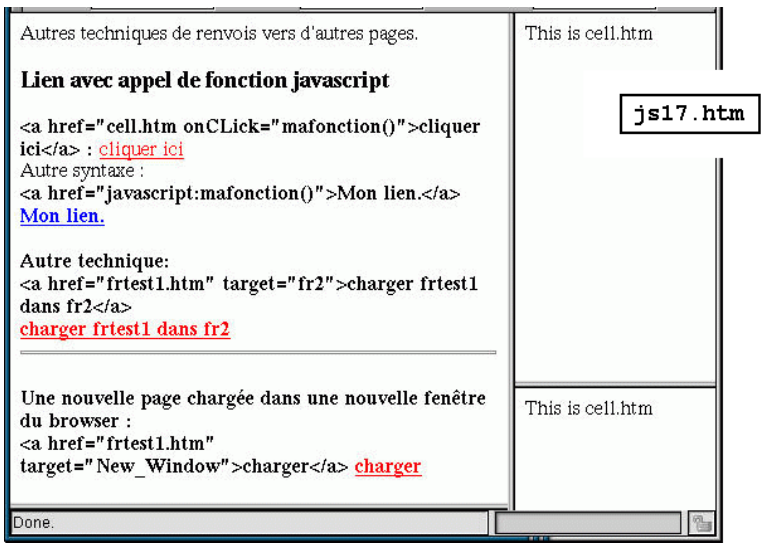


FIG. 131: js17.htm et js171.htm

Opérateurs javascript : js18.htm

js18.htm

```
1  <HTML><HEAD><!-- fichier js18.htm --> <!---->
2  </HEAD>
```

js18.htm (suite)

```
 3  <BODY>
 4  <h3>Quelques opérateurs javascript :</h3>
 5  <tt><pre>
 6  if (x>3)
 7    if (x&lt;10)
 8      doanything();    &lt;-- do if x>3 and x&lt;10
 9
10  if (x>3 && x&lt;10) doanything();  &lt;-- do if x>3 and x&lt;10
11
12  if (x==5 || y==17) doanything(); {&lt;-- do if x== 5 or y==17
13                                    {&lt;-- also if x==5 and y==17
14
15  Il y a aussi :  &lt; , > , &lt;= and >=  and!=
16  </pre></tt>
17  </BODY>
18  </HTML> <!---->
```

Validation d'entrées dans une forme : js19.htm

js19.htm

```
 1  <html> <!---->
 2  <head><!-- fichier js19.htm -->
 3  <script language="JavaScript">
 4  function test1(form) {
 5    if (form.text1.value == "")
 6      alert("Please enter a string!")
 7    else {
 8     alert("Hi "+form.text1.value+"! Form input ok!");
 9    }
10  }
11  function test2(form) {
12    if (form.text2.value == "" ||
13        form.text2.value.indexOf('@', 0) == -1)
14          alert("No valid e-mail address!");
15    else alert("OK!");
16  }
17  </script>
18  </head><body>
19
20  <form name="first">
21
22  Enter your name :<br>
23
24  <input type="text" name="text1">
25
26  <input type="button" name="button1"
27         value="Test Input" onClick="test1(this.form)">
28
29  <P>
30
```

js19.htm (suite)

```
31  Enter your e-mail address :<br>
32
33  <input type="text" name="text2">
34
35  <input type="button" name="button2"
36         value="Test Input" onClick="test2(this.form)">
37
38  </body>
39  </html> <!---->
```

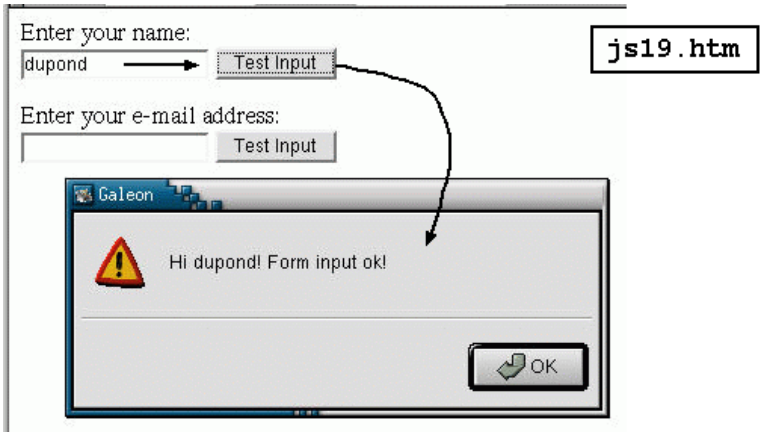


FIG. 132: js19.htm

11.3 JavaScript, CSS et les menus

On trouve sur le web de nombreux exemples de création de menus et barres de menus réalisés en JavaScript.

Toutefois, il est possible, dans les browsers récents **et** conformes aux normes du W3C, de réaliser des menus :

- en utilisant *uniquement* les CSS,
- en utilisant un mélange de CSS et de JavaScript.

Menus exemple 1 - Pure CSS : ex1.html + ex1.css

ex1.html

```
1  <!---->
2  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
3  "http ://www.w3.org/TR/html4/loose.dtd">
4  <html>
5  <head>
6  <title>Ex1 - Pure CSS Menu</title>
7  <meta http-equiv="Content-Type" content="text/html;
8           charset=ISO-8859-1">
9  <link rel="stylesheet" type="text/css" href="ex1.css">
```

ex1.html (suite)

```
10  </head>
11  <body>
12    <h2>ex1.html + ex1.css - Pure CSS Menu</h2>
13    <p>
14    Exemple de menus utilisant uniquement les CSS. Début.
15    </p>
16
17    <div id="menubar">
18      <ul class="css-menu">
19        <li><a href="#">Link</a></li>
20        <li class="submenu">
21          <a href="#">Submenu</a>
22          <ul>
23            <li><a href="#">Subitem 1</a></li>
24            <li><a href="#">Subitem 2</a></li>
25          </ul>
26        </li>
27      </ul>
28    </div>
29
30      <ul class="css-menu">
31        <li><a href="#">Link1</a></li>
32        <li><a href="#">Link2</a></li>
33        <li class="submenu"><a href="#">Submenu1</a><ul>
34            <li><a href="#">Subitem 1/1</a></li>
35            <li><a href="#">Subitem 1/2</a></li>
36        </ul></li>
37        <li class="submenu"><a href="#">Submenu2</a><ul>
38            <li><a href="#">Subitem 2/1</a></li>
39            <li><a href="#">Subitem 2/2</a></li>
40            <li><a href="#">Subitem 2/3</a></li>
41        </ul></li>
42      </ul>
43
44    <p>
45    Exemple de menus utilisant uniquement les CSS. Fin.
46    </p>
47  <hr>
48  </body>
49  </html><!----
```

ex1.css

```
1  /*  ex1.css  */
2
3  /* la barre de menu */
4  ul.css-menu
5  { display : block;  background-color : aqua;
6    /* margin : espace entre 2 barres;
7       padding : espace autour des objets contenus */
8    margin : 2px;  padding : 6px;
```

ex1.css (suite)

```
 9    /* cadre noir autour de la barre */
10    border-style : solid; border-width : 1px; color : black;
11  }
12
13  /* une entrée dans la barre de menus */
14  ul.css-menu li  /* les "li" sous "ul.css-menu" = top level */
15  { /* afficher en ligne (horizontale) les top-levels */
16    display : inline;
17    /* list-style : none;  inutile en inline */
18    position : relative;
19    margin : 3px;  padding : 3px;
20    /* cadre rouge autour de chaque entrée de la barre */
21    border-style : solid; border-width : 1px; color : red;
22  }
23
24  /* la partie anchor d'une entrée de la barre */
25  ul.css-menu li a
26  { margin : 1px;  padding : 3px;  background-color : silver;
27    /* texte des entrées de la barre (dans les "a") */
28    text-decoration : none;
29  }
30
31  /* changer apparence entrée top niv sélectionnée */
32  ul.css-menu li a :hover
33  { color : #FFF;    /* selection : texte blanc sur fond rouge */
34    background-color : red;  /* fond élément menu sélectionné */
35  }
36
37  /* sous-menu qui s'ouvre (en vert) sous une entrée toplevel */
38  ul.css-menu li ul  /* "ul" sous un "li" = sous-menu */
39  { display : block;
40    position : absolute;
41    left : 0;
42    top : 1em; /*required for NS7.x/Gecko 1.0.x but not Gecko 1.3*/
43    visibility : hidden;  /* sous-menu cachés par défaut */
44    margin : 5px;  padding : 5px;  background-color : lime;
45    /* width : 9em; */
46    z-index : 1000;
47  }
48
49  /* fix up the submenu itemss voice-family lines screen correct
50     CSS values from browsers that improperly lay out block-level
51     boxes and have broken CSS parsers (IE5.5/Win)
52  */
53  ul.css-menu li ul li
54  { margin : 0 0 0 -1.5em;
55    padding : 0;
56    display : block;
57    width : 100%;
58    voice-family : "\"}\""; voice-family : inherit;
59    margin-left : 0;
60  }
```

ex1.css (suite)

```
61
62  /* une entrée dans un sous-menu */
63  ul.css-menu li ul li a
64  { display : block;
65    margin : 0;
66  /*   padding : 0 0 0 5%;
67    width : 95%; */
68  }
69
70  /* pour sous-menu ouvert, zone active assez haute,
71     sinon reste pas ouvert */
72  ul.css-menu li.submenu :hover
73  { padding-bottom : 10em;
74  }
75
76  /* quand entrée barre est activée, rendre visible les
77     "ul" en-dessous */
78  ul.css-menu li.submenu :hover ul
79  { left : 0;
80    visibility : visible;
81  }
82   *
```

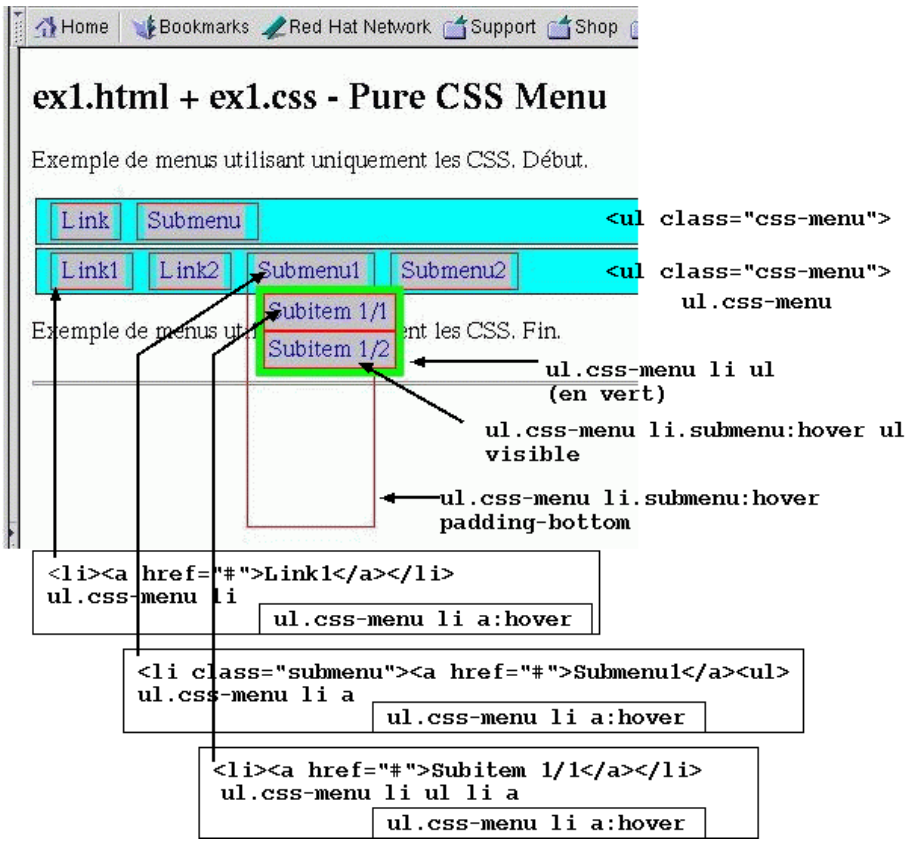


FIG. 133: ex1.html + ex1.css

Menus exemple 2 - Pure CSS : ex3.html + ex3.css

ex3.html

```
<!---->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http ://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ex3 - CSS Menu</title>
<meta http-equiv="Content-Type" content="text/html;
        charset=ISO-8859-1">
<link rel="stylesheet" type="text/css" href="ex3.css">
</head>

<body>
  <h2>ex3.html + ex3.css - Pure CSS Menu</h2>
  <p>
  Exemple de menus utilisant uniquement les CSS. Début.
  </p>

  <div id="menubar">
    <ul class="css-menu">
      <li><a href="#">Link1</a></li>
      <li><a href="#">Link2</a></li>
      <li class="submenu"><a href="#">Submenu1</a><ul>
          <li><a href="#">Subitem 1/1</a></li>
          <li><a href="#">Subitem 1/2</a></li>
      </ul></li>
      <li class="submenu"><a href="#">Submenu2</a><ul>
          <li><a href="#">Subitem 2/1</a></li>
          <li><a href="#">Subitem 2/2</a></li>
          <li><a href="#">Subitem 2/3</a></li>
      </ul></li>
    </ul>
  </div>

  <p>
  Exemple de menus utilisant uniquement les CSS. Fin.
  </p>
<hr>
</body>
</html><!---->
```

ex3.css

```
/* ex3.css */

/* la barre de menu */
ul.css-menu
{ display : block;
  margin : 3px;  padding : 5px;
```

ex3.css (suite)

```
 7    background-color : lime;
 8  }
 9
10  /* une entrée dans la barre de menus */
11  ul.css-menu li  /* top level items */
12  { margin : 0;
13    border-style : solid; border-width : 1px;
14    padding : 3px;
15    margin : 3px;
16    margin : 0;
17    padding : 0;
18    background-color : yellow;
19    display : inline;
20    list-style : none;
21    position : relative;
22  }
23  /* idem utiliser ul.css-menu ul ou de mettre les attributs
24     dans ul.css-menu li ul */
25  /* ul.css-menu ul
26  { display : block;
27    margin : 3px; padding : 5px;
28    background-color : lime;
29  } */
30
31  ul.css-menu li ul   /* liste "sous" le top-level */
32  {
33    display : block;
34    position : absolute;
35    left : 0;
36    top : 1em; /*required for NS7.x/Gecko 1.0.x but not Gecko 1.3*/
37    visibility : hidden;
38    width : 9em;
39    z-index : 1000;
40    margin : 3px; padding : 5px;
41    background-color : lime;
42  }
43
44  /* fix up the submenu itemss voice-family lines screen correct
45     CSS values from browsers that improperly lay out block-level
46     boxes and have broken CSS parsers (IE5.5/Win)
47  */
48  ul.css-menu li ul li
49  { margin : 0 0 0 -1.5em;
50    padding : 0;
51    display : block;
52    width : 100%;
53    voice-family : "\"}\""; voice-family : inherit;
54    margin-left : 0;
55  }
56
57  ul.css-menu li ul li a
58  { display : block;
```

ex3.css (suite)

```
 59    margin : 0;  padding : 0 0 0 5%;
 60    width : 100%;
 61    voice-family : "\"}\""; voice-family : inherit;
 62    width : 95%;
 63  }
 64
 65
 66  /* la partie anchor d'une entrée de la barre */
 67  ul.css-menu li a
 68  { display : inline;
 69    text-decoration : none;
 70    margin : 0;
 71    color : black;
 72    color : red; /* texte des éléments menu */
 73    font-size : .9em;
 74    padding : 5px; /* éloigner le cadre de chq item du texte */
 75  }
 76
 77  ul.css-menu li a :hover /* top-level menu sélectionné */
 78  { color : #FFF;
 79    background-color : red; /* fond d'un élément menu sélectionné */
 80  }
 81
 82  ul.css-menu li.submenu ul a
 83  { color : black;
 84  }
 85
 86  ul.css-menu li.submenu ul a :hover /* sub menu sélectionné */
 87  { color : white;
 88    background-color : navy; /* bleu fonçé */
 89  }
 90
 91  ul.css-menu li.submenu :hover ul
 92  { left : 0;
 93    visibility : visible;
 94  }
 95
 96  /* pour sous-menu ouvert, zone active */
 97  ul.css-menu li.submenu :hover
 98  { padding-bottom : 10em;
 99    background-color : transparent;
100    border-width : 0;
101    padding-right : 5em;
102  }
103
```

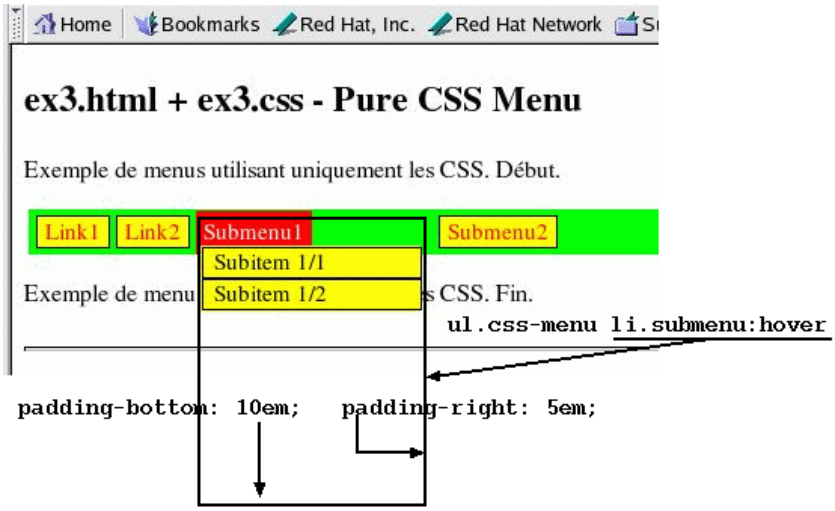


FIG. 134: ex3.html + ex3.css

Menus exemple 3 - CSS + JavaScript : exmix.html

exmix.html

```
<!---->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http ://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Exemple Menu Mixte CSS-JS</title>
<meta http-equiv="Content-Type" content="text/html;
        charset=ISO-8859-1">
<link rel="stylesheet" type="text/css" href="cssjsmenu.css">
<script type="text/javascript" src="cssjsmenu.js"></script>
<script type="text/javascript">
  <!--
  function init()
  { cssjsmenu('navbar');
  }
  // -->
</script>
</head>
<body onload="init()">
  <h3>Exemple Menu Mixte CSS-JS</h3>
  <div id="navbar">
    <ul class="nde-menu-system">
      <li><a href="#">Link1</a></li>
      <li><a href="#">Link2</a></li>
      <li class="submenu">
      <a href="#">Submenu1</a>
        <ul>
          <li><a href="#">Subitem 1/1</a></li>
          <li><a href="#">Subitem 1/2</a></li>
        </ul>
      </li>
      <li class="submenu2">
```

exmix.html (suite)

```
33        <a href="#">Submenu</a>
34          <ul>
35            <li><a href="#">Subitem 2/1</a></li>
36            <li><a href="#">Subitem 2/2</a></li>
37            <li><a href="#">Subitem 2/3</a></li>
38            <li><a href="#">Subitem 2/4</a></li>
39         </ul>
40        </li>
41        <li class="submenu3">
42        <a href="#">Submenu</a>
43          <ul>
44            <li><a href="#">Subitem 3/1</a></li>
45         </ul>
46        </li>
47        <li class="submenu4">
48        <a href="#">Submenu</a>
49          <ul>
50            <li><a href="#">Subitem 4/1</a></li>
51         </ul>
52        </li>
53      </ul>
54    </div>
55
56    <p>
57    Exemple de menu mixte CSS/Javascript, inspiré du site
58    "Netscape DevEdge".
59    <br>http ://devedge.netscape.com/central/css/
60    <br>http ://devedge.netscape.com/viewsource/2003/
61  devedge-redesign-css/
62    </p>
63
64  </body></html><!---->
```

cssjsmenu.css

```
 1  /* barre de menus */
 2  ul.nde-menu-system
 3  {
 4    background-color : rgb(75%,75%,75%);
 5    padding : 8px; /* faire barre plus grande que top levels */
 6    background-color : yellow; /* fond de la barre de menus */
 7  }
 8
 9  ul.nde-menu-system ul
10  {
11    display : block;
12    margin : 0; /* nécessaire pour garder l'ul visible */
13    /* sinon disparait quand souris passe dans la marge! */
14    padding : 5px;
15    background-color : red;
16  }
```

cssjsmenu.css (suite)

```
17
18  /* top level items in ul are inline to display horizontally
19     across page  */
20  ul.nde-menu-system li
21  {
22    display : inline;
23    list-style : none;
24    position : relative;
25    margin : 0;
26    border-style : solid; /* ok */
27    border-width : 1px;   /* ok */
28    padding : 3px; /* espace entre texte toplevel et boite */
29    margin : 2px; /* espacement entre 2 boites toplevel */
30    background-color : silver; /* fond éléments menu toplevel */
31  }
32
33  /*  nested lists inside of the top level items are initially
34      not displayed  */
35  ul.nde-menu-system li ul
36  {
37    display : block;
38    position : absolute;
39    left : 0;
40    top : 1em; /*required for NS7.x/Gecko 1.0.x but not Gecko 1.3*/
41    visibility : hidden;
42    width : 9em;
43    z-index : 1000;
44  }
45
46  /* fix up the submenu items voice-family lines screen correct
47     CSS values from browsers that improperly lay out block-level
48     boxes and have broken CSS parsers (IE5.5/Win)
49  */
50  ul.nde-menu-system li ul li
51  {
52    margin : 0 0 0 -1.5em;
53    padding : 0;
54    display : block;
55    width : 100%;
56    voice-family : "\"}\""; voice-family : inherit;
57    margin-left : 0;
58  }
59
60  ul.nde-menu-system li ul li a
61  {
62    display : block;
63    margin : 0;
64    padding : 0 0 0 5%;
65    width : 100%;
66    voice-family : "\"}\""; voice-family : inherit;
67    width : 95%;
68  }
```

cssjsmenu.css (suite)

```
 69
 70  /* apparence des différents éléments de menu */
 71  ul.nde-menu-system *
 72  {
 73    font : 1em verdana sans-serif;
 74  }
 75
 76  ul.nde-menu-system li a
 77  {
 78    display : inline;
 79    text-decoration : none;
 80    margin : 0;
 81    color : black;
 82    font-size : .9em;
 83  }
 84
 85  ul.nde-menu-system li a :hover
 86  {
 87    color : #FFF;
 88    background-color : blue;
 89  }
 90
 91  ul.nde-menu-system li.submenu ul a
 92  {
 93    color : black;
 94  }
 95
 96  ul.nde-menu-system li.submenu ul a :hover
 97  {
 98    color : white;
 99    background-color : blue;
100  }  *
```

cssjsmenu.js

```
  1  //-------------------------------------------------
  2  function cssjsmenu(menuid)
  3  {
  4    var i;
  5    var j;
  6    var node;
  7    var child;
  8    var parent;
  9
 10    // if the browser doesn't even support
 11    // document.getElementById, give up now.
 12    if (!document.getElementById)
 13    { return true;  }
 14
 15    // check for downlevel browsers
 16    // Opera 6, IE 5/Mac are not supported
```

cssjsmenu.js (suite)

```
17    var version;
18    var offset;
19
20    offset = navigator.userAgent.indexOf('Opera');
21    if (offset != -1)
22    {
23      version=parseInt('0'+navigator.userAgent.substr(offset+ 6),10);
24      if (version < 7)
25      {  return true;    }
26    }
27
28    offset = navigator.userAgent.indexOf('MSIE');
29    if (offset != -1)
30    {
31      if (navigator.userAgent.indexOf('Mac') != -1)
32      { return true;    }
33    }
34  //---------------------------------------------------
35    // commence ici création des menus
36    var menudiv = document.getElementById(menuid);
37
38    // ul    liste des barres de menus
39    var ul = new Array();
40
41    for (i = 0; i < menudiv.childNodes.length; i++)
42    {
43      node = menudiv.childNodes[i];
44      if (node.nodeName == 'UL')
45      {
46        ul[ul.length] = node;
47      }
48    }
49
50    // ul > li    liste des toplevels
51    var ul_gt_li = new Array();
52
53    for (i = 0; i < ul.length; i++)
54    {
55      node = ul[i];
56      for (j = 0; j < node.childNodes.length; j++)
57      {
58        child = node.childNodes[j];
59        if (child.nodeName == 'LI')
60        {
61          ul_gt_li[ul_gt_li.length] = child;
62          child.style.display = 'inline';
63          child.style.listStyle = 'none';
64          child.style.position = 'static';
65        }
66      }
67    } // ul > li
68
```

cssjsmenu.js (suite)

```
  69    // ul > li > ul   liste des toplevels ayant un sous-menu
  70    var ul_gt_li_gt_ul = new Array();
  71
  72    for (i = 0; i < ul_gt_li.length; i++)
  73    {
  74      node = ul_gt_li[i];
  75      for (j = 0; j < node.childNodes.length; j++)
  76      {                            // pour chaque toplevel
  77        child = node.childNodes[j];
  78        if (child.nodeName == 'UL')  //a-t-il un sous-menu?
  79        {
  80          ul_gt_li_gt_ul[ul_gt_li_gt_ul.length] = child;
  81          child.style.position = 'absolute';
  82          child.style.left = '-13em';
  83          child.style.visibility = 'hidden';
  84
  85          // attach hover to parent li
  86          parent = child.parentNode;
  87          // attacher à chaque toplevel une action javascript
  88          // qui va rendre le sous-menu visible/caché suivant souris
  89          parent.onmouseover = function (e)
  90          {
  91            var i;
  92            var child;
  93            var point;
  94
  95            // stop the pure css hover effect
  96            this.style.paddingBottom = '0';
  97
  98            for (i = 0; i < this.childNodes.length; i++)
  99            {
 100              child = this.childNodes[i];
 101              if (child.nodeName == 'UL')
 102              {
 103                point = getPageXY(this);
 104                setPageXY(child,point.x, point.y+ this.offsetHeight);
 105                child.style.visibility = 'visible';
 106              }
 107            }
 108            return false;
 109          };
 110          parent.onmouseout = function (e)
 111          {
 112            var relatedTarget = null;
 113            if (e)
 114            {
 115              relatedTarget = e.relatedTarget;
 116              // work around Gecko Linux only bug where related
 117              // target is null when clicking on menu links or
 118              // when right clicking and moving into a context menu.
 119              if (navigator.product == 'Gecko' &&
 120                  navigator.platform.indexOf('Linux')!= -1 &&
```

cssjsmenu.js (suite)

```
121                !relatedTarget)
122              {
123                relatedTarget = e.originalTarget;
124              }
125            }
126            else if (window.event)
127            {
128              relatedTarget = window.event.toElement;
129            }
130
131            if (elementContains(this, relatedTarget))
132            {
133              return false;
134            }
135
136            var i;
137            var child;
138            for (i = 0; i < this.childNodes.length; i++)
139            {
140              child = this.childNodes[i];
141              if (child.nodeName == 'UL')
142              {
143                  child.style.visibility = 'hidden';
144              }
145            }
146            return false;
147          };
148        } // pr les toplevels ayant un sous-menu (UL)
149      } // pr chq toplevel
150    }  // ul > li > ul   les entrées de menus
151    var mess='cssjsmenu fin '+ ul.length+' '
152    window.status=mess+ul_gt_li.length+' '+ul_gt_li_gt_ul.length;
153    return true;
154  }// fin function cssjsmenu(menuid)
155
156  //-------------------------------------------------
157
158  function elementContains(elmOuter, elmInner)
159  {
160    while (elmInner && elmInner!= elmOuter)
161    {
162      elmInner = elmInner.parentNode;
163    }
164    if (elmInner == elmOuter)
165    {
166      return true;
167    }
168    return false;
169  }
170
171  function getPageXY(elm)
172  {
```

cssjsmenu.js (suite)

```
173   var point = { x : 0, y : 0 };
174   while (elm)
175   {
176     point.x += elm.offsetLeft;
177     point.y += elm.offsetTop;
178     elm = elm.offsetParent;
179   }
180   return point;
181 }
182
183 function setPageXY(elm, x, y)
184 {
185   var parentXY = {x : 0, y : 0 };
186
187   if (elm.offsetParent)
188   {
189     parentXY = getPageXY(elm.offsetParent);
190   }
191
192   elm.style.left = (x - parentXY.x) + 'px';
193   elm.style.top  = (y - parentXY.y) + 'px';
194 }
195
```

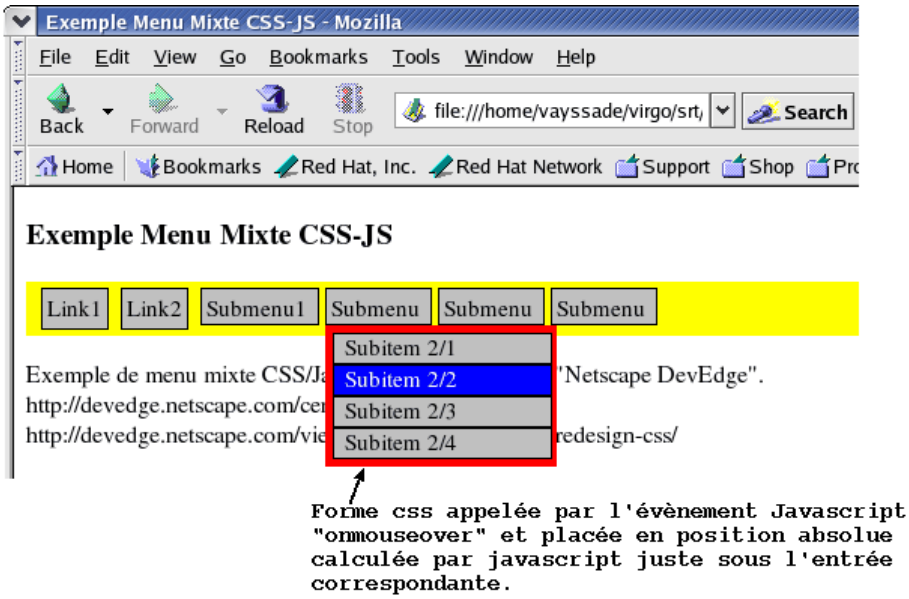


FIG. 135: exmix.html + cssjsmenu.css + cssjsmenu.js

Menus exemple 4 - CSS + JavaScript : exmix3.html

exmix3.html

```
1  <!---->
2  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
```

exmix3.html (suite)

```
 3  "http ://www.w3.org/TR/html4/loose.dtd">
 4  <html>
 5  <head>
 6  <title>Exemple Menu Mixte CSS-JS</title>
 7  <meta http-equiv="Content-Type" content="text/html;
 8          charset=ISO-8859-1">
 9  <link rel="stylesheet" type="text/css" href="cssjsmenu.css">
10  <script type="text/javascript" src="cssjsmenu.js"></script>
11  <script type="text/javascript">
12    <!--
13    function init()
14    { cssjsmenu('navbar');
15    }
16    // -->
17  </script>
18  </head>
19  <body onload="init()">
20    <h3>Exemple Menu Mixte CSS-JS</h3>
21    <div id="navbar">
22      <ul class="nde-menu-system">
23  <!-- mettre 2 barres "navbar" ne fonctionne pas : tester avec
24       une barre qui "déborde" : ça fonctionne, mais, petit pb
25       de recouvrement des toplevels qui pose problème pour
26       descendre dans les sous-menus (il faut passer le souris
27       "entre" les toplevels de la rangée du bas) -->
28
29        <li><a href="#">Link1</a></li>
30        <li><a href="#">Link2</a></li>
31        <li class="submenu">
32        <a href="#">Submenu1</a>
33          <ul>
34            <li><a href="#">Subitem 1/1</a></li>
35            <li><a href="#">Subitem 1/2</a></li>
36          </ul>
37        </li>
38        <li class="submenu2">
39        <a href="#">Submenu</a>
40          <ul>
41            <li><a href="#">Subitem 2/1</a></li>
42            <li><a href="#">Subitem 2/2</a></li>
43            <li><a href="#">Subitem 2/3</a></li>
44            <li><a href="#">Subitem 2/4</a></li>
45         </ul>
46        </li>
47        <li class="submenu3">
48        <a href="#">Submenu</a>
49          <ul>
50            <li><a href="#">Subitem 3/1</a></li>
51         </ul>
52        </li>
53        <li class="submenu4">
54        <a href="#">Submenu</a>
```

exmix3.html (suite)

```
 55          <ul>
 56            <li><a href="#">Subitem 4/1</a></li>
 57          </ul>
 58        </li>
 59
 60        <li><a href="#">2 Link1</a></li>
 61        <li><a href="#">2 Link2</a></li>
 62        <li class="submenu">
 63        <a href="#">2 Submenu1</a>
 64          <ul>
 65            <li><a href="#">2 Subitem 1/1</a></li>
 66            <li><a href="#">2 Subitem 1/2</a></li>
 67          </ul>
 68        </li>
 69        <!-- un br pour éviter toplevl à cheval sur 2 lignes -->
 70        <br>
 71        <br> <!-- et un br pour éviter recouvrement vertical
 72                  des toplevels -->
 73        <li class="submenu2">
 74        <a href="#">2 Submenu</a>
 75          <ul>
 76            <li><a href="#">2 Subitem 2/1</a></li>
 77            <li><a href="#">2 Subitem 2/2</a></li>
 78            <li><a href="#">2 Subitem 2/3</a></li>
 79            <li><a href="#">2 Subitem 2/4</a></li>
 80          </ul>
 81        </li>
 82        <li class="submenu3">
 83        <a href="#">2 Submenu</a>
 84          <ul>
 85            <li><a href="#">2 Subitem 3/1</a></li>
 86          </ul>
 87        </li>
 88        <li class="submenu4">
 89        <a href="#">2 Submenu</a>
 90          <ul>
 91            <li><a href="#">2 Subitem 4/1</a></li>
 92          </ul>
 93        </li>
 94
 95      </ul>
 96    </div>
 97
 98    <p>
 99    Exemple de menu mixte CSS/Javascript, inspiré du site
100    "Netscape DevEdge".
101    <br>http ://devedge.netscape.com/central/css/
102    <br>http ://devedge.netscape.com/viewsource/2003/
103  devedge-redesign-css/
104    </p>
105
106  </body></html><!---->
```

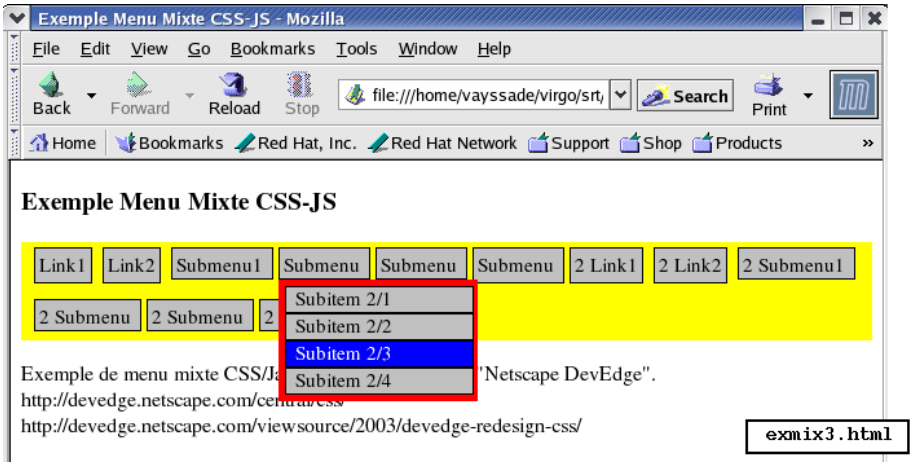


FIG. 136: exmix3.html + cssjsmenu.css + cssjsmenu.js

SR03 2004 - Cours Architectures Internet - PHP

12 SR03 2004 - Cours Architectures Internet - PHP

12.1 PHP : un langage de script côté serveur

Premier script : p01.php3

p01.php3

```
 1  <HTML><HEAD> <!---->
 2  <TITLE>fichier p01.php3</TITLE>
 3  </HEAD>
 4  <BODY BGCOLOR="#FFFFFF">
 5  <font size="+1"> Début du fichier php01.htm <br>
 6  Ici code html.<br>
 7  <hr>
 8  <?php
 9   echo "<P>Hello World ! écrit par du code PHP.";
10   echo "</P>";
11   ?>
12  Retour au code html.<br>
13  </BODY></HTML> <!---->
```

L'exécution dans un navigateur donne :

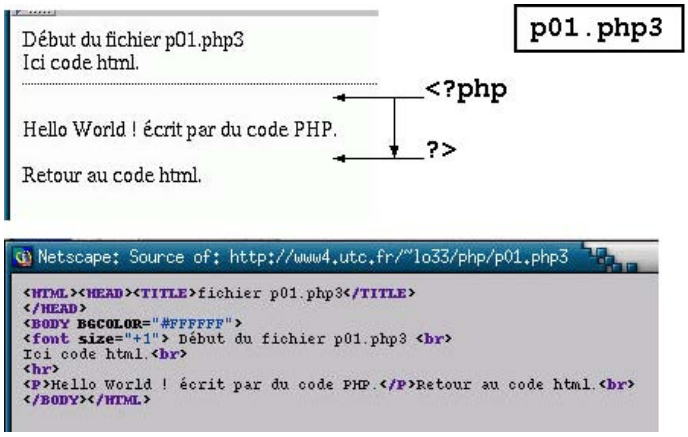


FIG. 137: p01.php3

Explications

Le but de PHP est de permettre au concepteur de site web d'exécuter un traitement avant d'envoyer une page au navigateur. Ce traitement pourra tenir compte de la requête. Les instructions PHP sont exécutées **par le serveur web**. Elles vont permettre de construire des pages **créées dynamiquement** : la page est construite par le serveur après réception de la requête, puis envoyée au client.

Attention : le code PHP ne fonctionne que si PHP à été installé sur le serveur web qui "sert" la page contenant du PHP. Le fichier source doit porter l'extension ".php3" ou ".php4" pour que le serveur web prenne l'initiative de faire exécuter son contenu par l'interpréteur PHP.

Important : le code source HTML envoyé au navigateur par le serveur ne contient **aucune trace** du code PHP utilisé pour fabriquer la page.

Le code PHP est placé directement dans le code source HTML, encadré par deux balises spéciales destinées à être reconnues par l'interprêteur PHP3.

Tout le texte situé à l'extérieur des balises < ?php et ?> est envoyé au navigateur. Un même fichier source php peut contenir une **alternance** de parties "html" et de parties "php".

La syntaxe de PHP ressemble à celle du C. Les commentaires peuvent être insérés soit par /* ... */ pour les blocs, soit par // pour les fins de lignes.

12.2 Types de données PHP : p02.php3

p02.php3

```
 1  <HTML><HEAD> <!---->
 2  <TITLE>fichier p02.php3</TITLE>
 3  </HEAD>
 4  <BODY BGCOLOR="#FFFFFF">
 5  <font size="+1"> Début du fichier p02.php3 <br>
 6  Démo des types de données PHP.<br>
 7  <hr>
 8  <?php
 9   echo "<P>écrit par PHP :<br>";
10
11  $foo = "123";  // $foo is string (ASCII 49,50,51)
12   print ("<br>foo= $foo\n");
13  $foo++;       // $foo is the string "124"
14   print ("<br>foo++= $foo\n");
15  $foo += 3;   // $foo is now an integer (127)
16   print ("<br>foo + 1= $foo\n");
17  $faa = "abc $foo def";
18   print ("<br>faa = $faa\n");
19  $foo = $foo + 1.3;  // $foo is now a double (3.3)
20   print ("<br>foo + 1.3= $foo\n");
21  $foo = 5 + "10 Little Piggies"; // $foo is integer (15)
22   print ("<br>foo= $foo\n");
23  $foo = 5 + "10 Small Pigs";     // $foo is integer (15)
24   print ("<br>foo= $foo\n");
25
26   echo "</P>";
27  ?>
28  Retour au code html.<br>
29  </BODY></HTML> <!---->
```

PHP possède trois types de données : les entiers (integer), les réels (double) et les chaines de caractères (string). C'est un langage **faiblement typé** : on peut mettre la chaine "123" dans une variable, lui **ajouter** 4, on obtient, **suivant le contexte** soit le nombre 127, soit la chaine

"127" (voir $foo += 3 ; et $faa dans l'exemple ci-dessus).

PHP : les opérateurs

12.3 PHP : les opérateurs

Opérateurs arithmétiques

$a + $b	Addition
$a - $b	Soustraction
$a * $b	Multiplication
$a / $b	Division
$a % $b	Modulo

Opérateurs sur les chaines : concaténation

```
$a = "Hello " ;
$b = $a . "World !" ;
// maintenant $b contient "Hello World !"
```

Il exite aussi tout une collection de fonctions portant sur les chaines, ainsi qu'un support des expressions régulières. L'utilisation de l'opérateur arithmétique d'addition en vue de concaténer deux chaînes de caractères est interdite.

Opérateurs binaires "bit à bit"

$a & $b	ET	bits à 1 dans a ET b mis à 1
$a | $b	OU	bits à 1 dans a OU b mis à 1
~$a	NON	bits inversés
<<	décalage	bits décalés vers la gauche
>>	décalage	bits décalés vers la droite

binop.txt

```
1  $a = 5;          // 0000 0101
2  $b = 12;         // 0000 1100
3  $c = $a & $b;    // $c vaut donc  0000 0100 soit 4
4  $d = $a | $b;    // $d vaut donc  0000 1101 soit 13
5  $e = ~ $a;
6  // $e contient le complément de 5, soit 1111 1010
7  // (avec devant autant de 1 que vous voulez) ce qui vaut -6
```

Opérateurs logiques (booléens)

$a and $b	ET	vrai si a ET b vrais
$a or $b	OU	vrai si a OU b vrai
$a xor $b	OU exclusif	vrai si a OU b vrais, mais par les 2 vrais
! $a	NON	vrai si a faux et inverse
$a && $b	ET	vrai si a ET b vrais
$a || $b	OU	vrai si a OU b vrai

Note : && et || font la même chose que "and" et "or", mais avec une précedence supérieure, ce qui évite parfois de mettre des parenthèses. Astuce vaseuse à éviter pour la lisibilité du code.

On dispose également tous les **opérateurs de comparaison** classiques :
`==` , `!=` , `<` , `<=` , `>` , `>=` .

12.4 PHP : instructions de contrôle

p03.php3

```
 1  <HTML><HEAD> <!---->
 2  <TITLE>fichier p03.php3</TITLE>
 3  </HEAD>
 4  <BODY BGCOLOR="#FFFFFF">
 5  <font size="+1"> Début du fichier p03.php3 <br>
 6  Ici code html.<br>
 7  <hr>
 8  <?php
 9   echo "<P>Hello World! écrit par du code PHP.</P>";
10   $a = "12";
11   $b = "34";
12   if ($a > $b) print "<br>a est > b";
13   if ($a < $b) print "<br>a est &lt; b";
14   if ($a > $b) {
15     print "<br>a plus grand que b";
16     print "<br>donc b plus petit que a";
17   } else {
18     print "<br>a plus petit que b";
19     print "<br>donc b plus grand que a";
20   }
21   $a = "123";
22  ?>
23  <br>Retour au code html.<br>
24
25  <?php if ($a=="123") :?>
26  <br>A = 123<br>
27  <?php endif;?>
28
29  </BODY></HTML> <!---->
```

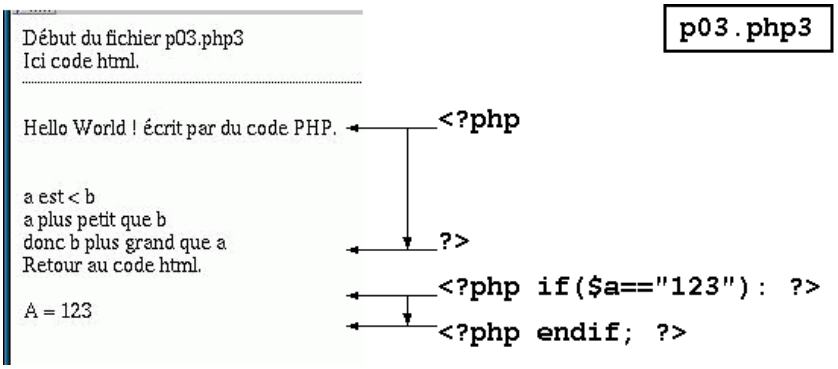


FIG. 138: p03.php3

Instructions de contôle diverses

(condition) ? (valeur si vrai) : (valeur si faux)
if .. elseif .. elseif .. else ..
switch(expr) .. case expr : .. default : ..
while(expr) ...
do...while(expr) ;
for(init ; test ; incr ;)...
break ; quitter une boucle

12.5 PHP : autres possibilités du langage

Création et appels de fonctions
Création et utilisation de tableaux : ceux-ci peuvent être indexés par des nombres (tab[5]) ou par des chaines (tab["abc"]), on parle alors de tableaux associatifs.
Création de classes et instanciation d'objets
Fonctions d'Entrée/Sortie et accès aux fichiers sur le disque du serveur : fopen(), fputs(), fclose(), ...

Fonctions

Par ailleurs, PHP dispose de très nombreuses fonctions de bibliothèque (plusieurs centaines) :

- gestion des échanges avec le navigateur,
- gestion des images,
- création de graphiques,
- accès à des bases de données,
- accès à des serveurs IMAP et LDAP,
- ...

12.6 Interaction entre PHP et le navigateur client

PHP facilite beaucoup la tâche du programmeur de scripts CGI en lui donnant automatiquement et simplement accès aux différents champs des formulaires. En effet, quand un formulaire a pour cible (paramètre "action=") un script PHP, les valeurs des champs du formulaire sont affectées à des variables PHP ayant le même nom que le champ : par exemple, la valeur du champ "nom" se retrouve dans la variabme PHP "$nom".

Exemple avec un formulaire

f1.php3

```
 1  <!----><?
 2  print("
 3  <HTML><HEAD>
 4  <TITLE>f1.php3</TITLE>
 5  </HEAD><BODY>
 6  ");
 7
 8          print("<FORM ACTION=\"f1.php3\" METHOD=\"POST\" >\n");
 9  /* est-ce le premier appel ou le retour du formulaire? */
10  if ( !$premier ) {
11       $premier = 1;
12       print("<FORM ACTION=\"f1.php3\" METHOD=\"POST\" >\n");
13       print("<B>Nom : </B>");
14       print("<INPUT TYPE=\"text\" NAME=\"nom\" >\n");
15       print("<B>Prénom : </B>");
16       print("<INPUT TYPE=\"text\" NAME=\"prenom\" >\n");
17       print(
18       "<INPUT TYPE=\"hidden\" NAME=\"premier\" VALUE=1 >\n");
19       print(
20  "<INPUT TYPE=\"submit\" NAME=\"envoi\" VALUE=\"envoyer\" >\n");
21       print("</FORM>\n");
22       print("</BODY></HTML>\n");
23
24  } else {
25       /* retour */
26
27       print("<B><P>Vous avez entré :</B><BR>\n");
28       print(
29     "<P>nom = $nom   <B>et</B>   prénom = $prenom\n");
30       print("<br><hr>");
31       print("</BODY></HTML>\n");
32  }
33   ?> <!---->
```

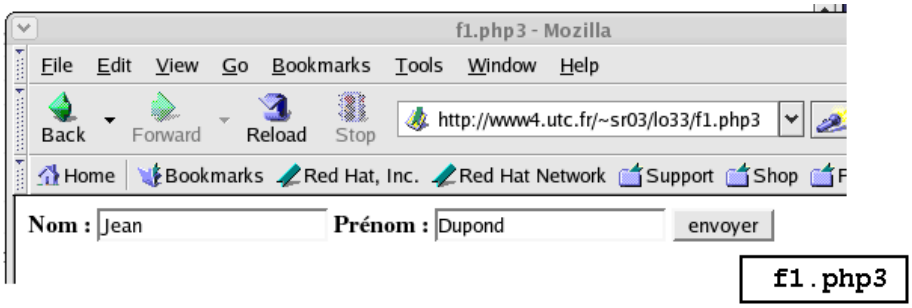


FIG. 139: f1.php3 - Premier appel au script PHP

Le code source HTML envoyé au navigateur est le suivant :

f1-1.html

```
 1  <!---->
 2  <html><head>
 3  <title>f1.php3</title>
 4  </head><body>
 5  <form ACTION="f1.php3" METHOD="POST" >
 6  <b>Nom : </b><input TYPE="text" NAME="nom" >
 7  <b>Prénom : </b><input TYPE="text" NAME="prenom" >
 8  <input TYPE="hidden" NAME="premier" VALUE=1 >
 9  <input TYPE="submit" NAME="envoi" VALUE="envoyer" >
10  </form>
11  </body></html>
12    <!---->
```

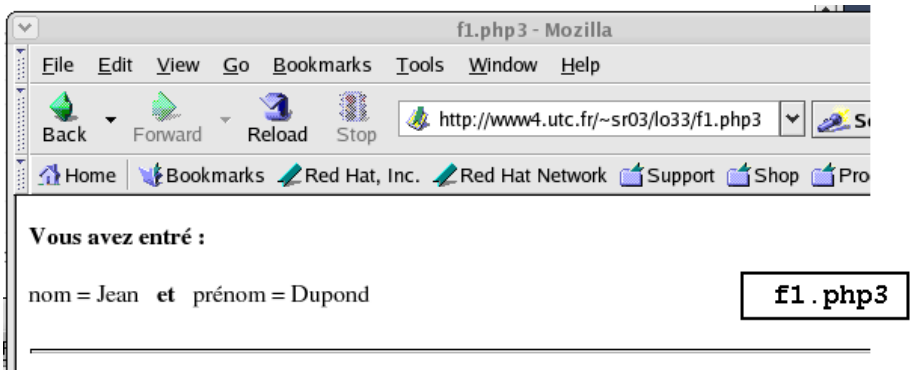


FIG. 140: f1.php3 - Deuxième appel au script PHP

Le code source HTML envoyé au navigateur est le suivant :

f1-2.html

```
1  <!---->
2  <html><head>
3  <title>f1.php3</title>
4  </head><body>
5  <b><p>Vous avez entré :</b><br>
6  <p>nom = Jean   <b>et</b>   prénom = Dupond
7  <br><hr></body></html>
8    <!---->
```

SR03 2004 - Cours Architectures Internet - PHP
Fin du chapitre.

SR03 2004 - Cours Architectures Internet - JavaRMI

13 SR03 2004 - Cours Architectures Internet - JavaRMI

13.1 Java RMI : Remote Method Invocation

C'est l'implémentation du RPC en Java, permettant d'appeler une méthode d'un objet géré dans une autre machine virtuelle (JVM) que celle du code appelant.

- Permettre à du code d'un objet tournant dans une machine virtuelle d'invoquer une méthode d'un objet tournant *dans une autre JVM*.
- les deux JVM peuvent être deux machines tournant dans des process différents sur la même machine ou sur des machines différentes reliées par un réseau TCP/IP.
- la machine contenant l'objet appelé est le serveur,
- la machine contenant l'objet appelant est le client.
- —->
  - il suffit d'une ligne de code en plus côté client pour obtenir une référence sur l'objet distant,
  - il suffit de deux lignes de code côté serveur pour "enregistrer" l'objet appelé et "exposer" ses méthodes,
  - le serveur doit définir la classe et instancier un objet distant,
  - le client ET le serveur doivent avoir accès à une interface qui déclare les méthodes qui peuvent être invoquées à distance,
  - le client et le serveur doivent définir un "RMISecurityManager()" ;
- par l'invocation de méthodes sur l'objet distant, le client peut passer des objets Java en paramètres et recevoir des objets en retour : ceci est réalisé par la capacité de "sérialisation" des objets de Java ;
- le client et le serveur peuvent être compilés sur la même machine (puis les .class déplacés) ou bien compilés séparément,
- le mécanisme de "bootstrap" (point de départ) pour qu'un client obtienne des références à des objets distants est basé sur l'enregistrement de noms d'objets dans un registre, qui peut ensuite être accédé par les clients à l'aide de fonctions et d'une syntaxe de type "URL" : **rmi ://host :port/name**
- **fonctions de la classe "Naming" :**
  - **bind** (String_name, RemoteObj) associe un objet à un nom,
  - **list** (String_URL) renvoie un tableau des noms d'objets dans le registre,
  - **lookup** (String_URL_obj) renvoie une référence sur l'objet,
  - **rebind** (String_name, RemoteObj) remplace un objet par un nouveau,
  - **unbind** (String_name) supprime un nom du registre ;

13.2 Java RMI : premier exemple

Comme premier exemple, on montre une version RMI d'un "Hello World" ; c'est donc un programme minimal. Il requiert 4 fichiers sources :

- **Rmi01Cli.java :** le programme client qui va invoquer une méthode d'un objet distant,

- **Rmi01Ser.java :** le programme serveur qui va instancier l'objet distant et l'enregistrer pour le rendre accessible,
- **Rmi01RemInt.java :** un fichier de définition d'une interface qui déclare les signatures des méthodes d'un objet distant qui peuvent être invoquées par un client à distance,
- **Rmi01RemObj.java :** un fichier de définition de la classe des objets distants dont certaines méthodes (déclarées dans Rmi01RemInt.java) seront invocables à distance ; nouveau,

En plus des 4 sources, il faut deux utilitaires : **rmic** et **rmiregistry**.

- **rmic :** il fait la "compilation" des fichiers "_Stub" et "_Skel" : à partir de Rmi01RemObj.class , il fabrique :
  - **Rmi01RemObj_Stub.class** (pour le client ET le serveur),
  - **Rmi01RemObj_Skel.class** (pour le serveur).
- **rmiregistry :** il crée et maintient un registre des objets d'un serveur, objets dont des méthodes peuvent être invoquées à distance par des clients ; Il peut être invoqué de 2 façons :
  - depuis le serveur à l'exécution par : LocateRegistry.createRegistry(1099) ;
  - comme un process indépendant par : > rmiregistry &
- le serveur enregistre un objet par : Naming.rebind("helloObj", theRemoteObj) ;
- le client accède à un objet par :
  Rmi01RemInt c1 = (Rmi01RemInt)Naming.lookup(partOfUrl+ "helloObj") ;

Compilation et exécution

rmi01-compil.txt

```
 1 _____________________________________________________________________
 2
 3 Rmi01-compilrun.txt
 4 _____________________________________________________________________
 5
 6 /rmi1> ls *.java
 7 Rmi01Cli.java    Rmi01RemInt.java  Rmi01RemObj.java  Rmi01Ser.java
 8 /rmi1> javac *.java
 9 /rmi1> ls *.class
10 Rmi01Cli.class  Rmi01RemInt.class Rmi01RemObj.class  Rmi01Ser.class
11
12 /rmi1> rmic Rmi01RemObj
13 /rmi1> ls *.class
14 Rmi01Cli.class     Rmi01RemObj.class       Rmi01RemObj_Stub.class
15 Rmi01RemInt.class  Rmi01RemObj_Skel.class  Rmi01Ser.class
16
17 ==> création de Rmi01RemObj_Stub.class (côté client et serveur)
18                 et      Rmi01RemObj_Skel.class (côté serveur)
19
20 /rmi1> java Rmi01Ser
21 Remote obj helloObj ready to use
22
23 /rmi1> java Rmi01Cli
24 Hello à vous ! Bonjour !
25
26 _____________________________________________________________________
27
```

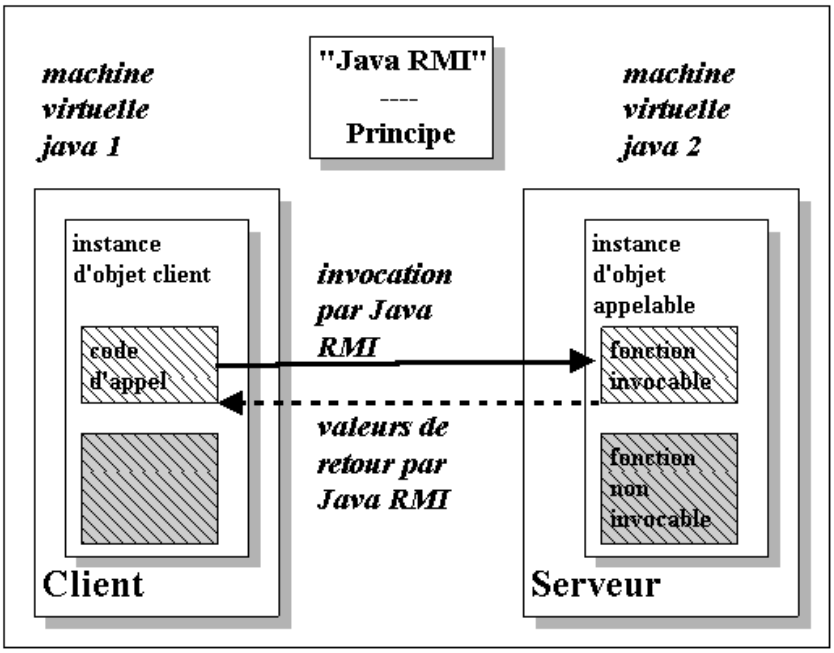


FIG. 141: Exemple rmi01 : exécution

Le client : l'instruction Naming.lookup(partOfUrl + "helloObj") ; obtient une référence sur un objet enregistré sous le nom "helloObj" par le serveur.

Rmi01Cli.java

```
 1  /* Rmi01Cli.java : an RMI hello world program. */
 2
 3   import java.rmi.*;
 4   import java.rmi.server.*;
 5
 6   public class Rmi01Cli{
 7
 8     public static void main(String[] args){
 9       System.setSecurityManager(new RMISecurityManager());
10       String partOfUrl = "rmi ://localhost/";
11       // remplacer par "rmi ://ultraxxx.utc/" si besoin
12       try{
13         Rmi01RemInt c1 =
14               (Rmi01RemInt)Naming.lookup(partOfUrl + "helloObj");
15         System.out.println(c1.helloRemoteObj("Bonjour !"));
16       }catch(Exception e){System.out.println("Erreur " + e);}
17       System.exit(0);
18     }//end main
19   }//end class Rmi01Cli
```

Le serveur : LocateRegistry.createRegistry(1099) ; crée un "registre" (un serveur de noms d'objets) sur le port 1099, puis, Naming.rebind("helloObj", theRemoteObj) ; enregistre une instance de Rmi01RemoteObj avec le nom "helloObj"

Rmi01Ser.java

```
 1  /* Rmi01Ser.java : le serveur distant appelé par RMI */
 2  /* ce serveur "contient" l'objet "helloObj" qui possède
 3     la méthode helloRemoteObj         */
 4   import java.rmi.*;
 5   import java.rmi.server.*;
 6   import sun.applet.*;
 7   import java.rmi.registry.LocateRegistry;
 8
 9   public class Rmi01Ser{
10
11     public static void main(String args[]){
12       System.setSecurityManager(new RMISecurityManager());
13
14       try{
15         Rmi01RemObj theRemoteObj = new Rmi01RemObj();
16
17         LocateRegistry.createRegistry(1099);
18         Naming.rebind("helloObj", theRemoteObj);
19
20         System.out.println("Remote obj helloObj ready to use");
21       }catch(Exception e){System.out.println("Erreur : " + e);}
22     }//end main
23   }//end class Rmi01Ser
```

L'interface : "extends Remote" est obligatoire : seuls les objets qui implémentent "Remote" peuvent être invoqués par RMI.

Dans l'objet distant on trouve :

public class Rmi01RemObj extends UnicastRemoteObject
implements Rmi01RemInt{....

Rmi01RemInt.java

```
1  //File Rmi01RemInt.java
2  import java.rmi.*;
3
4  public interface Rmi01RemInt extends Remote{
5    String helloRemoteObj(String client)
6                                      throws RemoteException;
7  }//end Rmi01RemInt definition d'interface
```

L'objet distant : "extends UnicastRemoteObject" est obligatoire : hérite de la capacité à se comporter comme une extrémité d'un lien de communication.

Rmi01RemObj.java

Rmi01RemObj.java (suite)

```
 1  /* Rmi01RemObj.java : l'objet serveur RMI pour hello world */
 2
 3   import java.rmi.*;
 4   import java.rmi.server.*;
 5
 6   public class Rmi01RemObj extends UnicastRemoteObject
 7   implements Rmi01RemInt{
 8
 9   public Rmi01RemObj() throws RemoteException{
10   super();
11   }//end constructor
12
13   public String helloRemoteObj(String client)  // méthode invocable
14   throws RemoteException{
15   return "Hello à vous ! " + client;
16   }//end method()
17   }//end class Rmi01RemObj
```

13.3 Appel RMI avec échange d'objet distant et d'objet ordinaire

Compilation et exécution

rmi05-compil.txt

```
 1  ____________________________________________________________
 2
 3  /rmi5> ls
 4  Rmi05Server.java   Rmi05RemIntA.java  Rmi05RemObjA.java
 5  Rmi05Client.java   Rmi05RemIntB.java  Rmi05RemObjB.java
 6
 7  /rmi5> javac *.java
 8  Note : Rmi05Client.java uses a deprecated API.
 9  Recompile with "-deprecation" for details.
10  1 warning                   //ignorer ce warning
11  /rmi5> rmic Rmi05RemObjA    //construire _Stub et _Skel
12  /rmi5> rmic Rmi05RemObjB
13  ____________________________________________________________
14
15  /rmi5> java Rmi05Server                 //lancer serveur
16           -- init objet Rmi05RemObjB --
17  Remote obj A ready to use
18
19
20  /rmi5> java Rmi05Client                 //1er run client
21           -- obtenir une ref sur obj serveur --
22           -- 1er test -- objet ordinaire --
23           Demander au serveur un OBJET date ordinaire.
24  demander la date a cet objet date= 1
25  faire date=32 dans objet date
```

rmi05-compil.txt (suite)

```
re-demander la date a cet objet date= 2
re-demander l'OBJET DATE au serveur
demander la date au nouvel OBJET DATE : 1
         -- 2eme test-- objet remote --
         Demander au serveur un OBJET date remote.
demander la date a objet date remote= 99
faire date=33 dans objet date remote
re-demander la date a objet date remote= 33
RE-Demander au serveur l'OBJET date remote.
re-demander date au nouveau objet date remote= 33

/rmi5> java Rmi05Client              //2eme run client
         .....
Demander au serveur un OBJET date remote.
demander la date a objet date remote= 33
.....
--> l'objet serveur a conserve la valeur 33 du 1er run
--> mais l'objet ordinaire Date est "recopié" sur le
    client à partir de la même instance initialisée sur
    le serveur
```

Cet exemple est destiné à montrer la différence entre les objets "ordinaires" (i.e. non-distants) et les objets "distants' ("Remote") lorsque des objets sont passés en argument ou en valeur de retour lors d'un appel RMI.

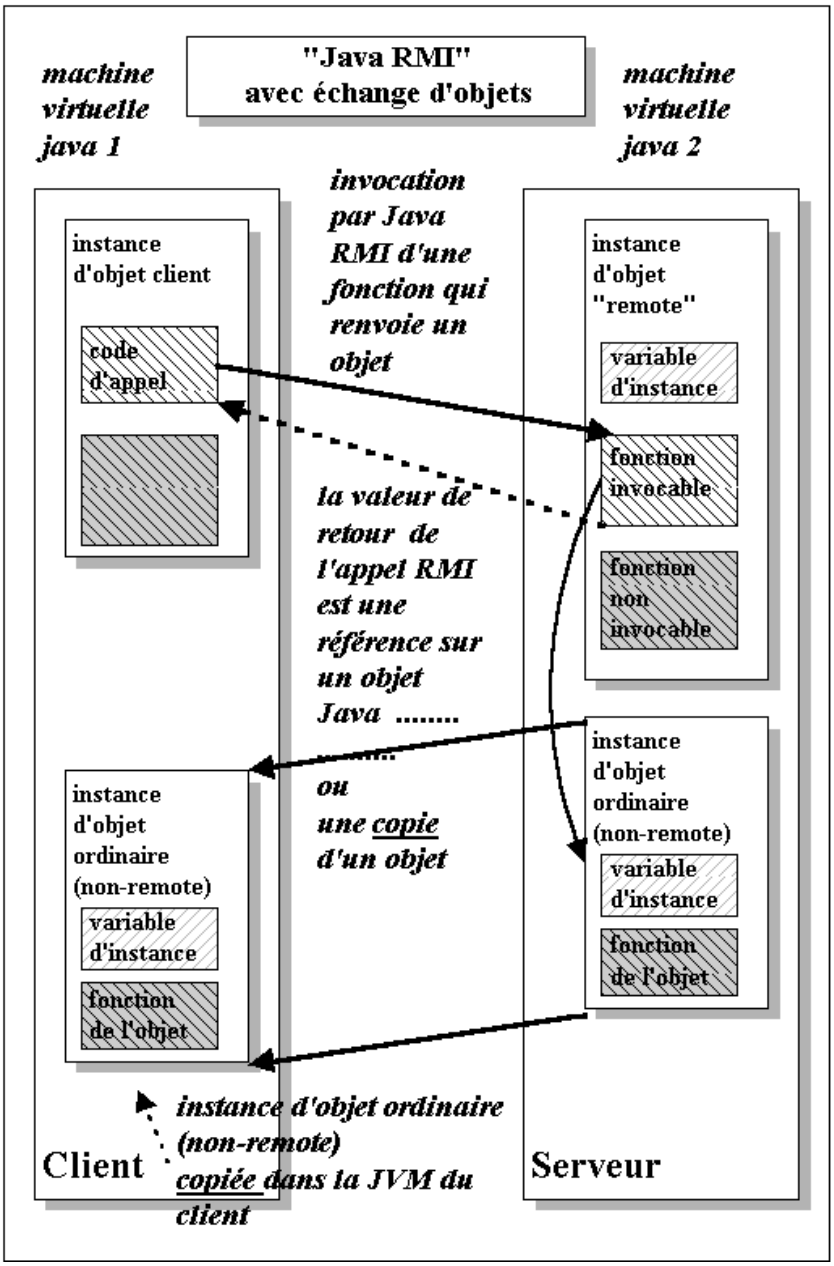


FIG. 142: Exemple rmi05 : exécution

Le client : invoque successivement une méthode sur un objet ordinaire, puis sur un objet "remote".

Rmi05Client.java

```
 1  /* Rmi05Client.java */
 2   import java.rmi.*;
 3   import java.rmi.server.*;
 4   import java.util.*;
 5
 6   public class Rmi05Client{
 7       public static void main(String[] args){
 8       System.setSecurityManager(new RMISecurityManager());
 9       String partOfUrl = "rmi ://localhost/";
10
```

Rmi05Client.java (suite)

```
11       try{
12       System.out.println("-- obtenir une ref sur obj serveur --");
13         Rmi05RemIntA refToObjA =
14            (Rmi05RemIntA)Naming.lookup(partOfUrl + "unObjetA");
15
16      System.out.println("-- 1er test -- objet ordinaire --");
17         System.out.println(
18               "Demander au serveur un OBJET date ordinaire.");
19         Date theDate = refToObjA.getDateObj();
20         System.out.println("demander la date a cet objet date= "
21                                       + theDate.getDate());
22
23         System.out.println("faire date=32 dans objet date");
24         theDate.setDate(32);
25         System.out.println("re-demander la date a cet objet date= "
26                                       + theDate.getDate());
27
28         System.out.println("re-demander l'OBJET DATE au serveur");
29         theDate = refToObjA.getDateObj();
30         System.out.println("demander la date au nouvel OBJET DATE : "
31                                       + theDate.getDate());
32
33      System.out.println("-- 2eme test-- objet remote --");
34         System.out.println(
35               "Demander au serveur un OBJET date remote.");
36         Rmi05RemIntB theObj = refToObjA.getRemObj();
37         System.out.println("demander la date a objet date remote= "
38                                        + theObj.getData());
39
40         System.out.println("faire date=33 dans objet date remote");
41         theObj.setData(33);
42         System.out.println(
43               "re-demander la date a objet date remote= "
44                                        + theObj.getData());
45
46         System.out.println(
47               "RE-Demander au serveur l'OBJET date remote.");
48         theObj = refToObjA.getRemObj();
49         System.out.println(
50               "re-demander date au nouveau objet date remote= "
51                                        + theObj.getData());
52
53       }catch(Exception e){System.out.println("Error " + e);}
54       System.exit(0);
55     }//end main
56   }//end class Rmi05Client
```

Les interfaces :

Rmi05RemIntA.java

Rmi05RemIntA.java (suite)

```
 1  /* Rmi05RemIntA.java */
 2  /*
 3   Définition d'une interface RMI destinée à montrer la
 4   différence entre les objets "ordinaires" (i.e. non-distants)
 5   et les objets "distants' ("Remote") lorsque des objets sont
 6   passés en argument ou en valeur de retour.
 7   Objet "Remote" = objet qui implémente l'interface Remote ou
 8   qui implémente une interface qui en dérive (extends Remote).
 9  */
10
11   import java.rmi.*;
12   import java.util.*;
13
14   public interface Rmi05RemIntA extends Remote{
15     Date getDateObj() throws RemoteException;
16     Rmi05RemIntB getRemObj() throws RemoteException;
17
18   }//end Rmi05RemIntA definition
```

Rmi05RemIntB.java

```
 1  /* Rmi05RemIntB.java */
 2  /* Voir commentaires dans fichier Rmi05RemIntA.java */
 3
 4   import java.rmi.*;
 5
 6   public interface Rmi05RemIntB extends Remote{
 7     void setData(int data) throws RemoteException;
 8     int getData() throws RemoteException;
 9
10   }//end Rmi05RemIntB definition
```

Le serveur :

Rmi05Server.java

```
 1  /* Rmi05Server.java */
 2  /*
 3   Un serveur destiné à montrer la différence entre les objets
 4   "ordinaires" (i.e. non-distants) et les objets "distants"
 5   ("Remote") lorsque des objets sont passés en argument ou en
 6   valeur de retour.
 7  */
 8   import java.rmi.*;
 9   import java.rmi.server.*;
10   import sun.applet.*;
11   import java.rmi.registry.LocateRegistry;
12
```

Rmi05Server.java (suite)

```
13  public class Rmi05Server{
14     public static void main(String args[]){
15       System.setSecurityManager(new RMISecurityManager());
16
17       try{
18         Rmi05RemObjA remoteObjA = new Rmi05RemObjA();
19
20         LocateRegistry.createRegistry(1099);
21         Naming.rebind("unObjetA", remoteObjA);
22
23         System.out.println("Remote obj A ready to use");
24       }catch(Exception e){System.out.println("Error : " + e);}
25     }//end main
26   }//end class Rmi05Server
```

Les implémentations des classes de service :

Rmi05RemObjA.java

```
 1  /* Rmi05RemObjA.java */
 2  /*
 3   Un exemple de classe d'objet distant pour montrer le passage
 4   d'objets distants ou ordinaires dans un appel RMI
 5   Cette classe implemente Remote indirectement à travers
 6   l'héritage par Rmi05RemIntA.
 7  */
 8   import java.rmi.*;
 9   import java.rmi.server.*;
10   import java.util.*;
11
12   public class Rmi05RemObjA extends UnicastRemoteObject
13                                implements Rmi05RemIntA{
14
15     private Date theDate = new Date();
16     private Rmi05RemObjB theRemoteObj;
17
18     public Rmi05RemObjA() throws RemoteException{
19       theRemoteObj = new Rmi05RemObjB(99);
20     }//end constructor
21
22     public Date getDateObj() throws RemoteException{
23       return theDate;
24     }//end getDateObj()
25
26     public Rmi05RemIntB getRemObj() throws RemoteException{
27       return theRemoteObj;
28     }//end getRemObj()
29
30   }//end class Rmi05RemObjA
31
32  /* commentaires :
```

Rmi05RemObjA.java (suite)

```
33   * l'objet Rmi05RemObjA déclare une variable d'instance "theDate"
34   * qui est une référence sur un objet, initialisée avec une
35   * instance d'un objet date du système :
36   *      private Date theDate = new Date();
37   * cette variable privée est initialisée lors de l'instanciation
38   * par le serveur de l'objet Rmi05RemObjA :
39   *      Rmi05RemObjA remoteObjA = new Rmi05RemObjA();
40   *
41   * puis, il instancie (dans le constructeur) un objet Rmi05RemObjB
42   * qui est du type remote en lui donnant une valeur d'initialisation
43   * que cet objet va stocker dans sa variable d'instance :
44   *      private int data; // variable d'instance de Rmi05RemObjB
45   *      this.data = data; // init dans le constructeur
46   * Ceci n'est fait qu'UNE fois : le serveur n'imprime qu'une fois
47   * "-- init objet Rmi05RemObjB --".
48   *
49   */
```

Rmi05RemObjB.java

```
 1  /* Rmi05RemObjB.java */
 2  /*
 3   Un exemple de classe d'objet distant pour montrer le passage
 4   d'objets distants ou ordinaires dans un appel RMI
 5   Cette classe implemente Remote indirectement a travers
 6   l'heritage par Rmi05RemIntA.
 7  */
 8
 9   import java.rmi.*;
10   import java.rmi.server.*;
11   import java.util.*;
12
13   public class Rmi05RemObjB extends UnicastRemoteObject
14                              implements Rmi05RemIntB{
15     private int data;
16
17     public Rmi05RemObjB(int data) throws RemoteException{
18       System.out.println("-- init objet Rmi05RemObjB --");
19       this.data = data; //initialize the object
20     }//end constructor
21
22     public void setData(int data) throws RemoteException{
23       this.data = data;
24     }//end setData()
25
26     public int getData() throws RemoteException{
27       return data;
28     }//end getData()
29   }//end class Rmi05RemObjB
```

13.4

SR03 2004 - Cours Architectures Internet - Java, le client-serveur et le web
©Michel.Vayssade@utc.fr – Université de Technologie de Compiègne.

14 SR03 2004 - Cours Architectures Internet - Java, le client-serveur et le web

14.1 Le client-serveur et le web

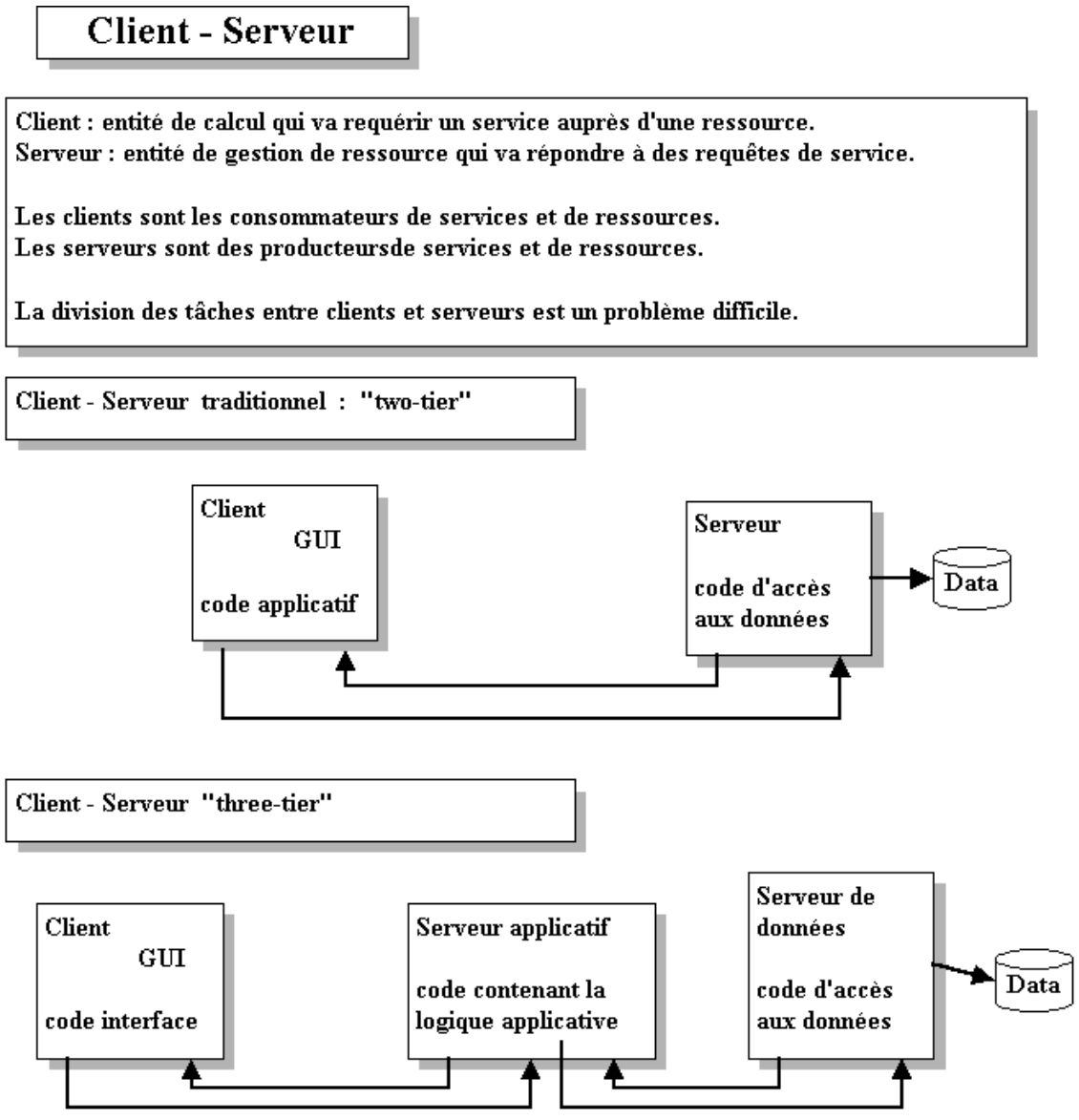


FIG. 143: Évolution du mode Client-Serveur

Le client-serveur traditionnel a évolué vers une architecture plus complexe, mais plus souple : le client-serveur dit "3 tiers", constitué de 3 éléments indépendants interagissant. Les différentes raisons de cette évolution seront détaillées plus loin dans le chapitre sur les **serveurs d'applications**.

Client - Serveur : la "collision" avec le Web

Les applications client/serveur traditionnelles faisaient communiquer une application sur un ordinateur personnel (PC ou Mac), chargé de gérer toute l'interactivité avec l'utilisateur avec un serveur chargé de stocker, conserver et consolider les données.

Puis à partir de 94 une nouvelle sorte d'application client/serveur s'est répendue à côté de l'existant : le web, avec un brower-client accédant des serveurs-web.

La simplicité d'accès au web (apprentissage = 0) a fortement poussé à une convergence. D'autant plus que le web était la seule application client-serveur facilement déployable sur l'Internet.

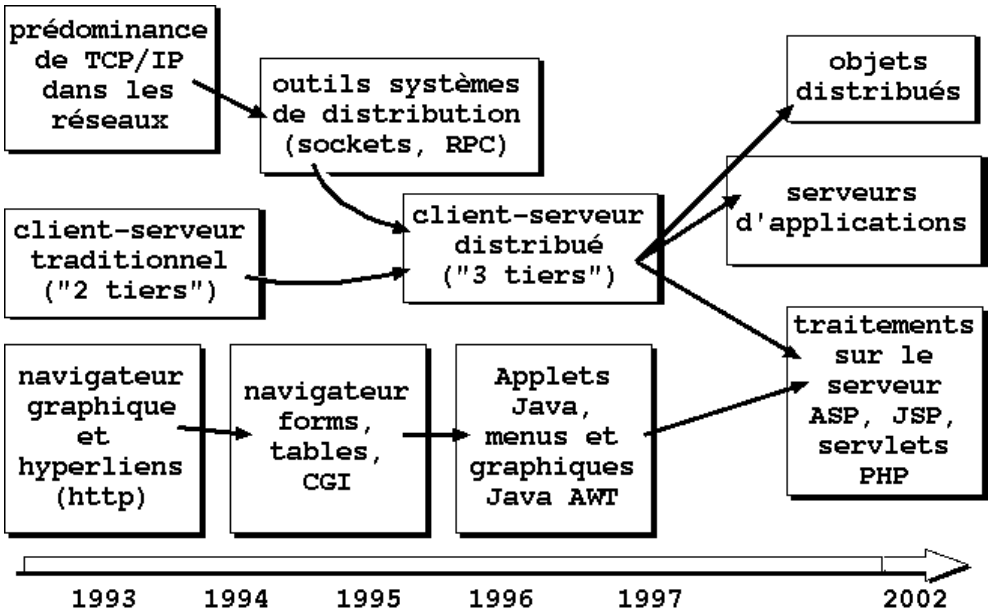


FIG. 144: Le Client-Serveur et le Web

14.2 Java et le web

En 5 ans l'Internet est passé du statut de "jouet privé" pour les univesitaires et chercheurs à celui d'une application client/serveur universelle.

L'application clé ("the killer application") qui a provoqué ce changement est le World Wide Web (www, ou le web, ou "la toile").

Le web a évolué rapidement depuis sa création qui peut être datée de l'apparition du browser Mosaic, fin 93. Au début de 97, le web comptait 120 000 serveurs et 20 millions d'utilisateurs, la taille du web doublant tous les 2 mois (+3000 serveurs chaque jour !) et le nombre d'utilisateurs croissant de 1 million par mois.

Parallèlement, les techniques d'interaction avec l'utilisateur du web ont augmenté en interactivité.

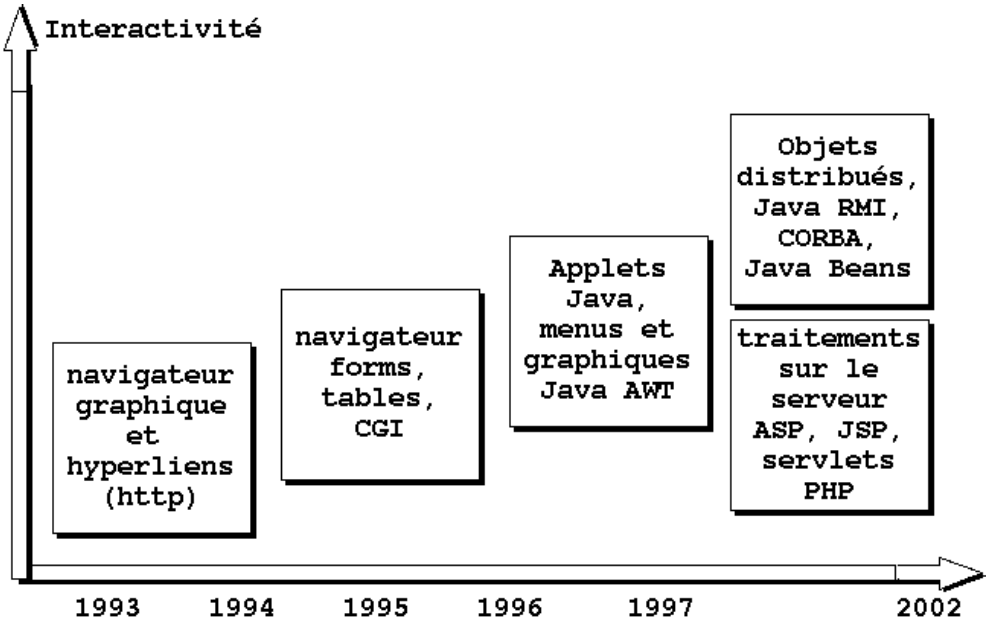


FIG. 145: L'interactivité sur le web

Java a été perçu comme un moyen de fournir au concepteur de sites interactifs les éléments dont manquaient les navigateurs de l'époque.

Java a permis un tout nouveau modèle d'interactivité sur le web : il permet d'écrire des "programmes composants", appelés "Applets", qui peuvent être téléchargés depuis un serveur web vers un client (dans un browser), qui, s'il est compatible java, pourra les exécuter localement sur le poste du client.

Les Applets permettent de distribuer à travers le web du contenu exécutable en même temps que des données.

Tout ceci explique pourquoi l'usage de java s'est développé alors qu'on avait pas vraiment besoin d'un langage de plus.

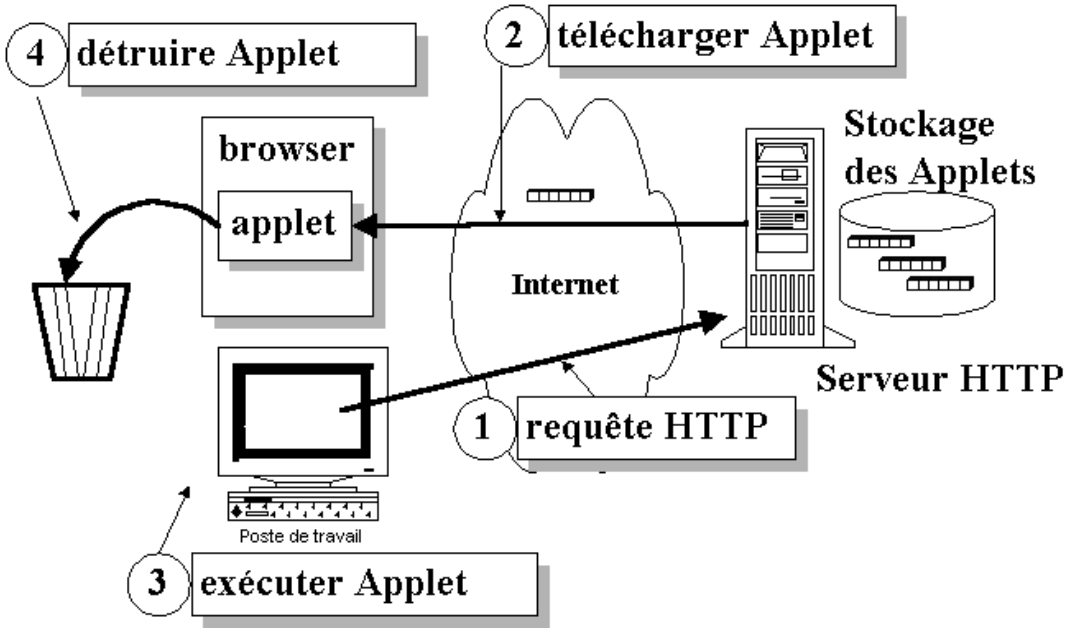


FIG. 146: Client-Serveur en Java

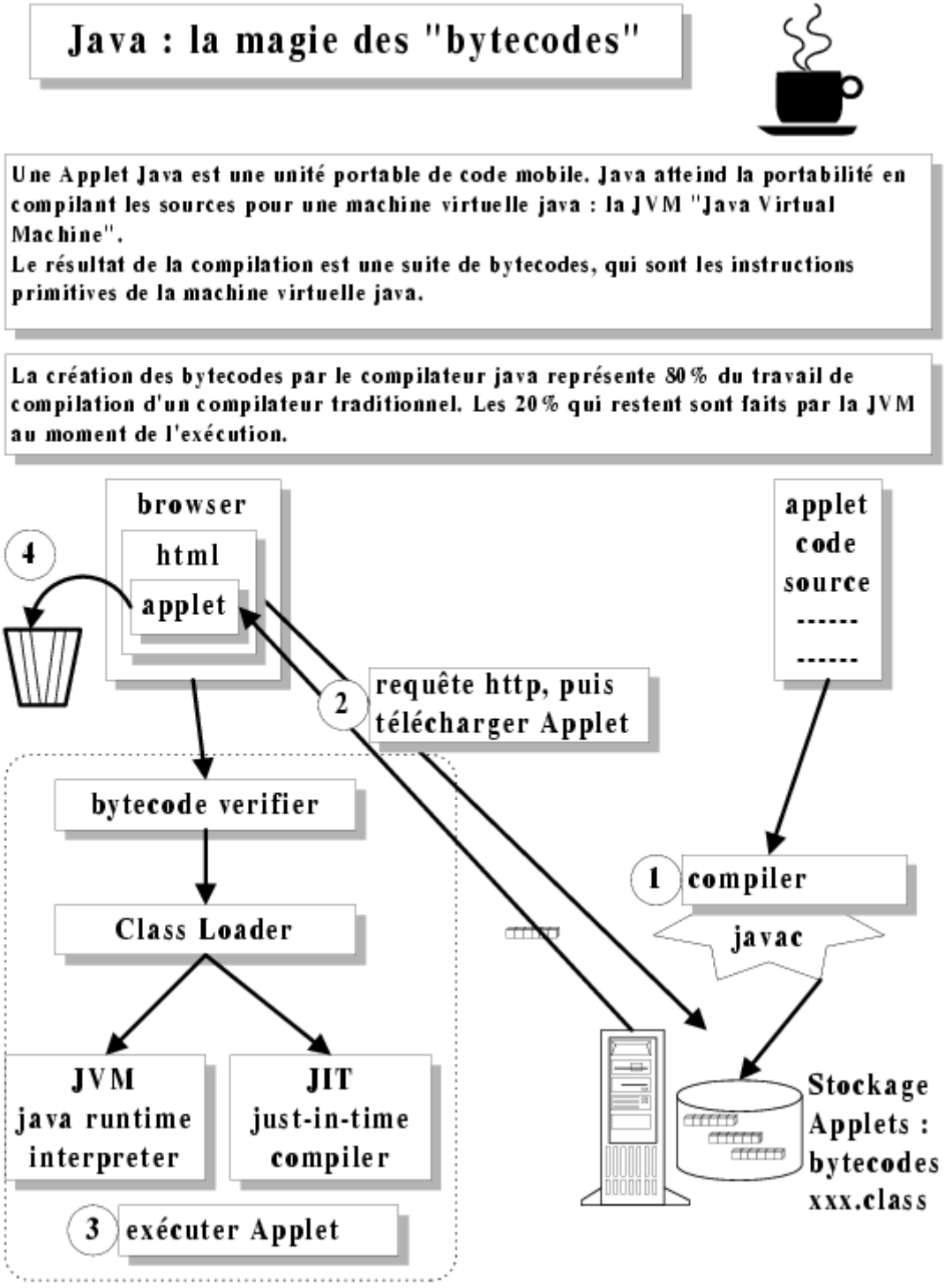


FIG. 147: Java : les bytecodes

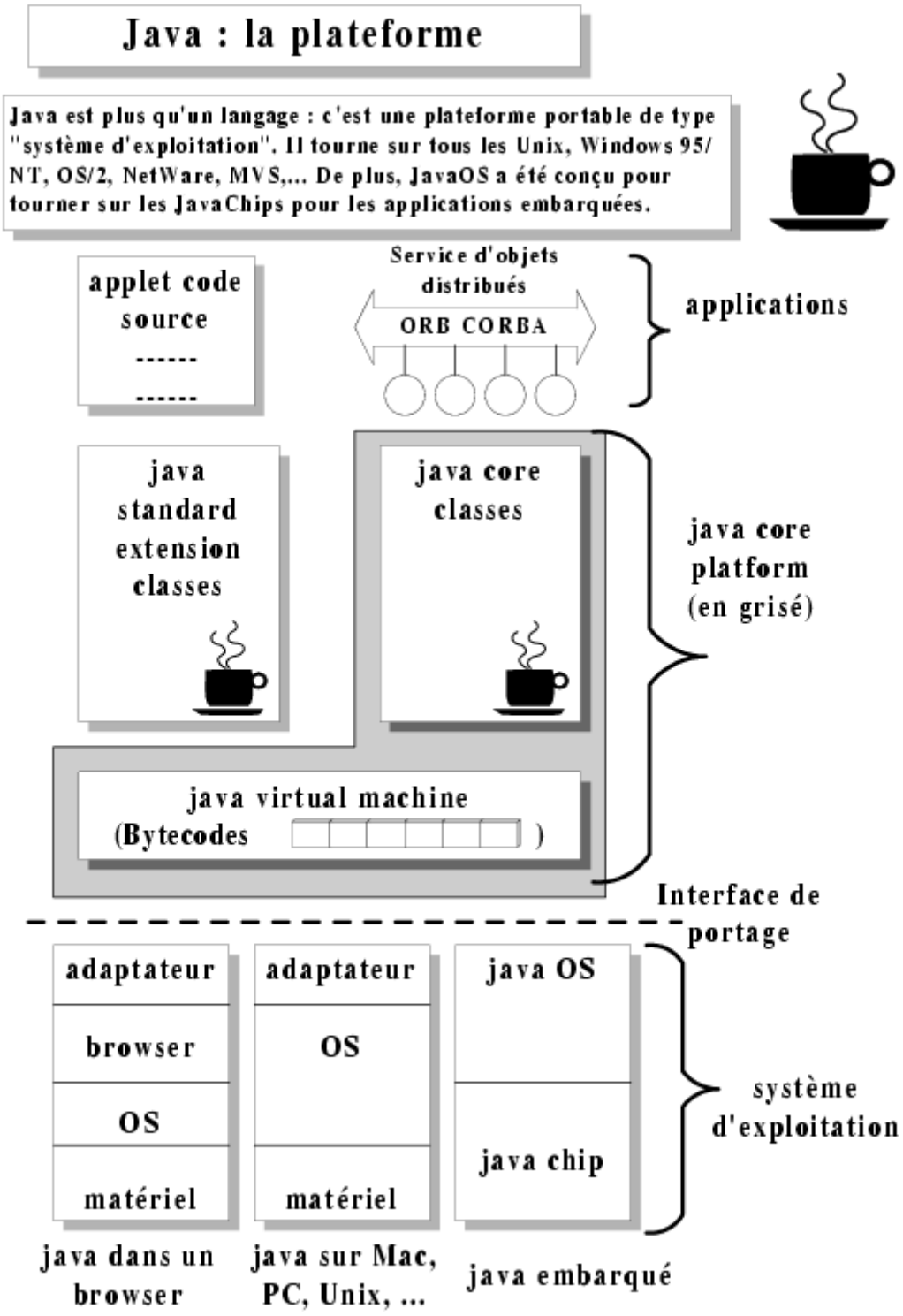


FIG. 148: La plateforme Java

14.3 Introduction au modèle de programmation du système Java

Introduction générale

Java n'est pas seulement un langage. C'est aussi une synthèse des évolutions de l'informatique, depuis la création de Smaltalk jusqu'à celle du web.

été 1993 : envol du web, développé au CERN ;

23 mai 1995 : annonce de Java par Sun ;

Java apparrait à la croisée de 3 évolutions : celle des ordinateurs, celle des langages et celle des réseaux.

Il est utile de distinguer le **langage java** du **système java** (on parle aussi de la **plateforme java**).

Le système java

- "Java is not a JAL" ! (Just Another Language),
- l'art de la programmation est dans la gestion de la complexité : complexité du problème à résoudre **plus** complexité de la machine avec laquelle on essaie de le résoudre,
- Java est sans doute le premier langage pour lequel un des objectifs majeurs de sa conception est de diminuer la complexité de conception et de maintenance de programmes,
- –> réduire le temps nécessaire au programmeur, et la difficulté, pour produire du code robuste,
- –> moitié moins long que pour produire un équivalent C++
- donc, plus facile de :
  - créer des programmes,
  - travailler en groupe à la création de programmes,
  - construire des interfaces utilisateur,
  - exécuter les programmes sur différentes machines,
  - écrire des programmes qui communiquent sur un Internet.
- il ne faut pas penser aux fonctions de Java en termes de **codage**, mais en termes de **design**.

Le langage java Java possède des caractéristiques modernes permettant d'atteindre des objectifs de productivié, rentablité, pérennité et sécurité. En particulier, java est :

- orienté objet,
- multithread,
- fortement et statiquement typé,
- à liaison dynamique,
- gérant la mémoire,
- simplifié,
- sécurisé.

Chacun de ces points sera détaillé plus loin dans le chapitre sur le langage java.

Du langage java à la plateforme Java

Il y a une autre des caractéristiques de Java dont on a pas encore parlé, et qui est une des plus importantes : la **portabilité**.

Les premiers essais de portabilité ont été ceux des langages procéduraux traditionnels : Cobol, Fortran, Pascal, C. Mais bien qu'un programme C, par exemple, soit en théorie portable sur de nombreuses machines, en pratique l'énorme diversité des bibliothèques dépendantes du système interdisent d'espérer qu'un programme de taille conséquente puisse fonctionner à l'identique par simple recompilation.

Ceci a conduit un certain nombre de projets à modifier la struture de la relation entre le langage et le système d'exploitation, en faisant "remonter" un maximum de fonctions, habituellement implantées dans les bibliothèques du système, au niveau du langage lui-même. On

peut citer dans cette optique : Smaltalk, Ada et Objective C / Next. Au passage, on réduit le système d'exploitation à ses fonctions de base (gestion mémoire, gestion multitâche, gestion des communications entre process) pour aboutir au concept de mico-noyau (Mach, Chorus).

La plateforme Java, en intercalant la machine virtuelle Java entre le système d'exploitation et le langage proprement dit et ses bibliothèques, franchit une étape de plus vers la portabilité universelle des applications.

Architecture de la plateforme Java

La plateforme Java est constituée d'un ensemble de composants qui sont tous, sauf la machine virtuelle, des bibliothèques de classes écrites en Java. Mais à la différence des bibliothèques C des systèmes d'exploitation, il ne s'agit pas d'une simple collection de routines juxtaposées, mais d'un ensemble hiérarchique rendu cohérent par la nature orientée objet du langage Java.

Architectures client-serveur et Java

La plateforme Java a été conçue pour faciliter l'implantation d'applications client-serveur. Pariculièrement, pour rendre facile le déploiement de la partie cliente sur un grand nombre de postes, et sa mise à jour, deux opérations qui, dans le cadre des applications client-serveur traditionnelles ont vite fait de tourner au cauchemar pour les administrateurs systèmes, à cause de l'instabilité de la plateforme client "Windows xx".

La section suivante va montrer comment le système Java permet de faire évoluer les architectures client-serveur.

14.4 Architectures client-serveur et Java

Les schémas présentent successivement différentes types d'architecture client-serveur, qui diffèrent par le type de code (GUI : gestion interface utilisateur, applicatif (traitements fonctionnels) ou accès aux données (base de données SQL ou objet)) qui est confié au programme client ou à la plateforme serveur. On montre successivement :

1. Architecture client-serveur traditionnelle à deux niveaux avec les traitements sur le client.
2. Architecture client-serveur traditionnelle à deux niveaux avec les traitements sur le serveur.
3. Architecture client-serveur à trois niveaux : séparation plus nette entre l'interface utilisateur, les traitements et l'accès aux données.
4. Architecture client-serveur à deux niveaux avec arpenteur, requête HTTP et CGI.
5. Architecture client-serveur à trois niveaux : navigateur, CGI, traitements et accès aux données.
6. Architecture client-serveur à deux niveaux avec arpenteur et téléchargement de code Java à la demande. Les objets Java peuvent ouvrir un canal de communication direct avec la source de données.
7. Architecture client-serveur à trois niveaux avec arpenteur et téléchargement de code Java à la demande. Les objets Java téléchargés sur le client gèrent l'interface et communiquent avec des objets applicatifs sur un serveur par des canaux ORB ou RMI. Les objets applicatifs ouvrent un canal de communication direct avec la source de données. Il y a indépendance entre le code de présentation, le code fonctionnel et le code d'accès aux données.

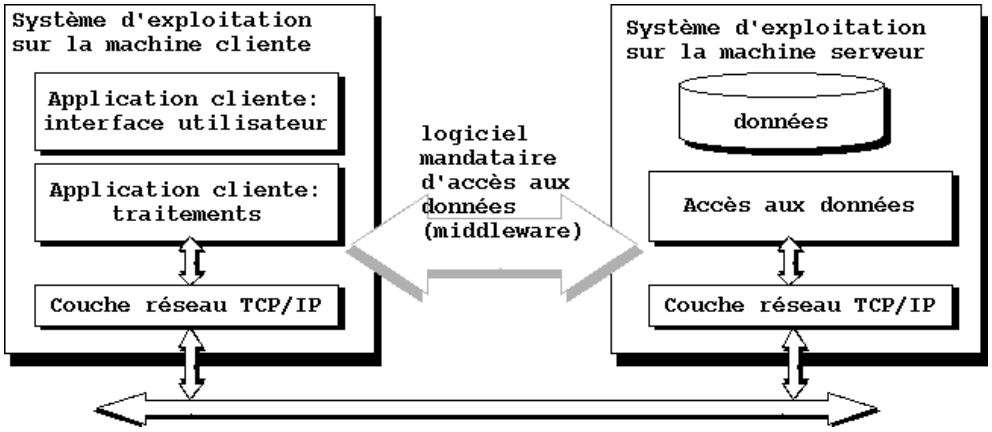


FIG. 149: Client-Serveur et Java 1/7

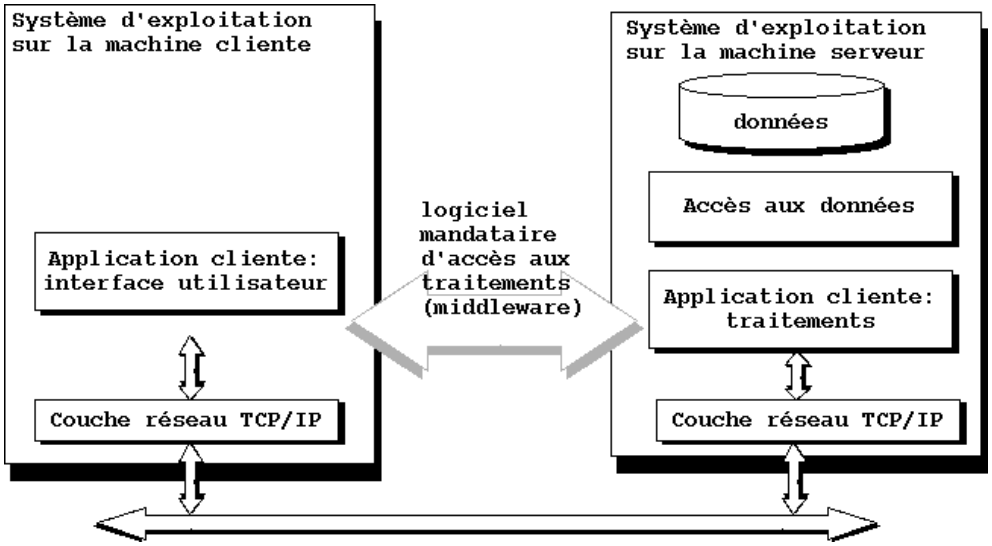


FIG. 150: Client-Serveur et Java 2/7

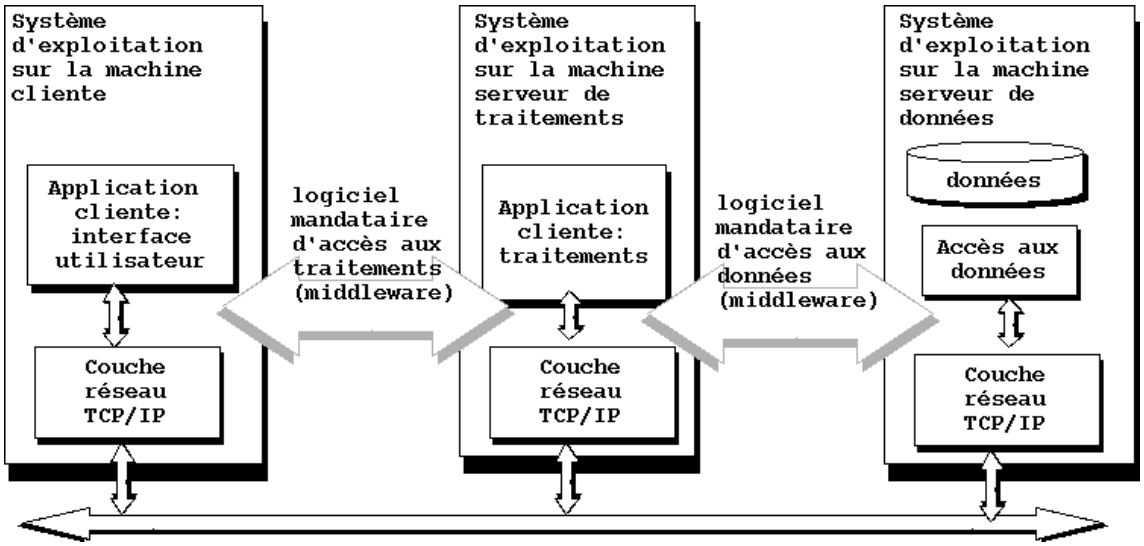


FIG. 151: Client-Serveur et Java 3/7

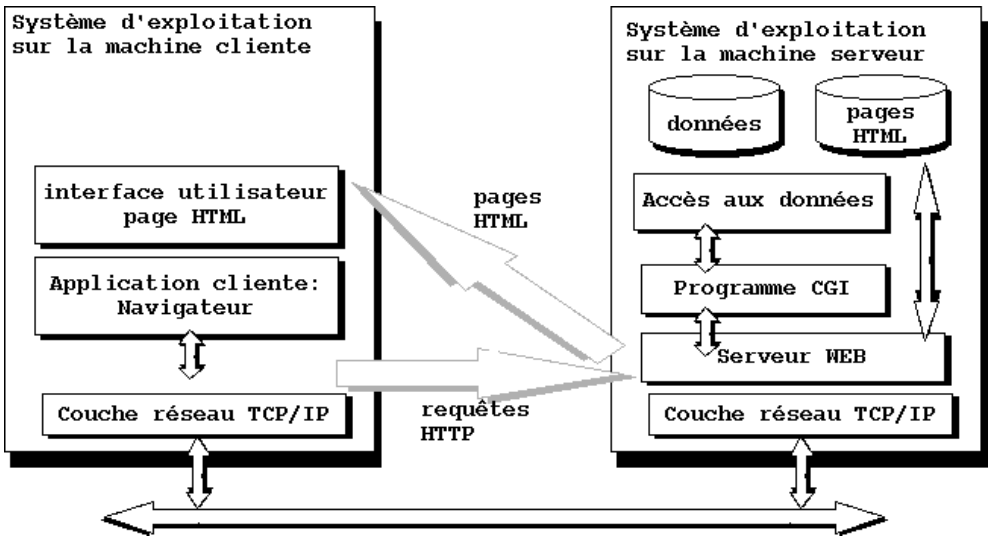


FIG. 152: Client-Serveur et Java 4/7

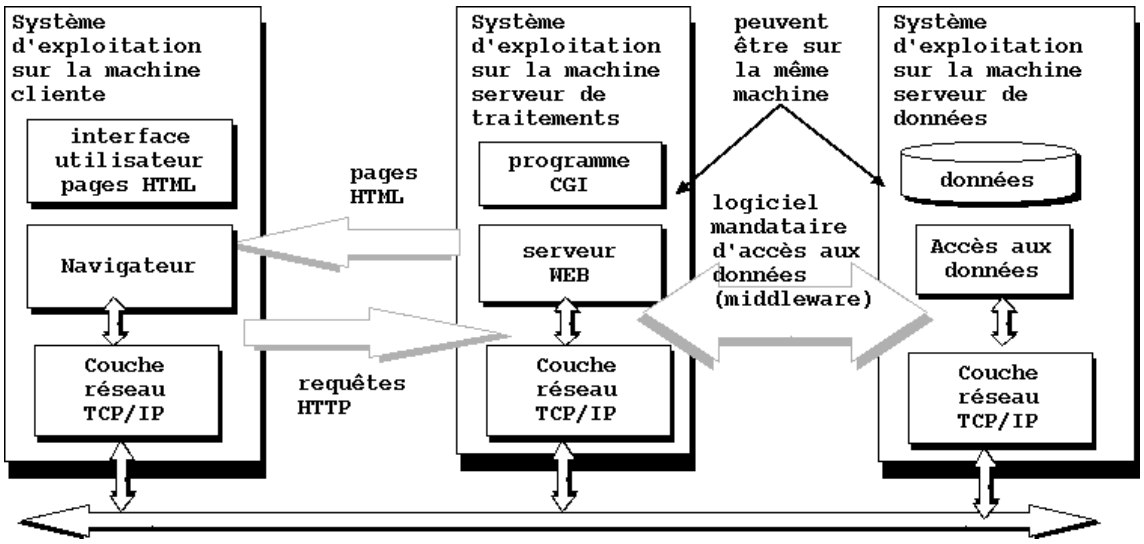


FIG. 153: Client-Serveur et Java 5/7

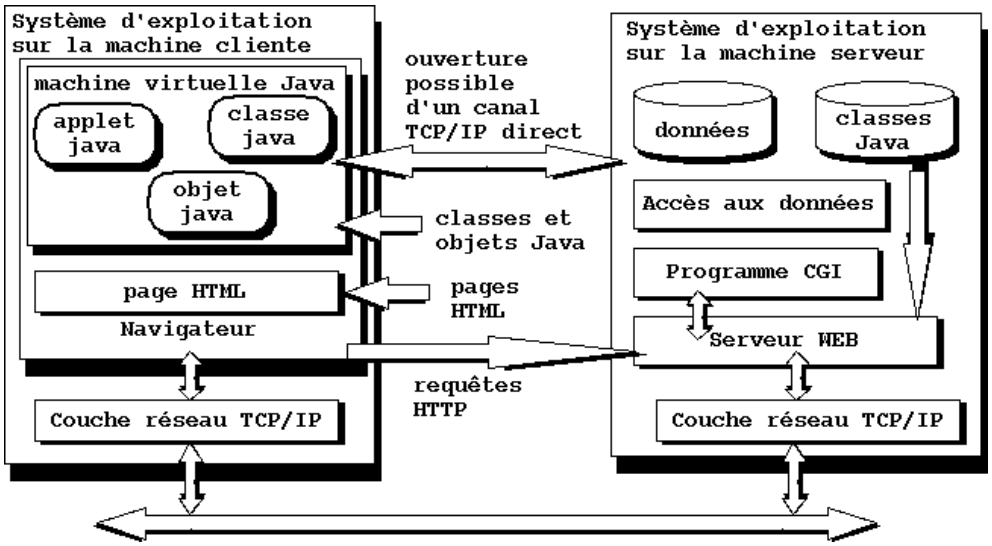


FIG. 154: Client-Serveur et Java 6/7

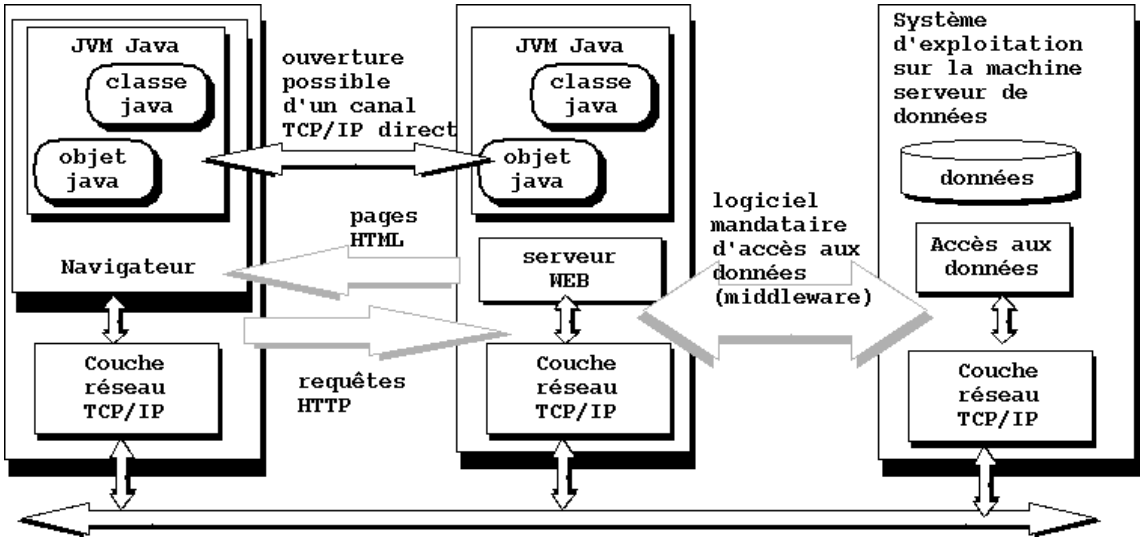


FIG. 155: Client-Serveur et Java 7/7

SR03 2004 - Cours Architectures Internet - Java, le client-serveur et le web
Fin du chapitre.

SR03 2004 - Cours Architectures Internet - Le système Java

15 SR03 2004 - Cours Architectures Internet - Le système Java

15.1 L'architecture du système Java : introduction

Le système Java c'est l'ensemble du langage Java, de la machine virtuelle java (la JVM), de l'API Java et des fichiers de classes Java.

On a toujours utilisé des outils pour aider le concepteur dans les tâches de programmation. Ce n'est que recemment ces outils sont devenus plus complexes, et surtout interconnectés. Les microprocesseurs fabriqués en très grande série n'ont pas seulement donné des stations de travail et des ordinateurs personnels, ils se sont aussi insérés dans un grand nombre d'objets, objets qui sont maintenant de plus en plus souvent connectés à un réseau : fax, matériels vidéo, systèmes de contrôle d'accès, automobiles, etc...

Ce nouvel environnement d'objets interconnectés et disposant d'une capacité de traitement local de l'information, crèe de nouvelles opportunités, mais aussi de nouveaux problèmes techniques pour la mise en œuvre.

Le système Java est un outil bien adapté à ce nouvel environnement et permet de résoudre ou contourner certaines difficultés.

Première difficulté : la diversité

Un réseau typique sera constitué de nombreuses machines de type différents, utilisant des systèmes d'exploitation différents. Java contourne cette difficulté en permettant de créer des programmes indépendants de la plate-forme sous-jacente. Ceci grâce à la machine virtuelle java.

Deuxième difficulté : la sécurité

En plus de leur potentiel "positif", les réseaux peuvent aussi être utilisés dans des buts délictueux : vol ou destruction d'informations, perturbation volontaire du bon fonctionnement du réseau ou des machines connectées au réseau. Java offre au programmeur d'applications différents moyens de contrôler l'environnement d'exécution (et donc les actions possibles) des programmes Java.

Écrire des logiciels complexes est une entreprise difficile et mal maîtrisée. Il en résulte de nombreux logiciels "fragiles" ou "instables" : les "plantages" de certains logiciels sont devenus tellement habituels que de nombreux utilisateurs nouveaux de l'informatique ne songent même plus à s'en offusquer. Imaginez un instant une automobile ou un poste de télévision qui tomberait en panne tous les deux jours ! L'architecture de Java incorpore un certain nombre de techniques qui visent à réduire cette fragilité, et à empêcher qu'elle ne se propage (un programme qui "se" plante, et qui en plus "plante la machine" !).

Aujourd'hui, presque tous les utilisateurs ont été confrontés au problème des "versions". On produit un document avec un logiciel, et on veut échanger ce document avec quelqu'un

d'autre ... qui ne peut pas le réutiliser parce que "ce n'est pas la bonne version". Il faut alors que l'un des deux "installe la bonne version" : les ennuis commencent. En effet une version donnée d'un logiciel exigera pour fonctionner la "bonne" version du système d'exploitation ou d'un composant du système d'exploitation, et cette bonne version pour le logiciel "A" ne sera peut-être pas la même que celle requise par le logiciel "B", dont l'utilisateur a AUSSI besoin pour travailler avec une troisième personne ... D'autre part pour installer une nouvelle version, il faut disposer d'une source : disquettes, cd-rom ou site Internet, contenant une copie du logiciel à installer. Cette source doit être construite et mise à jour pour toute version nouvelle de tout logiciel. De plus, l'utilisateur, n'a pas toujours la compétence nécessaire pour installer correctement le logiciel. Il faut alors que quelqu'un, dans l'organisation, se charge de faire ou d'organiser les installations multiples. Ceci peut devenir un vrai cauchemar sur des parcs de plusieurs centaines ou plusieurs milliers de machines.

Ce problème du déploiement d'une version nouvelle d'un logiciel sur de nombreuses machines, éventuellement de types différents, est fortement facilité par l'architecture de Java.

15.2 L'architecture du système Java : description

Le système Java c'est l'ensemble du langage Java, de la machine virtuelle java (la JVM), de l'API Java et des fichiers de classes, chacun de ces quatre éléments étant défini par une spécification publiée par Sun Microsystems.

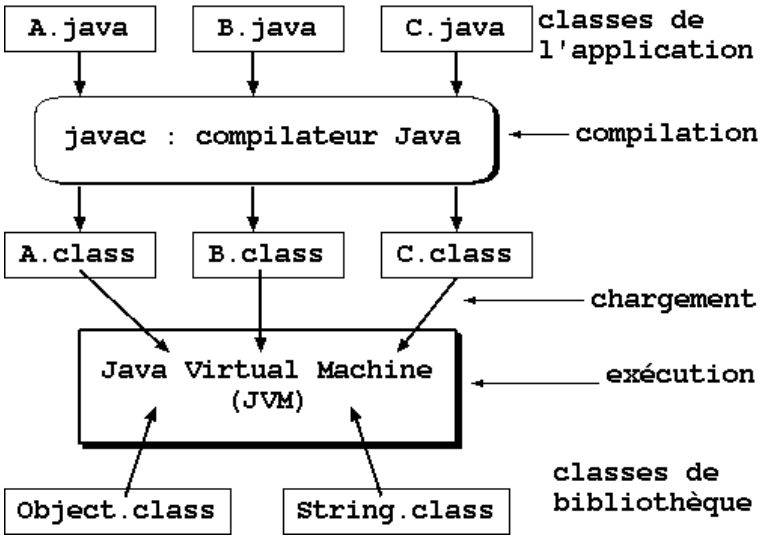


FIG. 156: Fonctionnement de Java

- Les programmes sources, écrits par le développeur, sont stockés dans des fichiers ".java" ;
- Le compilateur "javac" transforme les sources en "bytescode" et génère un fichier ".class" pour chaque classe java ;
- Les fichiers de classes sont envoyés du serveur au client, sur la machine locale ou à travers un internet (réseau TCP/IP),
- Les fichiers .class sont chargés dans la JVM ;
- La JVM interprète les "bytescode" contenus dans les .class. Ceux-ci peuvent faire appel à des objets pré-définis dans les bibliothèques des Java-API.
- Les bibliothèques d'objets pré-définis. Ces objets implémentent les accès aux ressources systèmes en fournissant aux programmes Java une interface commune, identique sur tous les systèmes.

Prises ensemble, la machine virtuelle java et les API java forment une "plateforme" pour laquelle les programmes Java sont compilés. La plateforme JVM + Java API est aussi appelée "Java Runtime System". Ce "système d'exécution java" peut lui-même être implémenté sous forme d'un logiciel et donc être porté sur toute sorte de matériels et de systèmes d'exploitation différents, comme illustré sur la figure ci-dessous.

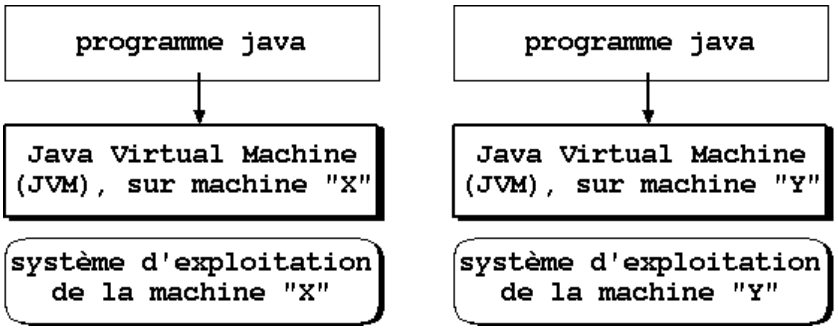


FIG. 157: Java exécute le même binaire partout

La machine virtuelle Java (JVM)

La JVM est le cœur du système Java. C'est une machine **abstraite** dont les spécifications définissent les caractéristiques que doivent posséder **toutes** les machines virtuelles Java. Mais ces spécifications laissent toute liberté aux concepteurs sur la façon de réaliser la machine : on peut faire des JVM uniquement logicielles (par exemple s'exécutant au sein d'un navigateur), mais aussi des machines entièrement matérielles, gravées dans un circuit intégré.

Le travail de la JVM est de charger les fichiers de classes contenant les bytecoodes à exécuter. Le chargeur de classes ("class loader") mets à disposition de la machine d'exécution les classes du programme utilisateur ainsi que toutes les classes de l'API Java utilisées par le programme (et seulement celles là).

JIT et JNI : Just In Time Compiler et Java Native Interface

La JVM peut être implantée de plusieurs façons :
- sous la forme d'un interpréteur : les bytecodes sont lus et interprétés "à la volée",
- sous la forme d'un "compilateur juste à temps" : les bytecodes sont lus et compilés à la volée, puis les classes compilées sont stockées localement par la JVM. Quand elles sont réexécutées, cette nouvelle exécution est beaucoup plus rapide,
- dans les deux cas ci-dessus, le machine peut être soit logicielle soit matérielle.

De plus, pour interagir avec le système d'exploitation de la machine hôte, la JVM appelle des **méthodes natives** implantées dans des bibliothèques précompilées pour la machine hôte. Toutefois, il faut être averti du fait que l'interface avec ces méthodes natives (le JNI) ne fait pas partie de la spécification obligatoire et donc que les programmes Java qui font appel à des méthode natives peuvent perdre le bénéfice de la portabilité du code. Dans ce cas, le concepteur doit arbitrer entre l'avantage de la vitesse d'exécution des méthodes natives et ceux de la portabilité, à moins de faire le portage de la bibliothèque native sur tous les systèmes cibles.

15.3 L'architecture du chargeur de classes de la JVM

La JVM possède un chargeur de classes dont l'architecture est flexible, ce qui permet à un programme de charger les classes selon une méthode choisie par lui. Une application Java peut soit utiliser le chargeur de classes primordial, soit utiliser des "objets chargeurs de classes". Le chargeur primordial fait partie de toute implémentation d'une JVM. C'est lui qui charge les classes "connues" ("trusted") et les classes de l'API Java, ces dernières généralement depuis le disque local.

À l'exécution, une application Java peut installer des objets chargeurs de classes qui vont charger des classes à leur façon, par exemple en les téléchargeant depuis une autre machine sur le réseau ("downloading").

La JVM considère comme connues (trusted) toutes les classes chargées à travers le chargeur primordial et comme suspectes, toutes les classes chargées par un objet chargeur. Les objets chargeurs sont écrits en Java, compilés en bytecode, chargés dans la machine virtuelle et instanciés comme n'importe quel autre objet. Ils sont juste une partie du code exécutable d'une application Java. Les objets chargeurs permettent d'étendre une application à l'exécution. Pour toute classe chargée, la JMV garde trace du chargeur par qui elle a été incluse dans l'application.

Quand une classe fait référence à une autre classe, la machine virtuelle appelle la classe référencée à travers le même chargeur que celui qui a chargé la classe appelante. Ceci, pour les invocations de classes depuis une autre classe. Une requête de chargement explicite, sans invocation de la classe, ne suit évidemment pas cette règle, sinon toutes les classes seraient "filles" du chargeur primordial.

Une conséquence de cette politique de chargement est que, par défaut, les classes ne "voient" que les autres classes chargées par le même chargeur de classes. Par ce biais, le concepteur d'application peut créer plusieurs espaces de nommage à l'intérieur d'une même application Java.

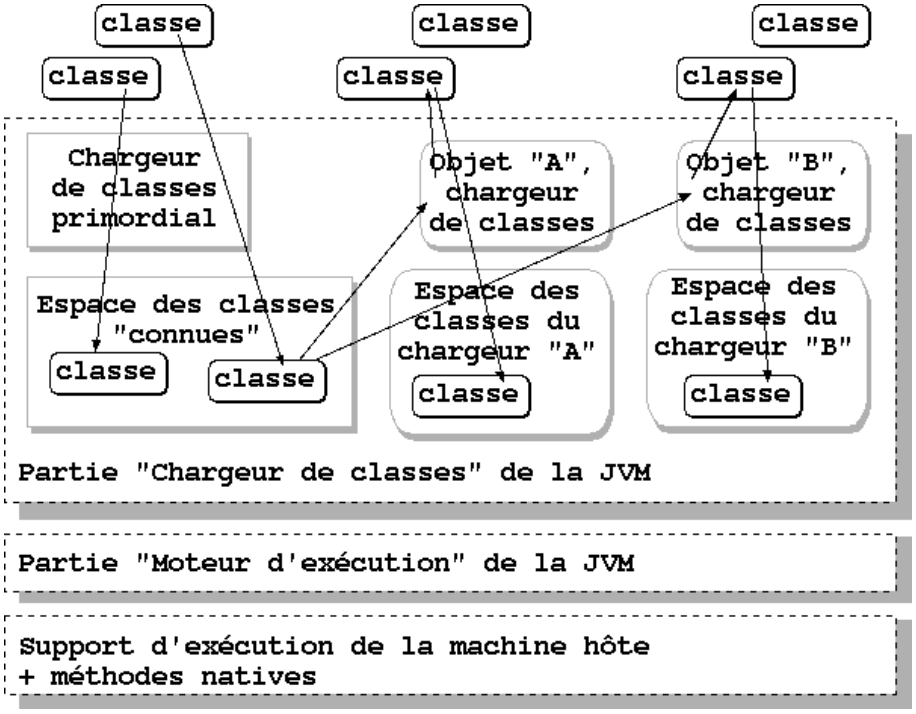


FIG. 158: Java : le chargeur de classes

Les classes chargées par des chargeurs de classes différents n'ont pas accès l'une à l'autre, sauf si l'application le permet explicitement. On peut donc dans une application faire une sé-

paration entre des classes chargées depuis des sources différentes, par exemple, grâce à ce mécanisme, et donc contrôler les interactions entre classes de provenances différentes. Ceci pour, par exemple, protéger des classes "amies" de classes "hostiles".

Exemple : dans un navigateur, on lance une application Java qui installe un chargeur de classes (un "applet class loader") qui "sait" comment chercher les fichiers de classe sur un serveur HTTP. L'application Java lançée par le navigateur va généralement créer un objet chargeur pour chaque source différente sur le réseau depuis laquelle elle va charger des classes. Ainsi les classes de deux sources différentes ne peuvent pas interférer, pusiqu'elles sont dans des espaces de nommage différents.

Le fichier de classes Java

Les fichiers de classes Java contiennent la forme binaire des programmes Java (le bytecode) produite par le compilateur Java et interprété par la machine virtuelle. Ce binaire est donc le même pour toutes les plateformes, puisque ce sont les machines virtuelles qui font l'interface entre le bytecode Java et la plateforme. Les machines virtuelles sont écrites une fois pour chaque type de plateforme, ensuite tous les bytecodes sont exécutables sur toutes les machines virtuelles. Le programmeur d'application Java n'a donc pas à se soucier de produire un exécutable pour chaque type de machine ou de système d'exploitation.

Le bytecode masque deux éléments dépendants de la plateforme matérielle : les instructions machines (assembleur) du processeur et l'ordre de rangement des octets dans les mots mémoire (big-endian comme sur le Power-PC ou little-endian comme sur la famille Intel X86). Dans un fichier de classe java, l'ordre est big-endian pour tous.

De plus, le bytecode a été conçu pour générer des fichiers compacts, facilement téléchargeables à travers un réseau.

Dernier avantage : les classes java étant liées dynamiquement (à l'exécution), ne sont téléchargées que celles qui seront effectivement utilisée lors d'une exécution particulière. Ceci réduit aussi la quantité de code binaire à charger à travers le réseau.

L'API Java

L'API Java est un ensemble de bibliothèques dynamiques (appelées à l'exécution) qui donnent accès de façon standard aux ressources du système hôte de la machine virtuelle. Quand on écrit un programme Java, on suppose que toutes les classes définies dans l'API Java sont disponibles dans la machine virtuelle sur laquelle le programme va s'exécuter. Cette supposition est raisonnable puisque l'API fait partie des spécifications requises de toute implantation d'une machine virtuelle.

C'est parceque les classes constituant les API Javi sont implantées pour chaque type de machine hôte, que les programmes Java qui n'utilisent que ces classes sont indépendants des machines hôtes.

De plus, avant de faire toute action potentiellement dangereuse, les classes de l'API Java demandent une permission au gestionnaire de sécurité ("security manager"). Celui-ci peut, par exemple, interdire l'accès au disque local.

15.4 Le langage Java

Le langage a parfois été défini sous une forme plaisant par C++– : il contient des spécificités de type "orienté-objet" comme le C++, mais certaines possibilités "dangeureuses" du C++ ont

été retirées et certaines modifiées, dans le but de rendre les programmes plus sûrs. C'est un langage complet et moderne (en 1997) qui inclut les fonctionnalités suivantes :

- orienté objet,
- multithread,
- fortement et statiquement typé,
- à liaison dynamique,
- gérant la mémoire,
- simplifié,
- sécurisé.

orienté objet : il respecte les principes d'encapsulation, d'abstraction et d'héritage ; il possède une notion de classe obligant le programmeur à structurer ses données. Toutefois, "l'objet" est une mode qui n'a pas toujours apporté les gains de productivité attendus parce qu'on a fortement sous estimé le changement de mentalité nécessaire pour concevoir et programmer avec les concepts objets.

Mais Java reste utile car ses autres caractéristiques, associées à la plateforme Java, compensent largement ce barrage initial :

- les ateliers de développement masquent en grande partie la complexité,
- il permet un bonne maintenance,
- il prend en charge les applications distribuées en réseau.

L'héritage entre les classes est "simple" : une classe ne peut hériter que **d'une** autre classe. L'héritage de propriétés mulitples, qui est parfois utile, peut se faire à travers l'héritage des **interfaces**, qui lui, peut être multiple.

Java résoud les problèmes de collision de noms des classes et méthodes en introduisant le concept de "package" (inventé par Ada), qui est un espace de nomage.

multithread : Java permet de créer très facilement des flots d'exécution concurrents ("threads"). Ceci permet d'écrire des programmes plus réactifs, plus efficaces du point de vue de la charge du système hôte, et qui utilisent les machines multiprocesseurs.

fortement et statiquement typé : Java effectue un contrôle statique (à la compilation) strict des types.

à liaison dynamique : En java, il n'y a pas d'édition de liens. L'existance des bibliothèques et des classes appelées dans ces bibliothèques se fait au moment de la compilation. Leur code est chargé dans la JVM au moment de l'exécution. Le compilateur et la JVM retrouvent les classes compilées en bibliothèque car celles-ci sont structurées en paquetages rangés dans une arborescence précise. Il est donc indispensable de découper les projets en hiérarchies de classes, structurées en paquetages.

gérant la mémoire : Java gère la mémoire (code et données) par un ramasse miettes (garbagge collector) qui libère la mémoire de tous les objets inutilisés. Le programmeur n'a donc plus besoin d'allouer et de relacher des zones de mémoire (malloc, dealloc), source très fréquente de bogues dans les programmes C et C++ ("memory leaks") ("delete" oubliés ou mal placés). De plus le langage Java ne possède pas de pointeurs. Ceci supprime une source de très nombreuses erreurs (en C et C++).

simplifié : la syntaxe est simple, les bizarreries d'autres langages ayant été supprimées.

sécurisé : de nombreuses vérifications sont faites par l'environnement avant d'exécuter du code Java : un champ "privé" d'une classe restera innacessible (pas de pointeurs), l'accès aux ressources locales est contrôlé, le débordement de la pile est contrôlé, etc ...

Java n'inclut aucune technique logicielle nouvelle ou expérimentale, mais plutôt fait une combinaison nouvelle de techniques connues, essayées et démontrées dans d'autres langages. Ceci en fait un outil puissant et général que l'on peut appliquer dans de nombreux cas indépendamment du fait que l'on travaille ou non sur une application de réseau.

La gestion des exceptions est incluse dans le langage : la documentation d'une méthode doit indiquer les exceptions qu'ell déclenche ("throw").

Certains ont pu dire que Java avait gardé les bons côtés de C++ et enlevé les éléments potentiellement dangereux.

15.5 Java ou C++ comme langage général ?

On estime que par rapport à C++, Java apporte un gain de productivité (ceci au prix d'une vitesse d'exécution plus faible). Cette productivite est essentiellement due aux restrictions que Java impose dans la manipulation directe de la mémoire (la très grosse majorité des bogues dans les programmes C ou C++ sont dus à des accès à des zones mémoires non-initialisées ou à des zones mémoires allouées ou désallouées à des moments incorrects : C++ est quasiment un "pousse au crime" à ce sujet).

Java possède un opérateur **new**, comme C++, pour allouer de la mémoire sur le tas ("heap") pour un nouvel objet. Mais, à la différence de C++, Java ne possède pas d'opérateur **delete** que l'on doit utiliser en C++ pour libérer la mémoire occupée par un objet dont on a plus besoin : il suffit de cesser de référencer l'objet et le ramasse miette ("garbagge collector") va, un certain temps plus tard récupérer cette zone mémoire. En C++, pour des objets complexes, il est souvent difficile de savoir quand un objet n'est plus référencé. On peut ainsi oublier de faire le delete, conduisant à un programme "mange mémoire" ("memory leak" : la mémoire restante devient faible), soit mettre le delete trop tôt, soit le mettre deux fois, conduisant à une mémoire corrompue qui provoque un "crash" du programme d'une façon qu'il est difficile de relier à la véritable cause de l'erreur.

Une autre particularité de Java protège la mémoire : Java ne permet pas de faire des conversions arbitraires de pointers ("cast"). Java impose un strict respect des types. Si on a une référence sur un objet de type "A", on ne peut manipuler l'objet que comme un objet de type "A". On ne pourra convertir la référence sur "A" en une référence sur un objet de type "B" que si "A" et "B" sont dérivées l'une de l'autre et que l'objet est réellement de la classe dérivée (donc des deux types).

Une troisième façon pour Java de protéger la mémoire est par la vérification des bornes des tableaux ("array bound checking"). En C, C++ ou Fortran, on peut déclarer un tableau TAB[100] et accéder à TAB[200] ou à TAB[-10]. Bien sûr ces accès en dehors du tableau vont se faire sur un autre objet rangé en mémoire. Ce type d'accès "hors limite" est une cause fréquente de bogues, lorsque les indices de tableaux sont le résultats de calculs, où d'indirections complexes. Lors d'un accès de ce type, Java va lever une exception.

Une dernière particularté de Java supprime aussi une cause de nombreuses erreurs : avant tout accès à un objet, Java va vérifier que la référence à l'objet n'est pas **null**, et lever une exception si c'est le cas. En C++, utiliser un pointeur nul entraine le "crash" du programme.

Inconvénients

Bien sûr, toutes ces fonctions de Java – test des bornes des tableaux, vérification de la cohérence des conversions de type, test des références null, et ramasse miettes – font qu'un programme Java sera intrinsèquement plus lent que son équivalent en C++.

Mais le principal effet sur la performance vient du caractère interprété du bytecode Java : en s'exécutant sur une machine virtuelle qui interprète le bytecode, un programme Java va tourner 10 à 30 fois moins vite que son équivalent C++ compilé en instructions natives de la machine hôte. Ce compromis sur la performance a principalement été accepté en échange de l'indépendance de la plateforme.

Heureusement des techniques intermédiaires peuvent limiter cette pénalité : un compilateur "juste à temps" procure un gain d'un facteur 7 à 10 par rapport à l'interprétation pure. Ceci permet de faire des applications qui s'exécutent **suffisamment** rapidement. De plus, la vitesse brute d'exécution dans l'unité centrale n'est **plus** le facteur le plus important dans la performance **perçue** par l'utilisateur : dans de nombreux cas, les programmes passent la plupart de leur temps à attendre des données en provenance de disques ou de réseaux.

Certains programmes ont toutefois réellement **besoin** de s'exécuter très vite. Une façon de contourner cette difficulté de la vitesse d'exécution, peut être d'identifier les paries critiques du code (là où on doit aller vite) et de les compiler en tant que méthodes natives et de les stocker dans les bibliothèques dynamiques de la JVM de la machine hôte. Il faudra bien sûr payer la performance obtenue par une compilation pour chaque type de système et une installation sur chaque machine cliente.

Une alternative "finale" est de compiler l'ensemble des classes utilisées par l'application en un fichier exécutable natif et monolithique qui sera utilisé comme un exécutable C ou C++. On abandonne l'indépendance vis-à-vis de la plateforme, mais on conserve toutefois les avantages en gain de productivité apportés par Java (tout le développement et la mise au point se fait en mode interprété).

Les compromis architecturaux de Java

Les concepteurs du langage ont fait les choix de compromis qui ont le plus de sens dans un contexte d'applications réparties sur un réseau. Le principal compromis est celui de la vitesse. Il est compensé dans de nombreux cas pratiques par le fait que la vitesse d'exécution du code n'est pas l'élément majeur dans le confort d'utilisation. Il est compensé aussi par les gains obtenus en matière de robustesse. Les programmes Java s'exécutent moins vite pour de nombreuses raisons, qui toutes apportent par ailleurs un bénéfice :

- l'interprétation du bytecode est 10 à 30 fois plus lent qu'un équivalent compilé en code natif,
- la compilation "juste à temps" apporte un gain de 7 à 10 sur l'interprétation, mais il reste une pénalité,
- les programmes Java sont liés dynamiquement (recherche à l'exécution d'une méthode invoquée),
- la machine virtuelle peut avoir à attendre qu'un fichier de classe lui parvienne à travers le réseau,
- les bornes des tableaux sont testées à chaque accès,
- tous les objets sont créés sur le tas (les objets éphémères ne sont pas créés sur la pile comme en C),
- toutes les utilisations d'une référence vers un objet sont vérifiées ne pas être nulles,
- le ramasse miette est connu comme moins efficace qu'une gestion mémoire directe,
- les types primitifs sont les mêmes sur toutes les plateformes au lieu d'être adaptés au plus efficace,
- les chaines de caractères sont toujours en Unicode qui est plus long à traiter que l'ASCII.

Un autre compromis est celui de la perte de contrôle de la gestion de la memoire. Le ramasse miette fait des programmes plus robustes et plus faciles à concevoir, mais ajoute un degré d'incertitude sur la performance à l'exécution. On ne peut pas décider quand le ramasse miette va

décider de récupérer les zones mémoires orphelines, ni combien de temps il va mettre. Ceci limite l'usage de Java dans les applications de type temps réel où il est important d'avoir un temps d'exécution prédictible.

Encore un autre compromis est celui qui provient de la volonté d'indépendance de la plateforme. Celle-ci est obtenue par les bibliothèques qui implantent les API. Mais ces API sont un intermédiaire vers des fonctionalités du système d'exploitation de la machine hôte. Or, tous les systèmes d'exploitation n'ont pas la même liste de fonctionnalité. Si l'une d'entre elles est présente sur certains systèmes et pas sur d'autres, faut-il écarter cette fonction de la liste des fonctions disponibles dans l'API, donc utilisable en Java ? Ou faut-il la simuler (plus ou moins bien) dans les implantations de l'API sur les systèmes qui n'ont pas cette fonction ? Si la deuxième solution est choisie, et que la façon dont la fonction manquante est simulée ou implantée, ne plait pas au programmeur Java, celui-ci doit-il l'écarter de ses programmes, se créant ainsi un sous-ensemble de l'API, "plus petit dénominateur commun" des fonctions disponibles ? Ceci est plus particulièrement vrai lors de l'écriture d'interfaces utilisateur faisant appel à l'API "AWT" (Abstract Window Toolkit).

Enfin un dernier compromis a été fait pour obtenir un systéme de liens dynamiques. Dans un programme C++ un appel de fonction est compilé en une adresse directement utilisable à l'exécution. En Java quand une classe référence une autre classe, cette référence est implantée par le stockage de la chaine de caractères du nom de la classe "appelée". Un fichier de classe Java contient des références symboliques (symbôles stockés sous forme de chaines de caractères) aux classes appelées, aux méthodes, aux arguments et à leurs types, aussi bien pour les classes "externes" au fichier, que pour les classes et méthodes internes, définies dans le fichier.

Tendances futures

La pénalité en performance des programmes Java actuels ne doit pas être considérée comme définitive : des techniques combinées de compilation "juste à temps" et de mesures de profils ("profiling") juste à temps pourraient permettre de s'approcher d'un rapport de pénalité de 2 ou 3 vis-à-vis d'un équivalent natif en C++. À ce niveau la pénalité sera indiscernable à l'utilisateur final pour de nombreuses applications.

De même dans le domaine des interfaces utilisateur, on commence à voir apparaitre des composants Java, fondés sur les API standard de Java (l'AWT) qui implémentent des fonctions d'interface non présentes dans l'AWT et qui les rendent ainsi disponibles sur toutes plateformes. On pourrait voir ainsi apparaitre une interface avec "l'aspect et le comportement" ("look and feel") de la plateforme Java, comme on a aujourd'hui "l'aspect Mac", "l'aspect Motif" ou "l'aspect Windows".

15.6 Indépendance vis-à-vis des plateformes

La raison clé du succès de Java dans un environnement distribué en réseau est le fait que le système Java permet de créer des binaires exécutables qui vont pouvoir tourner sur de multiples plateformes, tout en restant inchangés (compilés une seule fois).

Le système Java c'est l'ensemble du langage Java, de la machine virtuelle java (la JVM), de l'API Java et des fichiers de classes Java. L'architecture du "Système Java" permet d'écrire des programmes indépendants des plateformes, mais permet aussi d'écrire des programmes dépendants d'une plateforme. L'indépendance est une option.

Tous les composants du système Java (le langage, l'API, la JVM et les fichiers de classe) jouent un rôle dans l'obtention de binaires indépendants des plateformes. Mais le principal support provient de la plateforme Java : l'ensemble de la JVM et des API Java.

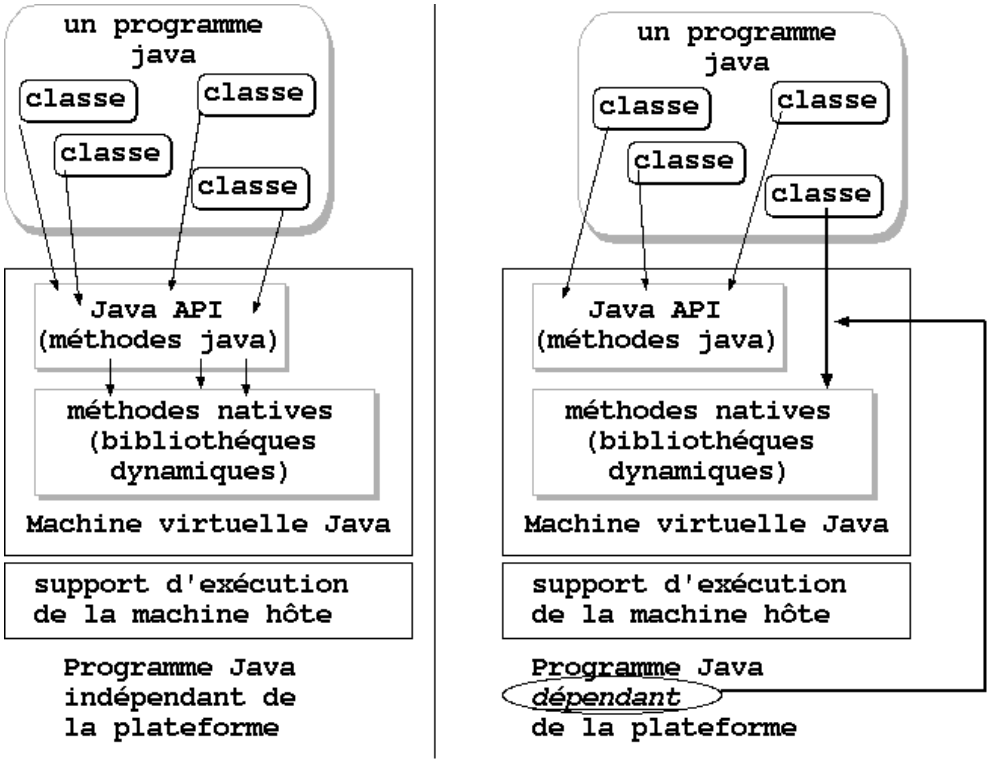


FIG. 159: Java : indépendance de la plateforme

Un programme Java qui n'interagit que avec les fonctions définies dans l'API Java pourra s'exécuter sur toute plateforme possédant une machine virtuelle java.

Le langage intervient dans l'indépendance en définissant l'étendue et le comportement de ses types primitifs. Alors que en C et C++ l'étendue ("range") d'un int est déterminée par sa taille, et sa taille est dépendante de la plateforme (4 octets sur certaines machines, 8 sur d'autres). Un int java est toujours un nombre signé sur 32 bits en complément à 2.

Les fichiers de classes définissent un format binaire spécifique de la machine virtuelle java, et non pas d'une plateforme particulière.

De plus, la plateforme java est extensible ("scalable") : elle peut être implantée aussi bien sur de grosse machines que sur des microprocesseurs embarqués, comportant très peu de ressources.

Les versions de la plateforme Java

L'ensemble des bibliothèques de base de l'édition standard de la plateforme java est appelé "Java Core API" et sa disponibilité est garantie sur toute plateforme java de l'édition standard. Mais il existe des extensions et des restrictions.

extensions : des extensions appelées "Java Standard Extensions API" ont été définies pour la téléphonie, le commerce, l'audio, la vidéo, la 3D, ...

restrictions : il existe 3 éditions de la plateforme java, qui comportent de moins en moins de fonctions prédéfinies dans les API. Elles sont destinées à des environnements embarqués disposant de très peu de ressources :

- l'édition embarquée : "Java Embedded Platform",
- l'édition personnelles : "Java personal Platform",
- l'édition carte à puce : "Java Card Platform".

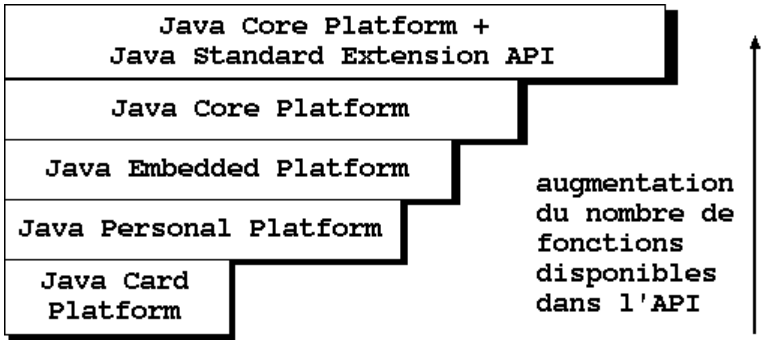


FIG. 160: Java : différentes éditions

Un autre facteur de complexité dans le support de l'indépendance des programmes est l'évolution des versions des APIs. Certaines fonctions peuvent être ajoutées, et (plus rarement) certaines devenir obsolètes. Un programme utilisant des fonctions récentes ne fonctionnera pas sur une machine virtuelle ancienne version. Les nouvelles versions de la machine virtuelles vont mettre un certain temps à se diffuser sur tous les ordinateurs connectés à un réseau.

L'utilisation des méthodes natives (ne faisant pas partie de l'API standard) est un autre facteur empêchant la portabilité. Si ces méthodes sont nécessaires et que l'on désire rester portable, il faut "porter" ces méthodes natives sur toutes les plateformes cibles.

Certains vendeurs fournissent également avec leur JVM des bibliothèques Java non standard qui viennent s'ajouter aux APIs standard de la JVM. Si ces méthodes Java sont implantées en utilisant uniquement des méthodes de l'API standard, elles pourront être téléchargées avec le programme pour permettre l'exécution de celui-ci. Il n'y a pas de perte de généralité, mais une pénalité sur le temps de chargement. Par exemple un programme Java qui utilise les AFC (Application Foundation Class) de Microsoft, ne pourra tourner sur Unix que si on charge avec lui les binaires des classes AFC utilisées par le programme.

La machine virtuelle étant implémentée différemment par différents vendeurs sur différentes machines, il est fortement conseillé, pour qu'un programme soit portable d'éviter les deux ecceuils suivants :

- ne pas dépendre de la vitesse de traitement pour assurer la correction du programme, (en particulier du moment où seront exécutés les routines de finalisation des objets non référençés, libérés par le ramasse miettes),
- ne pas dépendre de la priorité des threads pour assurer la correction du programme.

Ceci à cause de la liberté laissée aux implémenteurs dans la spécification du ramasse miettes et des threads.

Une partie très difficile d'une application Java qui doit fonctionner sur plusieurs plateformes est l'interface utilisateur. L'AWT ne contient qu'un ensemble de base de composants d'interface. Des bibliothèques telles que Microsoft AFC, Netscape IFC ou Sun JFC fournissent des composants plus élaborés. Mais créer une interface avec laquelle les utilisateurs des différentes plateformes se sentent tous à l'aise n'est pas une tâche aisée.

En conclusion, les programmes Java ne sont pas en général indépendants des plateformes à un degré tel que l'on puisse se contenter de mettre un programme au point sur une plateforme : il convient de le tester sur toutes les plateformes sur lesquelles on prétend que le programme fonctionne.

On peut fournir une liste d'étapes à suivre pour réaliser sans déboires un programme Java indépendant des plateformes :

1. choisir une liste de plateformes, sur lesquelles on va indiquer que le programme tourne,
2. choisir une version de la plateforme Java qui soit suffisamment diffussée sur ces systèmes,

3. sur chaque système cible, choisir une implémentation de la plateforme Java, sur laquelle on annoncera que le programme fonctionne ;
4. écrire le programme de telle sorte qu'il n'accède au système hôte que à travers l'API standard de Java,
5. écrire le programme de telle sorte qu'il ne dépende pas du moment d'exécution des routines de finalisation ni de la façon dont les priorités des threads sont gérées,
6. concevoir une interface utilisateur qui fonctionne correctement sur toutes les plateformes,
7. tester le programme sur toutes les plateformes cibles.

La politique de l'indépendance des plateformes

En plus de l'API standard, les vendeurs de JVM peuvent ajouter des extensions. Ces extensions peuvent être proposées pour "faciliter" la tâche du programmeur, mais aussi pour "vérouiller" celui-ci sur la plateforme du vendeur, et l'inciter à écrire des programmes qui ne fonctionnent que sur la plateforme du vendeur. Bien entendu, un vendeur concurrent va faire la même chose. Le développeur a alors le choix entre profiter des avantages des extensions ou faire l'effort de développer de façon indépendante et ne pas être lié à une plateforme unique.

15.7 La sécurité dans le système java

L'utilisation intensive des réseaux implique d'apporter une attention pqrticulière à la sécurité. Java ayant été conçu en vue d'une utilisation intensive sur des réseaux de machines diverses, a inclus dés sa conception un modèle de sécurité assez complet.

Le système Java inclut un modèle de sécurité qui a pour rôle premier de protéger l'utilisateur final de programmes hostiles qui seraient téléchargés à travers le réseau depuis une source non sûre, par exemple en chargeant une page html dans un navigateur. Pour cela Java fournit une "boîte à sable" ("sandbox") dans laquelle un programme Java non-sûr peut faire ce qu'il veut, mais dont il ne peut pas franchir les frontières. Cette "boîte à sable" interdit en particulier aux applets de :

- lire ou écrire sur le disque local,
- établir une connexion réseau vers toute machine sauf celle depuis laquelle elle vient ;
- créer un nouveau process,
- charger une nouvelle bibliohèque dynamique,
- appeler directement une méthode native.

Naturellement, si vous êtes raisonnablement paranoïaque, vous avez besoin d'être convaincu que la "boîte à sable" ("Sandbox") n'a pas de point de faiblesse avant de croire qu'elle peut protéger votre machine de tout programme hostile. Pour assurer cette absence de faiblesse, le modèle de sécurité Java implique tous les aspects de l'architecture.

Les composants responsables du bon fonctionnement de la "boîte à sable" sont :

- l'architecture du chargeur de classes,
- le vérificateur de fichiers de classes,
- les fonctions de sécurité incluses dans la machine virtuelle et dans le langage,
- le gestionnaire de sécurité et l'API Java.

L'une des forces du modèle de sécurité Java est que deux de ses composants, le chargeur de classes et le gestionnaire de sécurite, sont paramétrables. Ceci permet de créer une politque de sécurité adaptée pour une application particulière.

L'architecture du chargeur de classes

Le chargeur de classes est le première ligne de défense. Il garde la frontière des bibliothèques sûres en empêchant les classes non-sûres ("untrusted") de prétendre à être considérées comme "sûres". Il réalise ceci en fournissant des espaces de nommage différents pour des classes chrgées par différents chargeurs de classes. Un espace de noms est un ensemble de noms uniques pour des classes chargées. Il est maintenu par la machine virtuelle.

Dans la machine virtuelle, des classes chargées dans des espaces de noms différents ne peuvent même pas détecter leur présence mutuelle, sauf si on fournit explicitement un mécanisme qui leur permet d'interagir.

Quand on écrit un chargeur de classes, on crée un environnement dans lequel s'exécute le code chargé. Pour éviter que cet environnement ne comporte des trous de sécurité, il faut respecter un certain nombre de règles quand on écrit le chargeur. En général on désire que le chargeur protège les limites des classes reconnues sûres ("trusted"), par exemple l'API Java ou un package local.

Java permet aux classes d'un même paquetage ("package") de s'accorder l'une l'autre des droits d'accès particuliers qui ne sont pas accordés aux classes extérieures au paquetage. Or, rappelons nous que les classes se désignent (se référencent) par leur nom. Ainsi si une classe, chargée par le chargeur "mon_chargeur" reçoit une requête pour charger depuis le paquetage "inconnu" une classe qui se fait appeler "java.lang.virus", cette classe "virus" prétend par son nom faire partie de l'API Java. Si elle est chargée, elle se verra accorder les mêmes droits que ceux de l'API, ce qui n'est certainement pas ce que l'on désire. Le chargeur "mon_chargeur" doit donc être écrit pour empêcher cela.

Pour cela, le chargeur doir refuser de charger les classes qui prétendent faire partie d'un paquetage "reconnu sûr", mais qui ne sont pas stockées dans le container local lui-même reconnu sûr ("trusted local repository"). Le chargeur "mon_chargeur" doit alors utiliser la méthode suivante :

1. le chargeur personnel passe d'abord toutes les requêtes de chargement classes au chargeur primordial,
2. le chargeur primordial cherche dans l'entrepôt local "reconnu sûr" la classe demandée ; si elle est trouvée, c'est elle qui est fournie (ainsi un paquetage malicieux ne peut pas remplacer une classe d'une API connue). Si la classe n'est pas trouvée, le chargeur primordial renvoie un statut d'échec au chargeur personnel ;
3. le chargeur personnel doit alors vérifier si la classe demandée ne prétends pas faire partie d'un paquetage reconnu sûr (si elle le prétend et qu'elle en fait vraiment partie, alors le chargeur primortdial l'aurait trouvée) ; Si la classe le prétend, alors le chargeur personnel doit lever une exception de sécurité ("throw security exception") ;

Autre exemple : si vous ne voulez pas qu'une classe chargée par le chargeur personnel puisse charger une classe d'un paquetage local "mon_prog", le chargeur personnel doit vérifier les requêtes et lever une exception si l'une d'elle est adressée à une classe de "mon_prog" par une classe chargée par "mon_chargeur".

Comme c'est par le nom complet de la classe (tel que "java.lang.virus") que le chargeur peut savoir si la classe demandée est ou prétend être une classe d'un paquetage protégé, le chargeur devra être pourvu d'une liste des noms des paquetages protégés, restreints, interdits. Le chargeur devra alors :

1. si la classe demandée est dans un paquetage "interdit", lever une exception de sécurité, sinon aller en 2,

2. passer la requête au chargeur primordial ; si celui-ci fournit la classe, on continue, sinon on va en 3,
3. s'il existedes paquetages sûrs auxquels le chargeur n'est pas autorisé à ajouter des classes, et que la classe demandée fait partie d'un tel paquetage, il doit lever une exception, sinon aller en 4,
4. le chargeur personnel charge la classe selon sa méthode (par exemple téléchargement à travers un réseau) ; si c'est un succès, il retourne la classe, sinon il retourne le statut "no class definition found".

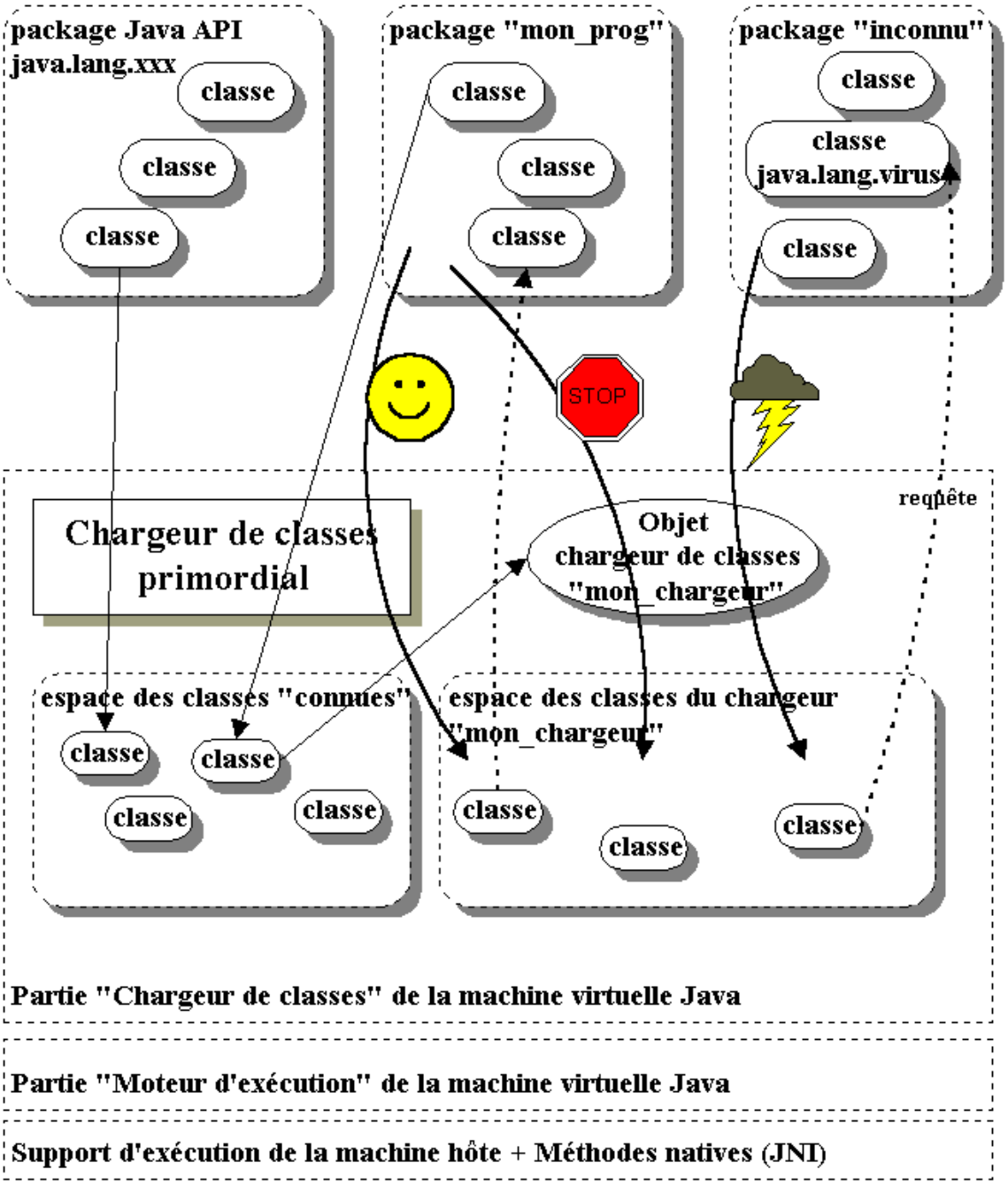


FIG. 161: Sécurité dans le chargeur de classes

Le vérificateur de fichiers de classes

Le vérificateur de fichiers de classes s'assure que les fichiers binaires ".class" contenant le bytecode exécutable ont une structure correcte. En effet, quand une classe est téléchargée depuis un réseau, on a aucun moyen de savoir si ce fichier a été créé par un compilateur java "normal" ou bien "forgé" par un pirate dans le but de compromettre le bon fonctionnement de la machine virtuelle.

Phase 1 : vérifications internes

Durant cette phase, on vérifie tout ce qu'il est possible de vérifier en regardant uniquement le contenu du fichier de classe. La première vérification concerne la robustesse : si une classe erronée contient une instruction de branchement qui saute en-dehors du code de la classe, son exécution pourrait conduire à un crash de la machine virtuelle. La JVM vérifie donc que toutes les instructions de branchement pointent sur une instruction valide à l'intérieur de la classe.

On va d'abord vérifier la structure : la classe doit commencer par les 4 octets du "nombre magique" : 0xCAFEBABE. Puis on teste si la classe n'est pas tronquée et si elle ne comporte pas d'octets supplémentaires. Ceci est possible car chaque composant d'un fichier de classe doit indiquer son type et sa taille. Ensuite on vérifie que chaque composant est une instance bien formée de son type. Par exemple, on vérifie que chaque descripteur de méthode est une chaine de caractère bien formée conforme à la grammaire de description des paramètres d'une méthode.

Le vérificateur s'assure aussi que la classe adhère à toutes les contraintes qui lui sont imosées par la spécification du langage Java. Par exemple, toute classe sauf la classe object doit avoir une super-classe.

Ensuite la JVM opère une analyse de flux de la chaine de bytecodes constituant les méthodes de la classe. Le flux de bytcodes est constitué d'une série de code opérations ("opcodes") de 1 octet, chacun suivi de 0, 1 ou plusieurs opérandes. L'activité consistant à exécuter les bytecodes l'un aprés l'autre constitue un flot d'exécution ("thread") à l'intérieur de la JVM. Chaque flot possède sa propre pile java, constituée de formes ("frame"). Chaque invocation d'une méthode possède sa propre forme qui est une section de mémoire pour les variables locales et les résultats intermédiaires. La partie de la forme où sont rangés les résultats intermédiaires est appelée "pile d'opérandes" ("operand stack"). Un bytecode peut utiliser des données stockées dans ses opérandes, dans les variables locales de la forme et dans la pile d'opérandes.

Le vérificateur de bytecode va faire un grand nombre de tests : que aucune variable locale n'est accédée avant d'être définie, que les champs d'une classe sont remplis avec des valeurs du bin type, que les méthodes sont invoquées avec le bon nombre d'arguments et que ceux-ci sont du bon type, ...

Phase 2 : vérification des références symboliques

La phase 1 est exécutée une fois, dés le chargement de la classe dans la JVM. La phase 2 est reportée juste avant l'exécution. Durant la phase 2, la JVM suit toutes les références contenues dans la classe à vérifier jusqu'aux classes référencées, pour être sur qu'elles sont correctes. Cette phase 2 est en fait une partie du processus de liaison dynamique.

La liaison dynamique est le processus qui va résoudre les références symbolique (une classe définie par son nom) en des références directes : la JVM va (1) trouver les classes référencées (et les charger si nécessaire), (2) remplaçer la référence symbolique par une référence directe (un pointeur). La JVM garde une table de correspondance des références symboliques et directes de façon à ne pas faire deux fois le travail de recherche si une classe est utilisée plusieurs

fois. Quand c'est un champ ou une méthode d'une classe qui est référençée, la JVM vérifie l'existance de ce champ ou de cette méthode avant de remplacer le symbôle par un lien direct.

Références symboliques et compatibité binaire

- Quand une classe référence un champ ou une méthode d'une classe en bibliothèque, il peut arriver que l'on doive modifier la classe référencée. Bien sûr, le compilateur n'a pas de moyen de savoir quelles classes possèdent une référence vers la classe qu'il est en train de recompiler ; il ne peut donc pas faire une recompilation automatique de toutes les classes utilisatrice de la classe de bibliothèque qui vient d'être modifiée. On distingue alors deux types de modifications : celles qui sont compatibles binaires, et celles qui ne le sont pas.
- Par exemple si une classe "a" contient la méthode "m1", référencée par une classe de "b", et que l'on change le nom de "m1" en "m2", l'invocation "a.m1()" va échouer. Cette modification n'est pas compatible binaire. Par contre, si on ajoute la méthode "m2" à "a" et que l'on modifie le contenu de la méthode "m1" en lui faisant appeler "m2", mais sans changer ni son nom ni sa liste d'arguments, l'invocation "a.m1()" continuera à fonctionner : c'est une modification compatible binaire. Dans ce dernier cas, il n'est pas nécessaire de recompiler toutes les classes utilisatrices de la bibliothèque pour qu'elles continuent à fonctionner correctement.
- Le langage Java spécifie les modifications que l'on peut faire à une classe, tout en lui conservant sa compatibilité binaire.

Les fonctions de sécurité incluses dans la JVM

La machine virtuelle n'accorde aux programmes que des moyens sûrs et structurés d'accéder à la mémoire :

- conversions de type vérifiées et sûres,
- accès mémoires structurés (pas d'arithmétique des pointeurs),
- ramasse miette automatique (pas de "de-alloc" intenpestif),
- test des bornes des tableaux (pas d'accès à Tab[-10], ...),
- test des références vis-à-vis de null (pas d'accès à travers un pointeur non initialisé).

L'interdiction de tout accès mémoire non structuré empêche un programme malicieux d'aller lire ou écrire n'importe où dans la mémoire gérée par la machine virtuelle. Ceci est non seulement imposé par les règles du langage, mais est aussi inlclut dans la définition des codes opératoires des bytecodes. Ainsi même en écrivant "à la main" une séquence de bytecodes, il est peu probable de pourvoir accéder en dehors des zones mémoires "officiellementé déclarées par la classe. Ceci constitue une solide barière contre la manipulation imprévue de la mémoire de la JVM.

Les barières de sécurité de la machine virtuelle peuvent cependant être pénétrées par l'utilisation des méthodes natives. Celles-ci sont en effet écrites en C ou C++ avec tous les défauts de contrôle d'accès à la mémoire de ces langages. Pour cette raison, le gestionnaire de sécurité dispose d'une méthode dont le rôle est de dire si un programme sera autorisé à charger une bibliothèque dynamique, ce qui est nécessaire avant d'invoquer une méthode native. Les applers, par exemple, ne sont pas autorisés à charger une bibliothèque dynamique, et donc ne peuvent pas installer leur propres méthodes natives. Ils peuvent cependant appeler une méthode native de l'API Java. Si cette méthode contient un trou (sous forme d'accès non prévu à la mémoire), la sécurité de la JVM est menaçée.

Le mécanisme final de sécurité est le traitement des exceptions, particulièrement des exceptions de sécurité : quand un flot lève une telle exception, c'est ce flot qui doit être arrêté, mais

pas la machine virtuelle toute entière.

Le gestionnaire de sécurité et l'API Java

Les chargeurs de classe permettent de protéger entre elles des classes chargées dans la machine virtuelle. Mais pour protéger les éléments extérieurs à la JVM on a besoin du gestionnaire de sécurité : celui-ci va définir (de façon paramétrable) les limites extérieures du "bac à sable" dans lequel évoluent les programmes Java.

Pour chaque action potentiellement non-sûre ("unsafe"), l'API Java interroge le gestionnaire de sécurité. Celui-ci posséde, pour chaque action de ce type, une méthode qui indique si l'action est permise depuis le "bac à sable". Le nom de chacune de ces méthodes commence par "check" : par exemple checkRead() indique si le flot appelant est autorisé à lire un fichier, etc ... Avanr d'essayer une action "sensible" l'API Java appelle la méthode "check" correspondante du gestionnaire de sécurité. Soit celle-ci autorise, elle fait alors un simple "return", soit elle refuse, elle lève alors une exception.

Les principales actions vérifiées avant exécution sont :

- accepter une connexion socket depuis un hôte et un port spécifiés,
- modifier les caractéristiques d'un flot (changer de priorité, arrêter, ...),
- ouvrir une connexion socket vers un hôte et un port spécifiés,
- créer un nouveau chargeur de classe,
- détruire un fichier spécifié,
- créer un nouveau process,
- sortir de l'application ;
- charger une bibliothèque dynamique contenant des méthodes natives,
- attendre une connexion sur un port local spécifié,
- charger une classe depuis un paquetage spécifié (concerne les chargeurs de classe) ;
- ajouter une nouvelle à un paquetage spécifié (concerne les chargeurs de classe),
- accède ou modifie certaines propriétés du système d'exploitation,
- lire un fichier spécifié,
- écrire dans un fichier spécifié.

Il reste, dans la version courante (1.1) deux actions potentiellement dangereuses qui ne sont pas vérifiées :

- allouer de la mémoire jusqu'à ce qu'il n'en reste plus,
- lançer de nouveaux flots (threads) jusqu'au nombre maximum possible.

Ces deux actions de type "déni de service" peuvent rendre un navigateur inutilisable par manque de ressources pour continuer normalement.

D'autre types d'applets hostiles peuvent :

- envoyer des e-mails intenpestifs depuis la machine de l'utilisateur,
- provoquer des bruits désagréables, même aprés que l'on ai quitté la page web,
- afficher des images ou des animations "choquantes".

Bien qu'on ne puisse installer qu'un seul gestionnaire de sécurité, on peut lui faire gérer plusieurs politiques de sécurité. Il est en effet possible de savoir, quand une méthode "check" est appelée, si la classe à l'origine de l'appel a été chargée par un chargeur ajouté, et si oui, par lequel. On peut aussi savoir d'où venait la classe qui a provoqué la vérification.

15.8 Mobilité du code dans le système Java

L'une des raisons fondamentales de l'importance et du succès de Java est sa capacité à produire du logiciel **mobile sur un réseau**. Cette capacité a été considérée par beaucoup comme un saut technologique.

Évolution de l'informatique et importance de la mobilité du code

L'informatique a subit de grandes évolutions à travers une série d'étapes et de modèles d'utilisation, liés aux évolutions de la technique, en particulier à celle des semi-conducteurs et des circuits intégrés (VLSI) :

- partage d'une machine centrale par de nombreux utilisateurs,
- utilisation de machines personnelles isolées et indépendantes,
- mise en réseau des machines personnelles et connexion à une machine centrale, avec un développement d'applications client-serveur 2 tiers (1 client - 1 serveur),
- généralisation de l'utilisation des réseaux, toutes les machines sont connectées, les applications deviennent plus complexes et évoluent vers le client-serveur 3 tiers (1 client sur une machine personnelle, 1 serveur intermédiaire responsable de traitement de données et 1 serveur "d'arrière plan" responsable du maintient de bases de données communes),
- les applications sur un ensemble de machines connectées effacent peu à peu la différence entre clients et serveurs (chaque partie jouant tour à tour l'un ou l'autre rôle) pour devenir de véritables applications distribuées ;
- la mobilité du logiciel sur le réseau représente donc une étape de plus dans l'évolution du modèle d'utilisation de l'informatique.

Cette évolution ne s'est pas faite sans inconvénients. Du côté des utilisateurs le côté positif est largement dominant : ceux-ci peuvent disposer d'applications mieux adaptées à leurs besoins et à leur façon de travailler. L'inconvénient est toutefois une complexité plus grande. Du côté des administrateurs de systèmes on peut raisonnablement affirmer que le bilan ne comporte que des aspects négatifs : la gestion de centaines de machines diverses à qui on doit faire exécuter correctement les mêmes programmes est un véritable cauchemar du point de vue de l'administration système. Ceci est fortement agravé par le caractère très primitif, sur le sujet de l'administration système, des systèmes d'exploitation disponibles sur les ordinateurs personnels.

C'est pourquoi, l'apparition d'un "système logiciel" permettant de rendre un nouveau programme disponible à l'identique pour des centaines, voire des milliers de machines, en ne l'installant que sur un seul serveur a été accueillit par les administrateurs de systèmes et de réseaux comme un quasi miracle. En 1998, on attend de voir si le miracle a une petite chance de se réaliser.

L'évolution du modèle d'utilisation vers des ordinateurs personnels a été rendue possible par la révolution des microprocesseurs, avec leur capacité en augmentation rapide et leurs prix, simultanément, en baisse rapide. Parallèlement, caché sous l'évolution des logiciels vers de plus en plus de mobilité et de connexion en réseau, se cache les capacités croissantes et les prix en baisse de la bande passante des réseaux de télécommunication. L'augmentation rapide de la bande passante solvable permet d'envisager de faire passer de nouveaux types d'information sur les réseaux informatiques : du son, de la voix, de la vidéo, des applications informatiques complètes, de la gestion d'interface utilisateurs, le tout résultant en un outil fortement interactif.

Dans ce nouveau modèle de code mobile, le code et les données perdent leur distinction pour devenir du contenu. Avant cette évolution, les utilisateurs étaient contraints à penser l'utilisation de l'informatique en terme de logiciels et de versions de ces logiciels. Ces versions évoluant, ils

avaient à décider quand passer à la version suivante et installer la ou les nouvelles versions de leurs logiciels, afin de disposer d'outils à jour. Avec le nouveau modèle l'utilisateur est poussé à penser en termes de contenu des services.

Le logiciel est alors fournit comme un flux auto-évolutif faisant partie d'un contenu interactif. On n'a plus besoin de le maintenir, il suffit de l'utiliser. Le logiciel prends alors l'aspect d'un bien de consommation ("a software appliance").

De nombreux services au contenu auto-évolutif vont (ou devront) partager avec les biens de consommation ménagers deux caractéristiques : une fonction dédiée et une interface simple. Quand vous décidez d'utiliser un grille-pain, vous ne vous attendez pas à devoir lire un manuel au préalable ! De même la fonctionalié de nombreux services de contenu devra être aussi dédiée (1 seul objectif) et l'utilisation aussi simple.

Un autre exemple est celui d'une page web : d'un certain point de vue, la page html ressemble à un programme avec des instructions et des paramètres, d'un autre point de vue, elle ressemble à des données. La distinction entre données et programme s'est estompée. Mais les personnes qui naviguent sur le web s'attendent à ce que les pages évoluent, sans qu'il soit besoin d'actions délibérées de leur part (tel que installer la nouvelle version !).

Le support de l'architecture Java pour la mobilité du code

Le support de l'architecture Java pour la mobilité du code commence avec le support de l'indépendance vis-à-vis des plateformes et celui de la sécurité. Bien que non strictement nécessaires, ces deux éléments aident à rendre praticable la mobilité du code. Le support de la mobilité est concentré sur le temps nécessaire à charger les codes binaires à travers le réseau. Java réponds à ce problème en rejetant le modèle de programme traditionnel (un gros fichier monolithique contenant tous les binaires suceptibles d'être utilisés par le programme). À la place, Java découpe le programme en un grand nombre de petites unités : les fichiers de classes Java.

Les fichiers de classes voyagent sur le réseau indépendamment les uns des autres. De plus comme Java supporte la liaison dynamique, l'utilisateur n'a pas besoin d'attendre que tout le programme soit chargé pour commençer son exécution. Le programme démarre dès que la première classe arrive.

L'exécution d'un programme Java débute à la méthode main() de l'une des classes. Les autres classes sont chargées et reliées dynamiquement au fur et à mesure que l'application en a besoin. Si une classe n'est jamais effectivement utilisée dans une exécution particulière du programme, elle n'est jamais téléchargée.

En plus de cette liaison dynamique, qui permet de réduire le traffic sur le réseau, les fichiers de classe sont conçus pour être compacts. Bien que ceci conduise à de petits fchiers, le foncionnement du protocole http 1.0 fait que chaque fichier de classe d'une applet est l'objet d'une requête http individuelle. La majeure partie du temps d'attente est alors occupée par les protocoles d'échange ("handshacking") pour chaque requête de fichier. Pour cette raison la version 1.1 de Java a défini des fichiers "JAR" (Java ARchive) capables de contenir plusieurs fichiers de classes, à envoyer à travers le réseau en une seule requête, ce qui réduit fortement le surcoût du traitement des requêtes. Plus, les données à l'intérieur du fichier "JAR" peuvent être compréssées, ce qui réduit encore le temps de transit.

L'Applet, un exemple de Java mobile sur un réseau

Java a été adopté si vite et si largement, non seulement parce qu'il est apparu la bonne technologie au bon moment, mais aussi parcequ'il a bénéficié d'un bon marketing.

1. il avait un nom facile à retenir ("cool" !),
2. il était, dans la pratique, gratuit (toujours un bon point pour un acheteur !),
3. il a été accroché au web au bon moment : juste quand Netscape cherchait à transformer "Navigator" d'un visualiseur d'hypertexte graphique en une plateforme de programmation à part entière.

... et la clé de ce succès a été la création d'une forme spéciale de programme Java, destinée à s'exécuter à l'intérieur d'un navigateur : l'applet.

L'applet Java exhibe tous les avantages de l'aspect "orienté réseau" de Java : indépendance de la plateforme, mobilité et sécurité. L'indépendance vis-à-vis de la plateforme a été l'une des principales motivations de la création du web, et l'applet Java s'y conforme exactement : un applet Java peut s'exécuter sur n'importe quelle plateforme sur laquelle un navigateur avec une machine virtuelle java y est installé.

Un navigateur exécute un appler java quand il rencontre une balise <applet ...> ... </applet> dans une page html. Cette balise fournit suffisament d'information au navigateur pour lui permettre d'afficher l'applet. Le navigateur passe l'information à la machine virtuelle Java qui initialise l'applet en appelant la méthode init() de la classe de démarrage de l'applet. Une fois l'initialisation de l'applet terminée, celui-ci apparait comme faisant partie de la page web.

15.9 La Machine Virtuelle Java (JVM)

L'expression "Machine Virtuelle Java" peut, selon le contexte, désigner trois choses différentes :

- la **spécification** de la Machine Virtuelle Java,
- une **implémentation** de la JVM,
- une **instance** d'exécution de la JVM.

La durée de vie d'une instance de JVM

Une instance de JVM a une mission claire : exécuter une application Java. L'instance nait quand une application est lançée, elle meurt quand l'application se termine. Si on lance simultanément 3 applications Java sur la même machine, chacune d'elles va s'exécuter dans sa propre instance de l'implémentation locale de la machine virtuelle.

Exemple

```
class Echo {
    public static void main (String[] args) {
         int len = args.length ;
         for (int i = 0; i < len; ++i) {
            System.out.print ("arg[" + i + "]= ");
            System.out.print (args[i] + " ") ;
         }
         System.out.println() ;
 }
}
```

Cette application va imprimer les arguments trouvés sur sa ligne de commande.

Elle est construite pour être lançée depuis une ligne de commande du système, **mais pas** depuis un navigateur.

La méthode **main(), doit** être "public static void" et prendre un String comme unique paramètre. Pour lancer cette application, il faut donner à l'instance de la machine créée le nom de la classe initiale (celle qui contient la méthode main()).

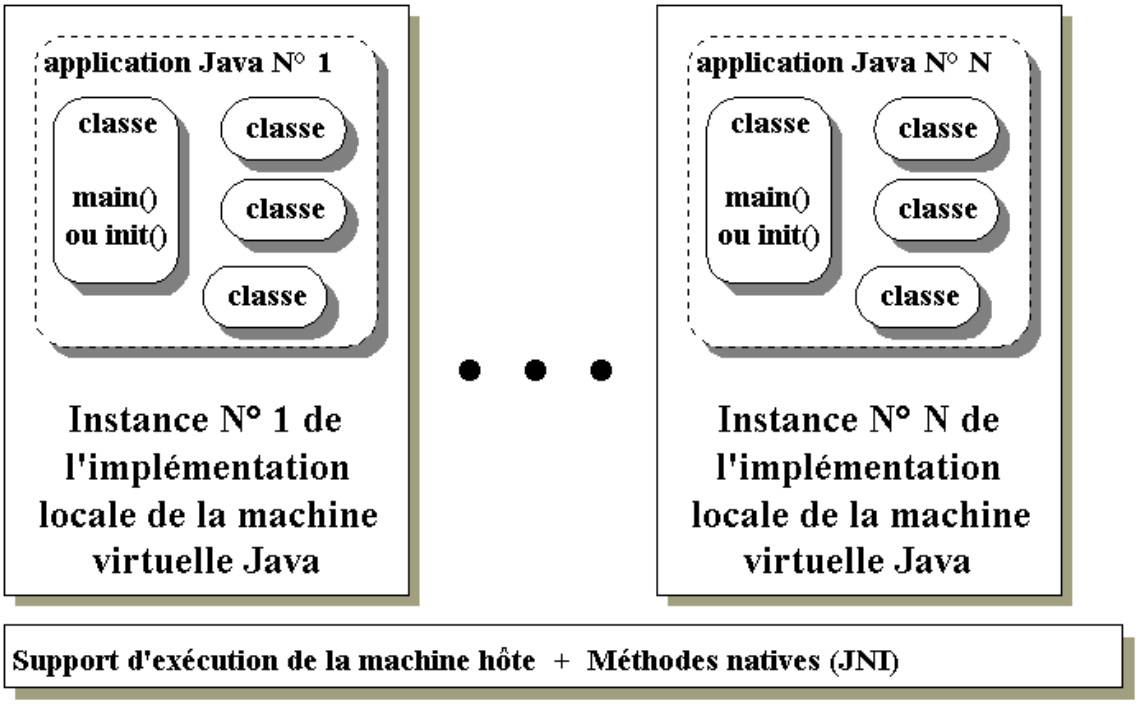


FIG. 162: La JVM

Par exemple : java Echo Hello World. La commande "java" lance une instance de la JVM, le premier argument "Echo" lui donne le nom de la classe initiale (il doit y avoir dans le répertoire courant un fichier Echo.class), les deux mots suivants sur la ligne sont considérés comme des arguments qui seront passés à main() dans le String "args".

Sous Win95, avec le JDK de Sun, il faut ajouter l'emplacement du JDK dans le PATH (dans le fichier autoexec.bat) :

```
PATH C:\WINDOWS;C:\WINDOWS\COMMAND;C:\;C:\DOS;C:\JDK1.1.4\BIN
```

On crée un fichier Echo.java contenant le source java. Le source est compilé par le compilateur "javac", et le résultat (le bytecode) stocké dans le fichier Echo.class

La commande "java Echo" va chercher dans le répertoire courant un fichier Echo.class qui doit contenir la classe "Echo", et l'exécuter.

Attention ! Les noms du fichier et de la classe doivent être identiques et respecter la casse.

```
>javac Echo.java
>java Echo Hello World.
arg[0]= Hello arg[1]= World.
```

L'architecture de la JVM

Dans la spécification de la JVM, son comportement est décrit en termes de sous-systèmes, zones mémoire, types de données et instructions. Le but est de décrire strictement le comportement des implémentations. La figure suivante décrit l'architecture de la JVM.

Parmi les zones mémoires d'une instance de machine virtuelle (donc pour une application java), certaines sont partagées par tous les flots (threads), par exemple la zone des méthodes (le

code exécutable) et le tas (heap) où sont stcokés les objets créés pendant l'exécution. Chaque flot créé possède sont propre registre et sa propre pile jave (contenant l'mpilement des formes correspondant aux invocations de méthodes).

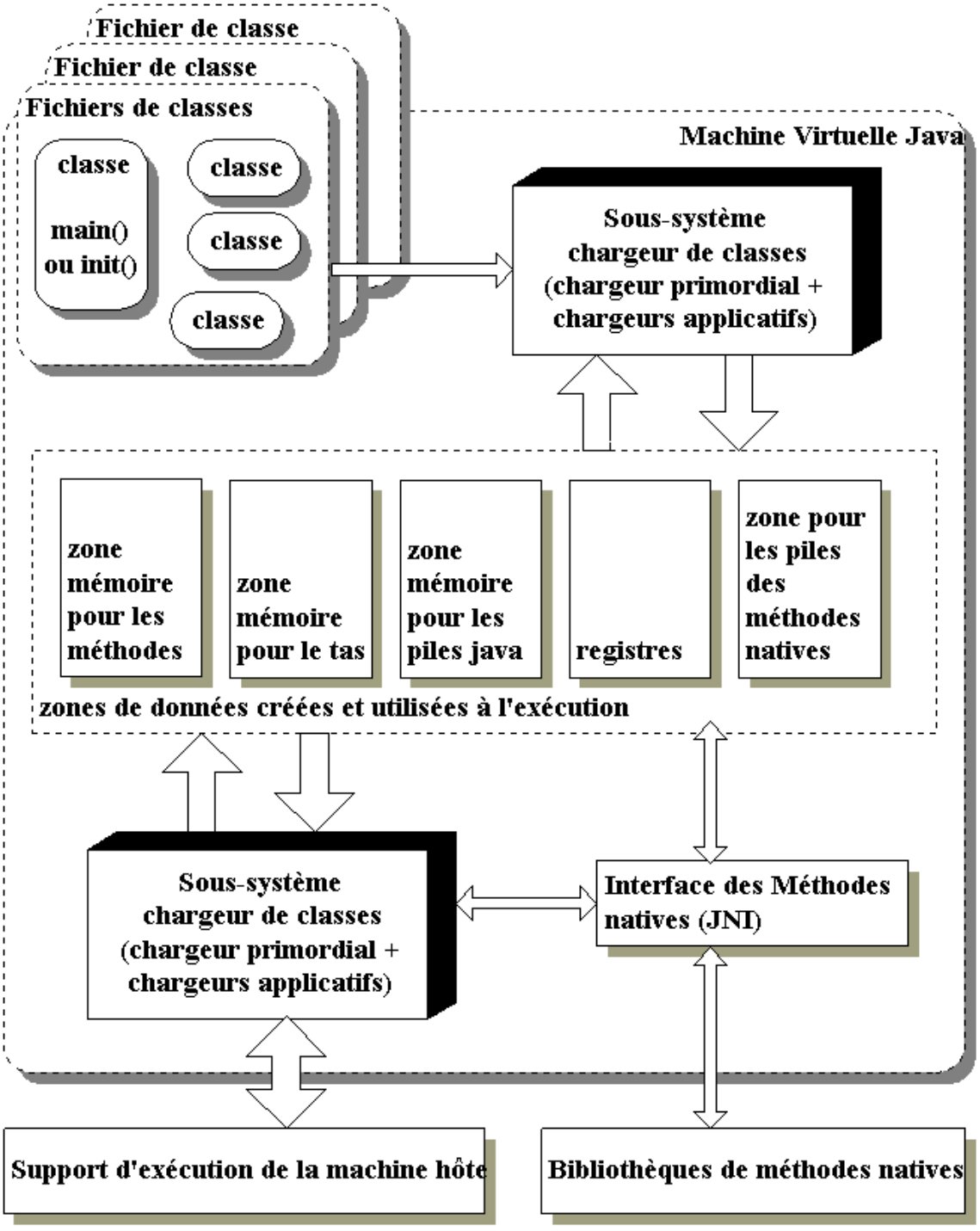


FIG. 163: Architecture de la JVM

Les piles java sont constituées d'une chaine de "formes" (stack frames), chacune contenant l'état d'une invocation de méthode java (l'équivalent d'un appel de fonction C). Chaque flot, n'a qu'un seul registre, le "pointeur courant" ("PC") indiquant la prochaine instruction à exécuter par le flot. Il n'y a pas de registres de données stockant des variables temporaires ou locales. Toutes les variables locales sont mises sur la pile java associée au flot. Ce choix a été fait pour simplifier la liste d'instructions de la machine virtuelle et permettre son implémentation dans

de très petites machines embraquées (e.g. JavaCard).

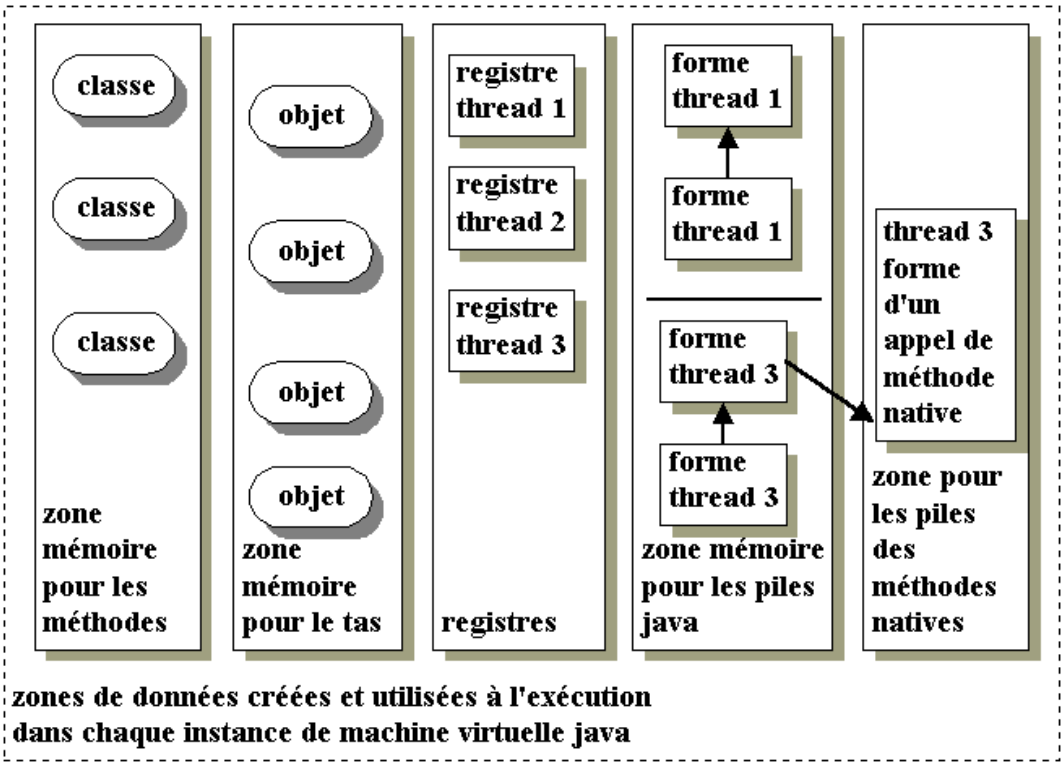


FIG. 164: Zones mémoire de la JVM

Les types de données de la JVM

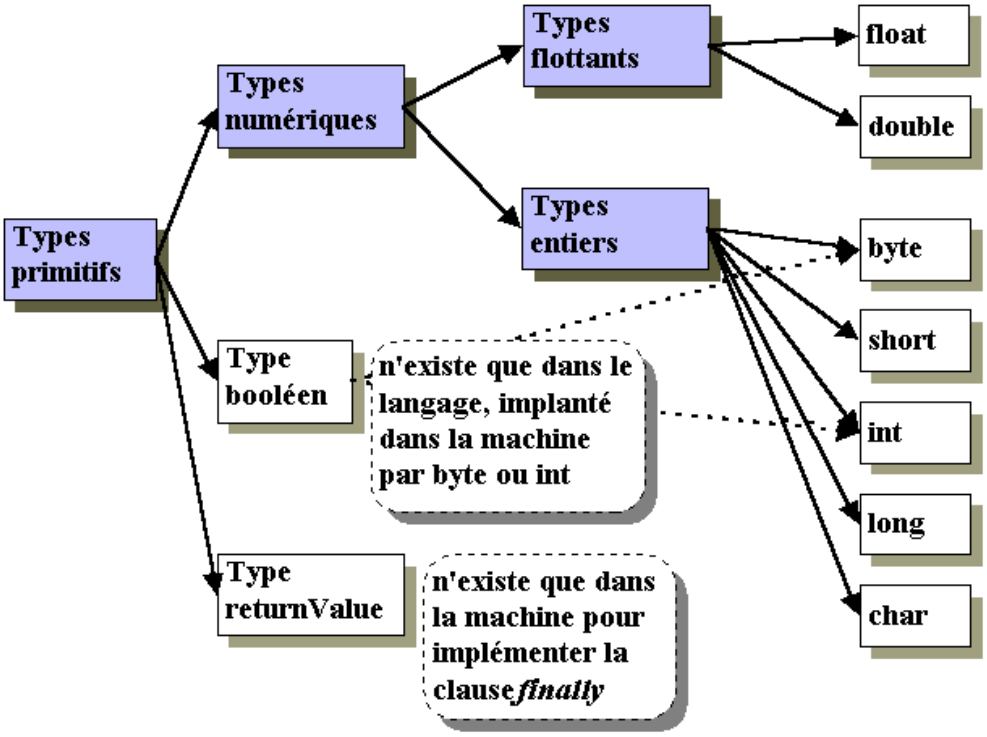


FIG. 165: Les types de données de la JVM

Dans la spécification de la JVM, les types de données et les opérations sur ces types sont strictement définies. Il y a deux grandes catégories de types : les types primitifs et les types référence. Les types primitifs sont décrits ci-dessous, ils existent à la fois dans la spécification du langage et dans celle de la machine virtuelle, sauf 2 exceptions annotées dans le tableau :

Les types "référence" contiennent des "pointeurs" vers des objets créés dynamiquement.

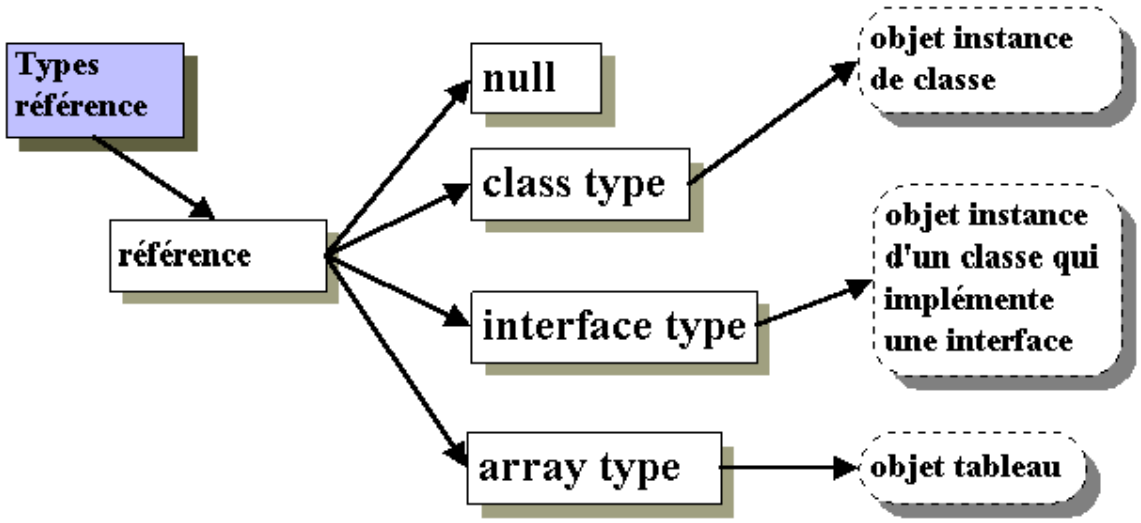


FIG. 166: les types référence de la JVM

La spécification de la JVM définit l'étendue des valeurs prises par chaque type de données, mais elle ne définit pas sa taille en mémoire. Cette dernière n'est pas accessible au programmeur et n'a pas d'influence sur le comportement du programme (il est choisit par l'implémenteur pour le meilleur compromis coût/efficacité sur le matériel considéré). Le tableau ci-dessous donne les étendues ("range") des types primitifs :

La spécification Java impose que les types de données de base aient toujours les définitions du tableau, quelque soient la taille du mot mémoire de la machine réelle qui implémente la machine virtuelle.

Type	**Étendue (range)**
byte	complément à 2 signé 8 bits (-2 7 à +2 7-1 inclus)
short	complément à 2 signé 16 bits (-2 15 à +2 15-1 inclus)
int	complément à 2 signé 32 bits (-2 31 à +2 31-1 inclus)
long	complément à 2 signé 64 bits (-2 63 à +2 63-1 inclus)
char	charactère UNICODE, non-signé 16 bits (0 à 2 16-1 inclus)
float	flottant simple précision 32 bits IEEE-754
double	flottant double précision 64 bits IEEE-754
reference	null ou référence vers un objet sur le tas
returnValue	adresse d'une instruction bytecode à l'intérieur de la même méthode

SR03 2004 - Cours Architectures Internet - Programmation Java
©Michel.Vayssade@utc.fr – Université de Technologie de Compiègne.

16 SR03 2004 - Cours Architectures Internet - Programmation Java

16.1 Application java et applet java

Le système Java permet de créer deux types de "programmes" :
– une "application java",
– une "applet java".

Nota : on verra plus loin d'autres façons d'exécuter du code Java : les JSP, les Servlets, les Beans et les EJBs. Toutefois dans ces quatres cas, il s'agit de portions de codes s'exécutant dans le cadre d'une **application**. La différence entre application et applet est essentiellement une différence **système** : quel est le process dans le cadre duquel le code s'exécute.

Une **application** est un programme java "standalone". Elle s'exécute dans le cadre d'une JVM instanciée dans un process indépendant, créé par le système d'exploitation. Elle s'exécute indépendamment d'un navigateur.

Une **applet** est un programme java téléchargeable" depuis un navigateur. Il s'exécute dans le cadre d'une instance de JVM créée par le navigateur.

application		
source	**compilation**	**exécution**
hello.java	javac hello.java	java hello
applet		
source	**compilation**	**exécution**
HelloWorld.java	javac HelloWorld.java	appel par browser

Exemple d'appel d'applet dans une page html :

```
<APPLET CODE="HelloWorld.class" WIDTH=150 HEIGHT=25></APPLET>
```

application java

```
(1) créer source :

367 lo33 ultra:~/public_html/java> more hello.java

class HelloWorldApp {
   public static void main(String[] args) {
        System.out.println("Hello World!"); //Display string.
   }
}

(2) compiler : javac hello.java

925 lo33 ultra:~/public_html/java> java hello
Can't find class hello
```

```
mv hello.class --> HelloWorldApp.class

le fichier doit avoir le nom de la classe

(3) exécuter :

919 lo33 ultra:~/public_html/java> java HelloWorldApp
Hello World!
```

---

applet java

```
(1) créer source

368 lo33 ultra:~/public_html/java> more HelloWorld.java

import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello world!", 50, 25);
    }
}

(2) compiler : javac HelloWorld.java

(3) créer un fichier HTML qui appelle l'applet compilé

926 lo33 ultra:~/public_html/java> more ../hel-java.html
<HTML>
<HEAD><TITLE> A Simple Program </TITLE></HEAD>
<BODY>
 Here is the output of my program:
<APPLET CODE="HelloWorld.class" WIDTH=150 HEIGHT=25>
</APPLET>

</BODY></HTML>

(4) l'exécution se fait quand on affiche le fichier
    HTML dans un browser

Note : l'applet (le fichier HelloWorld.class) doit se
====   trouver dans le même répertoire que le fichier HTML,
       sauf si on indique un autre répertoire dans le tag :
        <APPLET CODE=AppletSubclass.class CODEBASE=aURL
                WIDTH=anInt HEIGHT=anInt>
        </APPLET>

Note : avec le JDK, Sun fournit l'application "appletviewer":
====
930 lo33 ultra:~/public_html> appletviewer hel-java.html &
[5] 8079
```

Le point d'entrée de toute application java est la méthode "main" :
public static void main(String[] args) ...

Toute application java doit comporter une méthode "main". C'est elle qui "prends la main" et lance l'exécution d'autres méthodes.

Si main est absente, erreur :
In class Machin : void main(String argv[]) is not defined

main est déclarée :

- **public :** peut être appelée par tout objet,
- **static :** c'est une "class method",
- **void :** ne retourne aucune valeur.

Les arguments de la méthode main : un seul tableau "args" contenant "n" chaines :

```
public class Echo {
     public static void main (String[] args) {
         for (int i = 0; i < args.length; i++)
             System.out.println(args[i]);
     }
}
```

Seuls les arguments sont passés (pas le nom du programme) :

```
> java mon-appli arg-1 arg-2

args.length    contient 2
args[0]        contient la chaine "arg-1"
args[1]        contient la chaine "arg-2"
```

Toute applet java doit comporter l'une des trois méthodes "**init**" , "**start**" ou "**paint**" : c'est elle qui "prends la main" et lance l'exécution d'autres méthodes.

Paramètres d'une applet : ils sont aux applets ce que les arguments sont aux applications.

```
<APPLET CODE=AppletSubclass.class WIDTH=anInt HEIGHT=anInt>
<PARAM NAME=parameter1Name VALUE=aValue>
<PARAM NAME=parameter2Name VALUE=anotherValue>
</APPLET>
ex:
     <APPLET CODE="Animator.class" WIDTH=460 HEIGHT=160>
     <PARAM NAME="imageSource" VALUE="images/Beans">
     <PARAM NAME="backgroundColor" VALUE="0xc0c0c0">
     <PARAM NAME="endImage" VALUE=10>
     <PARAM NAME="soundSource" VALUE="audio">
     ......
     </APPLET>
```

Remarque : les "....." ci-dessus, peuvent contenir un code alternatif HTML, destiné aux navigateurs qui ne comprennent pas le tag <applet>.

Utilisation des classes et objets

En java une méthode ou une variable n'existe que dans une classe ou un objet (objet = instance d'une classe).

Tout programme java doit donc définir une ou des classes :
class ma_classe ...

La ligne : System.out.println("Hello World !") ; montre l'utilisation d'une variable de classe et d'une méthode d'instance.

D'une variable de classe : System.out
"out" est une variable de classe, associée à une classe et non pas à une instance d'une classe.

De même on peut créer des méthodes de classe :

– variable de classe appelée par : nom_classe.nom_var
– méthode de classe appelée par : nom_classe.nom_method

Les méthodes et variables qui ne sont pas "de classe" sont des méthodes d'instance et des variables d'instance : pour les utiliser, il faut y faire référence **DEPUIS UN OBJET**.

Explication de la ligne de code citée plus haut :
Quand la classe "System" est chargée dans l'application, elle instancie la classe "PrintStream" et assigne le nouvel objet "PrintStream" à la variable de classe "out". Comme on dispose d'une instance de "PrintStream", représentée par l'objet "System.out", on peut appeler l'une de ses méthodes d'instance, ici "println" en postfixant la référence d'objet par le nom de la fonction : System.out.println().

16.2 Un exemple d'applet

csbar.html

```
 1  <HTML>  <!---->
 2  <HEAD><TITLE>The cbar Applet</TITLE></HEAD>
 3  <BODY>
 4  <p> If you're using a Java-enabled browser, applet will show below.
 5  <p> If your browser can't run Java applets, you're out of luck.
 6  <p> Sorry!
 7  <P>
 8  <APPLET code="csbar.class" width=500 height=300>
 9  </APPLET>
10  </BODY></HTML>  <!---->
```

csbar.java

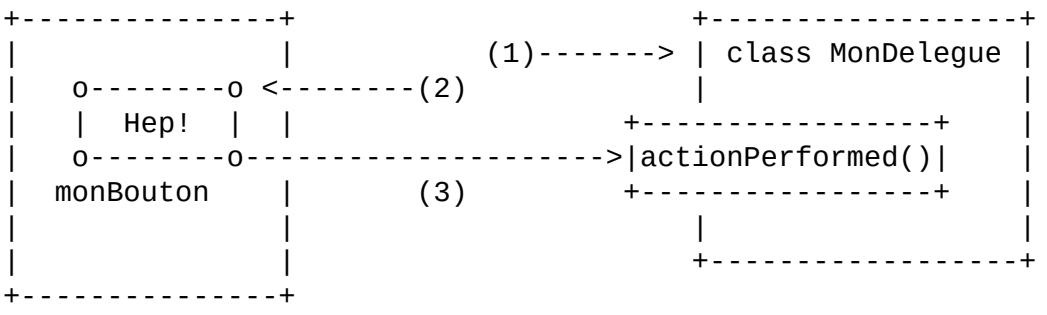
```
 1  /* Exemple d'applet simple : csbar.java */
 2  import java.applet.*;
 3  import java.awt.*;
 4
 5  public class csbar extends Applet {
 6
 7          private Color _light;
 8          private Color _dark;
 9
10     public void init() {
11          _light = getBackground().brighter().brighter();
12          _dark  = getBackground().darker().darker();
```

csbar.java (suite)

```
13          Graphics g = this.getGraphics();
14          paint(g);
15      }
16
17      public void paint ( Graphics g)
18      {
19          //Dimension d= getSize(); //1.1 : marche pas avec Comm.4?
20          Dimension d = size();
21          int length = d.height;
22          int pos = d.width/2;
23          g.drawString("length pos = "+length+" "+pos, length, pos);
24          // dessiner un rectangle autour surface
25          g.setColor ( _light );
26          g.draw3DRect(30, 30, d.width - 40 , d.height - 40, true);
27          g.setColor ( _dark );
28
29          int ix = d.width-44;
30          int iy = d.height-44;
31          g.drawString("ix , iy = "+ ix +" "+ iy, 300, 150);
32          g.drawString("A",ix,iy);
33          g.draw3DRect(32, 32, ix , iy , false);
34          g.draw3DRect(40, 80, 150 , 100 , true);
35
36          // dessiner une ligne en relief
37          g.setColor ( _light );
38          g.drawLine (pos, 5, pos, length-10);
39          g.setColor ( _dark );
40          g.drawLine (pos+1, 5, pos+1, length-10);
41
42          // dessiner une ligne en creux
43          g.setColor ( _dark );
44          g.drawLine (25, pos, length-10, pos);
45          g.setColor ( _light );
46          g.drawLine (26, pos+1, length-11, pos+1);
47
48          g.setColor ( _dark );
49          g.drawLine (25, 10, 25, 40);
50          g.setColor ( _light );
51          g.drawLine (26, 10, 25, 40);
52      }
53
54      public Dimension getPreferredSize()
55      {    return new Dimension (4, 8); }
56
57  }
```

16.3 Évènements en Java

Un premier exemple de gestion d'évènements

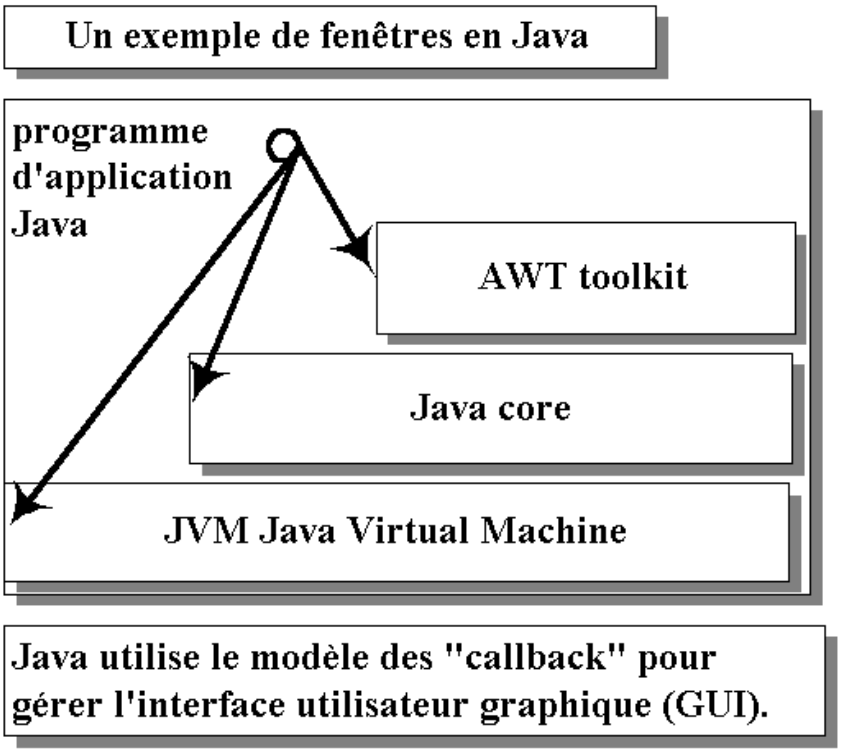


(1) créer une instance d'une classe déléguée
(2) enregistrer cette instance comme "actionListener"
du widget "monBouton"
(3) lors d'un clic sur le bouton, celui-ci appelle
la fonction actionPerformed() de l'instance déléguée

Simple.java

```
 1  // Simple.java : demo traitement évènements
 2  import java.awt.*; // accéder à Frame et Button
 3  import java.awt.event.*; // accéder à ActionListener
 4
 5  public class Simple extends Frame {
 6      public Simple () {  // constructeur
 7          Button monBouton = new Button ("Hep!"); //ajouter un bouton
 8          // créer un délégué
 9          // et l'enregistrer auprès du bouton
10          monBouton.addActionListener ( new MonDelegue() );
11          add ( monBouton );
12      }
13
14      public static void main (String args[]) { // entrée de l'appli.
15          Frame f = new Simple(); // créer la fenêtre
16          f.pack(); // l'afficher
17          f.setVisible (true);
18      }
19  }
20
21  // créer une classe déléguée gérant les évènements sur le bouton
22  class MonDelegue implements ActionListener {
23      public void actionPerformed (ActionEvent e) {
24          System.exit(0);
25      }
26  }
```

Un exemple de fenêtres et évènements en Java



Les différents conposants de l'AWT (Abstract Window Toolkit) peuvent générer des ***évènements***.
Par exemple :

Composant	Event généré	Signification
Button	ActionEvent	Clic sur un bouton
Checkbox	ItemEvent	Choix d'une case à cocher
MenuItem	ActionEvent	choix d'un item de menu

Le traitement des évènements dans le modèle Java 1.1 est basé sur le concept de `"event listener"` : un objet intéressé par la réception d'un évènement va s'enregistrer auprès de l'objet source de l'évènement et s'ajouter ainsi à la liste des "events listeners". Quand la source génère un évènement, elle va notifier à tous les objets listeners que cet évènement s'est produit.

FIG. 167: Fenêtres et évènements java 1/3

Un exemple de fenêtres en Java

La source notifie un objet "event listener" qu'un évènement s'est produit en invoquant une méthode de l'objet "intéressé" et en passant l'objet "event" en argument de cette méthode. Pour que la source soit certaine de pouvoir invoquer la méthode, tous les objets qui se déclarent comme "event listeners" doivent implémenter une interface correspondant à cet évènement.
À chaque évènement correspont une (ou 2) interface, par exemple :

Event généré	Listener Interface	Méthodes
ActionEvent	ActionListener	actionPerformed()
ItemEvent	ItemListener	itemStateChanged()
MouseEvent	MouseListener	mouseClicked()
		mousePressed()
		mouseRelease()
	MouseMotionListener	mouseDragued()

Par convention, toutes les méthodes d'un "event listener" reçoivent un argument unique qui est un objet évènement du type correspondant à celui du listener.

Pour chacune des interfaces "listener" comportant plus d'une méthode, java.awt.event définit une classe "adaptateur" qui contient un corps vide pour chacune des méthodes. Ainsi l'objet "event listener" n'a qu'à sous-classer l'adaptateur (héritage) et surcharger les seules méthodes qui l'intéresse.

FIG. 168: Fenêtres et évènements java 2/3

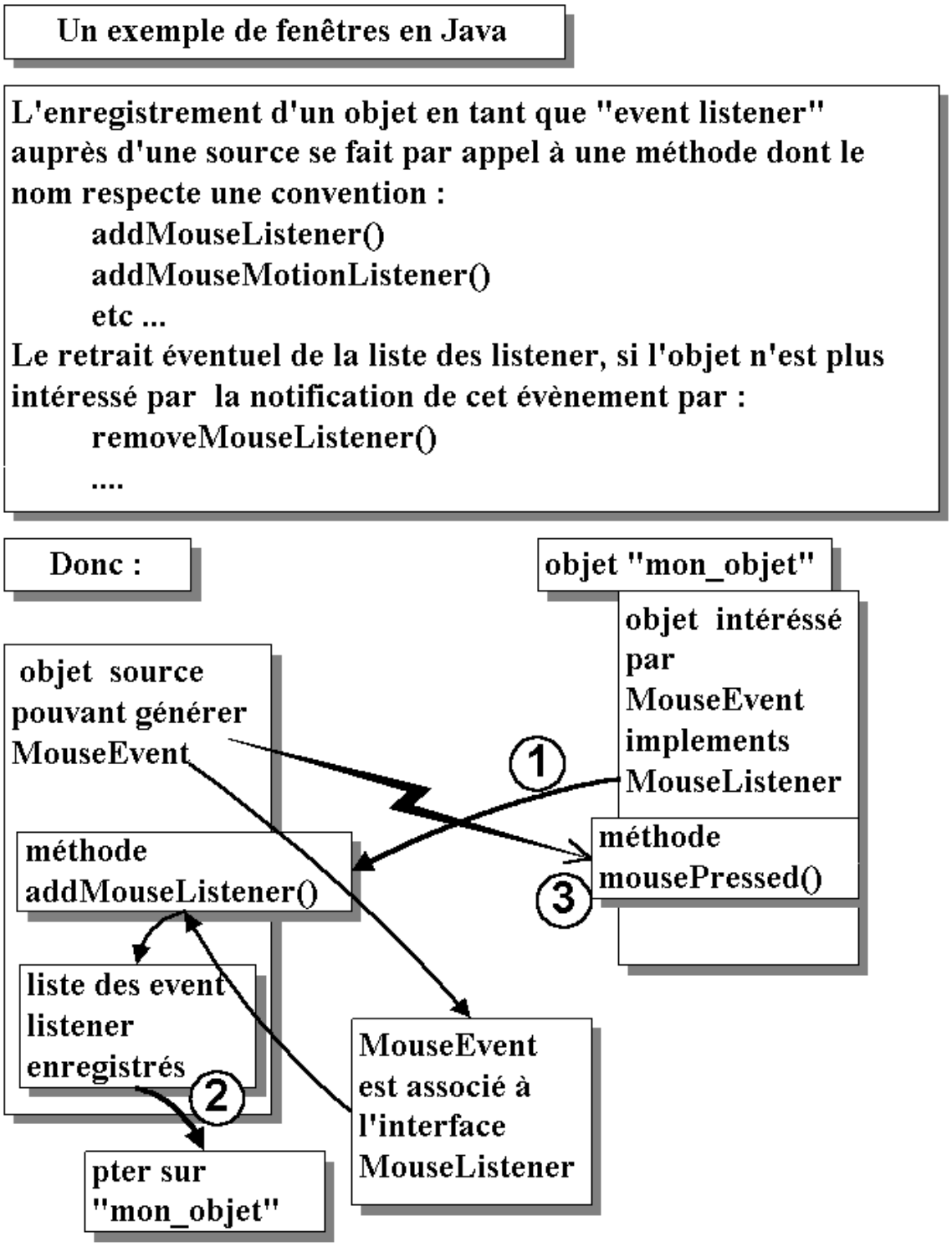


FIG. 169: Fenêtres et évènements java 3/3

Scribble2.html

```
1  <HTML>  <!---->
2  <HEAD><TITLE>The Scribble2 Applet</TITLE></HEAD>
3  <BODY>
4  <hr>
5  Scribble2 Applet dans le cadre ci-dessous :
```

Scribble2.html (suite)

```
 6  <table BORDER="1" ><tr><td>
 7  <APPLET CODE="Scribble2.class" WIDTH=400 HEIGHT=400></APPLET>
 8  </td></tr></table>
 9  <hr>
10  </BODY></HTML>  <!---->
```

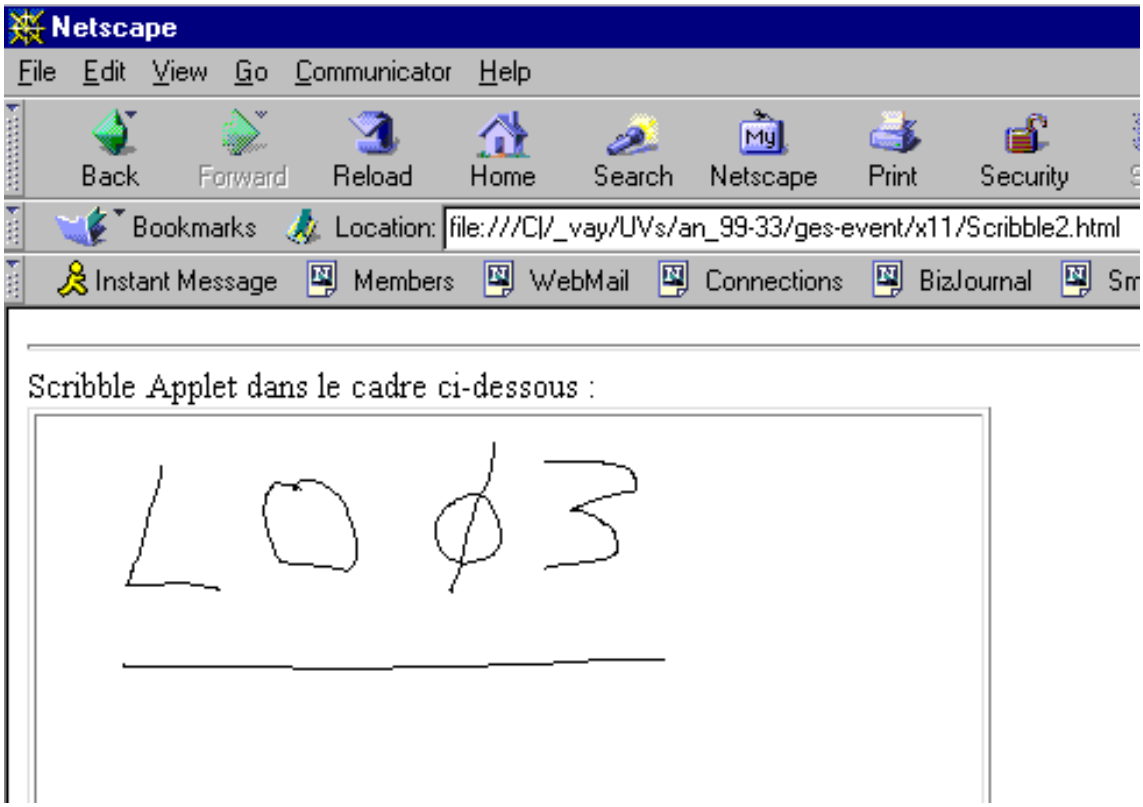


FIG. 170: Scribble2

Scribble2.java

```
 1  // This example is from _Java Examples in a Nutshell_.
 2  // http ://www.oreilly.com Copyright (c) 1997 by David Flanagan
 3  // This example is provided WITHOUT ANY WARRANTY either
 4  // expressed or implied. You may study, use, modify, and
 5  // distribute it for non-commercial purposes.
 6
 7  import java.applet.*;
 8  import java.awt.*;
 9  import java.awt.event.*;
10
11  /* A simple applet that uses the Java 1.1 event handling model */
12  public class Scribble2 extends Applet
13                 implements MouseListener,  MouseMotionListener {
14    private int last_x, last_y;
15
16    public void init() {
```

Scribble2.java (suite)

```
17      // Tell this applet what MouseListener and MouseMotionListener
18      // objects to notify when mouse and mouse motion events occur.
19      // Since we implement the interfaces ourself,
20      // our own methods are called.
21      this.addMouseListener(this);
22      this.addMouseMotionListener(this);
23    }
24
25    // A method from the MouseListener interface.  Invoked when the
26    // user presses a mouse button.
27    public void mousePressed(MouseEvent e) {
28      last_x = e.getX();
29      last_y = e.getY();
30    }
31
32    // A method from the MouseMotionListener interface.
33    // Invoked when the user drags the mouse with a button pressed.
34    public void mouseDragged(MouseEvent e) {
35      Graphics g = this.getGraphics();
36      int x = e.getX(), y = e.getY();
37      g.drawLine(last_x, last_y, x, y);
38      last_x = x; last_y = y;
39    }
40
41    // The other, unused methods of the MouseListener interface.
42    public void mouseReleased(MouseEvent e) {;}
43    public void mouseClicked(MouseEvent e) {;}
44    public void mouseEntered(MouseEvent e) {;}
45    public void mouseExited(MouseEvent e) {;}
46
47    // The other method of the MouseMotionListener interface.
48    public void mouseMoved(MouseEvent e) {;}
49  }
```

Un autre exemple d'évènements en Java

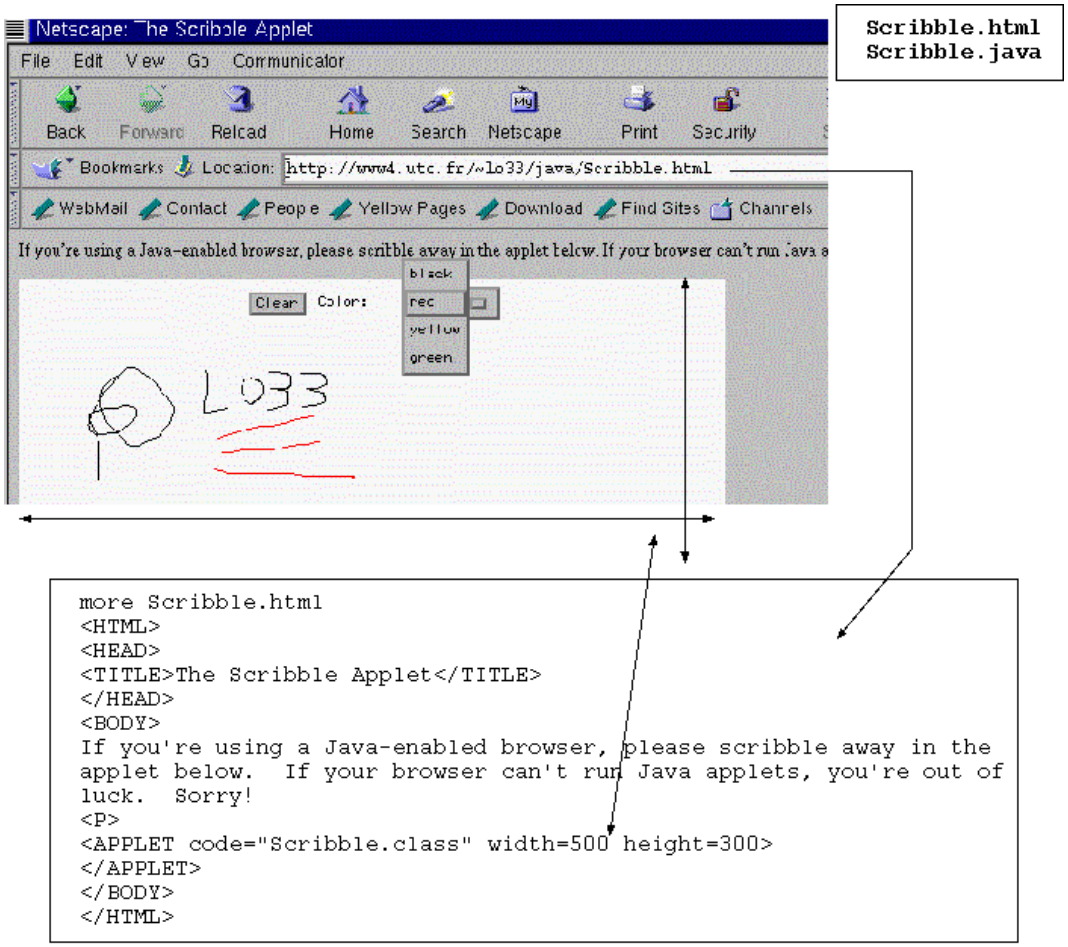


FIG. 171: Scribble

Scribble.java

```
 1  // This example is from the book _Java in a Nutshell_
 2  // Written by David Flanagan. Copyright (c)1996 O'Reilly
 3  // & Associates. You may study, use, modify, and distribute
 4  // this example for any purpose. Provided WITHOUT WARRANTY.
 5
 6  import java.applet.*;
 7  import java.awt.*;
 8
 9  public class Scribble extends Applet {
10      private int last_x = 0;
11      private int last_y = 0;
12      private Color current_color = Color.black;
13      private Button clear_button;
14      private Choice color_choices;
15
16      // Called to initialize the applet.
17      public void init() {
18          // Set the background color
```

Scribble.java (suite)

```
19          this.setBackground(Color.white);
20
21          // Create a button and add it to the applet.
22          // Also, set the button's colors
23          clear_button = new Button("Clear");
24          clear_button.setForeground(Color.black);
25          clear_button.setBackground(Color.lightGray);
26          this.add(clear_button);
27
28          // Create a menu of colors and add it to the applet.
29          // Also set the menus's colors and add a label.
30          color_choices = new Choice();
31          color_choices.addItem("black");
32          color_choices.addItem("red");
33          color_choices.addItem("yellow");
34          color_choices.addItem("green");
35          color_choices.setForeground(Color.black);
36          color_choices.setBackground(Color.lightGray);
37          this.add(new Label("Color : "));
38          this.add(color_choices);
39      }
40
41      // Called when the user clicks the mouse to start a scribble
42      public boolean mouseDown(Event e, int x, int y)
43      {
44          last_x = x; last_y = y;
45          return true;
46      }
47
48      // Called when the user scribbles with the mouse button down
49      public boolean mouseDrag(Event e, int x, int y)
50      {
51          Graphics g = this.getGraphics();
52          g.setColor(current_color);
53          g.drawLine(last_x, last_y, x, y);
54          last_x = x;
55          last_y = y;
56          return true;
57      }
58
59      // Called when the user clicks the button or chooses a color
60      public boolean action(Event event, Object arg) {
61          // If the Clear button was clicked on, handle it.
62          if (event.target == clear_button) {
63              Graphics g = this.getGraphics();
64              Rectangle r = this.bounds();
65              g.setColor(this.getBackground());
66              g.fillRect(r.x, r.y, r.width, r.height);
67              return true;
68          }
69          // Otherwise if a color was chosen, handle that
70          else if (event.target == color_choices) {
```

Scribble.java (suite)

```
71              String colorname = (String) arg;
72              if (arg.equals("black")) current_color= Color.black;
73              else if (arg.equals("red")) current_color= Color.red;
74              else if (arg.equals("yellow"))
75                                  current_color = Color.yellow;
76              else if (arg.equals("green"))
77                                  current_color = Color.green;
78              return true;
79          }
80          // Otherwise, let the superclass handle it.
81          else return super.action(event, arg);
82      }
83
84  }
```

16.4 Un exemple de gestion d'évènements "exceptions utilisateur"

throwtest.html

```
1  <HTML> <!---->
2  <HEAD><TITLE>MyException Applet</TITLE></HEAD>
3  <BODY>
4  If your browser can't run Java applets, you're out of
5  luck.  Sorry!
6  <P>
7  <APPLET code="throwtest.class" width=500 height=300>
8  </APPLET>
9  </BODY></HTML> <!---->
```

throwtest.java

```
 1  // Here we define some exception types of our own.
 2  // Exception classes generally have constructors but
 3  // no data or other methods. All these do is to call
 4  // their superclass constructors.
 5  class MyException extends Exception {
 6      public MyException() { super(); }
 7      public MyException(String s) { super(s); }
 8  }
 9  class MyOtherException extends Exception {
10      public MyOtherException() { super(); }
11      public MyOtherException(String s) { super(s); }
12  }
13  class MySubException extends MyException {
14      public MySubException() { super(); }
15      public MySubException(String s) { super(s); }
```

throwtest.java (suite)

```
16  }
17
18  // This class demonstrates defining, throwing and handling
19  // exceptions. Try invoking it in the following ways and
20  // try to understand the output :
21  //    java throwtest
22  //    java throwtest one
23  //    java throwtest 0
24  //    java throwtest 1
25  //    java throwtest 99
26  //    java throwtest 2
27  //    java throwtest 3
28  public class throwtest {
29      // This is the main() method.  Note that it uses two
30      // catch clauses to handle two standard Java exceptions.
31      public static void main(String argv[]) {
32          int i;
33
34          // First, convert our argument to an integer
35          // Make sure we have an argument and that
36          // it is convertible.
37          try {
38              i = Integer.parseInt(argv[0]);
39          }
40          catch (ArrayIndexOutOfBoundsException e) {
41              // argv is empty
42              System.out.println("Must specify an argument");
43              return;
44          }
45          catch (NumberFormatException e) {
46             // argv[0] not an integer
47              System.out.println("Specify an integer argument.");
48              return;
49          }
50
51          // Now, pass that integer to method a().
52          a(i);
53      }
54
55      // This method invokes b(), which is declared to throw
56      // one type of exception.  We handle that one exception.
57      public static void a(int i) {
58          try {
59              b(i);
60          }
61          catch (MyException e) {                    // Point 1.
62              // Here we handle MyException and
63              // its subclass MyOtherException
64              if (e instanceof MySubException)
65                  System.out.print("MySubException : ");
66              else
67                  System.out.print("MyException : ");
```

throwtest.java (suite)

```
 68              System.out.println(e.getMessage());
 69              System.out.println("Handled at point 1");
 70          }
 71      }
 72
 73      // This method invokes c(), and handles one of the
 74      // two exception types that that method can throw. The other
 75      // exception type is not handled, and is propagated up
 76      // and declared in this method's throws clause.
 77      // This method also has a finally clause to finish up
 78      // the work of its try clause.  Note that the finally clause
 79      // is executed after a local catch clause, but before a
 80      // a containing catch clause or one in an invoking procedure.
 81      public static void b(int i) throws MyException {
 82          int result;
 83          try {
 84              System.out.print("i = " + i);
 85              result = c(i);
 86              System.out.print(" c(i) = " + result);
 87          }
 88          catch (MyOtherException e) {                // Point 2
 89              // Handle MyOtherException exceptions :
 90              System.out.println(
 91                          "MyOtherException : " + e.getMessage());
 92              System.out.println("Handled at point 2");
 93          }
 94          finally {
 95              // Terminate output we printed above with a newline.
 96              System.out.print("\n");
 97          }
 98      }
 99
100      // This method computes a value or throws an exception.
101      // The throws clause only lists two exceptions, because
102      // one of the exceptions thrown is a subclass of another.
103      public static int c(int i) throws MyException,
104                                        MyOtherException {
105          switch (i) {
106              case 0 : // processing resumes at point 1 above
107                  throw new MyException("input too low");
108              case 1 : // processing resumes at point 1 above
109                  throw new MySubException("input still too low");
110              case 99 :// processing resumes at point 2 above
111                  throw new MyOtherException("input too high");
112              default :
113                  return i*i;
114          }
115      }
116  }
```

throwtest.txt

```
 1
 2  460 lo33 ultra :~/public_html/java/section2> java throwtest
 3  Must specify an argument
 4  461 lo33 ultra :~/public_html/java/section2> java throwtest aaa
 5  Must specify an integer argument.
 6  462 lo33 ultra :~/public_html/java/section2> java throwtest 0
 7  i = 0
 8  MyException : input too low
 9  Handled at point 1
10  463 lo33 ultra :~/public_html/java/section2> java throwtest 1
11  i = 1
12  MySubException : input still too low
13  Handled at point 1
14  464 lo33 ultra :~/public_html/java/section2> java throwtest 99
15  i = 99MyOtherException : input too high
16  Handled at point 2
17
18  465 lo33 ultra :~/public_html/java/section2> java throwtest 44
19  i = 44 c(i) = 1936
20  466 lo33 ultra :~/public_html/java/section2>
```

16.5 Un autre exemple : gestion d'évènements associés à des widgets

Scribble5.java

```
 1  // This example is from _Java Examples in a Nutshell_.
 2  // (http ://www.oreilly.com) Copyright 1997 by David Flanagan
 3  // You may study, use, modify, and distribute it for
 4  // non-commercial purposes. WITHOUT ANY WARRANTY
 5
 6  import java.applet.*;
 7  import java.awt.*;
 8  import java.awt.event.*;
 9
10  /** The application class.
11      Processes high-level commands sent by GUI */
12  public class Scribble5 {
13    /** main entry point. Just create an instance
14        of this application class */
15    public static void main(String[] args) { new Scribble5(); }
16
17    /** Application constructor : create an instance of our
18        GUI class */
19    public Scribble5() { window = new ScribbleGUI(this); }
20    protected Frame window;
21
22    /** This is the application method that processes commands
23        sent by GUI */
```

Scribble5.java (suite)

```
24    public void doCommand(String command) {
25      if (command.equals("clear")) {      // clear the GUI window
26        // It would be more modular to include this functionality
27        // in the GUI class itself.  But for demonstration
28        // purposes, we do it here.
29        Graphics g = window.getGraphics();
30        g.setColor(window.getBackground());
31        g.fillRect(0, 0, window.getSize().width,
32                         window.getSize().height);
33      }
34      else if (command.equals("print")) {} // not yet implemented
35      else if (command.equals("quit")) {   // quit the application
36        window.dispose();                      // close the GUI
37        System.exit(0);                        // and exit.
38      }
39    }
40  }
41
42  /** This class implements the GUI for our application */
43  class ScribbleGUI extends Frame {
44    int lastx, lasty;   // remember last mouse click
45    Scribble5 app;      // A reference to the application,
46                        // to send commands to.
47    /**
48     * The GUI constructor does all the work of creating the GUI
49     * and setting up event listeners. Note the use of local and
50     * anonymous classes.
51     */
52    public ScribbleGUI(Scribble5 application) {
53      super("Scribble");   // Create the window
54      app = application;   // Remember the application reference
55
56      // Create three buttons
57      Button clear = new Button("Clear");
58      Button print = new Button("Print");
59      Button quit = new Button("Quit");
60
61      // Set a LayoutManager, and add the buttons to the window.
62      this.setLayout(new FlowLayout(FlowLayout.RIGHT, 10, 5));
63      this.add(clear); this.add(print);  this.add(quit);
64
65      // Here's a local class used for action listeners
66      // for the buttons
67      class ScribbleActionListener implements ActionListener {
68        private String command;
69        public ScribbleActionListener(String cmd) {
70                  command = cmd; }
71        public void actionPerformed(ActionEvent e) {
72                  app.doCommand(command); }
73      }
74
75      // Define action listener adapters that connect the
```

Scribble5.java (suite)

```
 76      // buttons to the app
 77      clear.addActionListener(new ScribbleActionListener("clear"));
 78      print.addActionListener(new ScribbleActionListener("print"));
 79      quit.addActionListener(new ScribbleActionListener("quit"));
 80
 81      // Handle the window close request similarly
 82      this.addWindowListener(new WindowAdapter() {
 83        public void windowClosing(WindowEvent e) {
 84                            app.doCommand("quit"); }
 85      });
 86
 87      // High-level action events are passed to the application,
 88      // but we still handle scribbling right here.
 89      // Register a MouseListener object.
 90      this.addMouseListener(new MouseAdapter() {
 91        public void mousePressed(MouseEvent e) {
 92          lastx = e.getX(); lasty = e.getY();
 93        }
 94      });
 95
 96      // Define, instantiate and register a
 97      // MouseMotionListener object
 98      this.addMouseMotionListener(new MouseMotionAdapter() {
 99        public void mouseDragged(MouseEvent e) {
100          Graphics g = getGraphics();
101          int x = e.getX(), y = e.getY();
102          g.setColor(Color.black);
103          g.drawLine(lastx, lasty, x, y);
104          lastx = x; lasty = y;
105        }
106      });
107
108      // Finally, set the size of the window, and pop it up
109      this.setSize(400, 400);
110      this.show();
111    }
112  }
```

```
"Scribble5.java" : fonctionnement
main
   new ScribbleGUI()   crèe une Frame
-->
   constructeur de ScribbleGUI :
      créer 3 boutons
      pour chaque bouton "b" :
          b.addActionListener(new ScribbleActionListener("bbb")
          // la chaine "bbb" en argument servira à distinguer
          // chaque instance de Listener et sera repassée dans
          // l'appel app.doCommand(cmde) renvoyé vers le prog.
          // principal if(command.equals("bbb") ....
```

SR03 2004 - Cours Architectures Internet - Programmation Java
Fin du chapitre.

SR03 2004 - Cours Architectures Internet - Servlets et JSP

17 SR03 2004 - Cours Architectures Internet - Servlets et JSP

17.1 Introduction : Qu'est-ce qu'une servlet et une JSP ?

Servlets et JSP sont des composants logiciels destinés à être exécutés dans un **serveur d'application J2EE**.

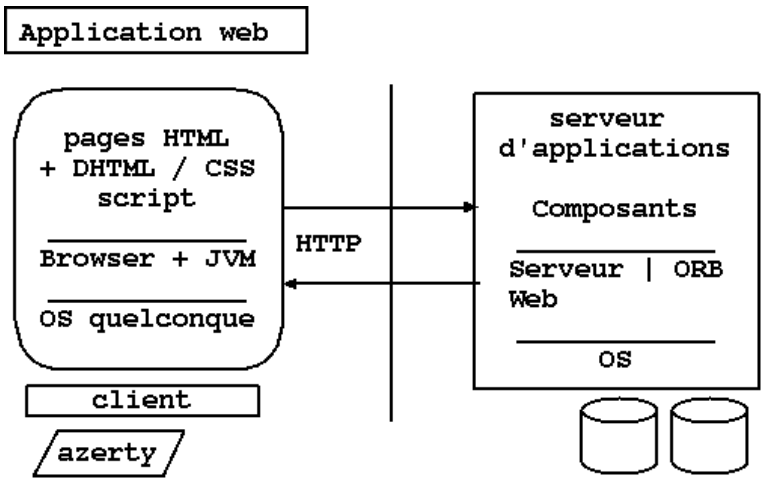


FIG. 172: Application web

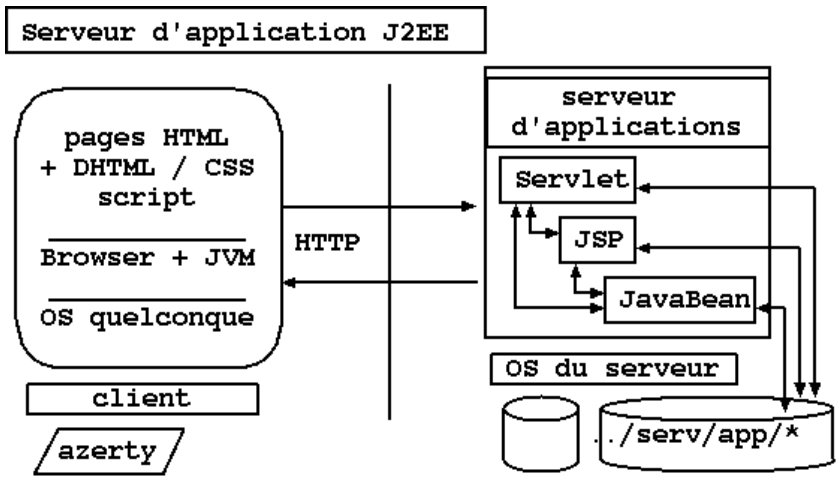


FIG. 173: Serveur d'application J2EE

Une servlet est une classe java qui est capabke de lire le contenu d'une requête HTTP et de produire (sur son flux de sortie) une réponse HTTP.

Une JSP est un squelette de page HTML qui contient des portions de code java, comme PHP, entre les balises <% et %>.

Servlets et JSP peuvent s'appeler les unes les autres.

Servlets, JSP et JavaBeans : aussi bien les servlets que les JSP peuvent appeler un JavaBean.

17.2 Les Servlets et les JSP avec Tomcat

Qu'est-ce que Tomcat ? Tomcat 4 est un conteneur Servlet/JSP. Il implémente "the Servlet 2.3 and JavaServer Pages 1.2 specifications from Java Software".

D'après ses auteurs, c'est "a useful platform for developing and deploying web applications and web services".

C'est un produit du projet "Jakarta" de Apache.

La documentation, sur l'installation de sunserv, est accessible à l'URL :
http ://sunserv.utc :8080/tomcat-docs/index.html

Sur sunserv, on peut vérifier que le serveur Tomcat est présent par la commande suivante :

```
1556 lo33 sunserv:~/mv/servlet> ps -ef | grep -i jak
http 18957    1  0 09:16:39 pts/47  0:09 /usr/java/bin/../bin/sparc/
               native_threads/java -classpath /usr/local/JAKARTA/ja
lo33 19016 7212  0 09:18:14 pts/26  0:00 grep -i jak

si non : su puis :
/etc/init.d/tomcat start
```

La commande "list" du gestionnaire de Tomcat permet de connaitre la liste des applications tournant sur le serveur :

```
Commande par entrée de l'URL :
         http://sunserv.utc:8080/manager/list
Réponse =

OK - Listed applications for virtual host localhost
/examples:running:0
/webdav:running:0
/kap6:running:0
/tomcat-docs:running:0
/manager:running:0
/:running:0
```

Le gestionnaire de Tomcat demande un Username et un Password. Ce sont des informations spécifiques à Tomcat.

Premiers exemples

Pour exécuter une "application web", il faut d'abord compiler cette application, puis **l'installer** dans Tomcat. Ceci se fait par la commande :

```
http://www4.utc.fr:8080/manager/install?
       path=/lo33&war=file:/usr/uvs/lo33/lo33/mv/servlet

la réponse de Tomcat est :

OK - Installed application at context path /lo33
```

Le résultat de cette commande (encore une fois envoyée à Tomcat comme une URL, par l'intermédiaire du navigateur), est que le contenu du répertoire de l'utilisateur sera considéré par le serveur Tomcat comme la **racine** des applications réportoriées sous le nom **/lo33**.

L'appel de l'URL : http ://sunserv.utc :8080/lo33/ va donner dans le navigateur la liste des fichiers du répertoire associé.

La mise en place de l'application :

Elle doit obéir à des règles précises concernant le placement des sources JSP, et des fichiers ".class" :

```
1684 lo33 sunserv:~/mv/servlet> ls -R
.:
HelloWorld.java       StringBean.jar        isprime.html
IsPrimeServlet.class  StringBean.java       manifest.tmp
IsPrimeServlet.java   StringBean.jsp        verifjsp.jsp
StringBean.class      WEB-INF

./WEB-INF:
classes

./WEB-INF/classes:
HelloWorld.class      IsPrimeServlet.class  cwp

./WEB-INF/classes/cwp:
StringBean.class
```

L'exemple ci-dessus contient deux servlets (HelloWorld.java et IsPrimeServlet.java), et un JavaBean (StringBean.java) appelé par une JSP (StringBean.jsp).

Les .class des servlets **doivent** être stockés dans le répertoire /racine/WEB-INF/classes . Un JavaBean situé dans le package "test" est compilé en un .class qui **doit** être stocké dans le répertoire /racine/WEB-INF/classes/test .

Une fois placé dans WEB-INF/classes le .class de la servlet va être appelée par l'URL : http ://sunserv.utc :8080/lo33/servlet/HelloWorld

Dans cette URL, "/lo33" désigne la racine de l'application installée dans le serveur Tomcat et "servlet/HelloWorld" peut être vu comme un nom logique (il n'y a aucun répertoire qui s'appelle "servlet") désignant racine/WEB-INF/classes/HelloWorld .

On pourra appeler la servlet par une URL de ce type dans un fichier html, comme par exemple, ci-dessous pour l'appel de IsPrimeServlet après avoir lu un argument dans une "form" html :

```
2009 lo33 sunserv:~/mv/servlet> more isprime.html
<HTML>
<HEAD><TITLE>servlet 2</TITLE></HEAD>
<BODY>
<H1>Test sur les nombres</H1>

<FORM Method="post"
      Action="http://sunserv.utc:8080/lo33/servlet/IsPrimeServlet">
Entrez un nombre:
<INPUT type=text size=10 name=number>
<INPUT type=submit value="Est-il premier?">
</FORM>
</BODY></HTML>
```

Compilation

Pour compiler une servlet, il faut ajouter au CLASSPATH l'emplacement de la bibliothèque (.jar) qui implémente l'API des servlets dans Tomcat :

```
setenv CLASSPATH \
   ".:/usr/local/JAKARTA/jakarta-servletapi-4/lib/servlet.jar"
compilation :
javac HelloWorld.java
```

Installation dans un répertoire privé

Normalement les applications sont installées dans le répertoire "webapps" de l'installation de Tomcat. Ceci pose un problème de droits d'accès au répertoire. Pour le contourner, on installe le produit dans un répertoire privé. Il faut que ce répertoire soit accessible à Tomcat et que les fichiers .class soient lisibles par Tomcat. Donc faire "chmod o+x" pour tous les répertoires et "chmod o+r" pour les .class et les JSP.

```
http://www4.utc.fr:8080/manager/install?
path=/lo33&war=file:/usr/uvs/lo33/lo33/mv/servlet

OK - Installed application at context path /lo33

si erreur : "cannot access directory"
chmod o+x mv/servlet
puis, recharger :
http://www4.utc.fr:8080/manager/reload?path=/lo33
OK - Reloaded application at context path /lo33

si on fait :

http://sunserv.utc:8080/manager/list
OK - Listed applications for virtual host localhost
/examples:running:0
/webdav:running:0
/lo33:running:1  <- - - - - - - - en plus
/tomcat-docs:running:0
/manager:running:0
/:running:0
```

Exécution d'une servlet

La servlet "HelloWorld.java" a été compilée, et le .class mis dans le répertoire **./WEB-INF/classes** situé sous la "racine" déclarée à Tomcat par la commande :
http ://www4.utc.fr :8080/manager/install ? donnée plus haut.

Quand on entre dans un navigateur l'URL http ://sunserv.utc :8080/lo33/servlet/HelloWorld Tomcat exécute dans sa machine virtuelle java le code java contenu dans :
./WEB-INF/classes/HelloWorld.class

Attention ! le mot "servlet" dans l'URL ci-dessus ne fait **pas** référence à un répertoire portant le nom "servlet". C'est un **marqueur logique**. Tomcat va le remplacer par le chemin réel qui lui a été donné dans la commande "install".

HelloWorld.java

```
 1  import java.io.*;
 2  import javax.servlet.*;
 3  import javax.servlet.http.*;
 4
 5  public class HelloWorld extends HttpServlet {
 6
 7      public void doGet(  HttpServletRequest request,
 8                          HttpServletResponse response)
 9      throws IOException, ServletException
10      {
11          response.setContentType("text/html");
12          PrintWriter out = response.getWriter();
13          out.println("<html>");
14          out.println("<body>");
15          out.println("<head>");
16          out.println("<title>Hello World!</title>");
17          out.println("</head>");
18          out.println("<body>");
19          out.println("<h1>Hello World!</h1>");
20          out.println("</body>");
21          out.println("</html>");
22      }
23  }
```

On remarque clairement dans le code de la servlet HelloWorld que celle-ci fabrique à la volée une page HTML en écrivant sur sa sortie standard, que Tomcat a connecté au socket TCP sur lequel est branché le navigateur client, le code source HTML décrivant la page.

17.3 Servlets et connexion à une base de données

Connexion à une base de donnée : Postgresql

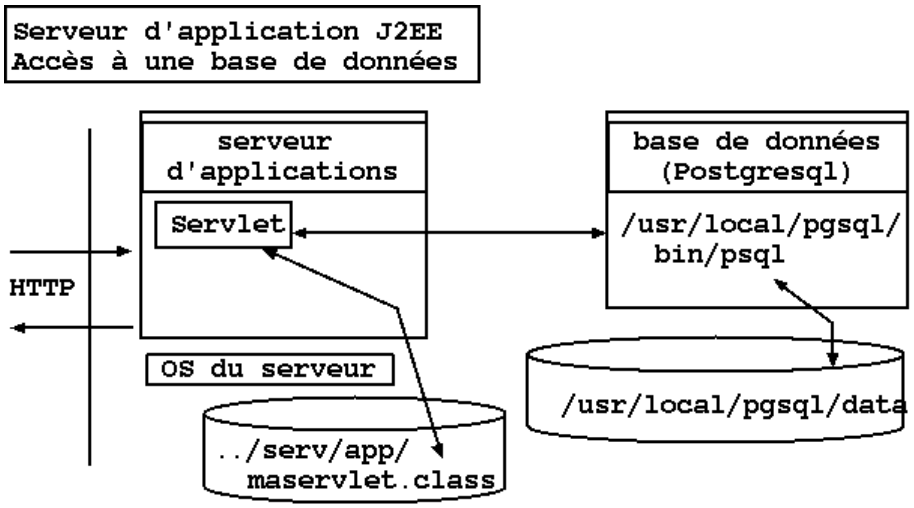


FIG. 174: Connexion à une base de données

Les gestionnaires de bases de données sont des systèmes complexes qui gèrent eux-mêmes leur espace disque et qui implantent un contrôle d'accès (Username/Password) propre à la base

de donnée. De plus sur une même installation, le programme gestionnaire gère **plusieurs** bases de données.

La connexion a une base de données va donc nécessiter un processus en trois étapes :

– Connexion au gestionnaire (MySQL, Oracle, PostGresql, Sybase, Informix, DB2),
– Choix de la base que l'on veut accéder,
– Identification : Username/Password <emph>de la base de données</emph>.

```
Exemple d'accès depuis un programme Java :

import java.sql.*;
..........
     username = "lo33xxx";
     password = "baselo33";
     // The URL to connect to the database
     // Syntax: jdbc:TYPE://machine:port/DB_NAME
     url = "jdbc:postgresql://sunserv:5432/baselo33";

     Connection conn;
     conn = DriverManager.getConnection(url, username, password);

Les insformations peuvent aussi être passées dans les arguments :
java ExecuteSQL -d postgresql.Driver \
-u lo33xxx -p yyyy jdbc:postgresql://sunserv:5432/baselo33
args parsed.
load driver. driver loaded.
open a connection.
connection opened.
sql> select * from auteur
+----------------------+----------------------------------------+
|        auteur        |                fichier                 |
+----------------------+----------------------------------------+
|  ........            | ......                                 |
```

Connexion à une base de donnée depuis la servlet

Le code montré ci-dessus utilise une URL pour l'accès à la base. Il est utilisable tel quel à l'intérieur de la servlet. Un exemple est donné dans les trois fichiers suivants.

La page html d'appel de la servlet

LanceRequete.html

```
 1  <html> <!---->
 2  <head><title>frasme de lance requete</title></head><body>
 3  <frameset cols="30%,*">
 4  <frame src = "FormLeft.html" name = "left">
 5  <frame src =
 6   "http ://sunserv.utc.fr :8080/lo33db/servlet/LanceRequete"
 7   name = "right">
 8  </frameset>
 9  </body></html> <!---->
```

FormLeft.html

```
 1  <html> <!---->
 2  <head><title>Lancer requete database</title></head>
 3  <body>
 4  <P>
 5  <form action=
 6    "http ://sunserv.utc.fr :8080/lo33db/servlet/LanceRequete"
 7    target="right">
 8  <input type=text size=30 name=requete>
 9  <br>
10  <input type=submit>
11  </form>
12  </body></html> <!---->
```

LanceRequete.java

```
 1  import java.io.*;
 2  import javax.servlet.*;
 3  import javax.servlet.http.*;
 4
 5  import java.sql.*;
 6  import java.io.*;
 7
 8  public class LanceRequete extends HttpServlet {
 9          String username;
10          String password;
11          String url;
12          String requete ="";
13          String nom;
14          int nbre;
15
16      public void doGet(  HttpServletRequest request,
17                          HttpServletResponse response)
18      throws IOException, ServletException
19      {
20          response.setContentType("text/html");
21          PrintWriter out = response.getWriter();
22          out.println("<html>");
23          out.println("<head>");
24          out.println("<title>Lance requete!</title>");
25          out.println("</head>");
26          out.println("<body>");
27
28          // ---- configure START
29          username = "lo17xxx";
30          password = "dblo17";
31          // The URL to connect to the database
32          // Syntax : jdbc :TYPE ://machine :port/DB_NAME
33          url = "jdbc :postgresql ://sunserv/dblo17";
```

LanceRequete.java (suite)

```
34          // ---- configure END
35
36          String requete;
37          requete = request.getParameter("requete");
38          if (requete!= null) {
39             // INSTALL/load the Driver (Vendor specific Code)
40             try {
41                 Class.forName("org.postgresql.Driver");
42                 } catch(java.lang.ClassNotFoundException e) {
43                 System.err.print("ClassNotFoundException : ");
44                 System.err.println(e.getMessage());
45                 }
46             try {
47                 Connection con;
48                 Statement stmt;
49                 // Establish Connection to the database at URL
50                 // with usename and password
51                 con = DriverManager.getConnection(url,
52                                                   username, password);
53                 stmt = con.createStatement();
54                 // Send the query and bind to the result set
55                 ResultSet rs = stmt.executeQuery(requete);
56                 ResultSetMetaData rsmd=rs.getMetaData();
57                 nbre=rsmd.getColumnCount();
58                 while (rs.next()) {
59                         for (int i=1; i<=nbre;i++){
60                         nom = rsmd.getColumnName(i);
61                         String s = rs.getString(nom);
62                         out.print(s);
63                         }
64                 out.print("<p>");
65                 }
66             out.println("</body>");
67             out.println("</html>");
68             // Close resources
69             stmt.close();
70             con.close();
71             }//try
72             // print out decent erreur messages
73             catch(SQLException ex) {
74                 System.err.println("==> SQLException : ");
75                 while (ex!= null) {
76                         System.out.println("Message :   " +\
77                                            ex.getMessage ());
78                         System.out.println("SQLState :  " +\
79                                            ex.getSQLState ());
80                         System.out.println("ErrorCode : " +\
81                                            ex.getErrorCode ());
82                         ex = ex.getNextException();
83                         System.out.println("");
84                         }
85             }//catch
```

LanceRequete.java (suite)

```
86                  }
87          }
88  }
```

17.4 Servlets, JSP et JavaBeans : interactions

Presque toutes les interactions sont possibles (sinon faciles) entre les servlets, les JSP et les JavaBeans :

- une page html appelle une servlet,
- une servlet propage une requête vers une JSP,
- une JSP appelle une servlet,
- une servlet instancie un JavaBean et lui passe de l'informations ;
- une JSP appelle un JavaBean pour lui demander un calcul,
- une JSP appelle un JavaBean déja instancié par une servlet et lui demande des informations que la servlet lui a passé,
- ...

Les spécifications des servlets et des JSP prévoient les API et les syntaxes pour réaliser toutes ces opérations.

Connexion d'une Servlet à une JSP

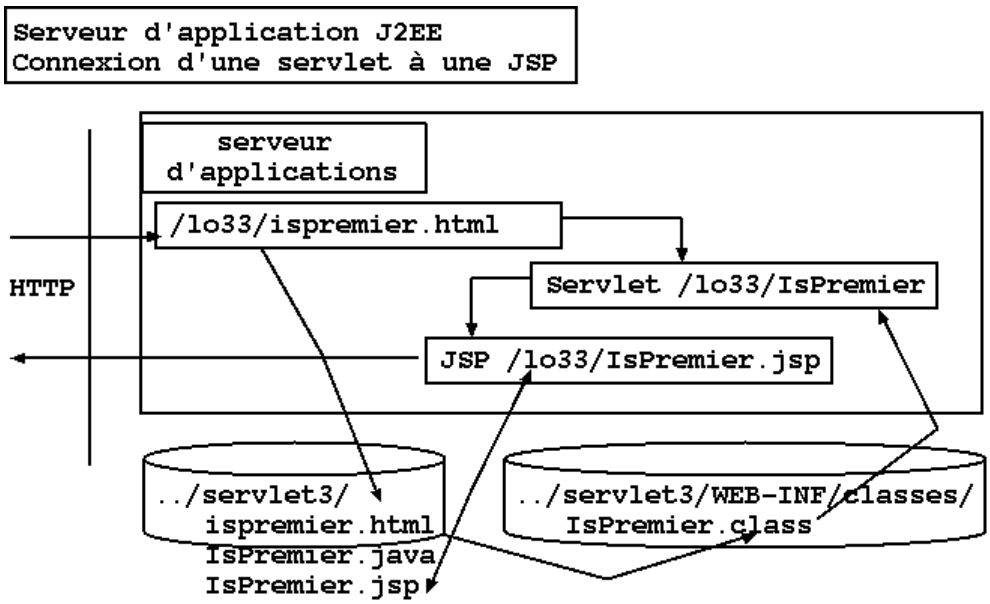


FIG. 175: Connexion d'une Servlet à une JSP

```
lo33 sunserv:~/servlet3> ls ispremier.html IsPremier.*
IsPremier.java  IsPremier.jsp   ispremier.html
lo33 sunserv:~/servlet3> echo $CLASSPATH
.:/usr/local/JAKARTA/jakarta-servletapi-4/lib/servlet.jar
lo33 sunserv:~/servlet3> javac IsPremier.java
lo33 sunserv:~/servlet3> mv IsPremier.class WEB-INF/classes/
```

```
http://www4.utc.fr:8080/manager/install?
path=/lo33&war=file:/usr/uvs/lo33/lo33/mv/servlet3
ou
http://www4.utc.fr:8080/manager/reload?path=/lo33
puis :
http://www4.utc.fr:8080/lo33/ispremier.html
==> affichage de la forme, saisie du nombre, envoi

Page JSP IsPremier.jsp appelée par servlet IsPremier.java

Nombre demandé était : 17 Ce nombre est premier.
```

ispremier.html

```
 1  <HTML><!---->
 2  <HEAD><TITLE>servlet 3 et JSP</TITLE></HEAD>
 3  <BODY>
 4  <H1>Test sur les nombres</H1>
 5
 6  <FORM Method="post"
 7   Action="http ://sunserv.utc :8080/lo33/servlet/IsPremier">
 8  Entrez un nombre :
 9  <INPUT type=text size=20 name="lenombre">
10  <INPUT type=submit value="Est-il premier?">
11  </FORM>
12
13  </BODY></HTML><!---->
```

IsPremier.java

```
 1  import java.util.*;
 2  import java.io.*;
 3  import javax.servlet.*;
 4  import javax.servlet.http.*;
 5
 6  public class IsPremier extends HttpServlet
 7  {
 8
 9  public void service(    HttpServletRequest req,
10                          HttpServletResponse res)
11      throws ServletException, IOException
12      {
13          String val;
14          int number = Integer.parseInt(
15                          (req.getParameterValues("lenombre"))[0]);
16          boolean prime = isPrime(number);
17          if (prime) {
18                  val = new String ("est premier");
19          } else {
20                  val = new String ("n'est pas premier");
```

IsPremier.java (suite)

```
21          }
22          req.setAttribute("estil", val);
23
24          gotoPage("/IsPremier.jsp", req, res);
25
26      }
27      // utility routine forwarding request
28      private void gotoPage( String url,
29                             HttpServletRequest request,
30                             HttpServletResponse response)
31      throws IOException, ServletException
32      {
33          RequestDispatcher dispatcher =
34                   getServletContext().getRequestDispatcher(url);
35          dispatcher.forward (request, response);
36      }
37
38
39  static boolean isPrime(int i)
40     {
41          int j;
42
43          if ((i<=0) || (i==1)) return false;
44          for (j = 2; j < i; j++)
45                  if ((i % j) == 0)
46                          return false;
47          return true;
48     }
49
50  public String getServletInfo()
51     {
52          return "A servlet to test for primeness";
53     }
54
55  }
```

IsPremier.jsp

```
 1  <!---->
 2  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
 3  <!--
 4  IsPremier.jsp : example de jsp appelée par une servlet
 5  -->
 6  <HTML>
 7  <HEAD><TITLE>Using JSP from Servlets</TITLE></HEAD>
 8  <BODY>
 9  <h3>Page JSP IsPremier.jsp appelée par servlet
10      IsPremier.java</h3>
11  <p>
12  Nombre demandé était : <%= request.getParameter("lenombre") %>
13  <br/>
```

IsPremier.jsp (suite)

```
14  Ce nombre <%= request.getAttribute("estil") %>.
15  <br/>
16  </p>
17  <hr><br/>
18  </body></html><!---->
```

Pour pouvoir appeler une page JSP depuis une servlet, on permet à la servlet de renvoyer la requête ("forward request") en utilisant un "RequestDispatcher".

On obtient un ré-émetteur en appelant la méthode "getRequestDispatcher" du "ServletContext" en lui fournissant en argument un URL relatif à la racine du serveur.

Exemple : String url = "/mespages/mapagejsp.jsp" ;
RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(url) ;

Une fois que l'on a un "dispatcher", on utilise sa méthode "forward()" pour transférer complètement le contrôle à l'URL associée à ce dispatcher : dispatcher.forward (request, response) ;

Ici request et response sont des variables contenant HttpServletRequest et HttpServletResponse.

On peut aussi **inclure** dans la réponse de la servlet la réponse produite par la page JSP "appelée", puis continuer le traitement dans la servlet : dispatcher.include (request, response) ;

Ces deux fonctions renvoient ServletException et IOException.

Passage d'information depuis la servlet vers la page JSP

1. passer l'info par des attributs de la requête :
   passer (depuis la servlet : req.setAttribute("estil", val) ;
   récupérer (dans la jsp) : <
2. passer par l'utilisation d'un Bean présent dans la dir de la servlet, ce Bean étant ensuite utilisé par la JSP.

Passage d'information par l'utilisation d'un Bean : il y a 3 méthodes : par HttpServletRequest, HttpSession ou ServletContext :

1. **HttpServletRequest :** data utilisables par la JSP seulement dans CETTE requête :

```
    UneClasse value = new UneClasse(...);
    request.setAttribute("key",value);
la page JSP récupère l'info par :
    <jsp:useBean id="key" class="UneClasse" scope="request" />
```

2. **HttpSession :** data utilisables par la JSP dans CETTE requête ET dans les requêtes suivantes du même client :

```
    UneClasse value = new UneClasse(...);
    HttpSession session = request.getSession(true);
    session.setAttribute("key",value);
la page JSP récupère l'info par :
    <jsp:useBean id="key" class="UneClasse" scope="session" />
```

3. **ServletContext :** data utilisables par la JSP dans cette requête ET dans les requêtes suivantes de n'importe quel client :

```
    UneClasse value = new UneClasse(...);
    getServletContext().setAttribute("key",value);
la page JSP récupère l'info par :
    <jsp:useBean id="key" class="UneClasse" scope="application" />
```

17.5 Exemple d'interaction html -> Servlets -> JavaBeans -> JSP -> JavaBeans

Cet exemple va montrer une interaction entre les servlets, les JSP et les JavaBeans : une page html (1)(Prem.html) appelle une servlet (2)(Prem.java) qui instancie un JavaBean (3)(Premier.java), lui passe un argument, puis renvoie la requête vers une JSP (4)(Premi.jsp) ; cette JSP appelle (5) le JavaBean instancié précédemment par la servlet, lui demande (5) des paramètres puis envoie (6) la page html formattée au navigateur.

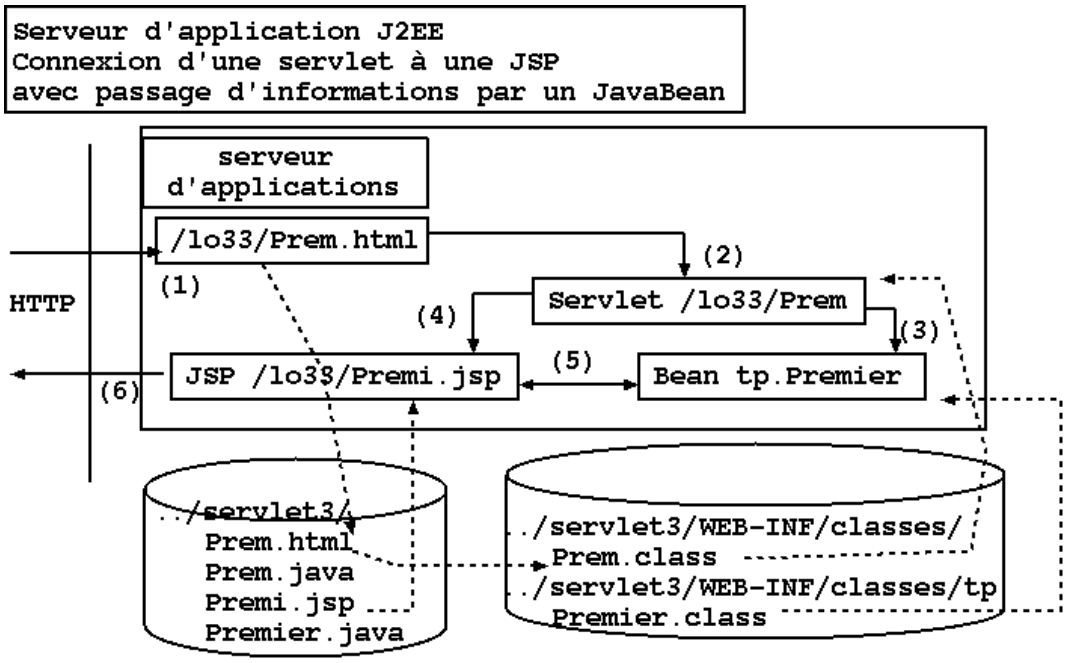


FIG. 176: Interactions Servlet, JSP, JavaBean

Prem.html

```
 1  <HTML><!---->
 2  <HEAD><TITLE>servlet 3 , bean et JSP</TITLE></HEAD>
 3  <BODY>
 4  <H1>Test sur les nombres</H1>
 5
 6  <FORM Method="post"
 7   Action="http ://sunserv.utc :8080/lo33/servlet/Prem">
 8  Entrez un nombre :
 9  <INPUT type=text size=10 name="lenombre">
10  <INPUT type=submit value="Est-il premier? ">
11  </FORM>
12
13  </BODY></HTML><!---->
```

Prem.java

```
 1  import tp.Premier;
 2
 3  import java.util.*;
 4  import java.io.*;
 5  import javax.servlet.*;
 6  import javax.servlet.http.*;
 7
 8  public class Prem extends HttpServlet
 9  {
10
11  public void service(    HttpServletRequest req,
12                          HttpServletResponse res)
13          throws ServletException, IOException
14          {
15
16  //      int number = Integer.parseInt(
17  //                      (req.getParameterValues("lenombre"))[0]);
18
19          Premier val = new Premier ();
20  //      val.setLeNum(number);
21  //      val.setUnAutre(123);
22          val.setLeNum((req.getParameterValues("lenombre"))[0]);
23
24          req.setAttribute("estil", val);
25
26          gotoPage("/Premi.jsp", req, res);
27
28          }
29      // utility routine forwarding request
30      private void gotoPage( String url,
31                             HttpServletRequest request,
32                             HttpServletResponse response)
33      throws IOException, ServletException
34      {
35          RequestDispatcher dispatcher =
36            getServletContext().getRequestDispatcher(url);
37          dispatcher.forward (request, response);
38      }
39
40  public String getServletInfo()
41          {
42          return "A servlet to test for primeness";
43          }
44
45  }
```

Premier.java

```
 1  package tp;
 2  /** A simple bean been called by a servlet
```

Premier.java (suite)

```
 3   *  store a value and give it to a calling JSP
 4   */
 5
 6  public class Premier {
 7    private int num = 0;
 8    private boolean is;
 9    private String is0 = "n'est pas premier";
10    private String is1 = "est premier";
11    private String isPrem = is0;
12    private String leNum;
13    private int unAutre = 0;
14
15  // a bean class must have a zéro-arg constructor,
16  // or no constructor; a bean class should have
17  // no public instance variables
18  // persitant values should be access with methods setXyz, getXyz
19
20    public void setLeNum(String n) {
21      // store init val in local storage
22      this.num = Integer.parseInt(n);
23      // calculate and keep result
24      this.is = isPrime(this.num);
25      isPrem=is0;
26      if (this.is) isPrem=is1;
27    }
28
29    public String getLeNum() {
30      this.leNum = ""+this.num;
31      return(this.leNum);
32    }
33
34    public String getIsPrem() {
35      return(this.isPrem);
36    }
37
38  static boolean isPrime(int i)
39          { int j;
40          if ((i<=0) || (i==1)) return false;
41          for (j = 2; j < i; j++)
42                  if ((i % j) == 0)
43                          return false;
44          return true;
45          }
46
47  }//end class Premier
```

Premi.jsp

```
1  <!---->
2  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
3  <!--
```

Premi.jsp (suite)

```
 4  Premi.jsp : exemple de jsp appelée par une servlet
 5  -->
 6  <HTML>
 7  <HEAD>
 8  <TITLE>Using JSP from Servlets</TITLE>
 9  </HEAD>
10  <BODY>
11  <h3>Page JSP Premi.jsp appelée par
12  servlet Prem.java</h3>
13  <p>
14  Nombre demandé était :
15  <%= request.getParameter("lenombre") %>
16  <br/>
17  Ce nombre <%= request.getAttribute("estil") %>.
18  <br/>
19  Appel du bean "Premier.java" du package "tp".
20  <br/>
21  <jsp :useBean id="estil"
22          class="tp.Premier" scope="request" />
23  Le nombre
24  <jsp :getProperty name="estil" property="leNum" />
25  <jsp :getProperty name="estil" property="isPrem" />
26  <br/>
27  Change
28  <jsp :setProperty name="estil"
29          property="leNum" value='<%= "5" %>' />
30  <br/>
31  New =  <jsp :getProperty name="estil" property="leNum" />
32  <jsp :getProperty name="estil" property="isPrem" />
33  </p>
34  <hr>
35  <br/>
36  </body></html><!---->
```

À la fin de la JSP, on a montré un exemple de modification d'un paramètre du Bean par l'accès depuis la JSP aux fonctions du Bean (<jsp :setProperty ..).

L'exécution donne :

```
Nombre demandé était : 3
Ce nombre tp.Premier@4845aa.
Appel du bean "Premier.java" du package "tp".
Le nombre 3 est premier
Change
New = 5 est premier
```

SR03 2004 - Cours Architectures Internet - Corba et les applications et objets distribués
©Michel.Vayssade@utc.fr – Université de Technologie de Compiègne.

18 SR03 2004 - Cours Architectures Internet - Corba et les applications et objets distribués

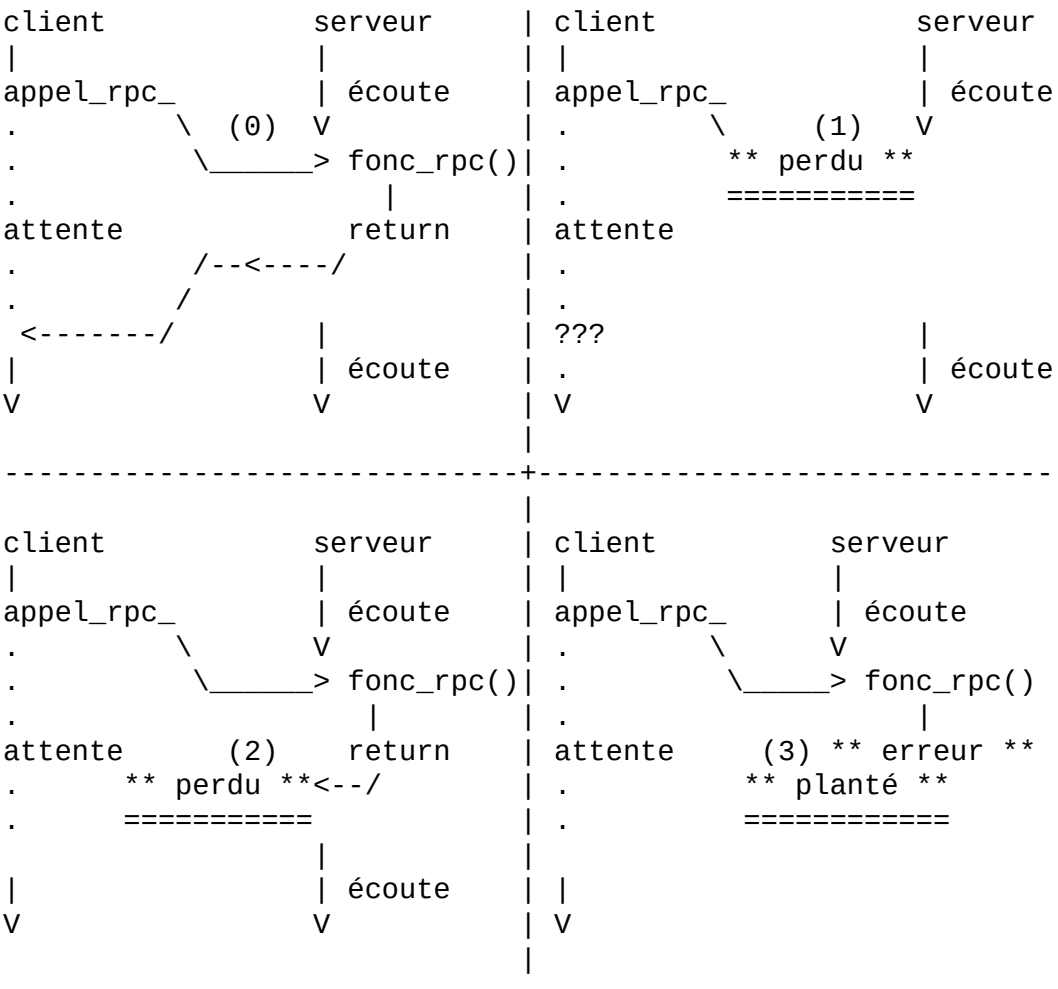
18.1 CORBA et les applications réparties

Généralités sur les applications réparties

Le premier type d'applications réparties est le RPC. Il y a plusieurs modes d'invocation du RPC :

- **synchrone :** c'est le RPC classique vu auparavant : le client est bloqué jusqu'à reception des résultats renvoyés par le serveur.
- **synchrone différé :** le client lance l'appel, puis continue ses traitements ; Il demande ensuite périodiquement à la souche si le serveur a envoyé les résultats.
- **asynchrone :** le client lance l'appel et poursuit son exécution. Une routine d'interruption est exécutée quand les données de retour du serveur parviennent sur le site du client.

Applications réparties et fiabilité



Un appel de RPC peut échouer pour diverses raisons. Dans ce cas, il peut être difficile de décider ce qui doit être fait. Le schéma ci-dessus montre différentes façons de se dérouler pour un appel RPC :

- 0 - appel réussi
- 1 - message "aller" perdu
- 2 - message "retour" perdu
- 3 - plantage dans le serveur

Si l'appel RPC déclenche un traitement côté serveur ayant une action sur une base de données :

- Pour (1) il faut refaire la demande ;
- Pour (2) il ne faut SURTOUT PAS la refaire,
- Pour (3) on est soit dans un cas de type (1) soit dans un cas de type (2) en fonction de l'endroit du plantage.

Dans ce cas (modif d'une base), il faut recourir à des mécanismes de type "transaction" permettant de passer d'un état initial cohérent à un état final cohérent. Si une erreur survient on pourra annuler la transaction et revenir à l'état cohérent de départ avant de réessayer.

CORBA et les applications réparties

Corba est une spécification d'un "bus à requêtes".

C'est une architecture **complexe** car elle permet d'assurer la portabilité, et l'interopérabilité entre environnements de fournisseurs différents.

Mais aussi de permettre l'écriture d'applications hétérogènes : invoquer depuis du code C++ par exemple des traitements effectués en Java sur la même machine ou sur une autre machine. Ceci en pouvant changer le client ou le serveur de machine, sans avoir à recompiler.

Le bus CORBA, s'appelle un ORB : Object Request Broquer.

Le mécanisme de base de CORBA est le RPC (pas LE RPC ONC de SUN ni le RPC DCE, mais un appel à distance ayant le même type de fonctionnement).

On a donc dans Corba :

- un fichier de description d'interface en IDL,
- un compilateur d'IDL qui va créer les amorces client et serveur (dans Corba l'amorce client s'appelle une souche et l'amorce serveur s'appelle un squelette),
- un fichier source du code client,
- un fichier source d'implémentation des objets serveurs,
- un fichier source du code du serveur.

En pratique, l'ORB se présente commme un bibliothèque qui va être linkée avec le client et avec le serveur et qui va effectuer les opérations de transcodage (marshalling) des arguments de l'appel, d'envoi des données sur un socket internet (après connexion au serveur), de récupération des résultats, etc ...).

La force de Corba est d'avoir définit des spécification précises (+ de 1000 pages) de tous les formats d'échange de données. C'est ceci qui permet d'invoquer un objet sur un serveur utilisant l'ORB du fournisseur "A" depuis un code linké avec l'ORB du fournisseur "B". Le client et le serveur pouvant se trouver sur des machines différentes (big/little endian, 32/64 bits) et être écrits dans des langages différents.

18.2 CORBA : Common Object Request Broquer Architecture

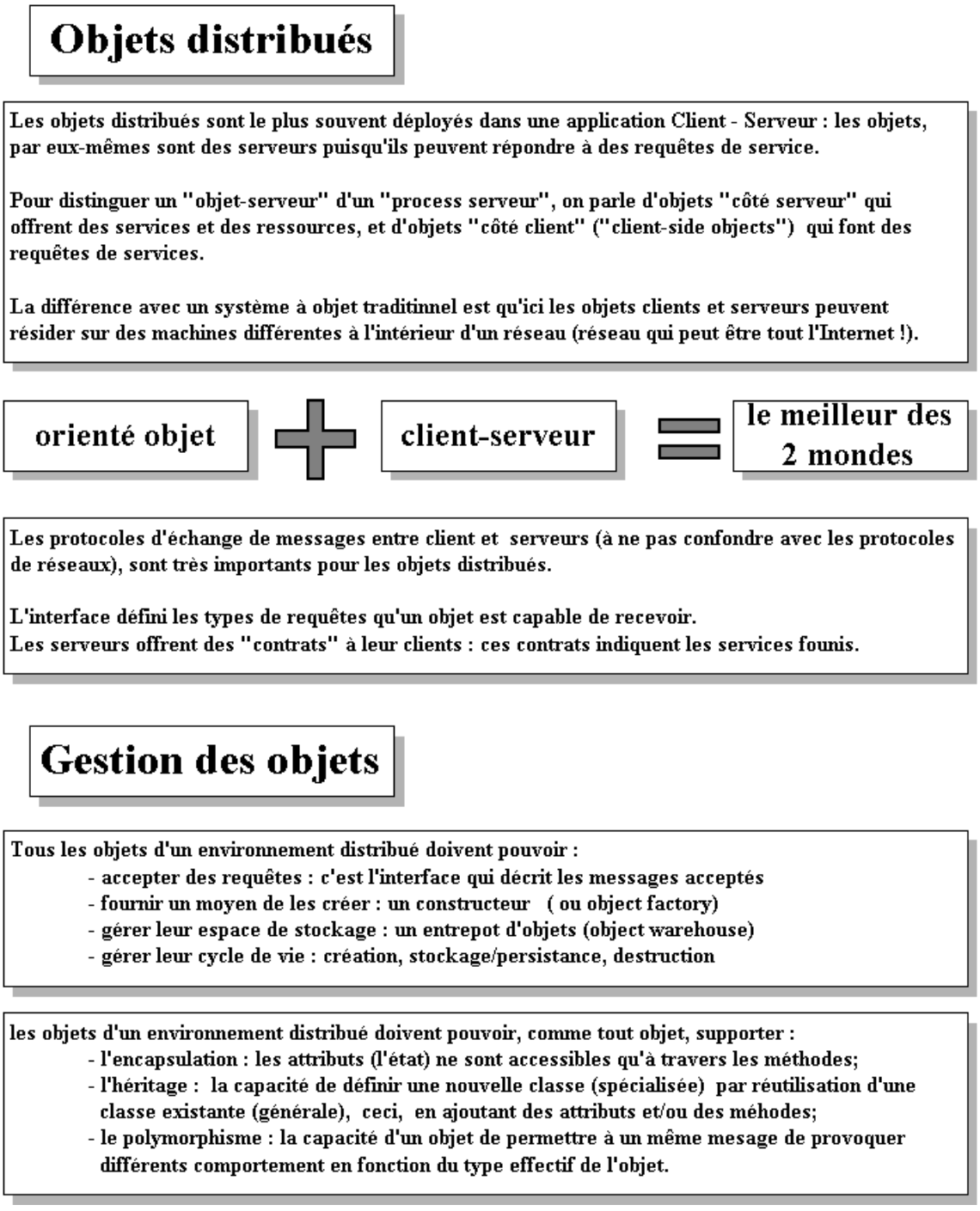


FIG. 177: Objets distribués

L'OMG : Object Management Group

L'OMG est un consortium fondé en 1989 pour promouvoir l'adoption de standards pour gérer des objets distribués.

OMG --> développé OMA Object Management Architecture
sur OMA --> développé un modèle de référence

Aujourd'hui l'OMG = 500 sociétés membres
Il a une bonne réputation pour adopter très vite des spécifications.

Un standard ne peut être adopté par l'OMG QUE s'il en existe une implémentation.
Si c'est adopté, c'est que quelqu'un quelque part l'a construit et a montré que "ça marche".

OMA : Object Management Architecture

OMA, Object Management Architecture, est constitué de quatre parties :
- un composant essentiel appelé ORB : Object Request Broker (mandataire de requêtes)
- des services pour développer les objest distribués : CORBAservices
- des services destinés à être utilisés par les objets des applications distribuées : CORBAfacilities
- les applications distribuées elles-mêmes;

L'ensemble, adopté comme standard, est appelé CORBA :
Common Object Request Broker Architecture

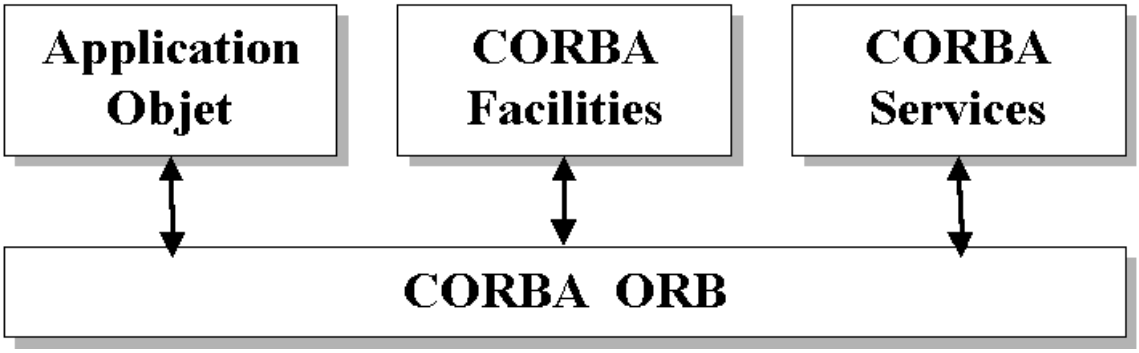


FIG. 178: L'OMG

Interface et implémentation

L'interface d'un objet spécifie son comportement. Elle déclare les services que l'objet est prêt à rendre. Ceci cache les détails d'implémentation grâc à l'encapsulation.
=> donc, tant que l'objet a le comportement spécifié par son interface, son implémentation peut changer. Ce principe est fondamental pour CORBA.

L'OMA adhère au modèle objet "classique": c'est-à-dire que toutes les méthodes sont contenues dans une classe. Dans CORBA les objets sont connus exclusivement par leur interface, sans rien savoir de leur implémentation.

Pour cela, CORBA inclut un langage IDL (Interface Definition Language) qui permet de décrire les interfaces de tous les objets.

L'interface est l'équivalent CORBA d'une classe.
Les requêtes à un objet CORBA sont toujours relayées à travers une référence à un objet. Toutes les invocations sont faites sur la réference d'un objet "ObjRef".
IDL n'est PAS un langage de programmation: c'est un langage pour exprimer des types, en particulier des types "interface". Il n'a ni construction de contrôle, ni itérateurs.

Les types IDL

types de base	boolean , char , octet , enum , short , unsigned short , long , unsigned long , float , double , any , object
types construits	string , struct , union , array , sequence

```
interface      Person {
      readonly attribute Gender sex;
      readonly attribute Date birthdate;
      attribute      string  name;
};
```

```
interface      Employee : Person {
      readonly attribute ssn;

      void addEmployer( Employer emp );
      void deleteEmployer( Employer emp );
      short numEmployers( );
      Employer Employer( short index ) throws exOutofBounds;
};
```

FIG. 179: L'IDL

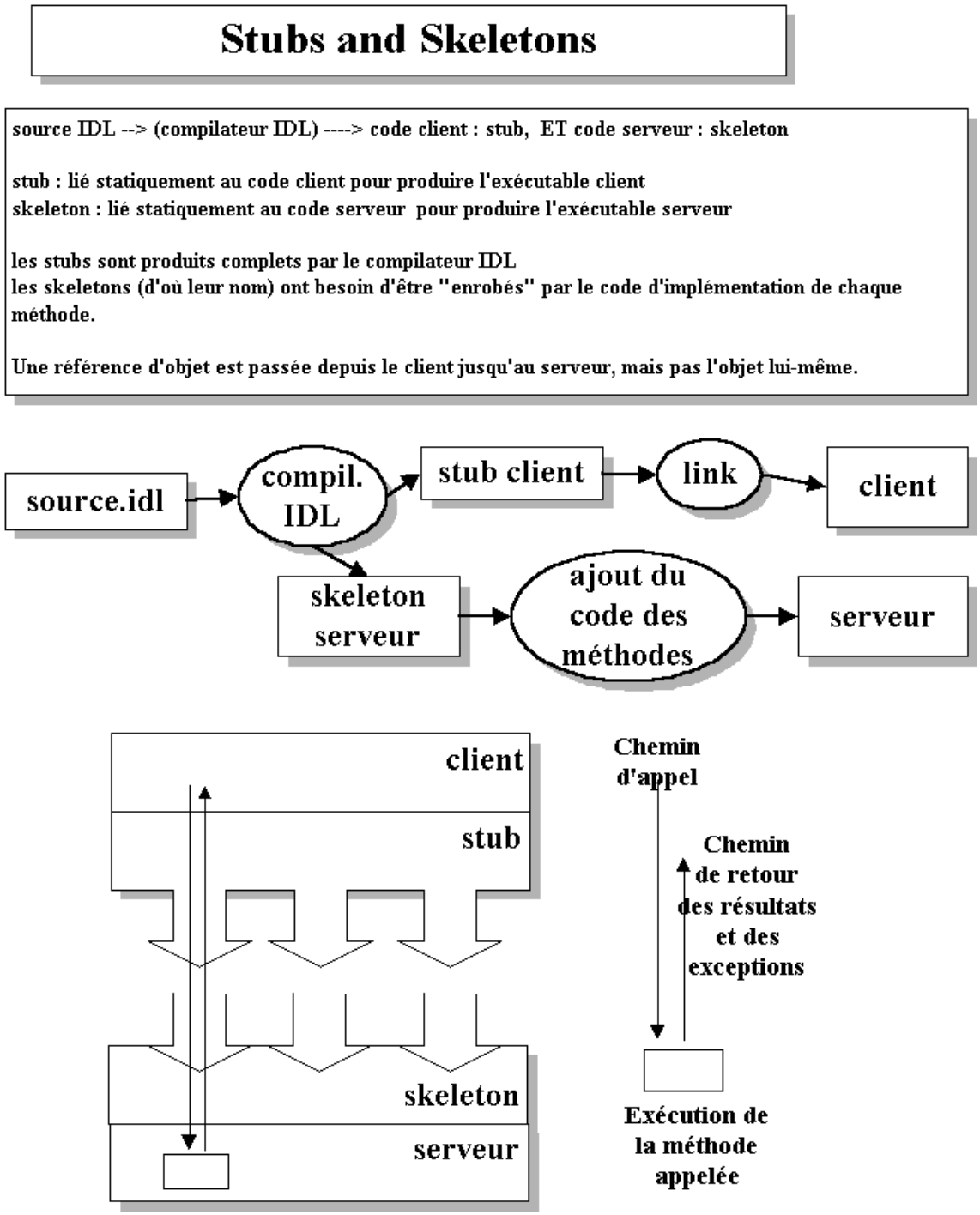


FIG. 180: Stubs et Skeletons

CORBA
Common Object Broker Architecture

les stubs constituent le mécanisme d'invocation d'un objet, côté client;
les squeltons l'interface côté serveur recevant les requêtes;

l'architecture spécifie que les objets CORBA sont accessibles sans spécifier leur localisation (location transparents); ==> l'implémentation peut être dans le même process, dans un autre process ou dans une autre machine; CORBA prends en charge le transport des requêtes aux objets serveurs quelque soit l'endroit où il est situé.

L'interface ORB fournit des services (supposé largement disponibles) aussi bien qux clients qu'aux serveurs.

Le BOA ("Basic Object Adapter") supporte différents styles d'implémentation des objets.

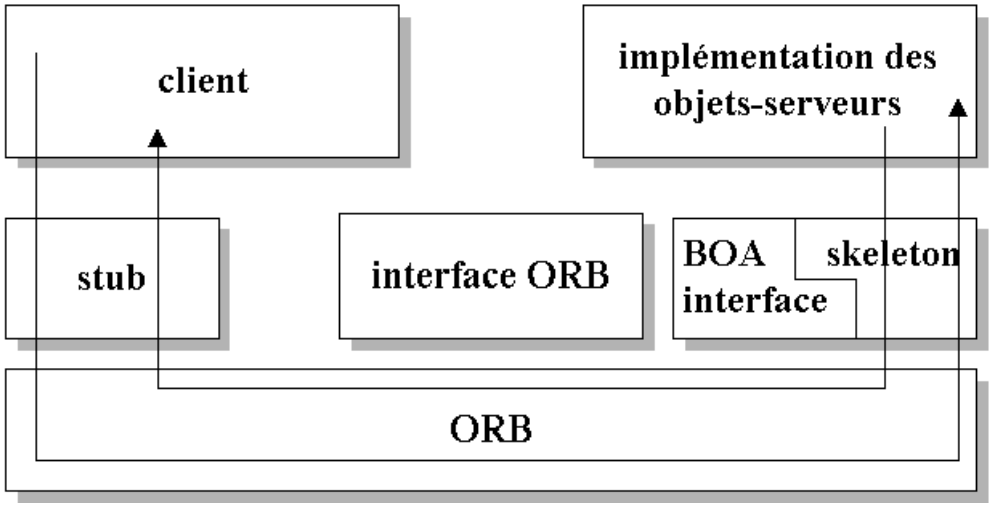


FIG. 181: L'ORB

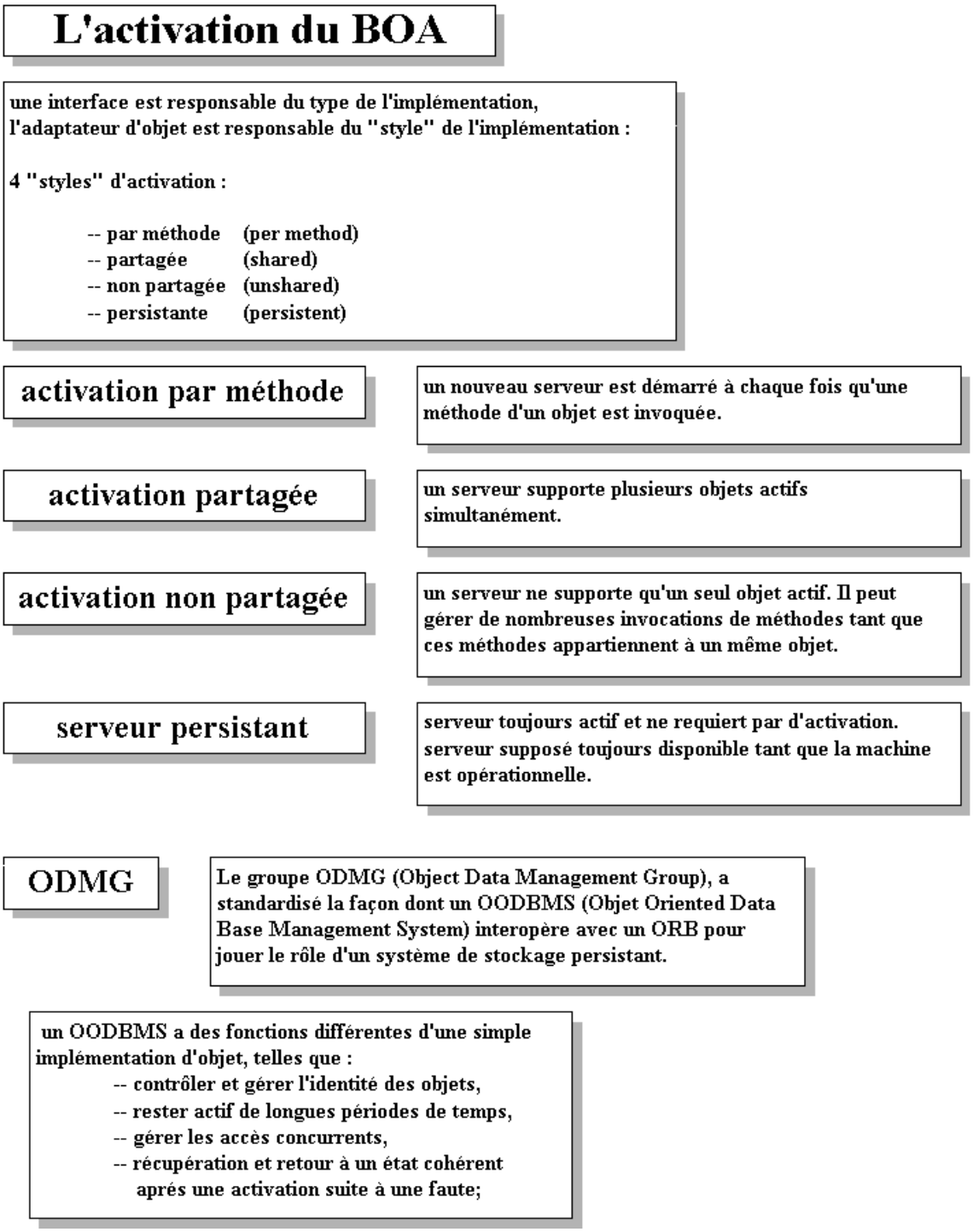


FIG. 182: Le BOA

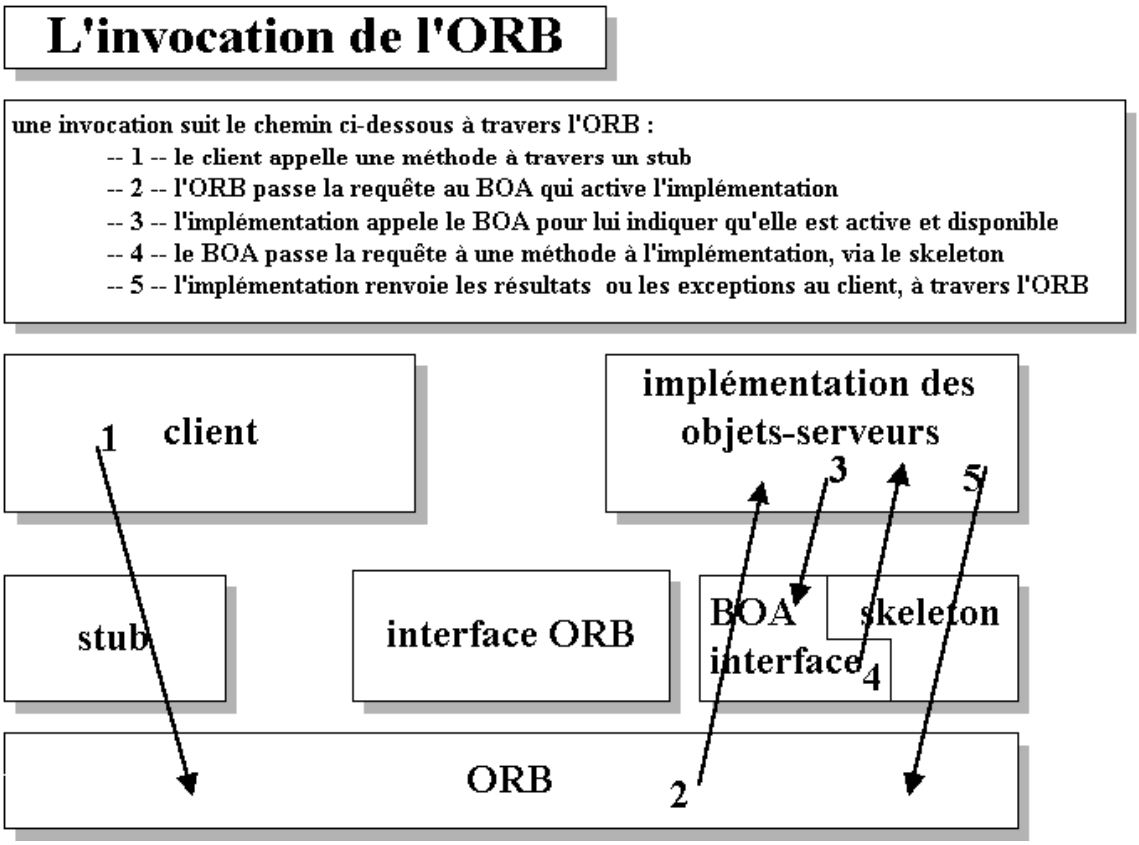
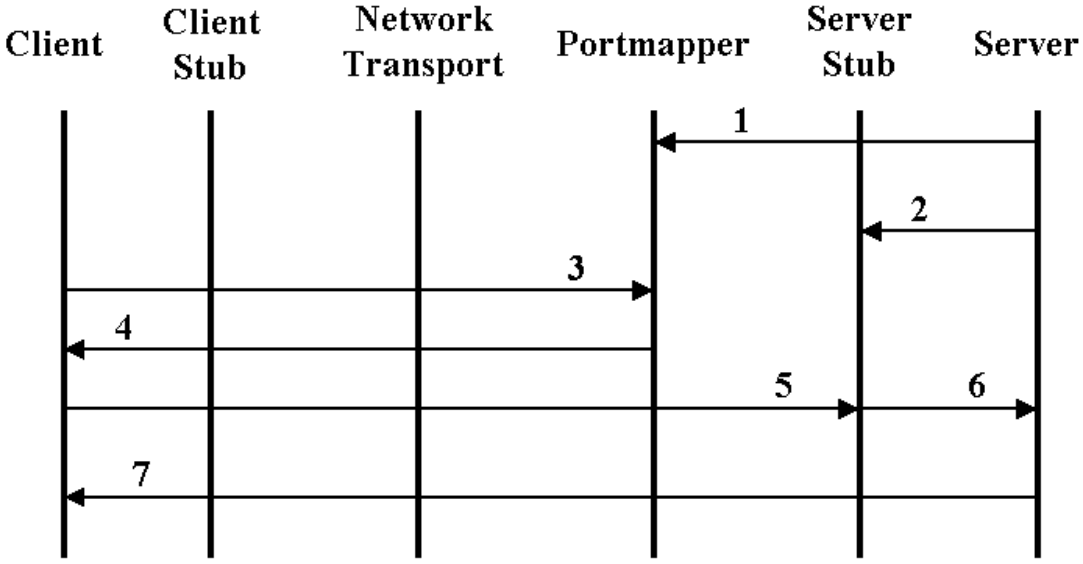


FIG. 183: Invocation de l'ORB

Rappel : Le RPC - Remote Procedure Call

Une invocation d'ORBest très proche de la technique du RPC: le RPC cache le réseau en déguisant l'appel distant comme un appel local. Un compilateur est utilisé pour transformer la spécification de l'interface de la procédure en un adaptateur (stub) côté client et un autre côté serveur.
La ressemblance s'arrête là : le RPC déguise des procédures, pas des objets.
De plus, avec le RPC, il faut connaître l'adresse de la machine et le numéro du port sur lequel la procédure du serveur est à l'écoute.



(1) le serveur enregistre son numéro auprès du portmapper
(2) le serveur se met en lecture bloquante sur son port
(3) le client demande le numéro de port du serveur au portmapper de la machine du serveur
(4) le portmapper répond en donnant le numéro de port du serveur
(5) le client contacte le serveur grâce au numéro d'hôte et au numéro de port
(6) le serveur sort du sa lecture bloquante
(7) le serveur renvoi des résultats ou une exception

CORBA a pour but d'encapsuler les services dans des objets et de fournir plus de facilité pour l'activation aussi bien des serveurs que des objets. Il élimine le besoin d'utiliser des RPCs.

FIG. 184: ORB et RPC

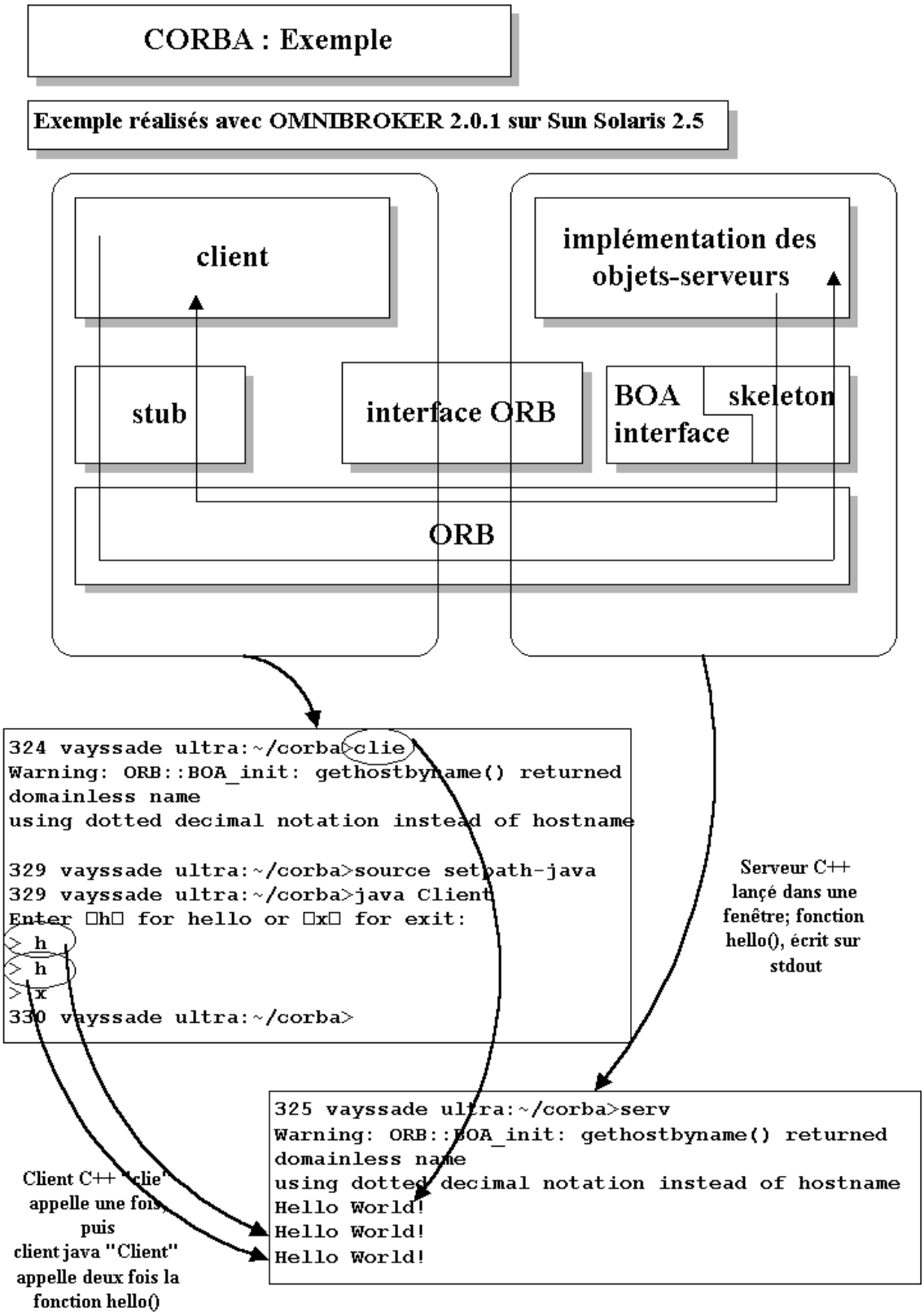


FIG. 185: Corba : exemple

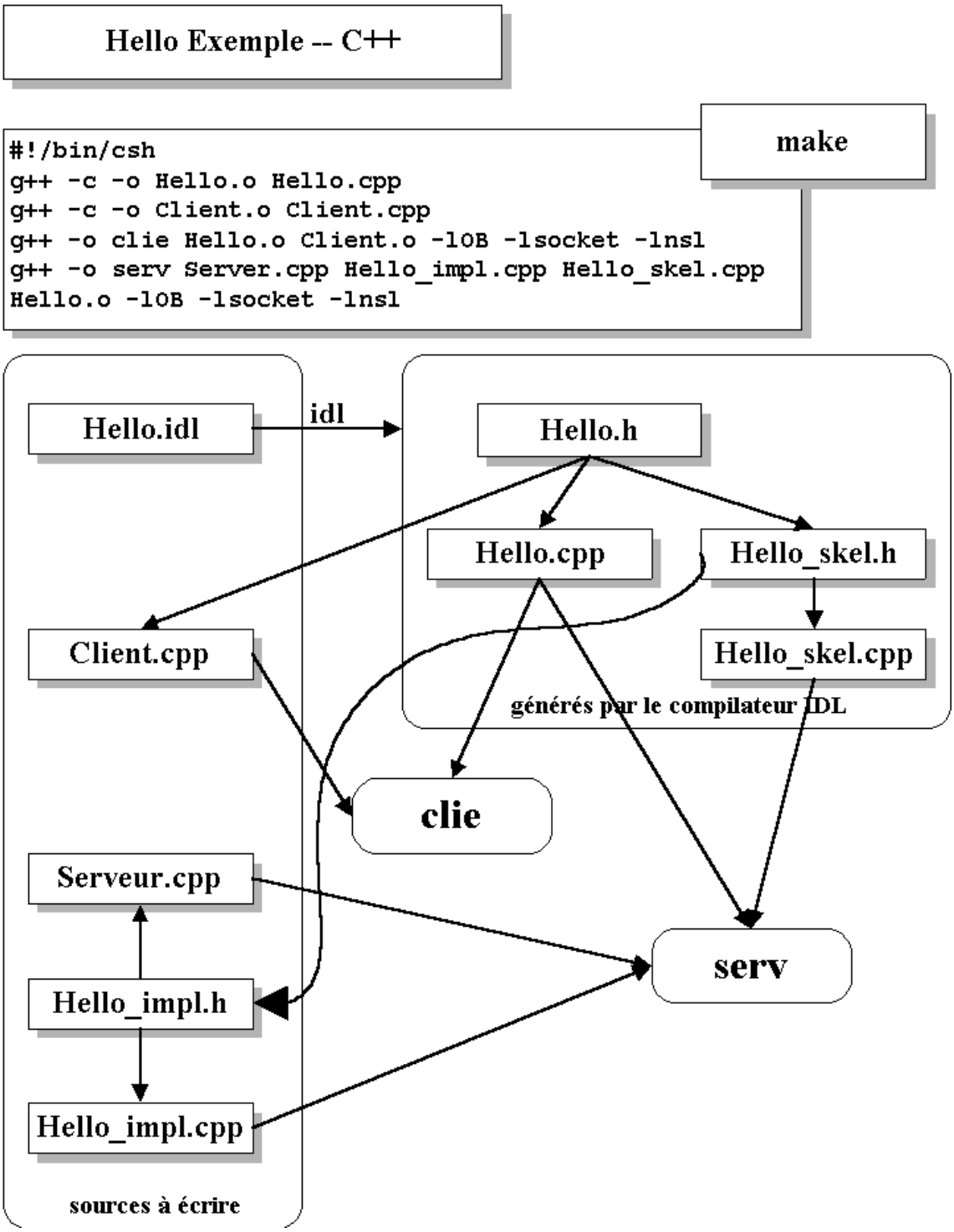


FIG. 186: Corba : exemple

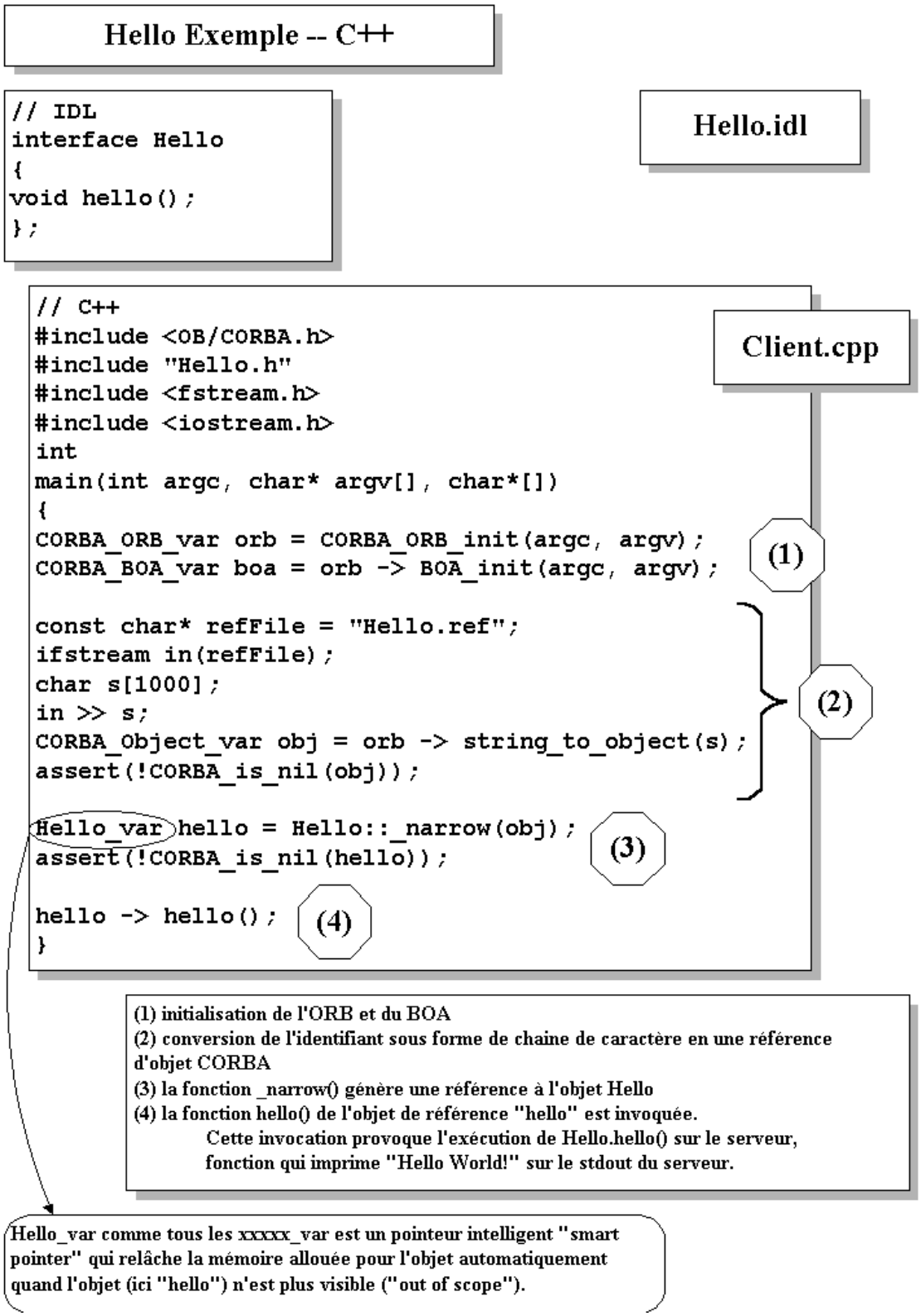
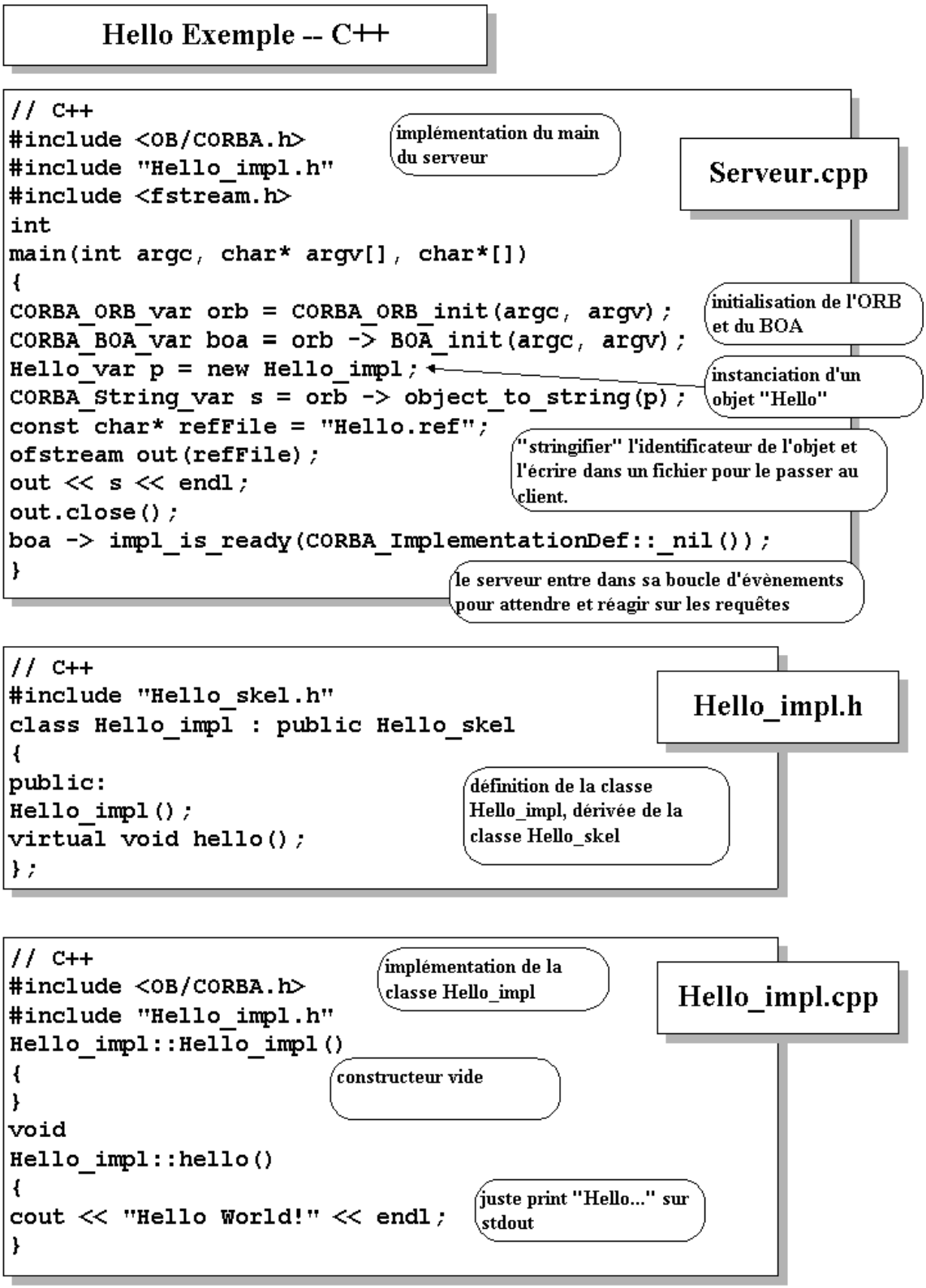


FIG. 187: Corba : exemple




```
// C++
#include <OB/CORBA.h>
#include "Hello_impl.h"
#include <fstream.h>
int
main(int argc, char* argv[], char*[])
{
CORBA_ORB_var orb = CORBA_ORB_init(argc, argv);
CORBA_BOA_var boa = orb -> BOA_init(argc, argv);
Hello_var p = new Hello_impl;
CORBA_String_var s = orb -> object_to_string(p);
const char* refFile = "Hello.ref";
ofstream out(refFile);
out << s << endl;
out.close();
boa -> impl_is_ready(CORBA_ImplementationDef::_nil());
}
```

```
// C++
#include "Hello_skel.h"
class Hello_impl : public Hello_skel
{
public:
Hello_impl();
virtual void hello();
};
```

```
// C++
#include <OB/CORBA.h>
#include "Hello_impl.h"
Hello_impl::Hello_impl()
{
}
void
Hello_impl::hello()
{
cout << "Hello World!" << endl;
}
```

FIG. 188: Corba : exemple

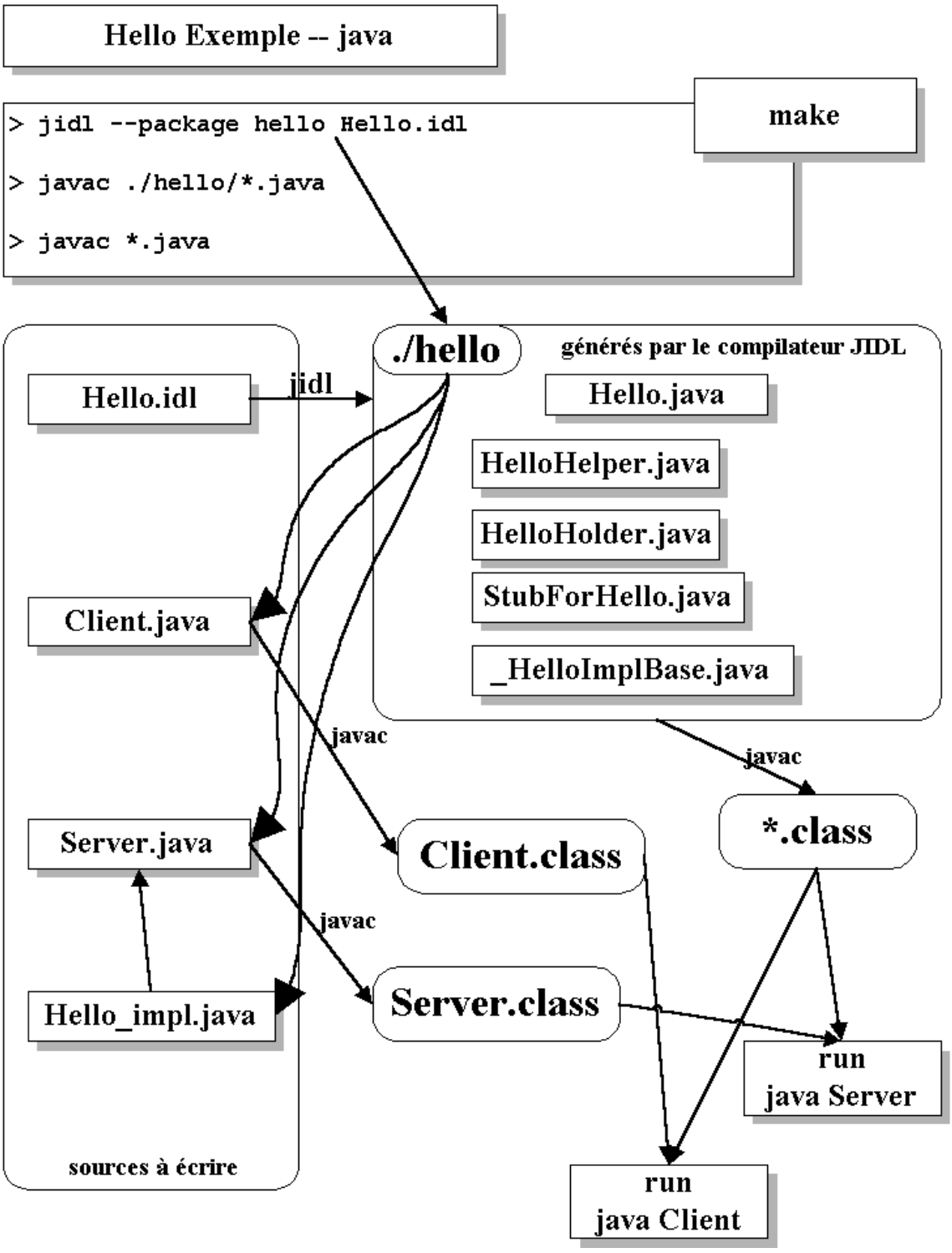


FIG. 189: Corba : exemple

Hello Exemple -- java

```
303 vayssade ultra:~/corba>java Server
java.lang.NoClassDefFoundError: org/omg/CORBA/ORB
        at hello.Server.main(Server.java:11)
304 vayssade ultra:~/corba>more setpath-java
setenv CLASSPATH .:hello:/usr/local/work/OB-2.0.4/jlib/
305 vayssade ultra:~/corba>source setpath-java
306 vayssade ultra:~/corba>java Server
Warning: ORB::BOA_init: hostname is a domainless name
using dotted decimal notation instead of hostname
^Z
Suspended
307 vayssade ultra:~/corba>bg
[1]    java Server &
308 vayssade ultra:~/corba>java client
Can't find class client
309 vayssade ultra:~/corba>java Client
Enter □h□ for hello or □x□ for exit:
> h
Hello World!
> h
Hello World!
> x
310 vayssade ultra:~/corba>

310 vayssade ultra:~/corba>jobs
[1]  + Running                       java Server
311 vayssade ultra:~/corba>kill %1

[1]    Terminated                    java Server
312 vayssade ultra:~/corba>serv
Warning: ORB::BOA_init: gethostbyname() returned domainless name
using dotted decimal notation instead of hostname
^Z
Suspended
313 vayssade ultra:~/corba>bg
[1]    serv &
314 vayssade ultra:~/corba>java Client
Enter □h□ for hello or □x□ for exit:
> h
Hello World!
> h
Hello World!
> x
315 vayssade ultra:~/corba>jobs
[1]  + Running                       serv
316 vayssade ultra:~/corba>
```

FIG. 190: Corba : exemple

Hello Exemple -- java

Hello.idl

```
// IDL
interface Hello
{
void hello();
};
```

Hello_impl.java

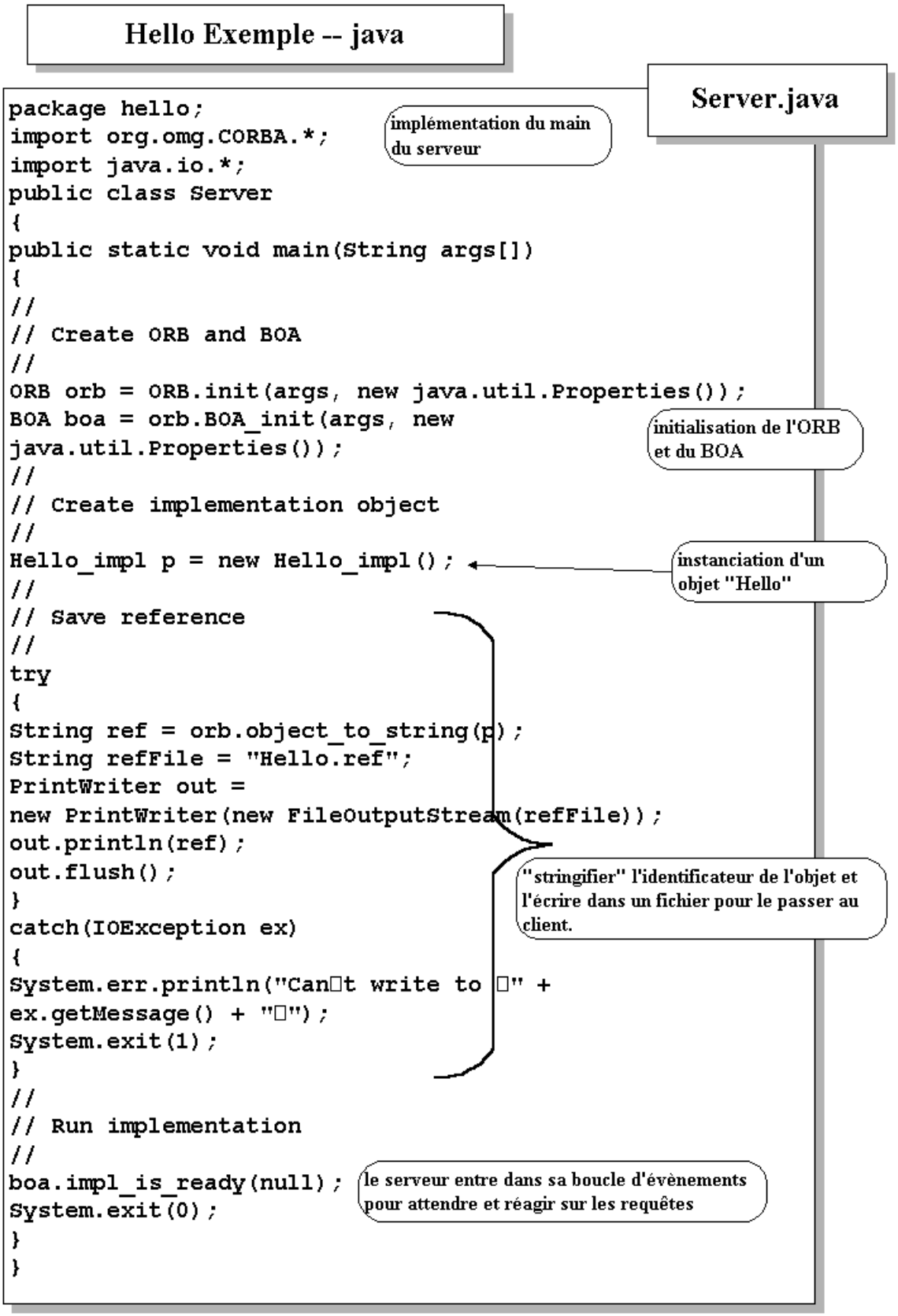
```
package hello;
import org.omg.CORBA.*;
public class Hello_impl extends _HelloImplBase
{
public void hello()
{
System.out.println("Hello World!");
}
}
```

Hello.ref

```
IOR:000000000000000e49444c3a48656c6c6f3a312e300000000
0000001000000000000003a000100000000000103139332e313034
2e3137382e31323000138900000000001a4f422f49442b4e554d0
049444c3a48656c6c6f3a312e30003000
```

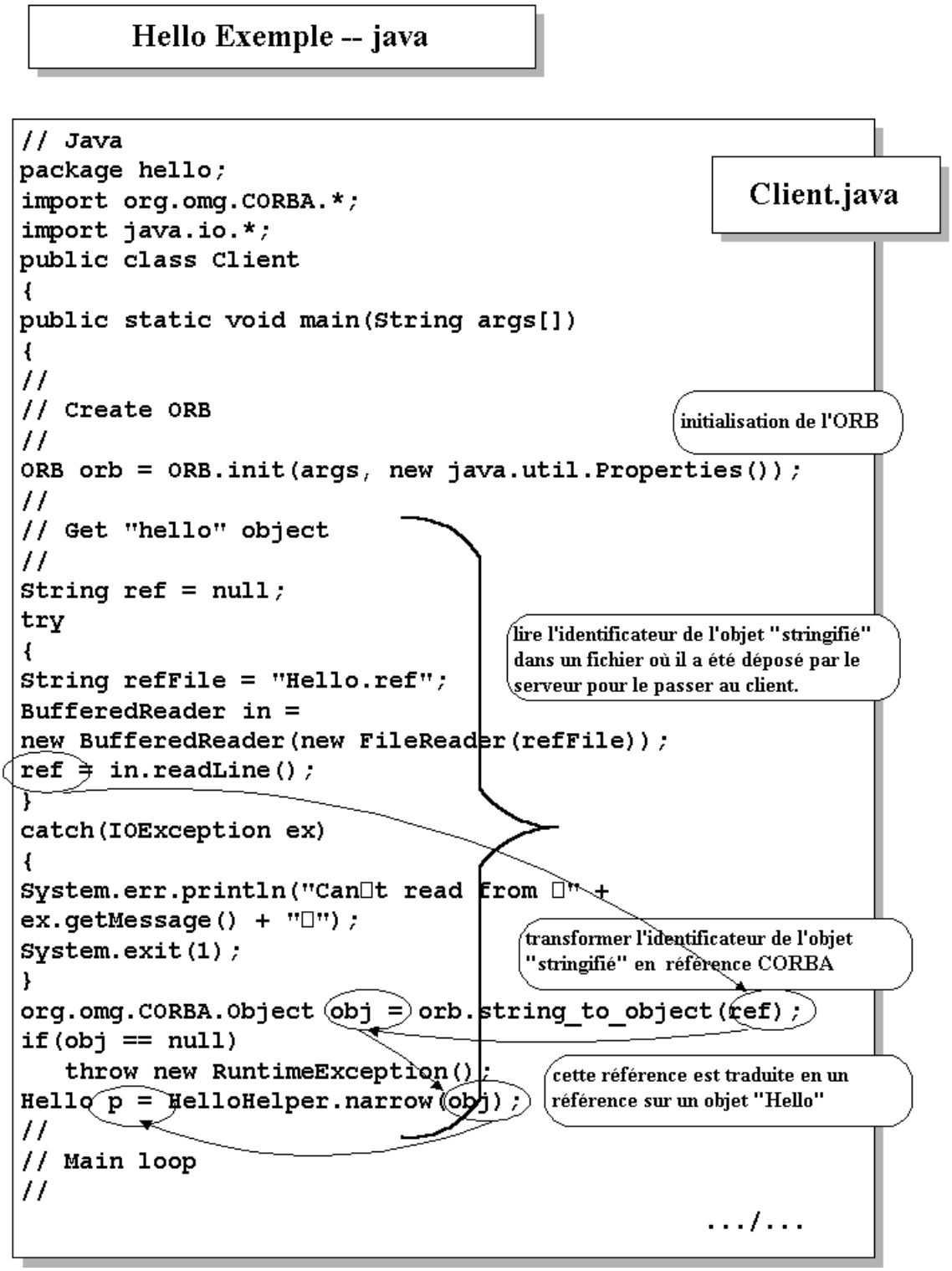
```
Chaine de caractères identifiant l'objet de façon
unique. Construite par l'appel au service CORBA :
     String ref = orb.object_to_string(p);
et utilisée pour accéder à l'objet par l'appel :
  org.omg.CORBA.Object obj = orb.string_to_object(ref);
```

FIG. 191: Corba : exemple




```
package hello;
import org.omg.CORBA.*;
import java.io.*;
public class Server
{
public static void main(String args[])
{
//
// Create ORB and BOA
//
ORB orb = ORB.init(args, new java.util.Properties());
BOA boa = orb.BOA_init(args, new
java.util.Properties());
//
// Create implementation object
//
Hello_impl p = new Hello_impl();
//
// Save reference
//
try
{
String ref = orb.object_to_string(p);
String refFile = "Hello.ref";
PrintWriter out =
new PrintWriter(new FileOutputStream(refFile));
out.println(ref);
out.flush();
}
catch(IOException ex)
{
System.err.println("Can□t write to □" +
ex.getMessage() + "□");
System.exit(1);
}
//
// Run implementation
//
boa.impl_is_ready(null);
System.exit(0);
}
}
```

FIG. 192: Corba : exemple




```
// Java
package hello;
import org.omg.CORBA.*;
import java.io.*;
public class Client
{
public static void main(String args[])
{
//
// Create ORB
//
ORB orb = ORB.init(args, new java.util.Properties());
//
// Get "hello" object
//
String ref = null;
try
{
String refFile = "Hello.ref";
BufferedReader in =
new BufferedReader(new FileReader(refFile));
ref = in.readLine();
}
catch(IOException ex)
{
System.err.println("Can□t read from □" +
ex.getMessage() + "□");
System.exit(1);
}
org.omg.CORBA.Object obj = orb.string_to_object(ref);
if(obj == null)
   throw new RuntimeException();
Hello p = HelloHelper.narrow(obj);
//
// Main loop
//
```

FIG. 193: Corba : exemple

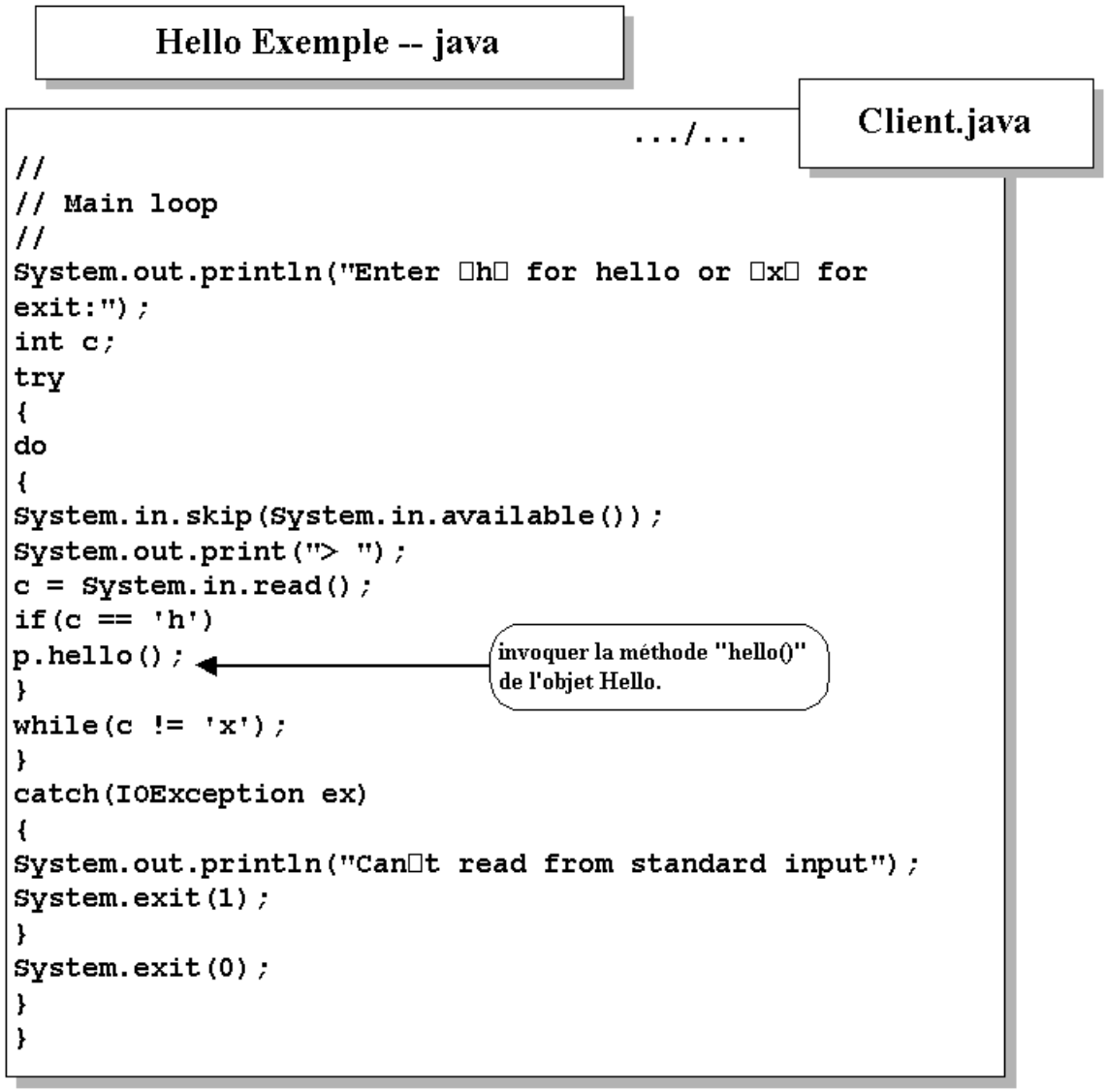


FIG. 194: Corba : exemple

RPC versus ORB

Tous les mandataires ("brokers") prélèvent un prix pour leurs services.

En quoi l'invocation d'une méthode à travers un ORB diffère d'un RPC ?

Les mécanismes sont très similaires :

- **appel de fonction à distance,**
- **passage par des adaptateurs ("stubs") côté client et côté serveur,**
- **empaquetage et dépaquetage des arguments par les adaptateurs,**
- **utilisation "transparente" du service de transport du réseau sous jacent,**
- **compilation et ligature séparées des codes clients et serveurs,**
- **description des fonctions appelées par un langage de définition (IDL);**

Avec le RPC on appelle une fonction spécifique, les données sont séparées.
Avec l'ORB, on appelle une méthode à l'intérieur d'un objet spécifique.

Différentes classes d'objet peuvent répondre de façon différente à un appel à la même méthode, par le biais du polymorphisme. La méthode appelée va travailler sur les données spécifiques à l'instance de l'objet appelé.

La méthode d'invocation par un ORB à une précision de scalpel : l'appel va à un objet précis qui contrôle des données précises et qui utilise la fonction spécifique associée à sa propre classe.
A contrario, l'appel RPC n'est pas spécifique : toutes les fonctions de même nom sont les mêmes.

Évidemment, la plupart des ORBs sont construits au-dessus d'un service RPC. On est donc amené à payer une pénalité en performances pour se niveau de service plus fin.

Ceci dit les implantations récentes d'ORB montrent que :
- dans le même espace d'adressage (invocation intra-process), les ORB ont des coûts équivalents à l'invocation d'une méthode virtuelle C++,
- dans un contexte distribué entre deux machines, un appel ORB est plus rapide qu'un appel PVM.

Si ces performances se confirment, on pourra utiliser les ORBs même pour objets à grain très fin, et pas seulement pour du calcul distribué à gros grain.

FIG. 195: RPC versus ORB

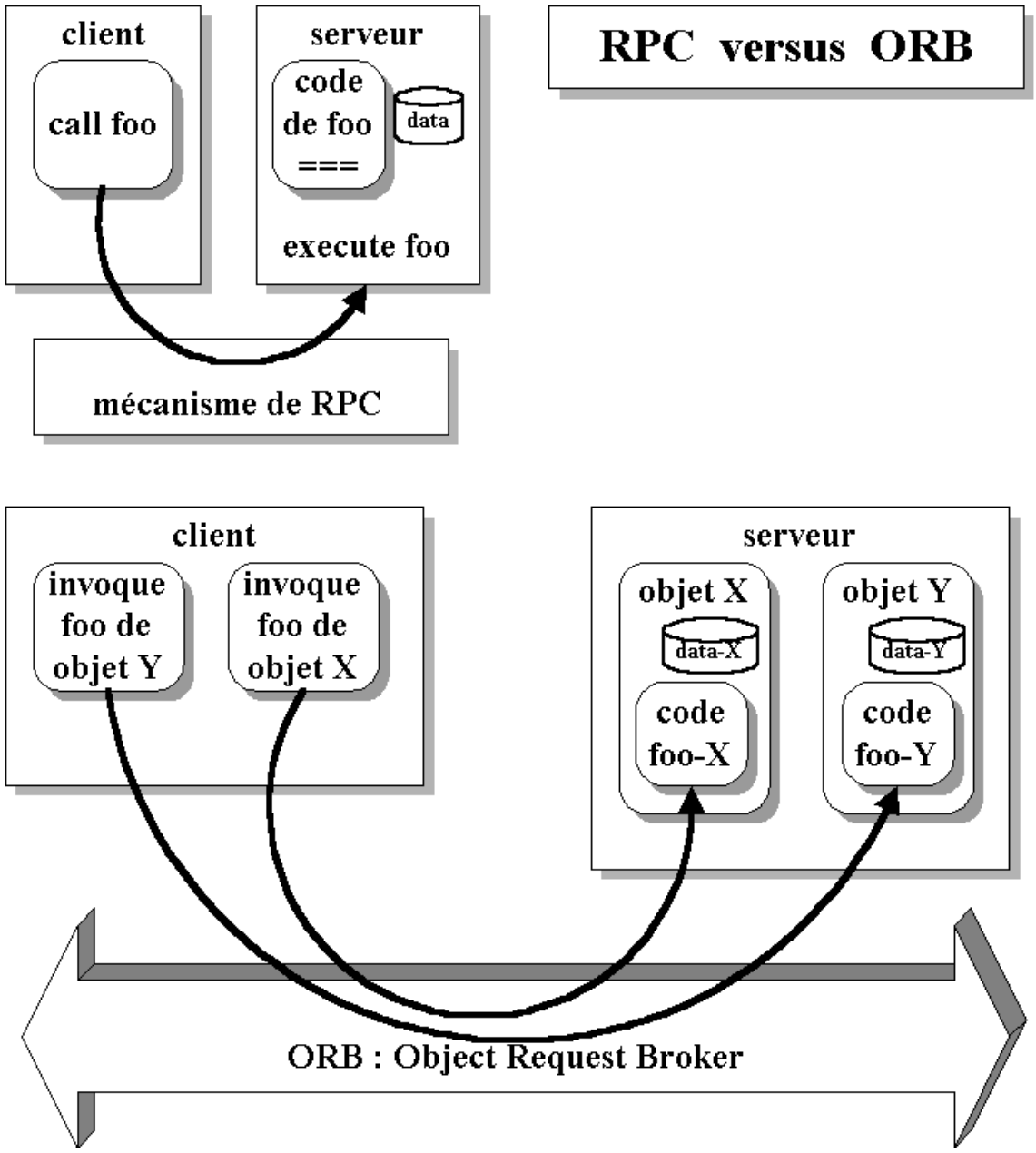


FIG. 196: RPC versus ORB

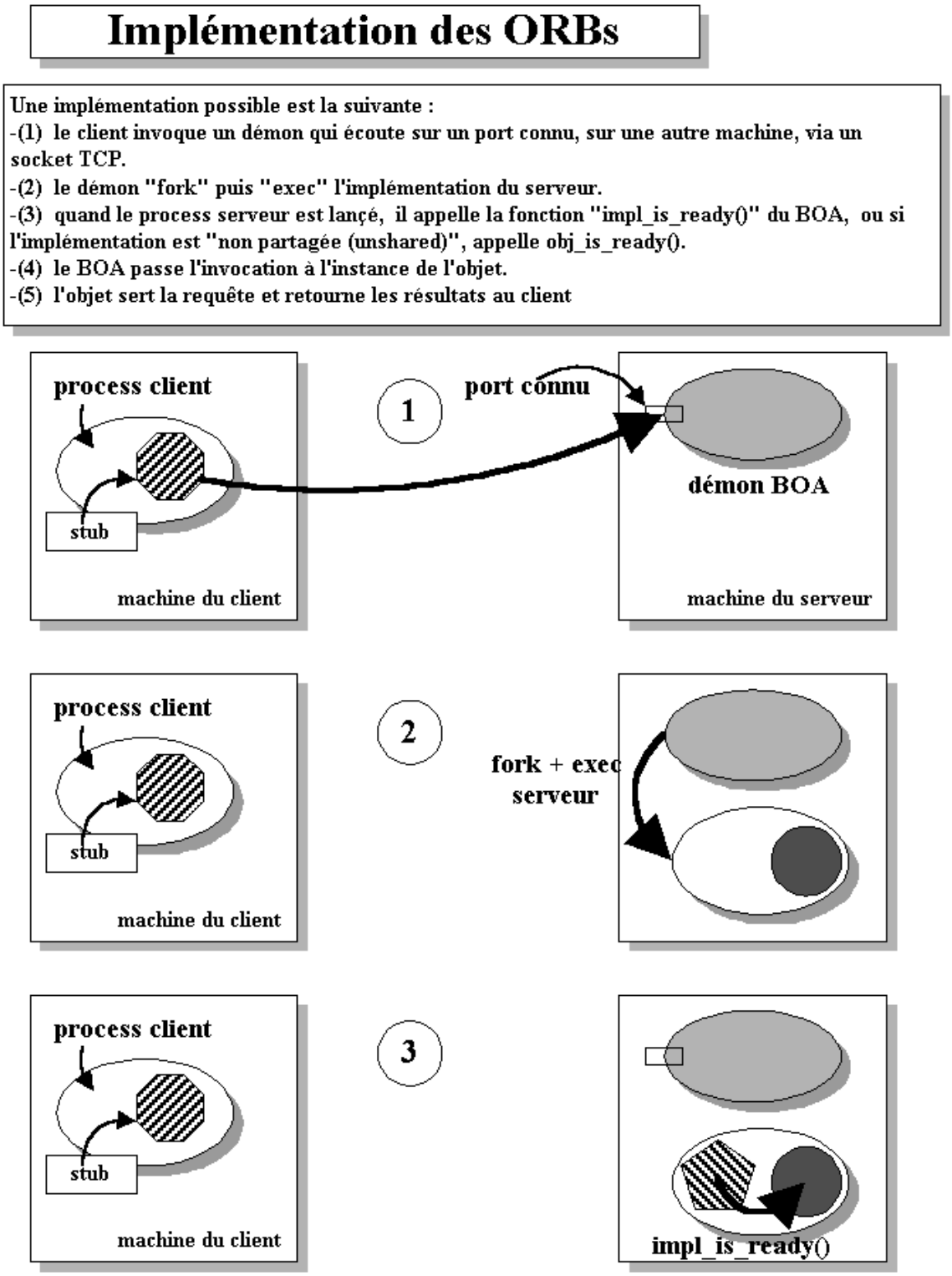


FIG. 197: Implémentation des ORBs

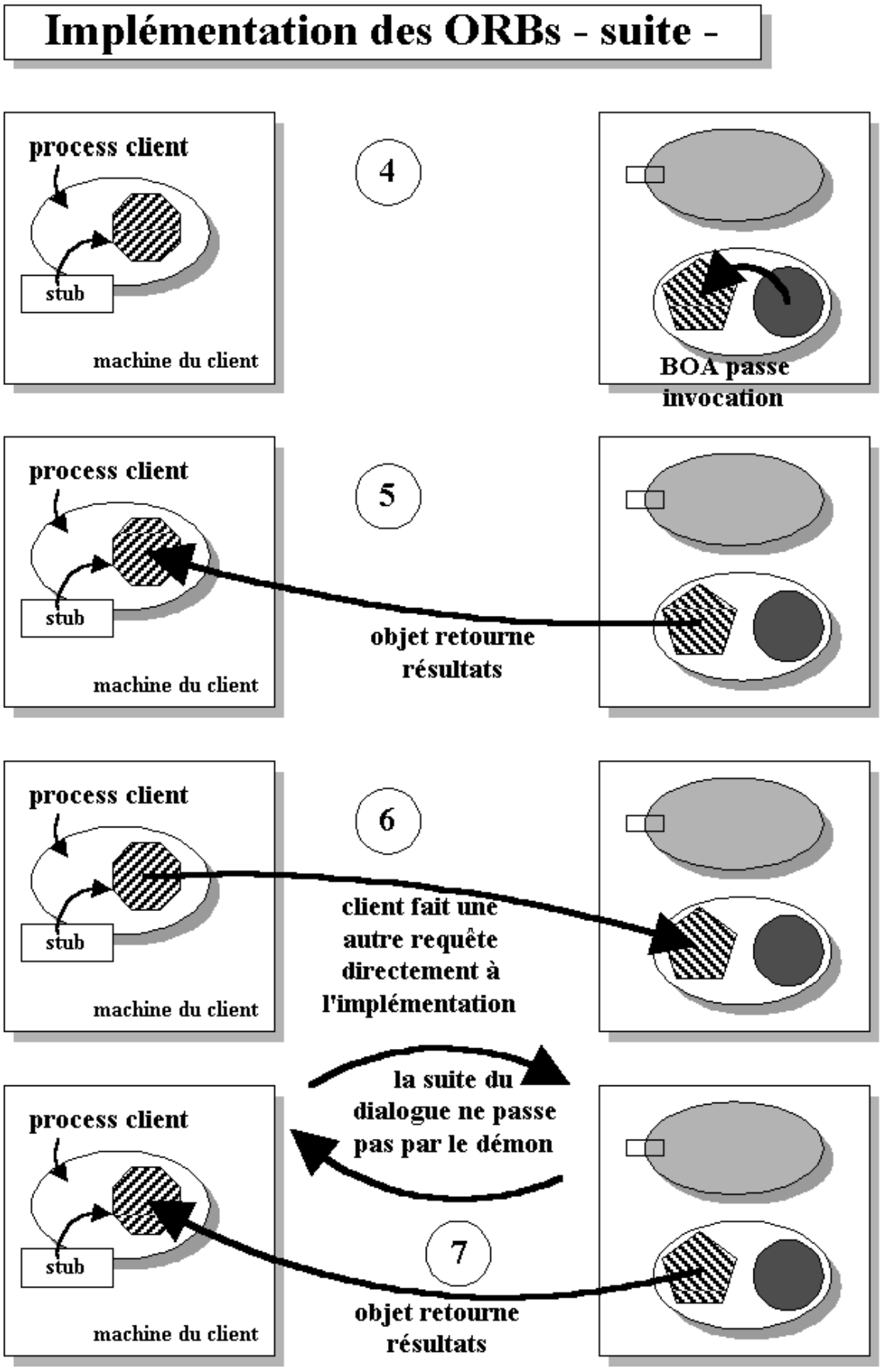
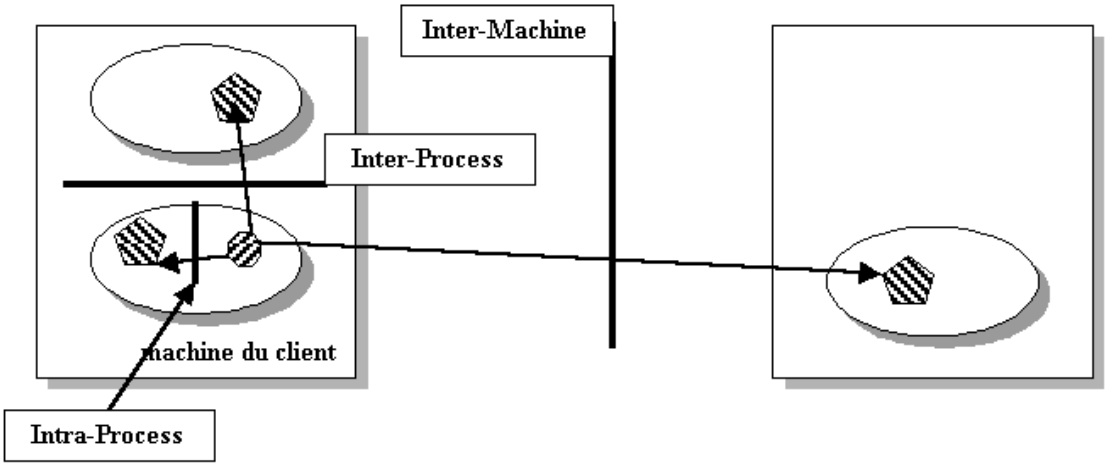


FIG. 198: Implémentation des ORBs

CORBA : "Location transparent" access.
Accés indépendant de la localisation de l'objet

L'un des buts de l'architecture OMA est qu'un programmeur puisse accéder à un objet sans avoir besoin de savoir à l'avance où, sur le réseau, l'objet existe : les détails sur la localisation de l'objet sont cachés.
Un objet doit apparaître comme étant le même, qu'il soit dans le même process, dans un autre process de la même machine où sur une machine différente.
Les détails sur la localisation ne font pas partie du "contrat" de l'objet spécifié par son interface : l'objet peut exister n'importe où sur le réseau et apparaître comme étant local.



Un effet de bord peut rendre la localisation "apparente" : des fautes ou des exceptions asynchrones peuvent survenir, et, en conséquence, le statut de terminaison d'une requête n'est pas toujours bien connu.
On ne peut pas toujours dire si l'objet a reçu la requête et commençé à travailler avant que la faute se produise.

FIG. 199: Accès transparent

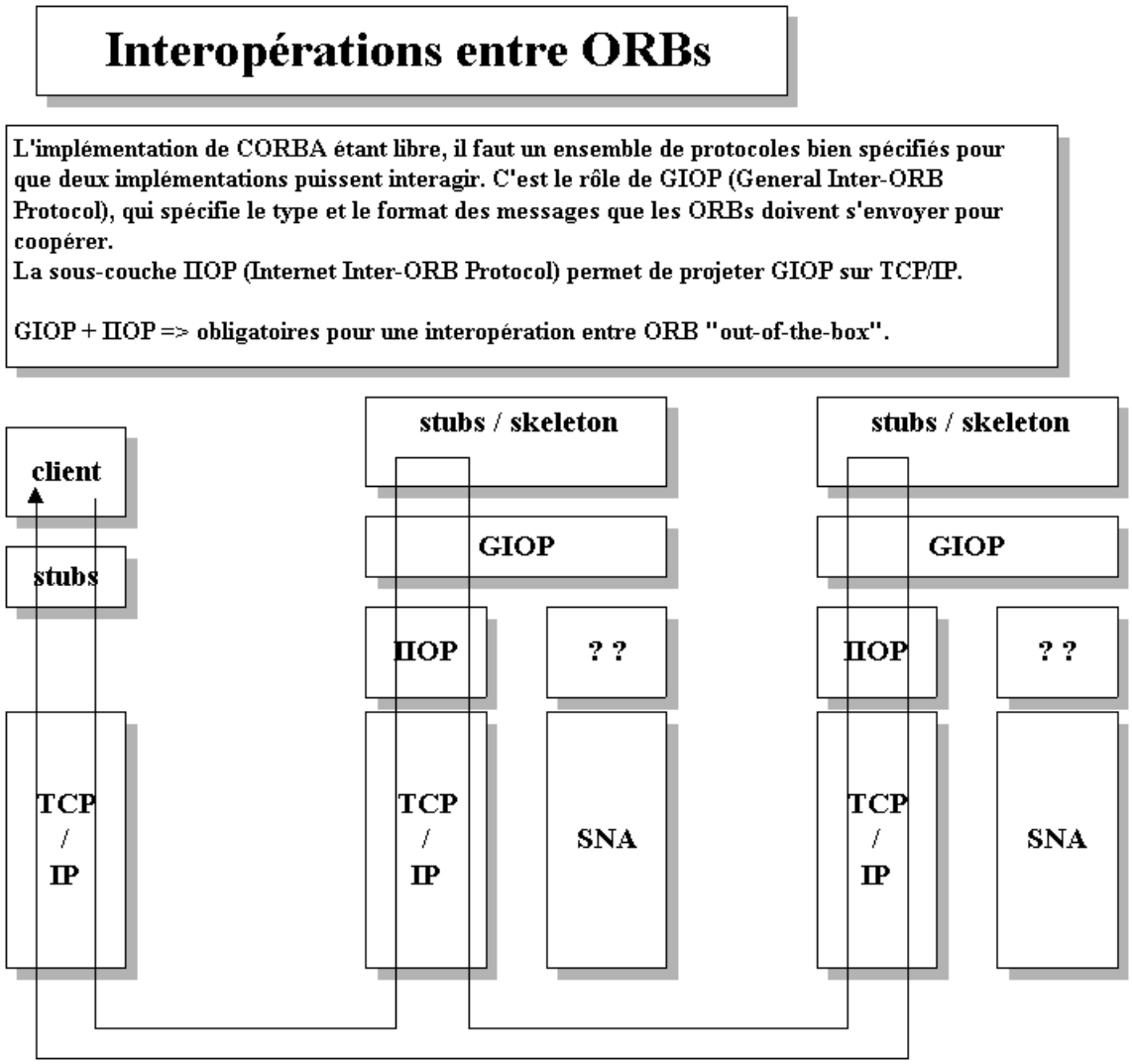


FIG. 200: Interopérations entre ORBs

CORBA

(Common Object Broker Architecture)

L'environnement

Un ORB CORBA 2.0 fournit plus que le simple mécanisme de passage de messages dont les objets ont besoin pour communiquer entre eux à travers des langages, des outils, des plateformes et des réseaux, tous hétérogènes.

Il fournit aussi l'environnement pour gérer ces objets, annoncer leur présence sur le réseau et décrire leurs métadonnées.

Un ORB CORBA 2.0 peut mandater des opérations entre objets qui résident dans un même process (comme un simple programme C++), aussi bien qu'entre objets interagissant à travers deux ORBs de constructeurs différents, sur des systèmes d'exploitation différents.

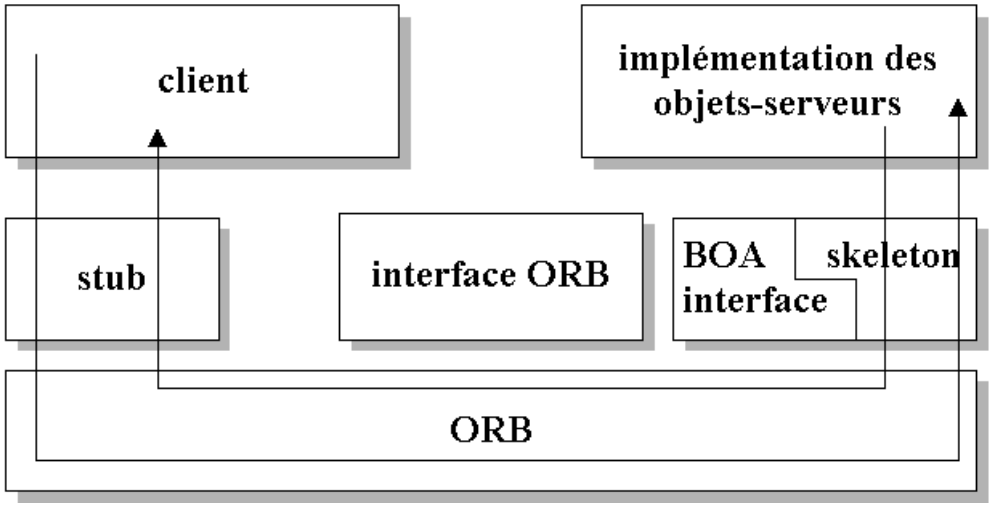


FIG. 201: L'environnement Corba

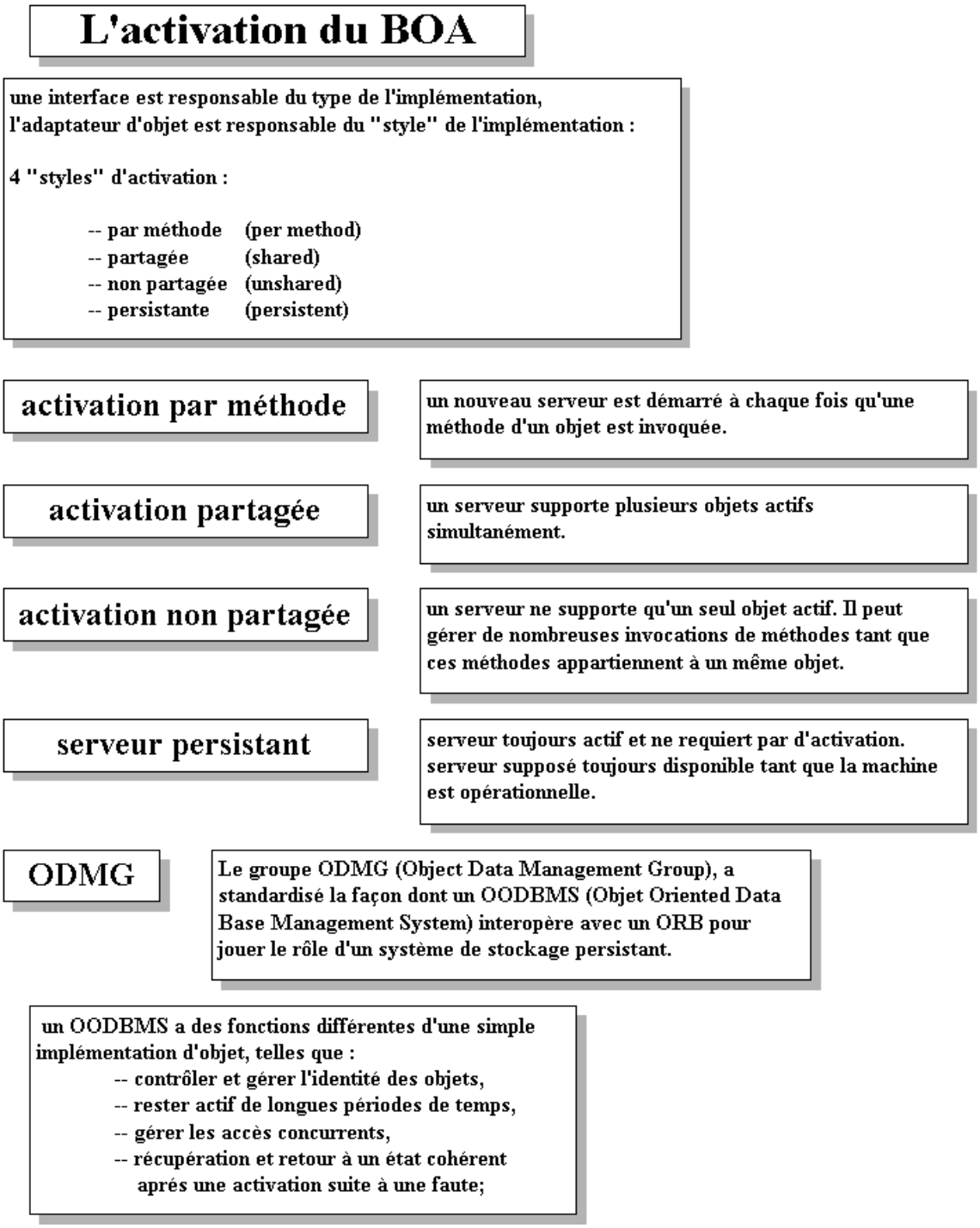


FIG. 202: L’activation du BOA

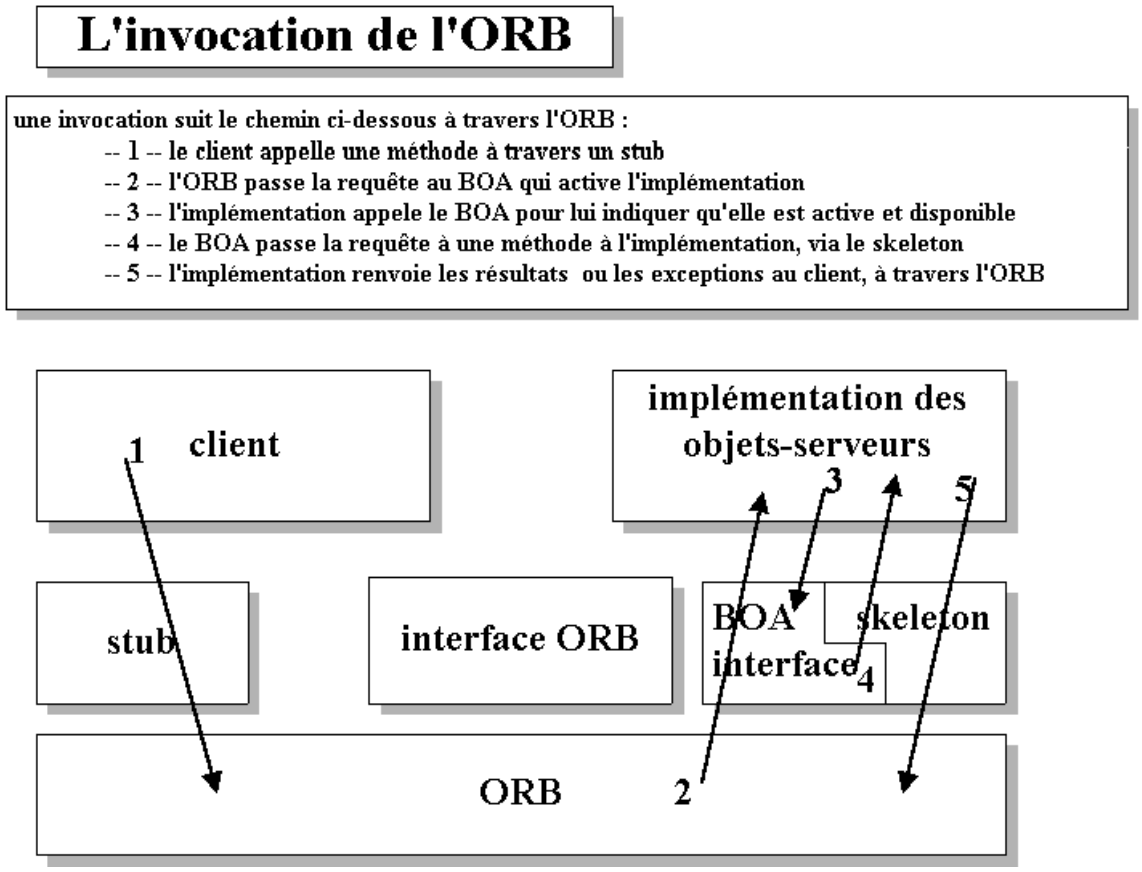


FIG. 203: L'invocation de l'ORB

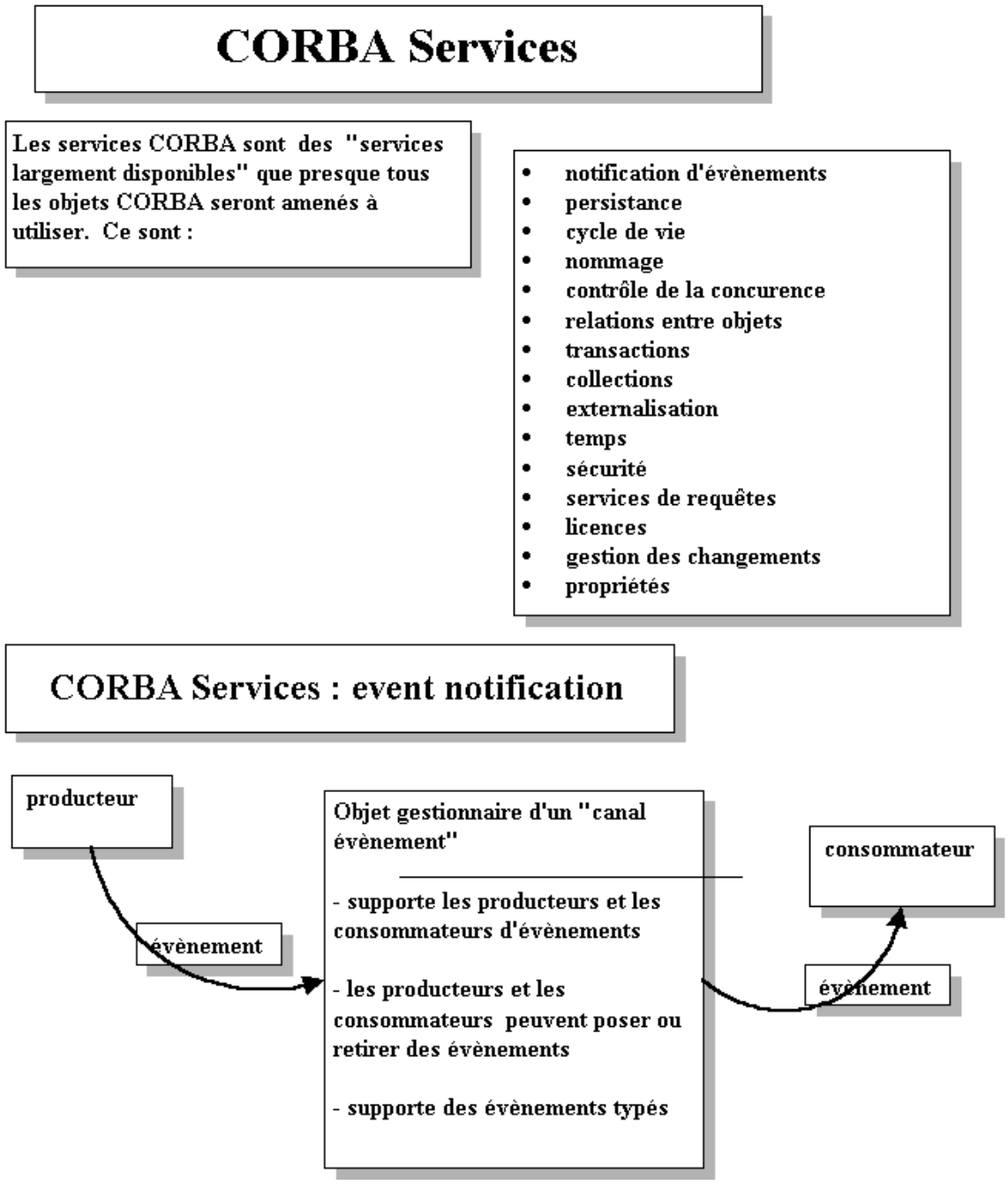


FIG. 204: Corba Services

19 SR03 2004 - Architectures Internet - Corba : exemple horloge ; IDL ; héritage et délégation

19.1 De l'IDL à l'application

Ce chapitre va montrer un premier exemple détaillé en Corba : l'application "horloge". C'est une application client - serveur utilisant une implémentation par héritage.

La description IDL du serveur décrit ce que le serveur propose aux clients.

Un serveur CORBA est une application qui met à disposition des objets. Chacun de ces objets implante une interface décrite au moyen d'un fichier IDL. Ce fichier IDl est compilé pour générer des fichiers sources qui devront être compilés (au sens des langages (C,C++,Java,...) et linkés avec le serveur d'une part et le client d'autre part.

Dans l'implantation par héritage, la classe d'implantation (écrite par l'auteur du serveur) va hériter du squelette (généré par le compilateur IDL).

Code de **Horloge.idl :**

```
interface Horloge
{
  string get_time();
};
```

Plus généralement :

```
interface Nom_de_l_interface
{
  // corps de l'interface
  attribute type un_attribut;
  ....
  type une_fonction (params);
  ....
};
```

Avec pour les fonctions :

```
Type_de_retour nom_fonction ( liste paramètres );
```

Et pour les structures :

```
struct nom_structure
{
  type x;
  type y;
  ...
};
```

On peut créer ses propres types :

```
      typedef sequence vecteur;
      vecteur v;
ou
typedef struct une_structure
{
  type x;
  ....
};
une_structure s;
```

Les types de base de CORBA sont les suivants :

```
boolean       octet
char          wchar           string          wstring
short         unsigned short      == short_java
long          unsigned long       == int_java
long long     unsignedlong long   == long_java
float         double              == float_java et double_java
long double                       ## pas_en_java
void  == retour vide d'une fonction
```

19.2 Compilation de la description IDL de l'interface

Le compilateur "idl2java" est utilisé pour compiler la description IDL de l'interface :

```
idl2java Horloge.idl

==> va créer plusieurs fichier parmi lesquels
    2 ont des noms normalisés :
        _HorlogeImplBase.java   amorce serveur = squelette
        _HorlogeStub.java       amorce client  = souche

ou en général pour  Xyyy.idl :
        _XyyyImplBase.java
        _XyyyStub.java
```

On va souvent appeler la classe d'implantation XyyyImpl.java (bien que ce ne soit pas obligatoire, on peut très bien l'appeler "truc.java"). Cette classe va hériter du squelette _HorlogeImplBase.java et contenir le code effectuant le travail "promis" par l'interface :

```
HorlogeImpl.java =
public class HorlogeImpl extends _HorlogeImplBase
{
  publis string get_time()
  {
    java.util.Calendar cal = new java.util.GregorianCalendar();
    String str = "["+ cal.get(java.util.Calendar.HOUR_OF_DAY);
    str = str + ":" + cal.get(java.util.Calendar.MINUTE) + "]";
  }
}
```

Le client va utiliser cette fonction de façon classique en instanciant un objet Horloge et en invoquant la fonction get_time() de cet objet :

```
public class Client
{
 public static void main (String [] args)
 {
 ..........// initialisations de CORBA
 //instanciation_à_distance_de_l_objet_CORBA_HorlogeImpl
 obj = orb.string_to_object(refobjhor);
 Horloge hor = HorlogeHelper.narrow(obj);
 System.out.println ( "Heure=" + hor.get_time() );
 }
}
```

19.3 Création du serveur

Avant de pouvoir utiliser l'objet, il va falloir créer l'application serveur qui va être linkée avec la bibliothèque CORBA pour pouvoir se mettre à l'écoute des connexions des clients sur le réseau, instancier les objets demandés et passer les appels (invocations de méthodes) des clients aux instances des objets.

Le serveur va comporter cinq étapes :

1. initialiser l'ORB
2. initialiser l'adaptateur d'objets (BOA ou POA)
3. créer un objet d'implantation (ici de type HorlogeImpl)
4. enregistrer et activer cet objet auprès de l'adaptateur d'objets
5. attendre les demandes des clients

La bibliothèque CORBA se présente comme un package contenant une classe org.omg.CORBA.ORB. Elle possède une fonction init() pouvant être appelée sous deux formes :

- org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init() ;
- org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(String [] args, java.util.Properties prop) ;

La deuxième forme permet de passer à l'ORB des arguments en provenance de la ligne de commande :

```
public static void main (String [] args)
{
  org.omg.CORBA.ORB orb= org.omg.CORBA.ORB.init(args, null);
  .....
}
```

L'initialisation de l'adaptateur d'objet dépend de l'adaptateur. Dans le cas d'un BOA elle est dépendante de l'ORB utilisé, mais la plupart ont une fonction ressemblant à :
org.omg.CORBA.BOA boa = orb.BOA_init (args , null) ;

Remarque : Avec le JDK1.2, qui possède une classe ORB, pour appeler la fonction de l'ORB que l'on veut utiliser (et non pas celui du JDK), il faut convertir la référence de votre ORB vers sa classe effective d'implantation :
org.omg.CORBA.BOA boa =
(com.ooc.CORBA.ORBSingleton) orb).BOA_init(arg,null);

Ici "com.ooc" est le nom de l'ORB ORBacus. Pour un autre ORB, il faudrait changer cette référence, par exemple pour l'ORB OpenORB on aurait :

org.omg.CORBA.BOA boa =
(**OpenORB**.CORBA.ORBSingleton) orb).BOA_init (args,null);

L'initialisation d'un adaptateur d'objet de type "POA" est plus compliquée mais normalisée. Le code peut être plus général et devient portable d'un ORB à un autre ORB. Cette initialisation sera montrée plus loin dans le chapitre sur le POA.

Pour créer un objet d'implémentation, il suffit d'invoquer le constructeur de l'objet :

```
HorlogeImpl hor = new HorlogeImpl();
```

Il faut ensuite enregistrer et activer cet objet auprès du BOA :

```
boa.connect ( hor ); // enregistrer
boa.obj_is_ready(hor); // activer
```

Ceci ajoute l'objet à la liste des objets connus et prêts à répondre aux demandes des clients (c'est un peu l'équivalent de l'enregistrement d'un serveur RPC dans le portmapper).

Ensuite, il ne reste qu'à attendre les demandes des clients en appelant la fonction impl_is_ready() de la classe BOA :

```
boa.impl_is_ready(hor); // appel "bloquant"
```

On ne ressort de cet appel que si une erreur se produit ou si le serveur est arrêté.

19.4 Création du client

Le client ne comporte que trois étapes :

1. initialiser l'ORB,
2. récupérer la référnce de l'objet que l'on veut utiliser,
3. utiliser l'objet (invoquer des méthodes de l'objet).

Le code du client aura la forme suivante :

```
public class Client
{
 public static void main (String [] args)
 {
 // initialisations de CORBA
 org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init( args, null );
 // récupérer la référnce de l'objet
 refobjdist = récupération_de_la_référence("identifiant_objet")
 objetdistant = traduction_de_la_référence(refobjdist)
 // convertir la réf. en objet du type souhaité
 Horloge hor = HorlogeHelper.narrow (obj);
 // utiliser l'objet
 System.out.println ( "Heure=" + hor.get_time() );
 }
}
```

Il faut maintenant détailler un peu les références d'objets et leur échange entre serveurs et clients ainsi que les relations entre classes utilisateurs et classes CORBA et les notions de "Helper" et "narrow".

Les références d'objets

Une référence d'objet contient toutes les données nécessaires pour retrouver cet objet. On peut la comparer à une URL.

Une référence est attribuée à un objet lors de l'opération boa.connect(obj). Elle permet de localiser l'objet : nom de la machine hôte du serveur, numéro du port sur lequel le serveur écoute les requêtes et un identifiant unique

qui permet de distinguer l'objet des autres objets gérés par le même serveur.

La référence est donc générée par le serveur. Il faut donc trouver un moyen de la transmettre aux clients potentiels.

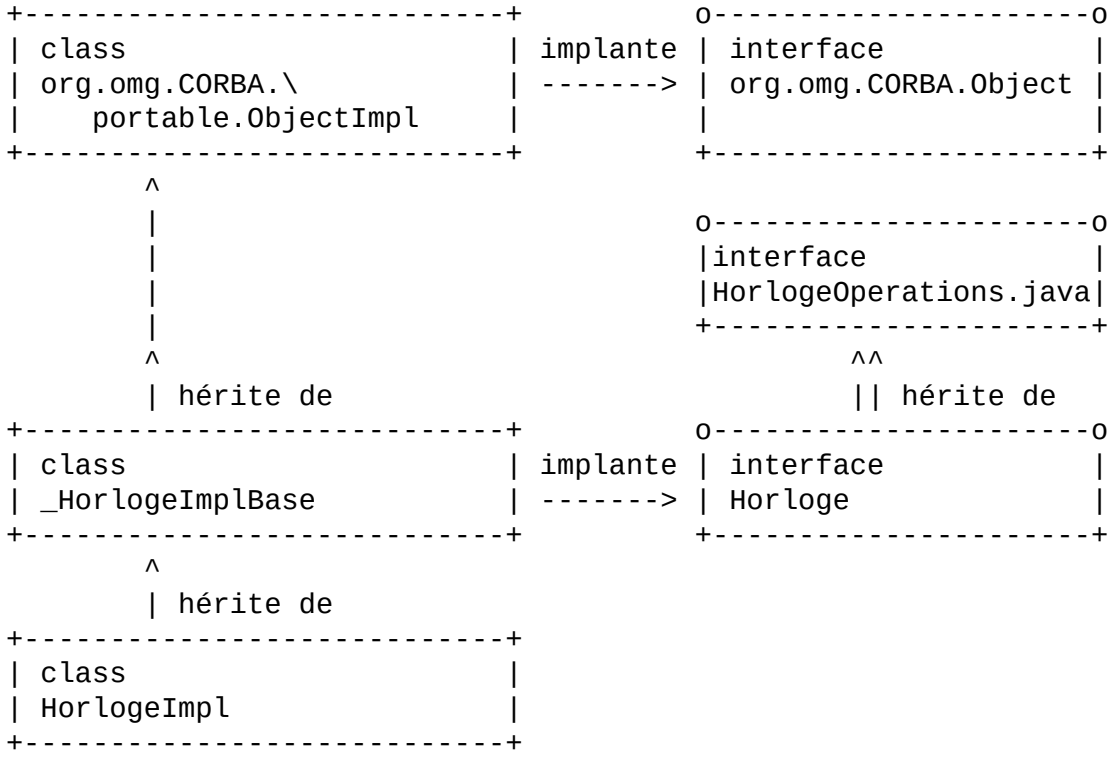
Dans les applications complexes déployées à grande échelle, on utilisera un service de noms ("naming service") fournissant un annuaire des objets disponibles.

Pour simplifier cette opération dans les premiers exemples on va utiliser deux fonctions de service de CORBA : object_to_string et string_to_object permettant de traduire une référence en une chaine de caractères et inversement.

Nous ferons passer cette chaine de caractères du serveur au client par un moyen extérieur à CORBA.

19.5 Les relations entre classes utilisateurs et classes CORBA et les notions de "Helper" et "narrow"

L'objet d'implantation (ici HorlogeImpl.java) hérite de _HorlogeImplBase.



La première ligne de la figure fait partie de l'ORB.

La deuxième ligne est générée par la compilation de l'IDL.

La troisième ligne est à écrire par l'auteur de l'application.

L'interface org.omg.CORBA.Object représente la référence d'objet.

La classe org.omg.CORBA.portable.ObjectImpl implante les méthodes de cette interface, et permet à l'adapteur d'objet de récupérer les informations caractérisant la référence de l'objet.

object_to_string :

```
HorlogeImpl hor = new HorlogeImpl();
String ref = orb.object_to_string(hor);
```

string_to_object :

```
org.omg.CORBA.Object obj = orb.string_to_object(ref);
```

Lorsque l'on recrèe la référence à partir de la chaine de caractère, on obtient une référence accessible uniquement à travers l'interface org.omg.CORBA.Object. Il va falloir convertir cette référence "générale" en une référence "spécifique" sur un objet de type (ici) Horloge.

Or "Horloge" n'hérite pas directement de "org.omg.CORBA.Object". On ne peut donc pas faire un transtypage (cast).

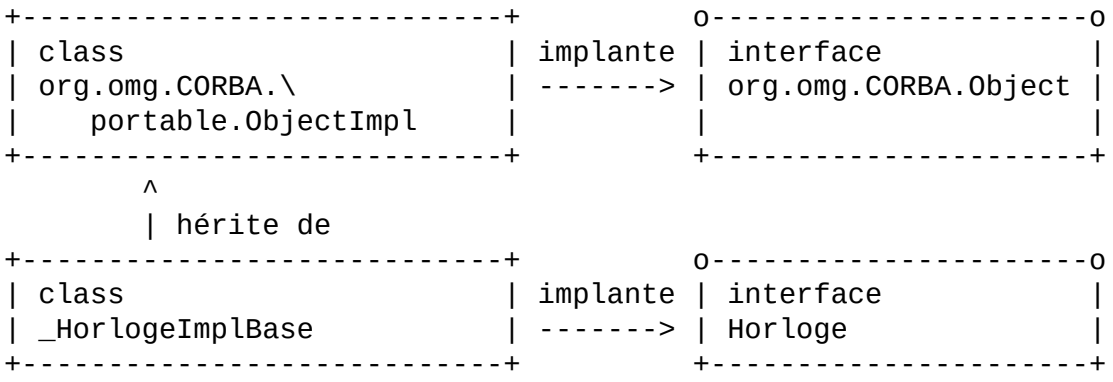
Pour contourner ce problème, on utilise une classe spéciale, générée par le compilateur IDL à partir de la description de l'interface Horloge. Cette classe s'appelle "classe Helper". Elle porte le nom de l'interface suivi de "Helper" : ici HorlogeHelper.

La classe "Helper" contient en particulier une méthode "narrow" spécialement construite pour convertir une référence d'objet en une référence vers l'interface Horloge. Il faut évidemment que l'objet que l'on donne en paramètre de "narrow" implante l'interface vers laquelle on essaie de le convertir. Ce sera le cas ici si l'objet est bien à l'origine du type HorlogeImpl qui hérite de _HorlogeImplBase qui elle-même implante Horloge.

Dans le cas contraire, lors de l'essai de conversion, on reçoit une exception de type : org.omg.CORBA.BAD_PARAM (exception système CORBA qui hérite de org.omg.CORBA.SystemException).

Que fait "narrow" ?

Quand on utilise "narrow" sur le client, on a la situation suivante :



La fonction "narrow" convertit la référence en une référence vers la souche qui implante l'interface Horloge. La souche contient donc toutes les méthodes de l'interface que l'on pourra alors invoquer :

```
// convertir la réf. en objet du type souhaité
Horloge hor = HorlogeHelper.narrow (obj);
// utiliser l'objet
System.out.println ( "Heure=" + hor.get_time() );
```

Remarque : À chaque appel à une méthode d'un objet distant, des erreurs peuvent se produire (serveur planté, timeout dû au réseau, etc ...), c'est pourquoi il est prudent de mettre ces appels dans un bloc try/catch :

```
try
{ Horloge hor = HorlogeHelper.narrow (obj);
  String heure = hor.get_time();
  ....
}
catch (org.omg.CORBA.SystemException ex)
{
  ex.printStackTrace();
}
```

19.6 La compilation et l'exécution

```
horloge> ls
Client.java   HorlogeImpl.java   Serveur.java   Horloge.idl

horloge> java org.openorb.compiler.IdlCompiler\
 -d . -boa -notie Horloge.idl

horloge> ls
Client.java          HorlogeHolder.java      Serveur.java
Horloge.idl          HorlogeImpl.java        _HorlogeImplBase.java
Horloge.java         HorlogeOperations.java  _HorlogeStub.java
HorlogeHelper.java

horloge> javac *.java
horloge>

running serveur with :

horloge> java -Dorg.omg.CORBA.ORBClass=org.openorb.CORBA.ORB
-Dorg.omg.CORBA.ORBSingletonClass=org.openorb.CORBA.ORBSingleton
Serveur
 Le serveur est pret...

running client with :

horloge> java -Dorg.omg.CORBA.ORBClass=org.openorb.CORBA.ORB
-Dorg.omg.CORBA.ORBSingletonClass=org.openorb.CORBA.ORBSingleton
Client
 Heure = [ 16:27 ]
```

19.7 Le code complet du serveur

Extrait du livre "Au coeur de Corba - Jérôme Daniel" aux éditions Vuibert.
**==> achetez ce livre si vous devez travailler sur Corba.

Serveur.java

```
 1  // -------------------------------------------------
 2  // Chapitre 2 - Notre première application CORBA
 3  // Jérôme DANIEL
 4  // -------------------------------------------------
 5  // Serveur.java
 6  /**
 7   * Implantation du serveur pour l'objet Horloge
 8   * @version 1.0
 9   */
10  public class Serveur
11  {
12    /**  Point d'entree du programme
13     */
14    public static void main( String [] args )
15    {
16        // Initialisation de l'ORB
17        org.omg.CORBA.ORB orb =
18                            org.omg.CORBA.ORB.init(args, null);
19        System.out.println("orb="+orb);
20        // Initialisation du BOA
21        // ( attention cette methode est propre a JavaORB,
22        // on aurait également pu ecrire :
23        //      org.omg.CORBA.BOA boa = orb.BOA_init(args,null);
24
25        org.omg.CORBA.BOA boa= org.omg.CORBA.BOA.init(orb,args);
26        System.out.println("boa="+boa);
27        // Creation de l'objet Horloge
28        HorlogeImpl horloge = new HorlogeImpl( );
29        System.out.println("horloge");
30        // Connexion et activation au sein du BOA
31        boa.connect( horloge );
32
33        boa.obj_is_ready( horloge );
34
35        // Exportation de la reference de l'objet Horloge
36        // dans un fichier
37        try
38        {
39            String ref = orb.object_to_string( horloge );
40            java.io.FileOutputStream file = new
41                              java.io.FileOutputStream("ObjectId");
42            java.io.PrintStream pfile=new
43                              java.io.PrintStream(file);
44            pfile.println(ref);
45            pfile.close();
46        }
47        catch ( java.io.IOException ex )
48        {
49            ex.printStackTrace( );
50            System.exit(0);
51        }
52
```

Serveur.java (suite)

```
53        // Mise en attente du serveur de requetes clientes
54        try
55        {
56            System.out.println(" Le serveur est pret...");
57            boa.impl_is_ready( );
58        }
59        catch ( org.omg.CORBA.SystemException ex )
60        {
61            ex.printStackTrace();
62        }
63    }
64 };
```

HorlogeImpl.java

```
 1 // ------------------------------------------------
 2 // Chapitre 2 - Notre première application CORBA
 3 // Jérôme DANIEL
 4 // ------------------------------------------------
 5 // HorlogeImpl.java
 6 /**
 7  * Implantation de l'interface Horloge definie en IDL
 8  * @version 1.0
 9  */
10 public class HorlogeImpl extends _HorlogeImplBase
11 {
12    /** Cette operation retourne l'heure courante.
13     */
14    public String get_time( )
15    {
16       java.util.Calendar calendar = new
17                 java.util.GregorianCalendar( );
18
19       String str = "[ " +
20             calendar.get(java.util.Calendar.HOUR_OF_DAY);
21       str = str + " :" +
22             calendar.get(java.util.Calendar.MINUTE) + " ]";
23       return str;
24    }
25 }
```

19.8 Le code complet du client

Client.java

```
 1 // ------------------------------------------------
```

Client.java (suite)

```
 2  // Chapitre 2 - Notre première application CORBA
 3  // Jérôme DANIEL
 4  // ------------------------------------------------
 5  // Client.java
 6  /**
 7   * Implantation du client de l'objet Horloge
 8   * @version 1.0
 9   */
10  public class Client
11  {
12     /** Point d'entrée du programme
13      */
14     public static void main( String [] args )
15     {
16         // Initialisation de l'ORB, pas de BOA puisque nous
17         // sommes uniquement client
18         org.omg.CORBA.ORB orb= org.omg.CORBA.ORB.init(args,null);
19
20         // Recuperation de la reference de l'objet Horloge
21         // depuis un fichier
22         org.omg.CORBA.Object obj = null;
23         try
24         {
25            java.io.FileInputStream file = new
26                           java.io.FileInputStream("ObjectId");
27            java.io.InputStreamReader input = new
28                           java.io.InputStreamReader( file );
29            java.io.BufferedReader reader = new
30                           java.io.BufferedReader(input);
31            String ref = reader.readLine();
32            obj = orb.string_to_object(ref);
33         }
34         catch ( java.io.IOException ex )
35         {
36            ex.printStackTrace( );
37            System.exit(0);
38         }
39
40         // Conversion de la reference a l'aide de l'operation
41         // "narrow", puis utilisation de l'objet
42         try
43         {
44                 Horloge horloge = HorlogeHelper.narrow( obj );
45
46                 System.out.println(
47                          " Heure = " + horloge.get_time());
48         }
49         catch ( org.omg.CORBA.SystemException ex )
50         {
51                 ex.printStackTrace( );
52         }
53     }
```

Client.java (suite)

```
54  }
```

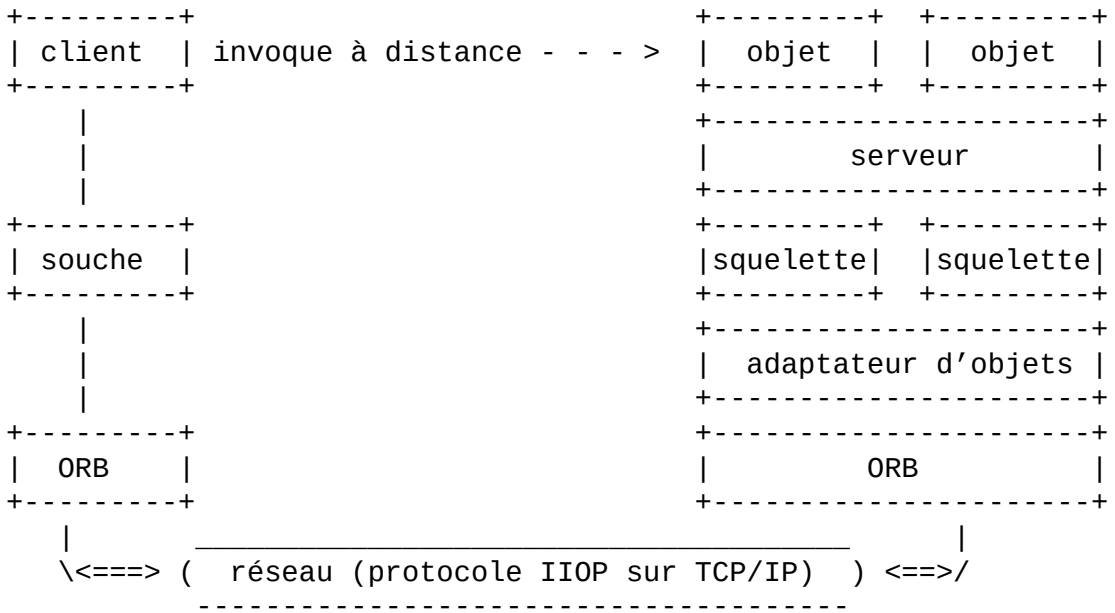
Le code de Horloge.java et de HorlogeOperations.java :

```
// Horloge.java
// Interface definition : Horloge
// @author OpenORB Compiler
//
public interface Horloge extends HorlogeOperations, \
org.omg.CORBA.Object, org.omg.CORBA.portable.IDLEntity
{
}
```

```
// HorlogeOperations.java
// Interface definition : Horloge
// @author OpenORB Compiler
//
public interface HorlogeOperations
{
        // Operation get_time
        //
        public java.lang.String get_time();
}
```

19.9 Conclusion partielle

On a la structure suivante entre client et serveur :



L'ORB traduit les requêtes entre clients et serveurs en un protocole général indépendant du langage d'implantation, ce qui permet d'appeler un objet java depuis un objet C++ ou l'inverse.

19.10 CORBA : les bases de l'IDL (Interface Definition Language)

Surcharge de noms interdite en IDL

Si une interface définit déja une fonction string = get_time(), et que l'on veut ajouter une fonction void get_time(hh,mm), il **faudra** lui donner un autre nom, la surcharge de noms étant interdite en IDL.

On a vu qu'il était possible de définir des structures en IDL : struct nom_structure

```
{
  type x;
  type y;
  ...
};
```

Ces structures peuvent être passées en paramètres des fonctions ou bien reenvoyées comme valeur de retour. struct heure_min

```
{
  long hh;
  long mm;
};
interface Time_hm
{
  heure_min get_time_hm();
};
```

Traduction des structures IDL en Java

Quand on compile la structure IDL ci-dessus, on va générer **trois** nouveaux fichiers : heur_min.java , heur_minHelper.java et heur_minHolder.java.

heur_min.java : Ce fichier contient la traduction en Java de la structure sous forme d'une classe, chaque membre de la structure devenant un attribut public de la classe. Des constructeurs sont ajoutés pour initialiser les attributs.

Exemple de fonction renvoyant une structure :

```
struct heure_min
{
 long hh;
 long mm;
};
interface Time_hm
{
  heure_min get_time_hm();
};
```

La fonction ci-dessus renvoie une structure contenant deux attributs.

Pour renvoyer deux valeurs nous aurions aussi pu écrire une fonction qui renverrait deux valeurs (équivalent du passage par référence en C) comme ci-après :

```
interface Heur_min
{
  void get_heur_min(out long hh, out long mm);
};
```

En effet, un paramètre d'opération en IDL peut être spécifié de trois façons :

```
long get_fonc (in  long param);
void get_fonc (out long param);
void get_fonc (inout long par);
```

19.11 Les classes Holder

Le choix de conception du langage Java fait que celui-ci ne peut passer les arguments à une fonction **que par valeur**. Ainsi, pour les types simples (int, long, double ...) une fonction Java ne peut pas modifier un paramètre d'appel et renvoyer la nouvelle valeur à son appelant.

Pour pouvoir faire cette opération, il faut passer à la fonciton une **référence sur une instance** d'une classe et modifier un attribut de la classe.

Donc pour pouvoir implanter les sémantiques **out et inout** de CORBA, il faut, en Java, encapsuler les arguments dans des classes : c'est le rôle des classes **holder**.

Exemple : Ci-dessous on montre un exemple **val.java** et un exemple **ref.java** qui illustrent cette particularité du langage Java :

val.java :

```
public class val
{
  public static int value = 5;
  public void change( int i)
  {
        i = i * 2;      //<=***** accès par VALEUR
  }
  public static void main(String args[])
  {
     val une_val = new val();
     System.out.println("value="+une_val.value);
     une_val.change(une_val.value);
     System.out.println("value="+une_val.value);
  }
}
1090 lo33 b116serv:~/mv/java> javac val.java
1091 lo33 b116serv:~/mv/java> java val
value=5
value=5         //<=***** accès par VALEUR
                //pas de modif à l'extérieur
```

ref.java :

```
public class ref
{
  private class hold
  {
     public int value;
     public hold ()     // constructeur
     {}
     // constructeur avec initialisation
     public hold (int init)
     {  value = init; }
     public void change( hold i)
     {
        i.value = i.value * 2;
     }
  }

  public static void main(String args[])
  {
     ref une_ref = new ref();
     // créer classe interne à ref
     hold une_val= une_ref.new hold(5);

     System.out.println("value="+une_val.value);
     une_val.value = 7;
     System.out.println("value="+une_val.value);
                          // appel avec une instance
// de classe en argument
                          // ==> passage par RÉFÉRENCE
     une_val.change(une_val);
     System.out.println("value="+une_val.value);
  }
}
1106 lo33 b116serv:~/mv/java> javac ref.java
1107 lo33 b116serv:~/mv/java> java ref
value=5
value=7
value=14        //<== valeur interne modifiée
```

Quand on passe an paramètre à une fonction java, non pas un type simple mais un objet, **on peut modifier la valeur des attributs de l'objet**.

Donc pour les paramètres out et inout de CORBA, il faut définir des classes "conteneur" pour chaque type utilisé dans un paramètre modifié. Ces classes sont les classes holder automatiquement créées par le compilateur IDL.

Ces classes portent le nom du type accolé au mot "Holder".

Exemple : heure_min fonc () ; ==> heure_minHolder.java

Pour les types de base de CORBA les bibliothèques CORBA possèdent déja la collection des classes holder correspondantes :

```
org.omg.CORBA.BooleanHolder
org.omg.CORBA.ShortHolder   etc ...
```

Les classes holder possèdent un attribut public "value" qui correspond à la valeur qu'elles contiennent.

19.12 Traduction des interfaces IDL en Java

Les interfaces IDL deviennent des interfaces Java et chaque opération de l'interface devient une méthode publique de l'interface Java.

Fichiers générés

La compilation d'une interface horloge.idl par idl2java génère les fichiers suivants :
- **_horlogeStub.java** : la souche côté client,
- **_horlogeImplBase.java** : le squelette côté serveur,
- **horlogeOperations.java** : une interface contenant les signatures des fonctions définies dans horloge.idl
- **horloge.java** : l'interface représentant l'objet horloge ; elle hérite de **horlogeOperations.java** et de **org.omg.CORBA.portable.IDLEntity**, ce qui en fait un objet CORBA accessible à distance,
- **horlogeHelper.java** : fournit l'opération narrow aux clients,
- **horlogeHolder.java** : permet d'utiliser un paramètre de type horloge en mode out ou inout.

Les attributs des interfaces IDL

L'IDL permet de définir des attributs :

```
attribute type_de_l_attribut nom_de_l_attribut ;
// exemple d'attribut de type long:
     attribute long age;
// un attribut peut être en lecture seule :
     readonly attribute long solde;
```

Le compilateur IDL va générer une ou deux fonctions pour représenter un attribut : une pour l'accès en lecture et une pour l'accès en écriture.

```
attribute long total; //===>
        long total();           // lecture
        void total(long val);   // écriture
```

Utilisation de références d'objets dans les interfaces IDL

Il est possible d'utiliser une référence d'objet comme type de paramètre d'une fonction ou comme membre d'une structure de données.

Une référence d'objet est symbolisée par le nom de l'interface correspondant à cet objet.

Par exemple, si on a créé une interface IDL "compte_bancaire" dans un fichier compte_bancaire.idl, il est possible d'utiliser des références à des objets CORBA de type "compte_bancaire" dans une autre interface, par exemple une interface "banque.idl".

```
interface banque
{  compte_bancaire creer_compte (
              in string titulaire, in string adresse);
};
```

Ici l'opération "creer_compte" renvoie une référence vers un objet CORBA de type "compte_bancaire".

19.13 Les héritages et les interfaces IDL

L'IDL supporte l'héritage : une interface peut hériter de une ou plusieurs interfaces. Exemple :

```
interface Base
{
  void f();
};
interface AutreBase
{
  void g();
};
interface PremierHeritier : Base
{
  void h();
};
interface DeuxièmeHeritier : Base, AutreBase
{
  void z();
};
```

Remarque 1 : la surcharge de méthodes est interdite en IDL. Ainsi PremierHeritier ne peut pas déclarer une opération "f()", même si sa signature est différente de celle de "Base :f()".

Remarque 2 : un objet de type DeuxièmeHeritier pourra être converit au choix en objet de type : DeuxièmeHeritier, Base, AutreBase ou org.omg.CORBA.Object

Problème d'implémentation en Java avec l'approche par héritage

Dans l'approche par héritage la classe d'implantation XyyyImpl.java hérite de la classe _XyyyImplBase.java générée par le compilateur IDL.

Or le langage Java interdit l'héritage multiple. Ainsi la classe PremierHeritierImpl va hériter de _PremierHeritierImplBase.

La classe PremierHeritier implémente l'interface PremierHeritier qui elle-même hérite de l'interface base. Donc elle doit implanter les fonctions définies dans PremierHeritier et dans Base. Donc la fonction "Base :f()" doit être définie dans PremierHeritier. Malheureusement, on ne peut pas la définir implicitement en faisant hériter PremierHeritierImpl de BaseImpl, ce qui serait normal dans la logique de l'héritage puisque BaseImpl (qui implémente l'interface Base) possède de ce fait une fonction f().

Il faut alors redéfinir "f()" dans l'implantation de PremierHeritierImpl.

```
public class BaseImpl extends _BaseImplBase
 {
   public void f()
   { ...//code de f...
   }
 }
public class PremierHeritierImpl extends _PremierHeritierImplBase
 {
   public void f()
   { ...//code de f...  à réimplanter car on ne peut pas
        //              hériter aussi de BaseImpl
   }
   public void h()
   { ...//code de h...
```

```
    }
}
```

Cette limitation est due à Java et à conduit l'OMG à définir une approche particulière pour conserver les bénéfices de l'héritage et de la réutilisation du code : l'approche par délégation ("tie"). Cette approche, détaillée en fin de chapitre, va permettre de se dégager de l'obligation pour une classe d'implémentation d'hériter du squelette du serveur. Ses fonctions (qui possèdent le code à exécuter) seront appelées par l'intermédiaire d'une classe "Tie" qui elle héritera de _PremierHeritierImplBase, proposera des fonctions de même signature que celles de PremierHeritierImpl, fonctions "vides" qui ne feront que transmettre l'appel à la vraie fonction de la classe PremierHeritierImpl.

19.14 Connexion d'objets par des fonctions de la classe ORB

L'adaptateur d'objet "BOA" n'a pas été suffisamment spécifié lors de sa normalisation. Il en est résulté des différences d'utilisation entre des ORBs de fournisseurs différents, empêchant la portabilité complète du code.

À partir de CORBA 2.3, le BOA ne fait plus partie de la spécification. Il a été remplacé par le POA : "Portable Object Adapter".

Toutefoia on a pu constater que dans la très grande majorité des cas seules 5 fonctions du BOA étaient utilisées :

- connexion d'un objet,
- activation d'un objet,
- déconnexion d'un objet,
- désactivation d'un objet,
- attente des requêtes clients.

On a donc ajouté 3 fonctions directement à la classe ORB pour simplifier l'écriture de serveurs simples :

- orb.connect(obj)
- orb.disconnect(obj)
- orb.run() pour se mettre en attente des clients

Les initialisations sont alors faites de façon transparentes et portables.

Toutefois, dans les application plus complexes, il est utile d'utiliser les fonctionnalités suplémentaires apportées par le POA.

19.15 Compléments sur l'IDL et l'environnement CORBA

Le langage IDL possède encore un certain nombre de carcatéristiques destinées à faciliter la réalistion d'applications complexes : règles de visibilité de l'IDL, les modules, les constantes, les énumérations, les exceptions, les opérations "oneway", les tableaux, les séquences, les alias (typedef), les unions.

L'environnement CORBA propose également : le type Any, les clauses de précompilation, les contextes d'opérations, les références initiales, le service d'annuaire interopérable, les URLs CORBA.

19.16 Exemple d'application CORBA renvoyant une exception

Calculatrice.idl

```
 1  // -------------------------------------------------
 2  // Chapitre 4  Une application gerant les exceptions
 3  // Jérôme DANIEL
 4  // -------------------------------------------------
 5  // modif MV avril 2001 pour compil avec OmniORB 3.1.2
 6
 7  // DIVISION PAR ZERO
 8  exception DivisionParZero
 9          float op1;
10          float op2;
11  };
12
13  // CALCULATRICE
14
15  interface Calculatrice
16  {
17          float add ( in float nb1, in float nb2 );
18          float div ( in float nb1, in float nb2 )
19                raises ( DivisionParZero );
20  };
21
```

Clientcalc.java

```
 1  // -------------------------------------------------
 2  // Chapitre 4  Une application gerant les exceptions
 3  // Jérôme DANIEL
 4  // -------------------------------------------------
 5  // modif MV avril 2001 pour compil avec OmniORB 3.1.2
 6
 7  // Client.java
 8  package calc;
 9  import org.omg.CORBA.*;
10  import java.io.*;
11
12  /**  Application client pour un objet de type Calculatrice
13   */
14  public class Client
15
16      /**  Point d'entree du programme
17       */
18      public static void main( String args[] )
19      {
20         // Initialise l'ORB   org.omg.CORBA.ORB orb =
21         // org.omg.CORBA.ORB.init(args,null);
22
23         ORB orb = ORB.init(args, new java.util.Properties());
```

Clientcalc.java (suite)

```
24
25          // Recupere la reference de l'objet de type Calculatrice
26          Calculatrice calc = null;
27          org.omg.CORBA.Object obj = null;
28          try
29          {
30              java.io.FileInputStream file = new \
31                              java.io.FileInputStream("ObjectId");
32              java.io.InputStreamReader myInput = new \
33                              java.io.InputStreamReader(file);
34              java.io.BufferedReader reader = new \
35                              java.io.BufferedReader( myInput );
36              String stringTarget = reader.readLine();
37
38              obj = orb.string_to_object(stringTarget);
39          }
40          catch ( java.io.IOException ex )
41          {
42              ex.printStackTrace();
43              System.exit(0);
44          }
45          // Conversion de la reference d'objet
46          calc = CalculatriceHelper.narrow(obj);
47
48          // Enfin, utilise l'objet Calculatrice
49          try
50          {
51              System.out.println(
52                   "5 + 3 = " + calc.add((float)5,(float)3) );
53              System.out.println("5 / 0 = " + calc.div(5,0) );
54          }
55          catch ( DivisionParZero ex )
56          {
57              System.out.println(
58                   "Une division par zero s'est produite...");
59              System.out.println(
60                   "La division demandee etait "+ex.op1+\
61                                   " / "+ex.op2);
62          }
63          catch ( org.omg.CORBA.SystemException ex )
64          {
65              ex.printStackTrace();
66          }
67      }
68  }
```

CalculatriceImpl.java

```
1  // ------------------------------------------------
```

CalculatriceImpl.java (suite)

```
 2  // Chapitre 4  Une application gerant les exceptions
 3  // Jérôme DANIEL
 4  // ------------------------------------------------
 5  // modif MV avril 2001 pour compil avec OmniORB 3.1.2
 6
 7  // CalculatriceImpl.java
 8  package calc;
 9  import org.omg.CORBA.*;
10
11  /**  Implantation de l'interface IDL Calculatrice
12   * @version 1.0
13   */
14  public class CalculatriceImpl extends _CalculatriceImplBase
15  {
16      /**  Operation IDL "add"
17       */
18      public float add(float nb1, float nb2)
19      {
20          System.out.println("Addition = "+nb1+" + "+nb2);
21          return nb1 + nb2;
22      }
23
24      /**  Operation IDL "div"
25       */
26      public float div(float nb1, float nb2)
27          throws DivisionParZero
28      {
29          System.out.println("Division = "+nb1+" / "+nb2);
30
31          if ( nb2 == 0 )
32                  throw new DivisionParZero(nb1,nb2);
33
34          return nb1 / nb2;
35      }
36  }
```

Serveurcalc.java

```
 1  // ------------------------------------------------
 2  // Chapitre 4  Une application gerant les exceptions
 3  // Jérôme DANIEL
 4  // ------------------------------------------------
 5  // modif MV avril 2001 pour compil avec OmniORB 3.1.2
 6
 7  // Serveur.java
 8  package calc;
 9  import org.omg.CORBA.*;
10  import java.io.*;
11  import java.util.*;
12
13  /**  Implantation du serveur pour l'objet Caculatrice
```

Serveurcalc.java (suite)

```
14   * @version 1.0
15   */
16  public class Serveur
17      /**  Point d'entree du programme
18       */
19      public static void main( String [] args )
20      {
21  // set properties necessary to use OB with java 1.2
22  java.util.Properties props = System.getProperties();
23  props.put("org.omg.CORBA.ORBClass", "com.ooc.CORBA.ORB");
24  props.put("org.omg.CORBA.ORBSingletonClass",
25            "com.ooc.CORBA.ORBSingleton");
26  System.setProperties(props);
27  //
28  // Create ORB and BOA
29  //
30  ORB orb = ORB.init(args, props);
31  BOA boa = ((com.ooc.CORBA.ORB)orb).BOA_init(args, props);
32
33         // Initialisation de l'ORB  org.omg.CORBA.ORB orb=
34         // org.omg.CORBA.ORB.init(args,null);
35
36         System.out.println("orb="+orb);
37         // Init du BOA (OpenORB)  org.omg.CORBA.BOA boa=
38         // org.omg.CORBA.BOA.init(orb,args);
39         System.out.println("boa="+boa);
40
41         // Creation de l'objet Horloge
42         CalculatriceImpl calc = new CalculatriceImpl( );
43         System.out.println("calc=");
44         // Connexion et activation au sein du BOA
45         //boa.connect( calc );
46
47         //boa.obj_is_ready( calc );
48
49         // Exportation de la reference de l'objet Calculatrice
50         // dans un fichier
51         try
52         {
53              String ref = orb.object_to_string( calc );
54              java.io.FileOutputStream file = new \
55                           java.io.FileOutputStream("ObjectId");
56              java.io.PrintStream pfile=new
57                           java.io.PrintStream(file);
58              pfile.println(ref);
59              pfile.close();
60         }
61         catch ( java.io.IOException ex )
62         {    ex.printStackTrace( );
63              System.exit(0);
64         }
65
```

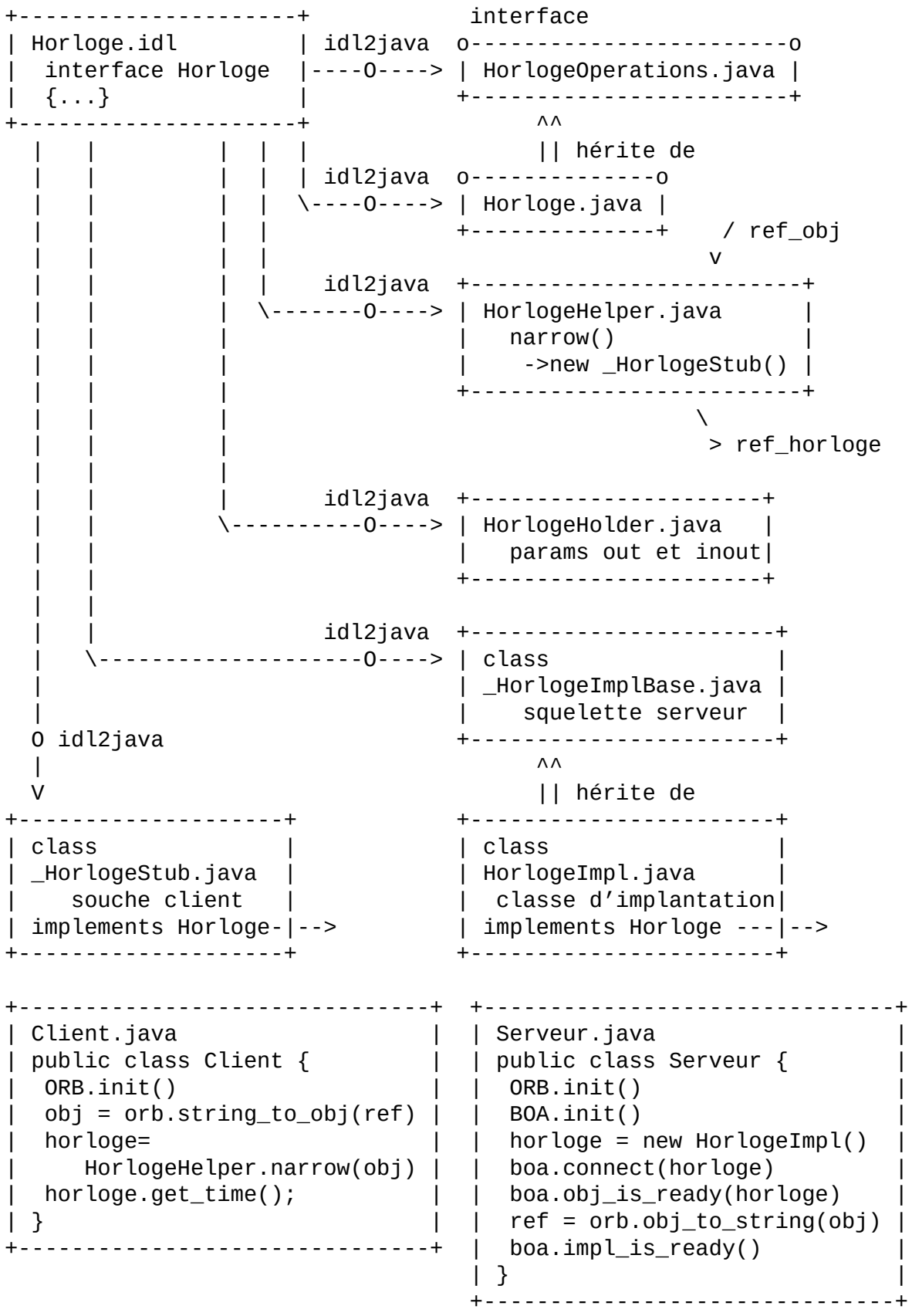
Serveurcalc.java (suite)

```
66          // Mise en attente du serveur de requetes clientes
67          try
68          {       System.out.println(" Le serveur est pret...");
69                  boa.impl_is_ready(null);
70          }
71          catch ( org.omg.CORBA.SystemException ex )
72          {       ex.printStackTrace();
73          }
74      }
75  };
```

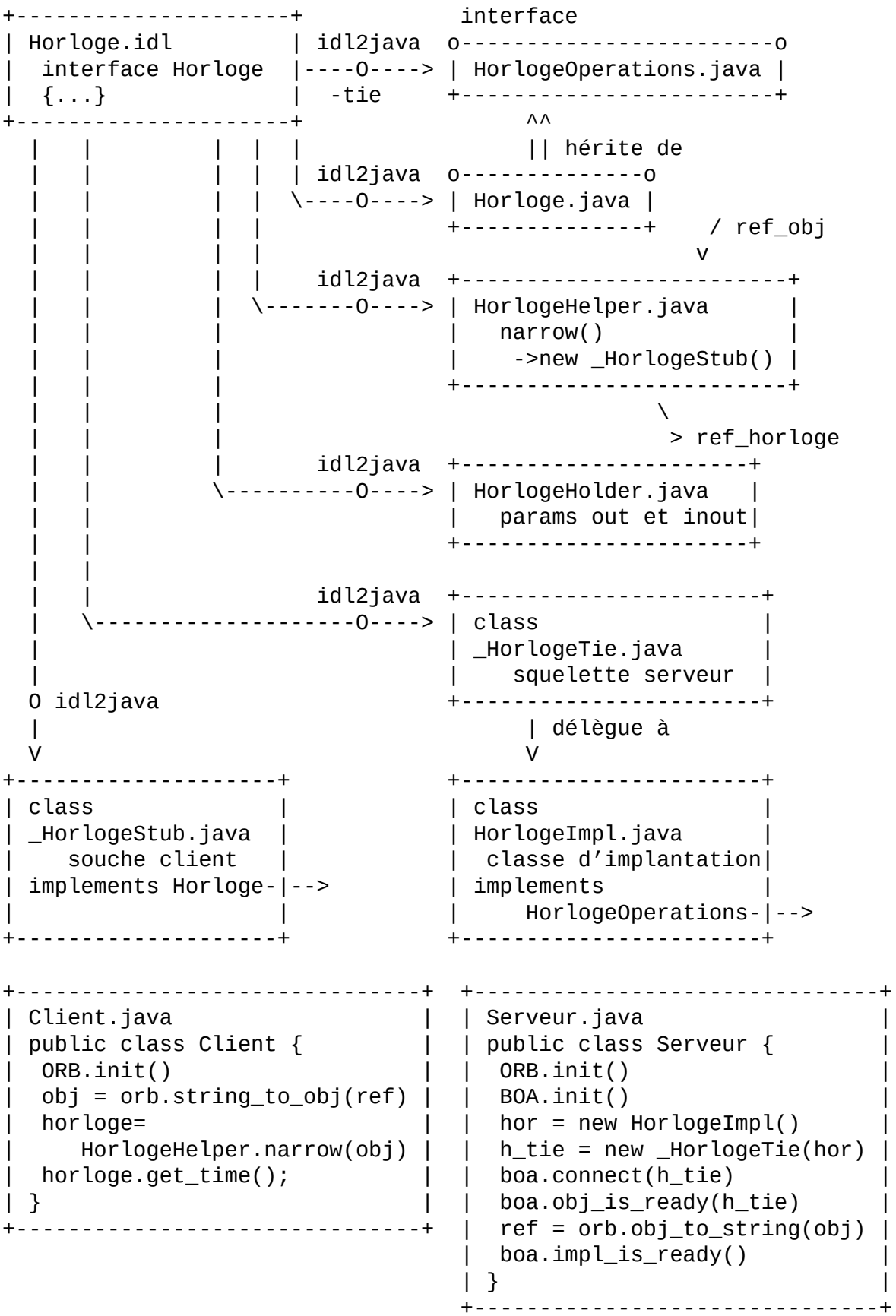
calc-exec.txt

```
 1  // ------------------------------------------------
 2  // MV avril 2001 compil avec OmniORB 3.1.2 sur b116serv
 3  // ------------------------------------------------
 4
 5  calc> setenv CLASSPATH
 6  . :calc :/usr/local/OB-3.1.2/J/OB.jar :\
 7  /usr/local/OB-3.1.2/J/OBNaming.jar
 8
 9  calc> rm nameserv.ref
10  calc> java com.ooc.CosNaming.Server \
11  -i -OAport 12000 > nameserv.ref
12  Running in non-persistent mode.
13
14  calc> java calc/Serveur \
15  -ORBservice NameService \
16  iiop ://b116serv :12000/DefaultNamingContext
17
18  orb=com.ooc.CORBA.ORB@beb2f6
19  boa=com.ooc.CORBA.BOA@1d74a66
20  calc=
21   Le serveur est pret...
22  Addition = 5.0 + 3.0
23  Division = 5.0 / 0.0
24  Addition = 5.0 + 3.0
25  Division = 5.0 / 0.0
26
27  calc> java calc/Client \
28  -ORBservice NameService \
29  iiop ://b116serv :12000/DefaultNamingContext
30
31  5 + 3 = 8.0
32  Une division par zero s'est produite...
33  La division demandee etait 5.0 / 0.0
34
```

19.17 Résumé de la création d'une application CORBA avec approche par héritage



19.18 Résumé de la création d'une application CORBA avec approche par délégation ("tie") pour serveur en Java



La délégation s'opère par association de l'instance hor = new HorlogeImpl() qui contient le

code d'implantation (car implements HorlogeOperations) qui devient le "tie object" associé à une instance de la classe _HorlogeTie par h_tie = new _HorlogeTie(hor), classe tie (c'est elle la classe de délégation), qui est le squelette du serveur et va "déléguer" les appels des clients vers "l'objet tie" (ici "hor") qui contient le code effectif.

Ainsi le code effectif (dans HorlogeImpl) peut-il être invoqué sans que l'on ai besoin de le faire hériter d'une classe générée par l'ORB.

La classe d'implantation est alors "libérée" de son obligation d'hériter du squelette : elle peut donc hériter d'une autre classe d'implantation, ce qui permet de construire une hiérarchie de classes d'implantation pour bénéficier de la réutilisation du code dans ces diverses classes.

Ceci est un problème spécifique à Java qui ne permet pas l'héritage multiple des classes (mais permet d'implémenter plusieurs interfaces).

Explications complémentaires

Dans le schéma par héritage, la classe HorlogeImpl hérite de la classe _HorlogeImplBase, qui elle-même implémente l'interface Horloge. Or, on voudrait pouvoir éviter cet héritage, de façon à conserver la possibilité pour la classe HorlogeImpl d'hériter d'une autre classe d'implantation.

La classe _HorlogeImplBase est une classe abstraite possédant des méthodes publiques. Le but est que ces méthodes soient disponibles sur les objets instanciés de classes héritant de _HorlogeImplBase, car ce sont les méthodes de base d'appel à distance par l'ORB (comme _invoke).

Donc, il faut enregistrer auprès du BOA/POA un objet qui hérite de ces méthodes de base.

Mais l'objet enregistré doit **aussi** posséder les fonctions de l'interface distante.

Or, la classe qui les possède est justement (par définition) la classe HorlogeImpl, celle dont on voudrait éviter de la faire hériter de _HorlogeImplBase pour se garder la possibilité de la faire hériter d'une autre classe.

L'astuce de la méthode par délégation consiste à créer une classe intermédiaire ("tie") qui va hériter de _HorlogeImplBase, qui va aussi posséder toutes les fonctions déclarées dans l'interface distante, et qui sera enregistrée auprès du BOA/POA.

Bien sûr, la classe "tie" ne possède pas l'implémentation des fonctions de l'interface distante (on ne va pas mettre deux fois l'implémentation !). Simplement, chacune des fonctions "d'interface" de la classe "tie" va se contenter, quand elle est appelée, de "renvoyer l'appel" vers la vraie fonction correspondante de la classe d'implémentation HorlogeImpl.

On peut voir dans le listing ci-dessous d'une classe "tie" créée par le compilateur IDL, comment celle-ci enregistre une référence sur un objet instancié de la classe d'implémentation HorlogeImpl pour pouvoir renvoyer les appels sur les fonctions de cet objet.

HorlogeTie.java

```
 1  //
 2  // Interface definition : Horloge
 3  // @author OpenORB Compiler
 4  //
 5  public class HorlogeTie extends _HorlogeImplBase
 6  {
 7          //
 8          // Private reference to implementation object
 9          //
10          private HorlogeOperations _tie;
```

HorlogeTie.java (suite)

```
11          //
12          // Constructor
13          //
14          public HorlogeTie( HorlogeOperations tieObject )
15          {
16                  _tie = tieObject;
17          }
18          //
19          // Get the delegate
20          //
21          public HorlogeOperations _delegate()
22          {
23                  return _tie;
24          }
25          //
26          // Set the delegate
27          //
28          public void _delegate( HorlogeOperations delegate_)
29          {
30                  _tie = delegate_;
31          }
32          //
33          // Operation get_time
34          //
35          public java.lang.String get_time()
36          {
37                  return _tie.get_time();
38          }
39  }
```

SR03 2004 - Cours Architectures Internet - Corba : le POA, fonctions et exemples

20 SR03 2004 - Cours Architectures Internet - Corba : le POA, fonctions et exemples

20.1 CORBA : le POA, fonctions

Le dialogue CORBA [Client–>Stub–>ORB–>..–>ORB–>Skeleton–>Server] peut se résumer par la figure ci-dessous :

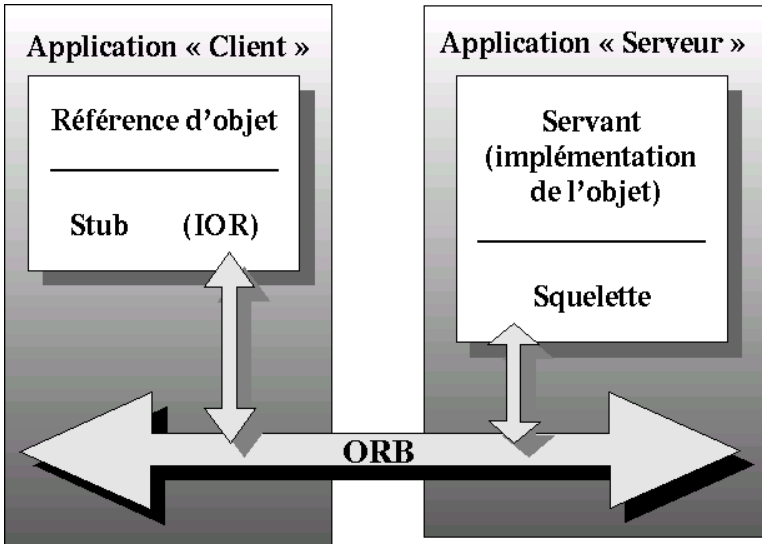


FIG. 205: Architecture Corba

En fait, ce schéma est la vision "fonctionnelle" de CORBA. Il masque deux éléments importants :

– la communication entre le client et le serveur,
– la mise en service et l'appel aux objets depuis l'ORB.

L'ORB est en pratique une bibliothèque linkée avec le client d'une part et le serveur d'autre part. Ces deux instances de la **bibliothèque-ORB** vont communiquer entre elles en échangeant des messages dans un protocole qui leur est propre et qui fait partie de la spécification de Corba. C'est la définition de ce protocole qui permet à deux bibliothèque-ORB de deux constructeurs différents de dialoguer entre elles.

Ce protocole est l'un des protocoles de la famille **GIOP**. Si les machines supportant le client et le serveur utilisent TCP/IP comme liaison réseau, alors le protocole utilisé par les ORBs est "**IIOP**".

Dans l'implantation d'un serveur Corba on trouve l'autre élément supplémentaire : **l'adaptateur d'objets** ("object adapter"). Il en existe deux sortes : **BOA** (Basic Object Adaptor) et **POA** (Portable Object Adaptor). La figure 207 positionne le BOA/POA dans l'architecture CORBA.

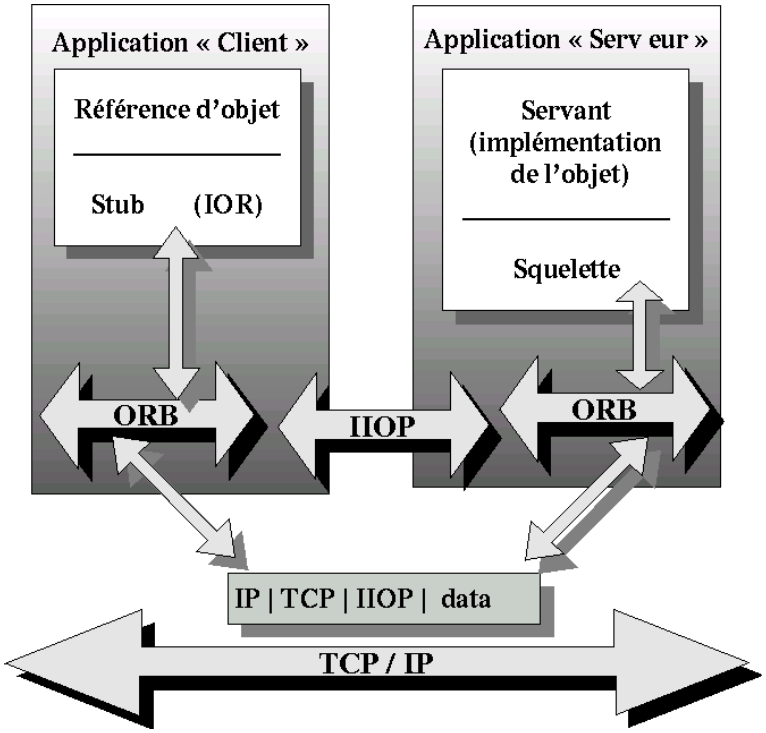


FIG. 206: Les protocoles d'échange dans Corba

Dans les premières verions de Corba, seul le BOA était spécifié. Toutefois, ayant été insuffisamment spécifié, les divers constructeurs ont pris des décisions d'implantation incompatibles entre elles, rendant le code de la partie serveur des applications non portables.

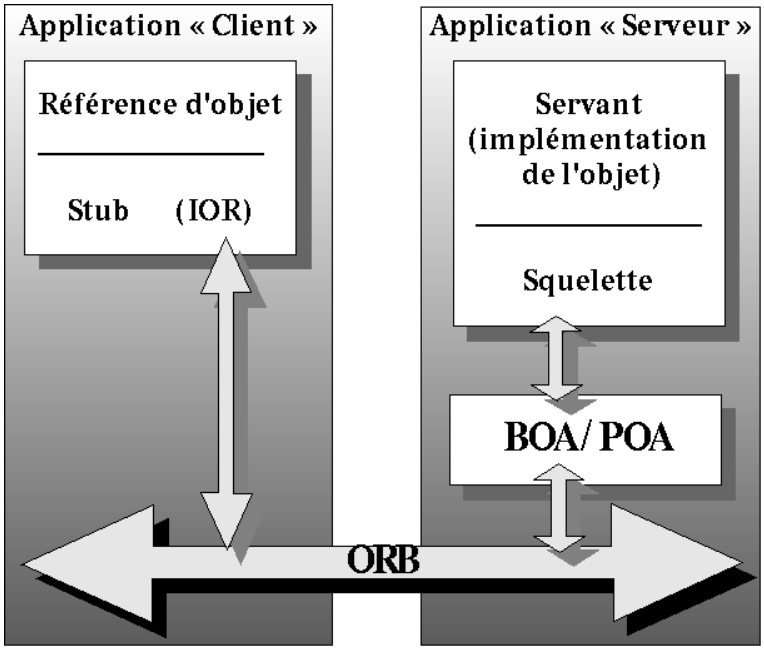


FIG. 207: Le POA dans Corba

En février 1998, la spécification CORBA 2.2 a retiré le BOA au profit du POA. Le POA fournit en plus un ensemble d'outils permettant d'affiner le comportement du serveur.

Que fait le BOA/POA ?

Son rôle est d'assurer la gestion des différents objets qui se situent à l'intérieur de l'application serveur. Il va :

- activer les objets (en fonction des demandes des clients),
- transmettre une invocation d'un client vers un objet particulier,
- créer et détruire des objets,
- déterminer l'état d'un objet,
- ...

Quand on passe du BOA au POA, seul le code du serveur a besoin d'être modifié. Les clients restent identiques.

Avec le POA on utilise une nouvelle terminologie pour prendre en compte les fonctions supplémentaires. Une application utilisant un POA possède donc les éléments suivants :

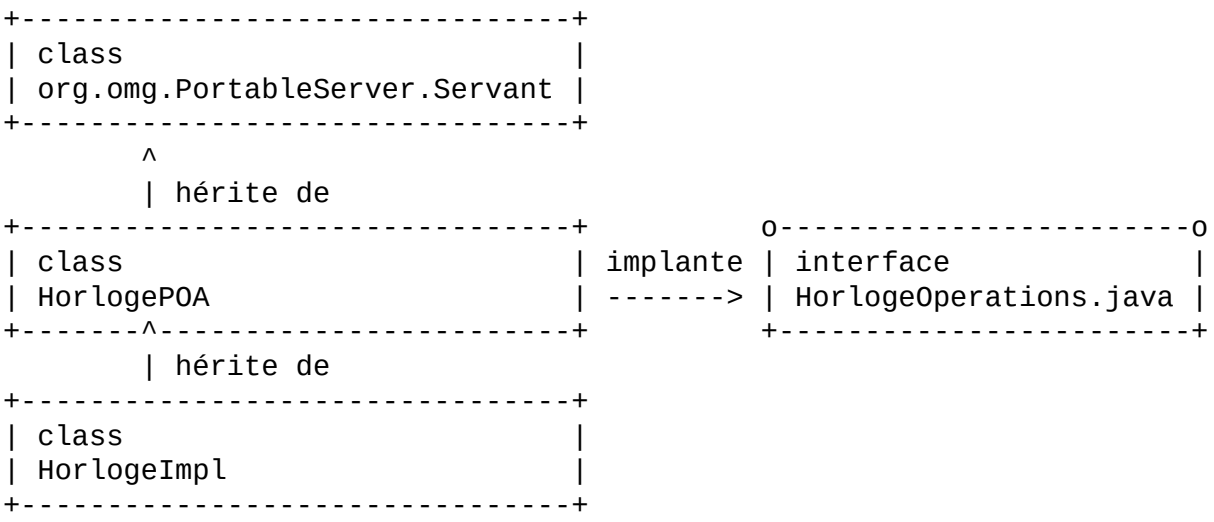
- **servant :** représente l'objet créé sur/par le serveur (anciennement appelé "implémentation de l'objet" ;
- **serveur :** c'est l'application qui loge un ensemble de servants et permet aux applications clientes d'invoquer ces objets ;
- **client :** c'est l'application qui invoque des services proposés par les objets servants ;
- **référence d'objet :** ce que le client voit d'un objet servant par l'intermédiaire de l'interface IDL ; le client utilise une référence d'objet pour accéder à un servant par l'intermédiaire du "stub", de l'ORB, du POA et du squelette qui, ensemble, masquent au client la non-proximité de l'objet réel. Cette référence est appelée **IOR** (Interoperable Object Reference).

20.2 Un servant avec une approche par héritage

Dans le cas du POA, les fichiers générés par le compilateur IDL sont un peu différents. On trouve pour XXX.idl :

- _XXXStub.java : la souche du client,
- XXX.java : l'interface XXX,
- XXXOperations.java : l'interface d'opérations de XXX,
- XXXPOA.java : le squelette côté serveur,
- XXXHelper.java et XXXHolder.java les classes Helper et Holder de XXX.

L'implantation du servant va hériter du squelette. Mais ici, il y a une différence fondamentale par rapport au BOA : le squelette n'implante pas l'interface XXX, mais **directement XXXOperations**.



La première ligne de la figure fait partie de l'ORB.

La deuxième ligne est générée par la compilation de l'IDL.

La troisième ligne est à écrire par l'auteur de l'application.

Le servant (la classe XXXImpl) n'a plus de lien avec l'interface XXX.

Le but est d'introduire une séparation entre la définition fonctionnelle de l'objet Corba (symbolisée par l'interface XXX) et l'implémentation qui en est faite (XXXImpl). Ceci, afin de pouvoir proposer différentes politiques de gestion des instanciations des objets servants au sein du serveur.

Ceci sera fait par les **politiques du POA** :

– politique de threading,
– politique de durée de vie,
– politique d'unicité d'identité d'objet,
– politique d'assignation d'identité,
– politique de rétention des servants,
– politique de gestion des requêtes,
– politique d'activation implicite.

Pour cela on va modifier l'association **référence <–> instance**.

Avec l'ancien BOA, on avait une relation bijective entre la référence d'objet passée/utilisée par le client et l'instance d'objet crée/appelée par le serveur :

```
Serveur
 xxx = new XXXImpl();
 ref = object_to_string(xxx);
  |
  \----------- "ref" ---------> Client
                                remotexxx = string_to_object(ref);
                                remotexxx.operation();
class XXXImpl                             |
public void operation () <--- invoque --/
```

Pour chaque "**new XXXImpl()**" il y a une référence et une seule.

Dans le cas du POA, on veut introduire une table d'indirection, gérée par le POA, et distinguer **trois notions** :
référence d'objet <–> Object-ID <–> Servant (instance).

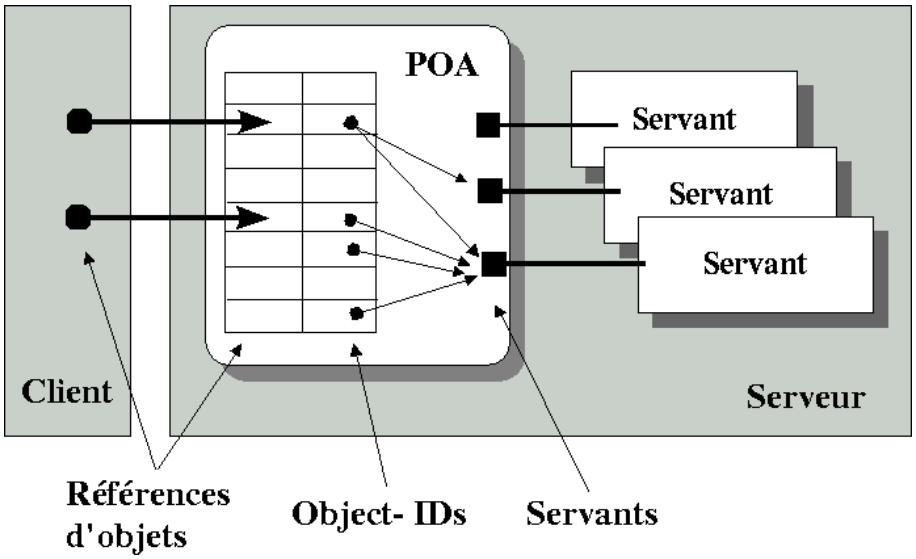
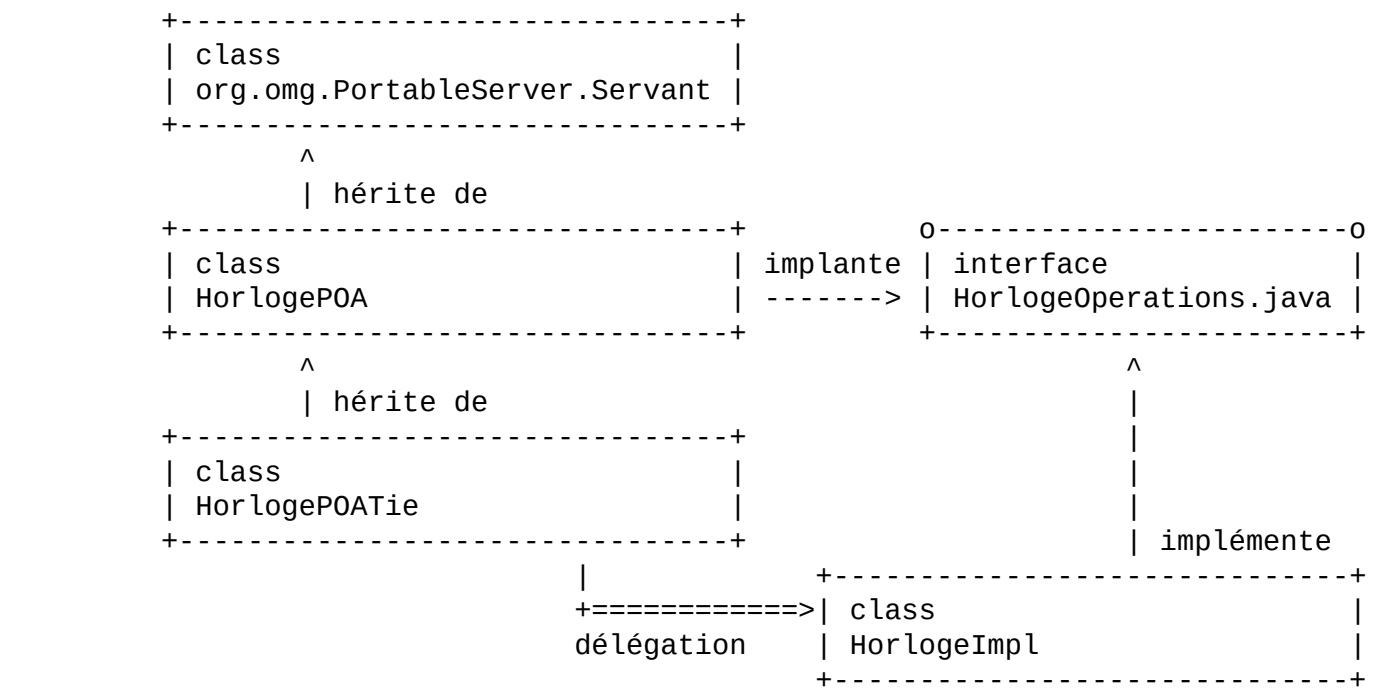


FIG. 208: Le POA et les servants

20.3 Un servant avec une approche par délégation

Pour les servants développés en Java, on peut, pour les mêmes raisons que dans le BOA vouloir utiliser l'approche par délégation.



La première ligne de la figure fait partie de l'ORB.

La deuxième et la troisième ligne sont générées par la compilation de l'IDL.

La quatrième ligne est à écrire par l'auteur de l'application.

L'utilisation se fait en passant un référence sur une instance de XXXImpl à une instance de la classe de délégation :

```
HorlogeImpl   hor = new HorlogeImpl();
HorlogePOATie tie = new HorlogePOATie(hor);
```

20.4 RootPOA et arborescence de POA

L'une des nouveautés du POA est que l'on puisse en avoir plusieurs, organisés en une hiérarchie à partir du RootPOA. Chaque POA peut avoir plusieurs POA-fils.

L'intérêt est lié au fait que chaque POA peut avoir des règles ou politiques de fonctionnement différentes.

Chaque POA gère un ensemble de servants et leur transmet les invocations en provenance des clients. La façon dont les servants et les invocations sont gérées est totalement paramétrable.

Chaque POA est géré par un **gestionnaire de POA**. Il y a au moins un gestionnaire de POA et il peut être unique et gérer tous les POAs du serveur. Le gestionnaire de POA ("**POAManager**") contrôle le traitement des requêtes en provenance des clients à destination du POA. Suivant l'état dans lequel il est, les requêtes seront rejetées, mises en attente ou traitées. Avant que des requêtes puissent être traitées il faudra donc faire passer le POA dans l'état actif.

La classe org.omg.PortableServeur fournit les méthodes pour convertir de l'un vers l'autre les trois types d'identifiants d'objets et pour associer un servant à un POA.

20.5 Exemple de serveur utilisant le POA

On reprends ici l'exemple "horloge" du chapitre 19, en transformant la partie serveur en utilisant le POA.

code de Horloge.idl et Client.java :

```
// c'est le même pour le BOA et le POA
// le passage au POA n'affecte que le serveur
```

code de HorlogeImpl.java :

```
// c'est presque le même pour le BOA et le POA
< public class HorlogeImpl extends HorlogePOA
---
> public class HorlogeImpl extends _HorlogeImplBase
```

Le code de serveurpoa.java est assez différent :

serveurpoa.java

```
 1  // ------------------------------------------------
 2  // Chapitre 2  Notre première application CORBA
 3  // Jérôme DANIEL
 4  // ------------------------------------------------
 5  // Modifs de MV pour init avec le POA
 6  // Serveurpoa.java
 7  /** Implantation du serveur pour l'objet Horloge
 8   *  @version 1.0
 9   */
10  public class Serveur
11  {
12     public static void main( String [] args )
13     { // rappel du code "BOA" laissé en commentaire
14       // pour comparaison :
15       // Initialisation de l'ORB puis du BOS
16       // org.omg.CORBA.ORB orb =
17       //               org.omg.CORBA.ORB.init( args, null);
18       // System.out.println("orb="+orb);
19       // org.omg.CORBA.BOA boa =
20       //               org.omg.CORBA.BOA.init( orb, args);
21       // System.out.println("boa="+boa);
22       // Creation de l'objet Horloge
23       // HorlogeImpl horloge = new HorlogeImpl( );
24       // System.out.println("horloge="+horloge);
25       // Connexion et activation au sein du BOA
26       // boa.connect( horloge );
27       // boa.obj_is_ready( horloge );
28
29          // Initialisation de l'ORB
```

serveurpoa.java (suite)

```
30          org.omg.CORBA.ORB orb =
31                       org.omg.CORBA.ORB.init(args, null);
32          System.out.println("orb="+orb);
33
34          // Recuperation du RootPOA
35          org.omg.CORBA.Object objPoa = null;
36          org.omg.PortableServer.POA rootPOA = null;
37          try
38          { objPoa =
39                   orb.resolve_initial_references("RootPOA");
40          }
41          catch ( org.omg.CORBA.ORBPackage.InvalidName ex )
42          {}
43
44          System.out.println("objPoa="+objPoa);
45          rootPOA =
46              org.omg.PortableServer.POAHelper.narrow(objPoa);
47          System.out.println("rootPoa="+rootPOA);
48
49          // Creation du servant HorlogeImpl
50          // Creation de l'objet Horloge
51          HorlogeImpl horloge = new HorlogeImpl( );
52          System.out.println("horloge créée");
53
54          // Activation du servant
55          byte[] servantId = null;
56          try
57          {   servantId = rootPOA.activate_object(horloge);
58          }
59          catch (
60            org.omg.PortableServer.POAPackage.WrongPolicy ex)
61          {    ex.printStackTrace();
62          }
63          catch (
64          org.omg.PortableServer.POAPackage.ServantAlreadyActive ex)
65          {    ex.printStackTrace();
66          }
67          System.out.println("horloge activé");
68
69          // Recuperation de la reference d'objet associee
70          // au servant
71          org.omg.CORBA.Object obj = null;
72          try
73          {     obj = rootPOA.id_to_reference(servantId);
74          }
75          catch (
76            org.omg.PortableServer.POAPackage.WrongPolicy ex )
77          {     ex.printStackTrace();
78          }
79          catch (
80            org.omg.PortableServer.POAPackage.ObjectNotActive ex )
81          {     ex.printStackTrace();
```

serveurpoa.java (suite)

```
 82          }
 83          System.out.println("ref obj servant");
 84
 85          // Exportation de la reference de l'objet Horloge
 86          // dans un fichier
 87          try
 88          {
 89              String ref = orb.object_to_string( obj );
 90              java.io.FileOutputStream file = new \
 91                          java.io.FileOutputStream("ObjectId");
 92              java.io.PrintStream pfile=new
 93                          java.io.PrintStream(file);
 94              pfile.println(ref);
 95              pfile.close();
 96          }
 97          catch ( java.io.IOException ex )
 98          {
 99              ex.printStackTrace( );
100              System.exit(0);
101          }
102
103          // Mise en attente du serveur de requetes clientes
104          // try
105          // {    System.out.println(" Le serveur est pret...");
106          //      boa.impl_is_ready( );
107          // }
108          // catch ( org.omg.CORBA.SystemException ex )
109          // {  ex.printStackTrace(); }
110
111          try
112          {       // Activation du POA manager
113                  rootPOA.the_POAManager().activate();
114                  System.out.println("Le serveur est pret...");
115
116                  // Attente des invocations clientes
117                  orb.run();
118          }
119          catch ( java.lang.Exception ex )
120          {
121                  System.out.println("Exception interceptee");
122                  ex.printStackTrace();
123          }
124      }
125  };
```

20.6 Exemple de client et serveur utilisant le POA et l'approche par héritage

Les fichiers de l'exemple

```
Les fichiers sources, avant compilation :
$ ls
Banque.idl       Client.java
BanqueImpl.java  CompteImpl.java  Serveur.java

Compilation :
idl2java Banque.idl -nopackage -poa
export CLASSPATH=$CLASSPATH:. (unix bash)
javac *.java

Les fichiers après les compilations :
$ ls
Banque.class             compte_bancaire.class
BanqueHelper.class       compte_bancaireHelper.class
BanqueHelper.java        compte_bancaireHelper.java
BanqueHolder.class       compte_bancaireHolder.class
BanqueHolder.java        compte_bancaireHolder.java
Banque.idl               compte_bancaire.java
BanqueImpl.class         compte_bancaireOperations.class
BanqueImpl.java          compte_bancaireOperations.java
Banque.java              compte_bancairePOA.class
BanqueOperations.class   compte_bancairePOA.java
BanqueOperations.java    _compte_bancaireStub.class
BanquePOA.class          _compte_bancaireStub.java
BanquePOA.java           CompteImpl.class
_BanqueStub.class        CompteImpl.java
_BanqueStub.java         ObjectId
Client.class             Serveur.class
Client.java              Serveur.java
```

Banque.idl

```
 5  // COMPTE BANCAIRE
 6
 7  interface compte_bancaire
 8  {
 9          readonly attribute string nom;
10          attribute string adresse;
11          readonly attribute float solde;
12
13          void debit( in float montant );
14          void credit( in float montant );
15  };
16
17  // BANQUE
18
```

Banque.idl (suite)

```
19  interface Banque
20  {
21    compte_bancaire creer_nouveau_compte(
22                  in string nom, in string adresse );
23
24    void ajouter_interet( in compte_bancaire cpt );
25  };
```

Client.java

```
 1  // Chapitre 7  Approche par heritage avec le POA
 2  // Jérôme DANIEL -- modifs MV avril 2001
 3
 4  // Client.java
 5  /**
 6   * Implantation du client de l'objet Banque
 7   * @version 1.0
 8   */
 9  public class Client
10  {
11     /**  Point d'entree du programme
12      */
13     public static void main( String [] args )
14     {
15        /**
16         * Initialisation de l'ORB
17         */
18        org.omg.CORBA.ORB orb =
19                  org.omg.CORBA.ORB.init( args, null );
20
21        /**
22         * Recuperation de la reference de l'objet Banque
23         * depuis un fichier
24         */
25        org.omg.CORBA.Object obj = null;
26        try
27        {  java.io.FileInputStream file =
28                        new java.io.FileInputStream("ObjectId");
29           java.io.InputStreamReader input =
30                        new java.io.InputStreamReader( file );
31           java.io.BufferedReader reader =
32                        new java.io.BufferedReader(input);
33           String ref = reader.readLine();
34           obj = orb.string_to_object(ref);
35        }
36        catch ( java.io.IOException ex )
37        {
38           ex.printStackTrace( );
39           System.exit(0);
40        }
41        try
```

Client.java (suite)

```
42        {  // Conversion de la reference
43           Banque bank = BanqueHelper.narrow( obj );
44           // Creation d'un nouveau compte
45           compte_bancaire cpt = bank.creer_nouveau_compte(\
46                       "Pierre DUPONT", "1 rue des granges");
47           // Manipule le compte
48           cpt.credit(200);
49           System.out.println("Compte = " + cpt.nom() );
50           System.out.println("Solde = " + cpt.solde() );
51           System.out.println("");
52
53           cpt.debit(50);
54           System.out.println("Compte = " + cpt.nom() );
55           System.out.println("Solde = " + cpt.solde() );
56
57           // Ajoute les interets
58           bank.ajouter_interet( cpt );
59
60           // Affiche la solde apres ajout des interets
61           System.out.println(
62                  "Solde avec interets = " + cpt.solde() );
63        }
64        catch ( org.omg.CORBA.SystemException ex )
65        {
66           ex.printStackTrace( );
67        }
68     }
69  }
```

BanqueImpl.java

```
 1  // Chapitre 7  Approche par heritage avec le POA
 2  // Jérôme DANIEL
 3  // -----------------------------------------------
 4  // BanqueImpl.java
 5  /**
 6   * Implantation de l'interface " Banque "
 7   * @version 1.0
 8   */
 9  public class BanqueImpl extends BanquePOA
10  {
11      // Taux d'interet a appliquer
12      private final float taux_interet = ( float ) 1.0225;
13
14      /**
15       * Operation IDL " creer_nouveau_compte "
16       * Cette opération crèee un nouvel objet et son délégué
17       * qu'elle connecte et active avant de le retourner
18       */
19      public compte_bancaire creer_nouveau_compte(
20                           String nom, String adresse )
```

BanqueImpl.java (suite)

```
21      {
22          CompteImpl cpt = new CompteImpl( nom, adresse, 0 );
23
24          try
25          {  _poa().activate_object( cpt );
26             // Premier solution
27             //
28             return cpt._this();
29
30             // Deuxieme solution
31             //
32             // org.omg.CORBA.Object obj
33             //    = _poa().servant_to_reference( cpt );
34             // return compte_bancaireHelper.narrow( obj );
35          }
36          catch (  java.lang.Exception ex )
37          {
38             ex.printStackTrace();
39             System.exit(0);
40          }
41
42          return null;
43      }
44
45      /**
46       * Operation IDL " supprimer_compte "   Cette operation
47       * recupère tout d'abord la classe d'implantation pour y
48       * effectuer d'éventuels traitements, puis elle désactive
49       * l'objet
50       */
51      public void ajouter_interet( compte_bancaire cpt )
52      {
53          org.omg.PortableServer.Servant srv = null;
54          try
55          {
56                  srv = _poa().reference_to_servant( cpt );
57          }
58          catch ( java.lang.Exception ex )
59          {
60                  ex.printStackTrace();
61                  System.exit(0);
62          }
63          CompteImpl impl = ( CompteImpl  ) srv;
64          impl._solde = impl._solde * taux_interet;
65      }
66  }
```

CompteImpl.java

```
1  // Chapitre 7  Approche par heritage avec le POA
2  // Jérôme DANIEL
```

CompteImpl.java (suite)

```
 3  // CompteImpl.java
 4  /**
 5   * Implantation de l'interface "compte_bancaire"
 6   */
 7  public class CompteImpl extends compte_bancairePOA
 8  {
 9      // Les attributs de la classe CompteImpl.
10      // Ces elements ne sont pas accessibles aux clients sauf
11      // si des operations IDL ou attributs IDL ont ete definis
12      // afin de les manipuler
13      private String _titulaire;
14
15      private String _adresse;
16
17      public float _solde;
18
19      /** Constructeur
20       * On crèe un nouveau compte bancaire en spécifiant
21       * à la fois le nom, l'adresse ainsi que son solde
22       */
23      public CompteImpl(
24              String nom, String adresse, float solde )
25      {
26              _titulaire = nom;
27              _adresse = adresse;
28              _solde = solde;
29      }
30
31      /** Attribut IDL "nom" en lecture
32       */
33      public String nom( )
34      {
35              return _titulaire;
36      }
37      /** Attribut IDL "adresse" en lecture
38       */
39      public String adresse( )
40      {
41              return _adresse;
42      }
43      /** Attribut IDL "adresse" en ecriture
44       */
45      public void adresse( String value )
46      {
47              _adresse = value;
48      }
49      /** Attribut IDL "solde" en lecture
50       */
51      public float solde( )
52      {
53              return _solde;
54      }
```

CompteImpl.java (suite)

```
55      /** Operation IDL " debit "
56       */
57      public void debit( float montant )
58      {
59              _solde -= montant;
60      }
61      /** Operation IDL " credit "
62       */
63      public void credit ( float montant )
64      {
65              _solde += montant;
66      }
67  }
```

Serveur.java

```
 1  // Serveur.java   Approche par heritage avec le POA
 2  /** Cette classe correspond a l'implantation du serveur @v1.0 */
 3  public class Serveur
 4  {
 5    /**  Point d'entree de l'application */
 6    public static void main( String args[] )
 7    { // Initialisation de l'ORB
 8       org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
 9       // Recuperation du RootPOA
10       org.omg.CORBA.Object objPoa = null;
11       org.omg.PortableServer.POA rootPOA = null;
12       try
13       {    objPoa = orb.resolve_initial_references("RootPOA");
14       }
15       catch ( org.omg.CORBA.ORBPackage.InvalidName ex )
16       {}
17
18       rootPOA = org.omg.PortableServer.POAHelper.narrow(objPoa);
19       // Creation du servant BanqueImpl
20       BanqueImpl bank = new BanqueImpl();
21
22       // Activation du servant
23       byte[] servantId = null;
24       try{    servantId = rootPOA.activate_object(bank);
25       }
26       catch ( org.omg.PortableServer.POAPackage.WrongPolicy ex )
27       {    ex.printStackTrace();
28       }
29       catch (
30        org.omg.PortableServer.POAPackage.ServantAlreadyActive ex)
31       {
32               ex.printStackTrace();
33       }
34       // Recuperation de la reference d'objet associee au servant
35       org.omg.CORBA.Object obj = null;
```

Serveur.java (suite)

```
36      try
37      {    obj = rootPOA.id_to_reference(servantId);
38      }
39      catch ( org.omg.PortableServer.POAPackage.WrongPolicy ex )
40      {    ex.printStackTrace();
41      }
42      catch (org.omg.PortableServer.POAPackage.ObjectNotActive ex)
43      {    ex.printStackTrace();
44      }
45      // Exportation de la reference au sein d'un fichier
46      String reference = orb.object_to_string(obj);
47      try{
48          java.io.FileOutputStream file = new \
49                              java.io.FileOutputStream("ObjectId");
50          java.io.PrintStream pfile=new java.io.PrintStream(file);
51          pfile.println(reference);
52          file.close();
53      }
54      catch ( java.io.IOException ex )
55      {   System.out.println("Impossible créer fichier ObjectId");
56      }
57      try{// Activation du POA manager
58          rootPOA.the_POAManager().activate();
59          System.out.println("Le serveur est pret...");
60
61          // Attente des invocations clientes
62          orb.run();
63      }
64      catch ( java.lang.Exception ex )
65      {   System.out.println("Exception interceptee");
66          ex.printStackTrace();
67      }
68    }
69  }
```

20.7 Exemple de client et serveur utilisant le POA et l'approche par délégation (tie)

Les fichiers de l'exemple

```
Les fichiers sources, avant compilation :
Banque.idl       Client.java
BanqueImpl.java  CompteImpl.java  Serveur.java

Les fichiers créés:
BanqueHelper.java      compte_bancaireHelper.java
BanqueHolder.java      compte_bancaireHolder.java
Banque.java            compte_bancaire.java
BanqueOperations.java  compte_bancaireOperations.java
BanquePOA.java         compte_bancairePOA.java
BanquePOATie.java      compte_bancairePOATie.java
_BanqueStub.java       _compte_bancaireStub.java
```

Banque.idl

```
 4  // ------------------------------------------------
 5  // COMPTE BANCAIRE
 6
 7  interface compte_bancaire
 8  {
 9          readonly attribute string nom;
10          attribute string adresse;
11          readonly attribute float solde;
12
13          void debit( in float montant );
14          void credit( in float montant );
15  };
16
17  // BANQUE
18
19  interface Banque
20  {
21     compte_bancaire creer_nouveau_compte(
22                    in string nom, in string adresse );
23
24     float ajouter_interet( in compte_bancaire cpt );
25  };
```

Client.java

```
 1  // ------------------------------------------------
 2  // Chapitre 7  Approche par heritage avec le POA
 3  // Jérôme DANIEL
 4  // ------------------------------------------------
```

Client.java (suite)

```
 5  // Client.java
 6  /**
 7   * Implantation du client de l'objet Banque
 8   * @version 1.0
 9   */
10  public class Client
11  {
12    private static float taux;
13    /**  Point d'entree du programme
14     */
15    public static void main( String [] args )
16    {
17       /**
18        * Initialisation de l'ORB   */
19       org.omg.CORBA.ORB orb =
20                   org.omg.CORBA.ORB.init( args, null );
21
22       /**
23        * Recuperation de la reference de l'objet Banque
24        * depuis un fichier
25        */
26       org.omg.CORBA.Object obj = null;
27       try
28       {
29          java.io.FileInputStream file =
30                      new java.io.FileInputStream("ObjectId");
31          java.io.InputStreamReader input =
32                      new java.io.InputStreamReader( file );
33          java.io.BufferedReader reader =
34                      new java.io.BufferedReader(input);
35          String ref = reader.readLine();
36          obj = orb.string_to_object(ref);
37       }
38       catch ( java.io.IOException ex )
39       {
40          ex.printStackTrace( );
41          System.exit(0);
42       }
43
44       try
45       {
46          // Conversion de la reference
47          Banque bank = BanqueHelper.narrow( obj );
48
49          // Creation d'un nouveau compte
50          compte_bancaire cpt = bank.creer_nouveau_compte(\
51                      "Pierre DUPONT", "1 rue des granges");
52
53          // Manipule le compte
54          cpt.credit(200);
55          System.out.println("Compte = " + cpt.nom() );
56          System.out.println("Solde = " + cpt.solde() );
```

Client.java (suite)

```
57
58          cpt.debit(50);
59          System.out.println("Compte = " + cpt.nom() );
60          System.out.println("Solde = " + cpt.solde() );
61
62          cpt.credit(10);
63          System.out.println("Solde = " + cpt.solde() );
64          cpt.debit(50);
65          System.out.println("Solde = " + cpt.solde() );
66
67          // Creation d'un nouveau compte
68          compte_bancaire cpt2 = bank.creer_nouveau_compte(\
69                                  "Jean Durand", "une adresse");
70          cpt2.credit(100);
71          System.out.println("Solde cpt2 = " + cpt2.solde() );
72          cpt2.debit(50);
73          System.out.println("Solde cpt2 = " + cpt2.solde() );
74
75          // Ajoute les interets
76          taux = bank.ajouter_interet( cpt );
77          System.out.println("taux = " + taux );
78          // Affiche le solde apres ajout des interets
79          System.out.println(
80                  "Solde avec interets = " + cpt.solde() );
81       }
82       catch ( org.omg.CORBA.SystemException ex )
83       {
84          ex.printStackTrace( );
85       }
86
87     } //main
88  }
```

BanqueImpl.java

```
 1  // -------------------------------------------------
 2  // Chapitre 7  Approche par delegation avec le POA
 3  // Jérôme DANIEL
 4  // -------------------------------------------------
 5  // modifs MV avril 2001
 6
 7  // BanqueImpl.java
 8  /**
 9   * Implantation de l'interface " Banque " @version 1.0
10   */
11  public class BanqueImpl implements BanqueOperations
12  {
13      // Taux d'interet a appliquer
14      private final float taux_interet = ( float ) 1.0225;
15
16      // Reference vers le rootPOA
```

BanquePOA.java (suite)

```
17      private org.omg.PortableServer.POA rootPOA;
18
19      /** Constructeur
20       */
21      public BanqueImpl( org.omg.PortableServer.POA root )
22      {
23              rootPOA = root;
24      }
25
26      /**
27       * Operation IDL " creer_nouveau_compte "
28       *
29       * Cette operation crèee un nouvel objet et son délégué
30       * qu'elle connecte et active avant de le retourner
31       */
32      public compte_bancaire creer_nouveau_compte
33                                  ( String nom, String adresse)
34      {
35        CompteImpl cpt = new CompteImpl( nom, adresse, 0 );
36
37        compte_bancairePOATie tie = new
38                                 compte_bancairePOATie( cpt );
39        try
40        {
41            rootPOA.activate_object( tie );
42            org.omg.CORBA.Object obj =
43                           rootPOA.servant_to_reference( tie );
44            return compte_bancaireHelper.narrow( obj );
45        }
46        catch (  java.lang.Exception ex )
47        {
48            ex.printStackTrace();
49            System.exit(0);
50        }
51        return null;
52      }
53
54      /**
55       * Operation IDL " supprimer_compte "
56       *
57       * Cette opération recupere tout d'abord la classe
58       * d'implantation pour y effectuer d'eventuels
59       * traitements, puis elle desactive l'objet
60       */
61      public float ajouter_interet( compte_bancaire cpt )
62      {
63         org.omg.PortableServer.Servant srv = null;
64         System.out.println("ajouter interet" );
65         try
66         {
67             srv = rootPOA.reference_to_servant( cpt );
68             System.out.println("got servant" );
```

BanqueImpl.java (suite)

```
69          }
70          catch ( java.lang.Exception ex )
71          {
72              System.out.println("err in ref_to_servant" );
73              //ex.printStackTrace();
74              return taux_interet;
75          }
76          compte_bancairePOATie tie =
77                              (compte_bancairePOATie) srv;
78
79          CompteImpl impl = ( CompteImpl) tie._delegate( );
80
81          impl._solde = impl._solde * taux_interet;
82          return taux_interet;
83      }//ajouter_interet
84  }
```

compte_bancairePOATie.java

```
 1  // compte_bancairePOATie.java
 2  // généré par le compilateur IDL, mets en place
 3  // le mécanisme de délégation (objet tie)
 4  //--------------------------------------------------------
 5  // Interface definition : compte_bancaire
 6  // @author OpenORB Compiler
 7  //
 8  public class compte_bancairePOATie extends compte_bancairePOA
 9
10          //
11          // Private reference to implementation object
12          //
13          private compte_bancaireOperations _tie;
14
15          //
16          // Private reference to POA
17          //
18          private org.omg.PortableServer.POA _poa;
19
20          //
21          // Constructor
22          //
23          public compte_bancairePOATie(
24                  compte_bancaireOperations tieObject )
25          {
26                  _tie = tieObject;
27          }
28
29          //
30          // Constructor
31          //
32          public compte_bancairePOATie
```

compte_bancairePOATie.java (suite)

```
33                          ( compte_bancaireOperations tieObject, \
34                                  org.omg.PortableServer.POA poa )
35          {
36                  _tie = tieObject;
37                  _poa = poa;
38          }
39
40          //
41          // Get the delegate
42          //
43          public compte_bancaireOperations _delegate()
44          {
45                  return _tie;
46          }
47
48          //
49          // Set the delegate
50          //
51          public void _delegate(
52                          compte_bancaireOperations delegate_)
53          {
54                  _tie = delegate_;
55          }
56
57          //
58          //
59          //
60          public org.omg.PortableServer.POA _default_POA()
61          {
62                  if ( _poa != null )
63                      return _poa;
64                  else
65                      return super._default_POA();
66          }
67
68          //
69          // Read accessor for nom attribute
70          //
71          public java.lang.String nom()
72          {
73                  return _tie.nom();
74          }
75
76          //
77          // Read accessor for adresse attribute
78          //
79          public java.lang.String adresse()
80          {
81                  return _tie.adresse();
82          }
83
84          //
```

compte_bancairePOATie.java (suite)

```
 85          // Write accessor for adresse attribute
 86          //
 87          public void adresse( java.lang.String value )
 88          {
 89                  _tie.adresse(value);
 90          }
 91
 92          //
 93          // Read accessor for solde attribute
 94          //
 95          public float solde()
 96          {
 97                  return _tie.solde();
 98          }
 99
100          //
101          // Operation debit
102          //
103          public void debit(float montant)
104          {
105                  _tie.debit( montant);
106          }
107
108          //
109          // Operation credit
110          //
111          public void credit(float montant)
112          {
113                  _tie.credit( montant);
114          }
115  }
```

CompteImpl.java

```
 1  // ------------------------------------------------
 2  // Chapitre 7  Approche par delegationavec le POA
 3  // Jérôme DANIEL
 4  // ------------------------------------------------
 5  // CompteImpl.java
 6  /**
 7   * Implantation de l'interface " compte_bancaire "
 8   * @version 1.0
 9   */
10  public class CompteImpl implements compte_bancaireOperations
11  {
12          // Les attributs de la classe CompteImpl.
13     // Ces elements ne sont pas accessibles aux clients sauf si
14     // des operations IDL ou attributs IDL ont ete definis afin
15     // de les manipuler
16          private String _titulaire;
17          private String _adresse;
```

CompteImpl.java (suite)

```
18          public float _solde;
19
20          /**  Constructeur
21           * On cree un nouveau compte bancaire en specifiant
22           * à la fois le nom, l'adresse ainsi que son solde
23           */
24          public CompteImpl( String nom,
25                             String adresse, float solde )
26          {
27                  _titulaire = nom;
28                  _adresse = adresse;
29                  _solde = solde;
30          }
31
32          /**
33           * Attribut IDL "nom" en lecture
34           */
35          public String nom( )
36          {
37                  return _titulaire;
38          }
39
40          /**
41           * Attribut IDL "adresse" en lecture
42           */
43          public String adresse( )
44          {
45                  return _adresse;
46          }
47
48          /**
49           * Attribut IDL "adresse" en ecriture
50           */
51          public void adresse( String value )
52          {
53                  _adresse = value;
54          }
55
56          /**
57           * Attribut IDL "solde" en lecture
58           */
59          public float solde( )
60          {
61                  return _solde;
62          }
63
64          /**
65           * Operation IDL " debit "
66           */
67          public void debit( float montant )
68          {
69                  _solde -= montant;
```

CompteImpl.java (suite)

```
70          }
71          /**
72           * Operation IDL " credit "
73           */
74          public void credit ( float montant )
75          {
76                  _solde += montant;
77          }
78  }
```

Serveur.java

```
 1  // -------------------------------------------------
 2  // Chapitre 7  Approche par delegation avec le POA
 3  // Jérôme DANIEL
 4  // -------------------------------------------------
 5  // modifs MV avril 2001
 6
 7  // Serveur.java
 8  /**
 9   * Cette classe correspond a l'implantation du serveur
10   * @version 1.0
11   */
12  public class Serveur
13  {
14     /**  Point d'entree de l'application
15      */
16     public static void main( String args[] )
17     {
18         // Initialisation de l'ORB
19         org.omg.CORBA.ORB orb =
20                  org.omg.CORBA.ORB.init(args, null);
21
22         // Recuperation du RootPOA
23         org.omg.CORBA.Object objPoa = null;
24         org.omg.PortableServer.POA rootPOA = null;
25         try
26         {
27            objPoa =
28               orb.resolve_initial_references("RootPOA");
29         }
30         catch ( org.omg.CORBA.ORBPackage.InvalidName ex )
31         {}
32
33         rootPOA =
34             org.omg.PortableServer.POAHelper.narrow(objPoa);
35
36         // Creation du servant BanqueImpl
37         BanqueImpl bank = new BanqueImpl(rootPOA);
38
39         // Creation du servant de delegation
```

Serveur.java (suite)

```
40          BanquePOATie tie = new BanquePOATie( bank );
41
42          // Activation du servant
43          byte[] servantId = null;
44          try
45          {
46              servantId = rootPOA.activate_object(tie);
47          }
48          catch (
49          org.omg.PortableServer.POAPackage.WrongPolicy ex)
50          {
51              ex.printStackTrace();
52          }
53          catch (
54          org.omg.PortableServer.POAPackage.ServantAlreadyActive ex)
55          {
56              ex.printStackTrace();
57          }
58
59          // Récuperation de la référence d'objet associée
60          // au servant
61          org.omg.CORBA.Object obj = null;
62          try
63          {
64                  obj = rootPOA.id_to_reference(servantId);
65          }
66          catch (
67          org.omg.PortableServer.POAPackage.WrongPolicy ex )
68          {
69                  ex.printStackTrace();
70          }
71          catch (
72          org.omg.PortableServer.POAPackage.ObjectNotActive ex )
73          {
74                  ex.printStackTrace();
75          }
76
77          // Exportation de la reference au sein d'un fichier
78          String reference = orb.object_to_string(obj);
79          try
80          {
81              java.io.FileOutputStream file = new \
82                              java.io.FileOutputStream("ObjectId");
83              java.io.PrintStream pfile=new
84                                      java.io.PrintStream(file);
85              pfile.println(reference);
86              file.close();
87          }
88          catch ( java.io.IOException ex )
89          {
90              System.out.println(
91                   "Impossible de créer le fichier ObjectId");
```

Serveur.java (suite)

```
 92         }
 93
 94         try
 95         {   // Activation du POA manager
 96             rootPOA.the_POAManager().activate();
 97             System.out.println("Le serveur est pret...");
 98
 99             // Attente des invocations clientes
100             orb.run();
101         }
102         catch ( java.lang.Exception ex )
103         {
104             System.out.println("Exception interceptee");
105             ex.printStackTrace();
106         }
107     }
108 }
```

Banque_Tie_exec.txt

```
 1  // -------------------------------------------------
 2  // MV avril 2001 - OpenORB V1.0.2 sur Linux avec JDK
 3  // version 1.2.2 de sun :
 4  // [vayssade@sirius tests]$ /usr/java/bin/java -version
 5  // java version "1.2.2"
 6  // Classic VM (build 1.2.2-L, green threads, nojit)
 7  // -------------------------------------------------
 8
 9  /livre/ch07> ls
10  Delegation  Heritage
11  /livre/ch07> cd Heritage/
12  /livre/ch07/Heritage> ls
13  Banque.idl       Client.java      LISEZMOI.txt
14  BanqueImpl.java  CompteImpl.java  Serveur.java
15
16  java org.openorb.compiler.IdlCompiler -d . -notie Banque.idl
17
18  /livre/ch07/Heritage> ls
19  Banque.idl                      LISEZMOI.txt
20  Banque.java                     Serveur.java
21  BanqueHelper.java               _BanqueStub.java
22  BanqueHolder.java               _compte_bancaireStub.java
23  BanqueImpl.java                 compte_bancaire.java
24  BanqueOperations.java           compte_bancaireHelper.java
25  BanquePOA.java                  compte_bancaireHolder.java
26  Client.java                     compte_bancaireOperations.java
27  CompteImpl.java                 compte_bancairePOA.java
28
29  /livre/ch07/Heritage> javac *.java
30  /livre/ch07/Heritage>
31
```

Banque_Tie_exec.txt (suite)

```
32  java -Dorg.omg.CORBA.ORBClass=org.openorb.CORBA.ORB
33  -Dorg.omg.CORBA.ORBSingletonClass=org.openorb.CORBA.ORBSingleton\
34   Serveur
35
36  Le serveur est pret...
37
38  java -Dorg.omg.CORBA.ORBClass=org.openorb.CORBA.ORB
39  -Dorg.omg.CORBA.ORBSingletonClass=org.openorb.CORBA.ORBSingleton\
40   Client
41
42  Compte = Pierre DUPONT
43  Compte = 1 rue des granges
44  Solde = 200.0
45
46  Compte = Pierre DUPONT
47  Solde = 150.0
48  Solde avec interets = 153.375
49  ____________________________________________________________________
50
51  /livre/ch07/Heritage> cd ../Delegation/
52  /livre/ch07/Delegation> ls
53  Banque.idl       Client.java      LISEZMOI.txt
54  BanqueImpl.java  CompteImpl.java  Serveur.java
55
56  java org.openorb.compiler.IdlCompiler -d . Banque.idl
57
58  /livre/ch07/Delegation> ls
59  Banque.idl                      LISEZMOI.txt
60  Banque.java                     Serveur.java
61  BanqueHelper.java               _BanqueStub.java
62  BanqueHolder.java               _compte_bancaireStub.java
63  BanqueImpl.java                 compte_bancaire.java
64  BanqueOperations.java           compte_bancaireHelper.java
65  BanquePOA.java                  compte_bancaireHolder.java
66  BanquePOATie.java               compte_bancaireOperations.java
67  Client.java                     compte_bancairePOA.java
68  CompteImpl.java                 compte_bancairePOATie.java
69
70  /livre/ch07/Delegation> javac *.java
71  /livre/ch07/Delegation> java \
72  -Dorg.omg.CORBA.ORBClass=org.openorb.CORBA.ORB
73  -Dorg.omg.CORBA.ORBSingletonClass=org.openorb.CORBA.ORBSingleton\
74   Serveur
75  Le serveur est pret...
76  ajouter interet
77  got servant
78  ajouter interet
79  got servant
80  Ĉ
81
82
83  /livre/ch07/Delegation> java -Dorg.
```

Banque_Tie_exec.txt (suite)

```
84  omg.CORBA.ORBClass=org.openorb.CORBA.ORB
85  -Dorg.omg.CORBA.ORBSingletonClass=org.openorb.CORBA.ORBSingleton\
86   Client
87  Compte = Pierre DUPONT
88  Solde = 200.0
89  Compte = Pierre DUPONT
90  Solde = 150.0
91  Solde = 160.0
92  Solde = 110.0
93  Solde cpt2 = 100.0
94  Solde cpt2 = 50.0
95  taux = 1.0225
96  Solde avec interets = 112.475006
97  ________________________________________________________________________
98
```

21 SR03 2004 - Cours Architectures Internet - XML et XSL

21.1 XML : eXtensible Markup Language

Introduction : XML est un **ensemble** de techniques : les documents XML eux-mêmes, les DTD (Document Type Definition) et les "feuilles" XSL (eXtensible Stylesheet Language).

Définition : Un document XML est constitué d'une suite de mots contenant des **données textuelles** et des **balises de marquage**. Ces balises construisent une structure constituée d'une **arborescence de noeuds**.

Le balisage doit se conformer à certaines **contraintes de forme** afin que le **parseur XML** puisse reconstruire l'arborescence rapidement et de façon non ambigüe. Dans ce cas le document XML est dit **bien formé**. Un document "mal formé" est inutilisable. Ce n'est pas un document XML. Face à un document mal formé, un parseur XML s'arrête avec un message d'erreur.

En plus des données de texte et des balises, on peut associer au document XML une **DTD** qui va jouer le rôle d'une grammaire indiquant des contraintes **sur le contenu**.

Exemple de contrainte imposée par une DTD : une balise <auteur> **devra** comporter : une balise <nom>, une balise <prénom> et une balise <e-mail>.

Un document XML conforme à sa DTD est dit **valide**

Un parseur sachant aussi vérifier si un document XML est conforme à sa DTD est dit **parseur validant**.

Première feuille XML : xml01.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE document SYSTEM "doc.dtd"[]>
<document>
<header><title>Titre du document</title>
<subtitle>Un sous-titre pour le document</subtitle>
<authors>
  <author>
    <name>Michel Vayssade</name><shortname>MV</shortname>
    <email>Michel.Vayssade@utc.fr</email>
  </author>
</authors>
<organization>
  <name>Université de Technologie de Compiègne</name>
  <shortname>UTC</shortname>
</organization>
<date>28 Mars 2002</date>
</header>
..........
</document>
```

Première feuille XML : la DTD : xml01.dtd

```
<?xml version="1.0" encoding="US-ASCII"?>
<!ELEMENT header (title,subtitle,keywords?,authors,organization,date)>
<!ELEMENT title (#PCDATA)*>
<!ELEMENT subtitle (#PCDATA)*>
<!ELEMENT keywords (keyword+)>
<!ELEMENT keyword (#PCDATA)>
<!ELEMENT authors (author+)>
<!ELEMENT author (name,shortname?,email?)>
<!ELEMENT name (#PCDATA)*>

Nota : * = 0 ou plus; ? = 0 ou 1; + = 1 ou plus;
```

Explications : la DTD stipule qu'une en-tête de document **doit** spécifier un titre et un sous-titre, qu'elle **peut** comporter une liste de mots clés (keywords ?), ... Puis qu'un auteur est décrit par un nom (obligatoire), un prénom (facultatif) et un e-mail (facultatif).

Ce document peut aussi être présenté sous la forme d'un arbre :

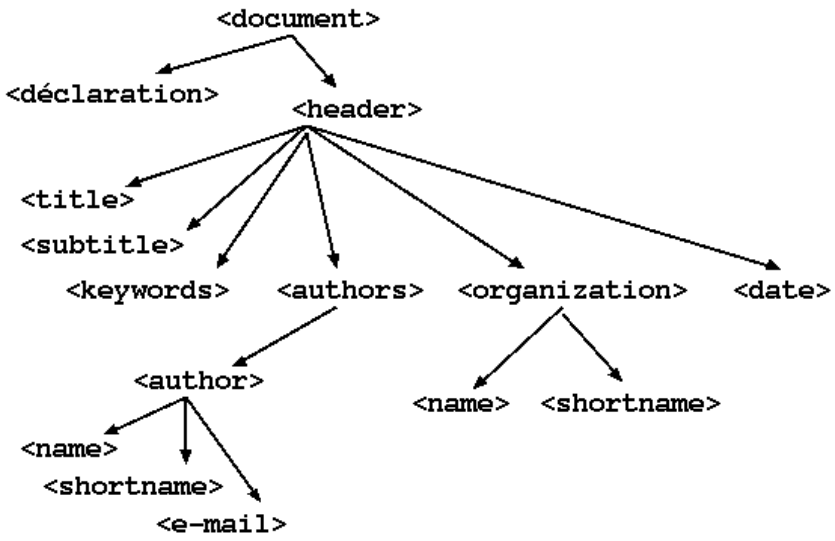


FIG. 209: Arbre XML

Nota : un document XML peut aussi contenir des **références** vers d'autres fichiers. Lors du traitement tout se passe comme si le fichier référencé était inclus (copié) dans le document initial. Cet autre document peut, lui aussi, contenir des références vers d'autres documents. Dans le vocabulaire XML le fichier référencé est appelé **entité régérence** ou **entité analysable**.

Nota : un document XML peut aussi contenir des **références vers des caractères** ; elles sont appelées **entités caractères**. Par exemple la chaine **&lt ;** est l'entité caractère chargée d'insérer dans le texte le caractère < qui ne peut pas y être inséré directement puisqu'il sert à délimiter les balises.

Remarque sur les parseurs XML : la spécification XML est très précise sur la façon dont un "parseur XML conforme" doit réagir aux erreurs du fichier source XML analysé. Les

parseurs ne doivent laisser passer aucune erreur. La conséquence est qu'un parseur XML est un **petit** programme. De plus, le passage du texte linéaire avec les balises à la représentation en arbre (ainsi que le passage inverse) sont faciles à réaliser. IL en résulte que les parseurs XML sont petits, rapides, fiables et faciles à trouver et gratuits.

Conséquence de la structure XML : un document XML étant constitué d'une suite de caractères, son transfert vers un autre programme est simplifié : un document XML est toujours prêt à circuler sur un réseau.

Contrat : si un document XML spécifie une DTD, celle-ci représente un contrat entre le fournisseur des données XML et le consommateur de ces données.

Traitement : un document XML peut être vu comme une arborescence. Le traitement des arbres est quelque chose de bien compris et de maitrisé. Avec XML a été développé un langage (**XPath**) qui permet de faire référence à des **ensembles de noeuds** (tous les descendants, tous les suivants, etc ...) ainsi qu'un langage (**XSLT**) permettant de décrire des transformations du document XML en un autre document (XML, HTML, texte, LaTeX, etc ...).

Utilisations typiques de XML :

- création de documents sources qui seront ensuite transformés vers des cibles diverses (HTML, LaTeX, PDF,...),
- encodage de données extraites ou à destination d'une base de données,
- configuration de programmes informatiques (voir l'outil ant de Apache),
- création de DTD et de vocabulaires (nouvelles balises) spécifiques à un domaine.

Par exemple, J2EE est utilise des conteneurs dans lesquels les composants d'une application sont déployés. La façon dont un composant interagit avec son conteneur est indiquée dans un fichier XML : c'est le **descripteur de déploiement**. De même, des fichiers XML sont utilisés pour l'échange de données entre conteneurs.

XML (avec XSLT) permettant une transformation aisée de formats de données constitue une passerelle très pratique entre des formats issus d'applications héritées d'une part, et des applications nouvelles d'autre part.

Standardisation de XML : XML est issu de spécifications publiées par le **W3C** (WWW Consortium) : **http ://www.w3.org**

21.2 Structure d'un document XML

Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- commentaire (après la ligne précédente, TOUJOURS preemière) -->
<!DOCTYPE document SYSTEM "doc.dtd"[]>  <!-- réf. vers la DTD -->
<document>                              <!-- racine du document -->
<header>                                <!-- élément -->
  <title>Titre du document</title>      <!-- élément fils de header -->
  <subtitle>Un sous-titre pour le document</subtitle>
  <authors>                    <!-- autre élément fils de header -->
    <author>                   <!-- élément fils de authors -->
```

```
      <name>Michel Vayssade</name><shortname>MV</shortname>
      <email>Michel.Vayssade@utc.fr</email>
    </author>
  </authors>
  <organization>
    <name>Université de Technologie de Compiègne</name>
    <shortname>UTC</shortname>  <!-- élément fils de organization -->
  </organization>
  <date>28 Mars 2002</date>     <!-- élément fils de header -->
</header>
..........
</document>
```

Un document XML "bien formé" obéit aux règles suivantes :

- toute balise d'ouverture doit possèder une balise de fermeture,
- les balises ne peuvent pas se chevaucher,
- un document XML n'a qu'un seul élément racine,
- les noms des éléments doivent être conformes aux conventions XML
- les noms des éléments (balises) sont sensibles à la casse.

Éléments vides :parfois un élément ne contient pas de données (par exemple <cover titlepage="no" tableofcontents="no" />). On le termine par "/>" accolés (pas d'espace entre / et >).

Entités caractères et entités prédéclarées

Les documents XML se composent donc de **balises** définissant des **éléments**, de données textuelles à analyser (**PCDATA** "Parsed Content DATA"), de données, valeurs d'attributs (**CDATA**) et de **sections CDATA** dont le contenu n'est pas analysé.

Deux caractères servant au balisage sont interdits partout : **<** et **&**. Dans un contenu d"élément, trois autres caractères doivent être déguisés : **>**, **"** et **'**. Ces caractères doivent être mis dans le document sous une forme déguisée dite "entité caractère". On utilise soit la valeur de leur représentation (par exemple pour < : #60 en décimal ou #x3C en hexa, soit un nom prédéfini (pour < c'est "lt"). Dans les deux cas, la référence esr délimitée par &... ;

On entre ainsi :

- pour **<** : **&lt ;**
- pour **&** : **&amp ;**
- pour **>** : **&gt ;**
- pour **"** : **&quot ;**
- pour **'** : **&apos ;**

Sections CDATA

Elles servent à déguiser une portion de texte contenant des caractères spéciaux tels que < ou &, par exemple un fichier XML servant d'exemple dans un exposé sur XML. Elles commencent par **<![CDATA[** et finissent par **]]>**.

```
<![CDATA[              <-- balise marquant le début de la section CDATA

        <?xml version="1.0" encoding="ISO-8859-1"?>
        <!-- un exemple XML -->
        <document>             <!-- racine du document -->
        ....
```

```
        </document>

        !-- balise marquant la fin de la section CDATA
]]>
```

Conséquence : la section CDATA ne doit pas contenir la séquence]]>. Si c'était le cas (comme par exemple dans le source XML qui a servi à générer ce document), cette séquence doit être masquée (en insérant un blanc :]] > ou découpée en deux séquences CDATA).

21.3 Les composantes de la "famille XML"L

Le **W3C** a publié une série de **recommandations** décrivant un ensemble de "briques de base" pour l'écriture d'applications web interopérables et d'échanges de données structurées.

- **XML 1.0 :** décrit la syntaxe des documents XML et les règles que doivent respecter les parseurs. Un document XML est **bien formé** s'il est conforme à la syntaxe XML.
- **DTD :** permet de créer des **modèles** pour un types de documents. Une DTD établit les règles de définition de la structure d'un document. Un document XML est **valide** s'il est conforme à sa DTD.
- **schémas :** la définition des DTD ayant quelques faiblesses d'expressivité, le W3C a travaillé sur une méthode plus complète de définition de la structure d'un document. Toutefois, les "schémas" sont très complexes à implémenter et à utiliser. Il n'existe pas encore de parseur validant vis-à-vis des schémas qui soit aujourd'hui utilisable en production aujourd'hui (début 2002).
- **espaces de noms :** ("namespaces") permettent d'utiliser plusieurs vocabulaires ayant des balises en commun dans un même document en préfixant la balise par le nom du namespace (par ex. <pers :title> et <xhtml :title>).
- **Xpath :** langage de requête permettant d'extraire des parties précises d'un document XML (en fait des portions bien délimitées de l'arbre associé au document).
- **XSL-XSLT :** XSLT est un langage de transformation qui va permettre d'extraire des portions d'un document source XML (en les désignant par une expression Xpath) et les transformer puis les écrire dans un document destination (qui peut être XML ou texte).
- **XSL-FO :** langage XML de description de pages imprimables de haute qualité.
- **Xlink et Xpointer :** création de liens entre documents XML.
- **DOM :** (Document Object Model) API de gestion de documents XML depuis un langage (C, Java,...).
- **SAX :** (Simple API for XML) API de gestion de documents XML depuis un langage (C, Java,...) DOM et SAX sont deux API complémentaires.

21.4 XSL : Introduction

XSL (eXtensible Stylesheet Language) est une **Recommandation** du **W3C** divisée en deux parties :

- **XSLT** (XSL Transform) langage XML servant à décrire les transformation d'un document XML ;

– **XSL-FO** (XSL Formating Objects) langage XML pour la description de pages imprimables de haute qualité typographique (Recommandation du 15 octobre 2001, qui n'est pas encore implémentée).

Ces deux langages ont pour but d'être intégrés dans une chaine de production de documents comme sur la figure ci-dessous :

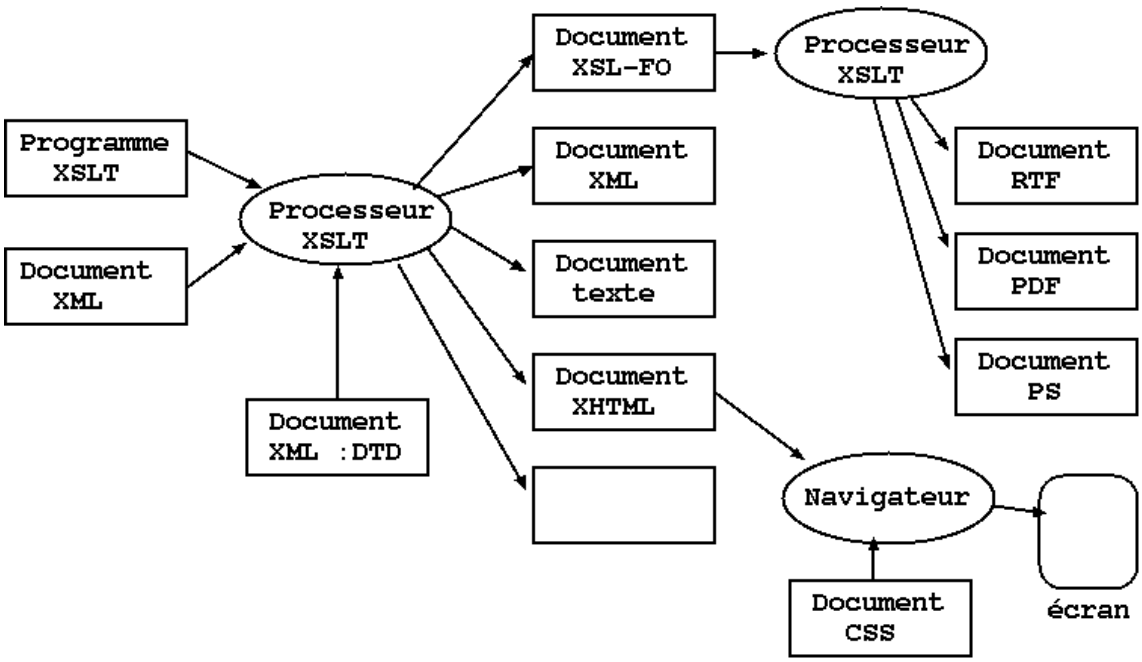


FIG. 210: Processeurs XML et XSLT

XSLT est un langage interprété. Il lui faut donc un interpréteur souvent appelé "processeur XSLT". En "Open Source" on trouve les processeurs **Xt, Xalan, Saxon et XSLTproc**

```
xsltproc  is  a  command  line  tool  for  applying  XSLT
stylesheets to XML documents. It is part of  libxslt,  the
XSLT  C  library for GNOME.
```

Donner une "petite" idée de XSLT est difficile car ce langage ne ressemble en rien aux langages procéduraux tels que C ou Java. Seuls des langages fonctionnels comme Lisp, Prolog ou Caml peuvent donner une idée : on déclare des règles qui seront appliquées à des éléments du fichier source pour produire des éléments du fichier destination.

Un programme XSLT est une suite d'instructions d'extraction de données. Une instruction XSLT va chercher un élément correspondant à un **modèle** ("template") et instancier la valeur de ce modèle dans le document destination.

Examinons un petit exemple constitué d'un fichier de données source XML, d'une DTD et de 5 programmes XSLT.

– **uv.xml** : fichier source XML
– **uv.dtd** : fichier contenant la DTD de uv.xml
– **uv-1.xsl** : premier programme XSLT
– **uv-1.out** : sortie générée par uv-1.xsl appliqué à uv.xml
  La commande avec xsltproc est :$ xsltproc –output uv-1.out uv-1.xsl uv.xml
  uv-1.xsl extrait tous les éléments de uv.xml , dans l'ordre de leur apparition dans uv.xml
– **uv-2.xsl** : programme XSLT qui extrait certains éléments seulement
– **uv-2.out** : sortie produite par uv-2.xsl

- **uv-3.xsl** : extrait les horaires de tous les groupes (pas seulement le premier)
- **uv-3.out** : sortie produite par uv-3.xsl
- **uv-4.xsl** : programme XSLT qui contrôle complètement la sortie (il n'y a pas de texte en dehors des instructions xsl),
- **uv-4.out** : sortie produite par uv-4.xsl
- **uv-5.xsl** : programme XSLT avec un meilleur contrôle de la sortie (pas d'exploration implicite d'un élément composite),
- **uv-5.out** : sortie produite par uv-5.xsl

uv.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE polytex SYSTEM "uv.dtd"[]>
<uv>
  <enseignants>
      <un-enseignant>
         <nom>Jean-Marc Philippe</nom>
         <statut>UTC</statut>
      </un-enseignant>
      <un-enseignant>
         <nom>Pierre Arthur</nom>
         <statut>extérieur</statut>
      </un-enseignant>
  </enseignants>
  <cours>
    <horaire>
       <jour>mercredi</jour>
       <heure>14h15</heure>
    </horaire>
    <enseignant>
       <nom>Jean-Marc Philippe</nom>
    </enseignant>
  </cours>
  <td>
    <groupe-td>
      <horaire>
         <jour>lundi</jour>
         <heure>08h00</heure>
      </horaire>
      <enseignant>
         <nom>Jean-Marc Philippe</nom>
      </enseignant>
      <inscrits>
        <étudiant>Jean Dupond</étudiant>
        <étudiant>Jacques Durand</étudiant>
      </inscrits>
    </groupe-td>
    <groupe-td>
      <horaire>
         <jour>mardi</jour>
         <heure>10h15</heure>
      </horaire>
      <enseignant>
         <nom>Pierre Arthur</nom>
```

```
         <statut>extérieur</statut>
      </enseignant>
      <inscrits>
        <étudiant>Jean Lefevre</étudiant>
        <étudiant>Paul Smith</étudiant>
      </inscrits>
    </groupe-td>
  </td>
</uv>
```

uv.dtd

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--* DTD pour un document UV *-->
<!--Nota : * = 0 ou plus; ? = 0 ou 1; + = 1 ou plus;-->

<!ENTITY % uv "(enseignants,cours?,td*)">

<!ENTITY % enseignants "(un-enseignant+)">
<!ENTITY % cours "(horaire,enseignant+)">
<!ENTITY % horaire "(jour,heure)">
<!ENTITY % un-enseignant "(nom,statut)">

<!ENTITY % td "(groupe-td+)">
<!ENTITY % groupe-td "(horaire,enseignant,inscrits)">
<!ENTITY % enseignant "(nom)">
<!ENTITY % inscrits "(étudiant+)">
<!ENTITY % étudiant "(#PCDATA)">
```

uv-1.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output cdata-section-elements="verbatim"
            doctype-system="uv.dtd" method="xml" indent="no"
    encoding="ISO-8859-1" />

<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

uv-1.out

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
          Jean-Marc Philippe
          UTC


          Pierre Arthur
          extérieur




        mercredi
        14h15


        Jean-Marc Philippe




          lundi
          08h00


          Jean-Marc Philippe


        Jean Dupond
        Jacques Durand




          mardi
          10h15


          Pierre Arthur
          extérieur


        Jean Lefevre
        Paul Smith
```

uv-2.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output encoding="ISO-8859-1" />
```

```
<xsl:template match="/">
écrire ce texte dans le fichier destination
puis extraire un élément du fichier source XML :
<xsl:value-of select="uv/cours/horaire/jour" />

extraire un élément composite :

<xsl:value-of select="uv/enseignants" />

puis un autre :
<xsl:value-of select="uv/td/groupe-td/horaire" />

</xsl:template>

</xsl:stylesheet>
```

uv-2.out

```
<?xml version="1.0" encoding="ISO-8859-1"?>

écrire ce texte dans le fichier destination
puis extraire un élément du fichier source XML :
mercredi

extraire un élément composite :



         Jean-Marc Philippe
         UTC


         Pierre Arthur
         extérieur



puis un autre :

         lundi
         08h00
```

uv-3.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output encoding="ISO-8859-1" />

<xsl:template match="/">
écrire ce texte dans le fichier destination
puis extraire un élément du fichier source XML :
```

```
<xsl:value-of select="uv/cours/horaire/jour" />

extraire un élément composite :

<xsl:value-of select="uv/enseignants" />

puis un autre :
<xsl:for-each select="uv/td/groupe-td/horaire">
  <xsl:value-of select="." />
</xsl:for-each>

</xsl:template>

</xsl:stylesheet>
```

uv-3.out

```
<?xml version="1.0" encoding="ISO-8859-1"?>

écrire ce texte dans le fichier destination
puis extraire un élément du fichier source XML :
mercredi

extraire un élément composite :



         Jean-Marc Philippe
         UTC


         Pierre Arthur
         extérieur



puis un autre :

         lundi
         08h00

         mardi
         10h15
```

uv-4.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output encoding="ISO-8859-1" />

<xsl:template match="/">
<xsl:text>écrire ce texte dans le fichier destination
```

```
puis extraire un élément du fichier source XML : </xsl:text>
<xsl:value-of select="uv/cours/horaire/jour" />
<xsl:text>
extraire un élément composite :</xsl:text>
<xsl:value-of select="uv/enseignants" />

<xsl:text>puis un autre :</xsl:text>
<xsl:for-each select="uv/td/groupe-td/horaire">
  <xsl:value-of select="." />
</xsl:for-each>

</xsl:template>

</xsl:stylesheet>
```

uv-4.out

```
<?xml version="1.0" encoding="ISO-8859-1"?>
écrire ce texte dans le fichier destination
puis extraire un élément du fichier source XML : mercredi
extraire un élément composite :

         Jean-Marc Philippe
         UTC


         Pierre Arthur
         extérieur

  puis un autre :
         lundi
         08h00

         mardi
         10h15
```

uv-5.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output encoding="ISO-8859-1" />

<xsl:template match="/">
<xsl:text>écrire ce texte dans le fichier destination
puis extraire un élément du fichier source XML : </xsl:text>
<xsl:value-of select="uv/cours/horaire/jour" />
<xsl:text>
extraire un élément composite :</xsl:text>
<xsl:value-of select="uv/enseignants" />

<xsl:text>puis un autre :</xsl:text>
<xsl:for-each select="uv/td/groupe-td/horaire">
```

```
  <xsl:text>&#10;</xsl:text>
  <xsl:value-of select="./jour" />
  <xsl:text> : </xsl:text>
  <xsl:value-of select="./heure" />
</xsl:for-each>

</xsl:template>

</xsl:stylesheet>
```

uv-5.out

```
<?xml version="1.0" encoding="ISO-8859-1"?>
écrire ce texte dans le fichier destination
puis extraire un élément du fichier source XML : mercredi
extraire un élément composite :

         Jean-Marc Philippe
         UTC


         Pierre Arthur
         extérieur

  puis un autre :
lundi : 08h00
mardi : 10h15
```

21.5 XPath : Introduction

XPath est un langage permettant de désigner de façon précise des ensembles d'éléments de l'arbre XML source, en vue de leur appliquer un traitement spécifié en XSLT.

Par le moyen **d'expressions XPath**, on définit un **chemin de localisation**. Ce chemin peut être relatif au noeud courant ou absolu (s'il commence par "/", dans ce cas il part de la racine du document).

Dans les exemples précédents on a vu des chemins de localisation dans les **attributs** "select" des balises xsl : select="uv/td/groupe-td/horaire".

Remarque 1 : la syntaxe d'une expression XPath n'est pas du XML. En effet, ces expressions sont appelées à être utilisées comme **valeurs** d'attributs de balises xsl. Or, on ne peut jamais mettre un balise XML à l'intérieur d'une autre balise : une syntaxe telle que <balise attribut=<a>valeur n'est pas valide en XML.

Remarque 2 : il y a une analogie entre les "chemins" XPath et les chemins vers un fichier dans l'arborescence d'un système de fichiers unix. Il faut toutefois se méfier de cette analogie : un chemin unix tel que "uv/td/groupe-td/horaire" pointe sur **un** fichier "horaire". Le chemin XPath ayant la même forme désigne **tous** les éléments <horaire> de **tous** des éléments <groupe-td> de tous les éléments <td>.

Il est souvent préférable de lire les chemins XPath à l'envers. Ainsi, "uv/td/groupe-td/horaire" se lira : "tous les éléments horaires rattachés à des éléments groupe-td rattachés à des éléments td rattachés à des éléments uv".

XPath est fait pour explorer des documents XML. Il va donc utiliser l'arbre associé au fichier XML. Les chemins de localisation permettent de se déplacer dans cet arbre en sélectionnant des ensembles de noeuds de l'arbre.

Il y a dans l'arbre XML, **sept** types de noeuds :

– **root : /** la racine de l'arbre. C'est un noeud "virtuel". Il n'est pas visible dans le document. Il est créé virtuellement par XPath et XSLT car un arbre a besoin d'une racine ! Ne pas confondre avec **l'élément racine** du document qui est seulement la balise de "plus haut niveau".

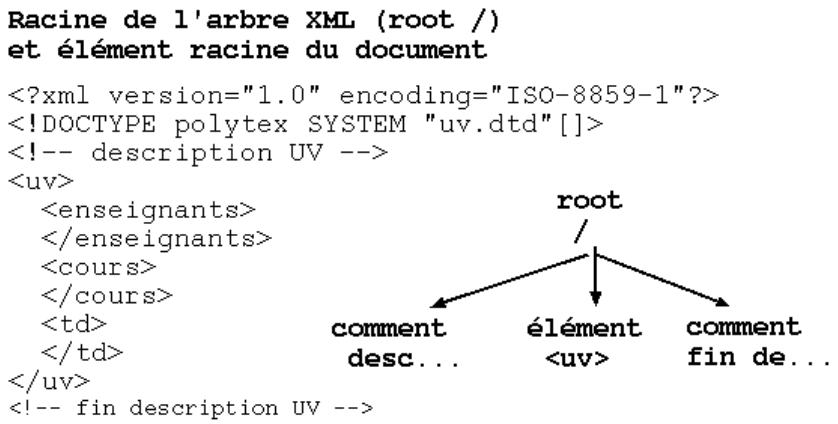


FIG. 211: Racine d'un document XML

– **élément : <balise>** un noeud élément XML
– **text : ..bla bla ..** un noeud texte, partie d'un élément
– **attribut : <xxx attrib="valeur">** un attribut d'un élément
– **namespace :** un nom permettant de qualifier (<nom :balise> des éléments, déclaré par exemple comme : xmlns :xi="http ://www.w3.org/2001/XInclude" qui crée le namesapce "xi",
– **processing-instruction : < ? traitement arg1 arg2 ?>** appel à un traitement externe au processeur XSLT
– **comment : <– ... –>** commentaire.

Noeud contexte : à partir d'un noeud de l'arbre (appelé noeud contexte), un chemin de localisation ("location path") permet de désigner un ensemble de **voisins** du noeud courant choisis dans une **direction** que l'on appelle un **axe de localisation**.

Forme d'un chemin de localisation :

```
LocationPath = "/"?, LocationStep, ("/",LocationStep)*
```

Un chemin de localisation est donc une suite d'étapes de localisation séparées par des "/" avec éventuellement un "/" au début qui indique alors un chemin **absolu** (sinon c'est un chemin **relatif**).

L'évaluation d'un chemin de localisation donne un **node-set** c'est-à-dire un ensemble de noeuds, sans répétition et non ordonné.

Chaque étape du processus consiste en fait à **éliminer** les noeuds de l'arbre qui ne correspondent pas à cette étape du chemin.

Forme d'une étape de localisation :

```
LocationStep = Axis, "::", NodeTest, Predicate*
exemple : child::horaire[attribute::type=été]
```

- **un Axe :** l'ensemble de départ est fait des noeuds choisis selon une direction : parents, enfants, frères en amon, frères en aval,...
- **un NodeTest (un déterminant) :** c'est un premier filtre permettant de ne garder que certains noeuds (par exemple tous les éléments <horaire>).
- **0 ou n prédicats :** filtre(s) suplémentaires (par exemple ne garder que les éléments ayant un attribut "truc=val").

Axes de localisation : XPath en définit treize.

- **child**
- **descendant**
- **parent**
- **ancestor**
- **self**
- **following-sibling**
- **preceding-sibling**
- **following**
- **preceding**
- **descendant-or-self**
- **ancestor-or-self**
- **attribute**
- **namespace**

Exemple (d'après XSLT fondamental, Ph.Drix, édi.Eyrolles 2002).

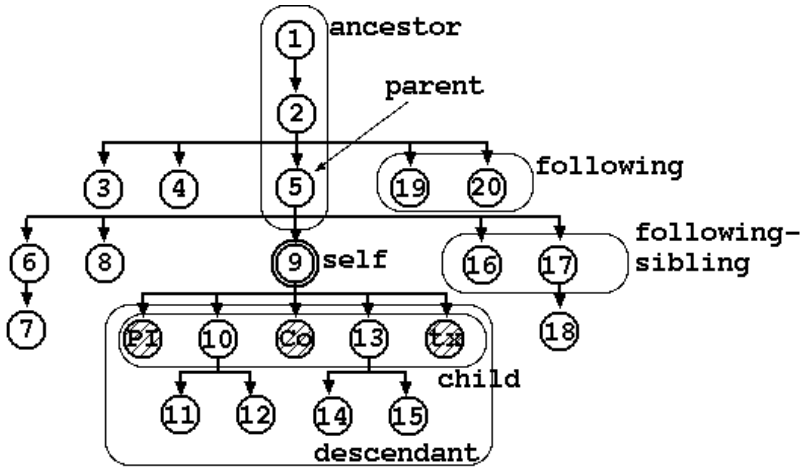


FIG. 212: Nœuds ancètres et descendants

L'arbre ci-dessus correspond au fichier ci-dessous :

```
<1>
   <2>
      <3 />
      <4 />
      <5>
        <6>
          <7 />
```

```
        </6>
        <8 />
        <9>
           <? processing-instruction ?>
    <10>
       <11 />
       <12 />
    </10>
    <!-- commentaire -->
    <13>
       <14 />
       <15 />
    </13>
    du texte
        </9>
        <16 />
        <17>
           <18 />
        </17>
      </5>
      <19 />
      <20 />
   </2>
</1>
```

L'ordre de lecture des éléments est celui de l'arbre. Les attributs et les domaines nominaux viennent après leur parent et avant les enfants de leur parent.

Indice de proximité : cet indice associé à chaque noeud d'un ensemble va donner l'ordre d'énumération des éléments de l'ensemble. Pour les axes directs, l'indice augmente quand on s'éloigne du noeud contexte en suivant l'ordre de lecture du document. Pour les axes rétrogrades, l'indice augmente quand on s'éloigne du noeud contexte vers le début du document.

Le NodeTest (ou déterminant) est une fonction, qui, appliquée à un noeud d'un ensemble désigné par un axe de localisation, va dire si le noeud doit ou non rester dans l'ensemble. Ce déterminant peut être :

- **un nom :** child : :horaire -> garder les éléments <horaire> fils du noeud courant.
- **une * :** child : :* -> garder tous les fils du noeud courant.
- **un type de noeud :** child : :text() -> garder tous les fils du noeud courant qui sont de type "text". Il y a cinq types possibles :
  - **text()**
  - **comment()**
  - **processing-instruction()**
  - **processing-instruction("xxx")** "pi" dont le nom est "xxx"
  - **node()** garder tous les noeuds de l'axe

Prédicats

Un prédicat est une fonction booléenne qui appliquée à un ensemble de noeud (node-set) produit un nouvel ensemble qui ne contient que les noeuds pour qui le prédicat est vrai.

Le résultat étant lui aussi un "node-set", on peut appliquer les prédicats "en cascade" :

```
Axe::NodeSet[prédicat-1][prédicat-2]...
```

```
Exemple:
       child::*[self::figure][position()=2]
       sélectionne le deuxième élément "figure" parmi les élements
       fils du noeud courant
Exemple:
       child::paragraphe[ child::*[self::figure][position()=2] ]
       sélectionne les éléments "paragraphe" fils du noeud courant
       qui possèdent au moins deux éléments de type "figure" (car
       s'il n'y a pas de figure en position 2, le node-set résultat
       est vide).
```

Évaluation d'un chemin de localisation

Si on part d'un node-set contenant "n" noeuds, pour évaluer une étape de localisation vis-à-vis de ce node-set, on évalue "n" fois cette étape par rapport à chacun des noeuds. Ces "n" évaluations produisent chacune un node-set. Le node-set global résultant est la réunion ensembliste (on enlève les doublons) de chacun des node-set produits par chacune des "n" étapes.

```
Exemple :
     uv/td/groupe-td/horaire
   1-  à partir du noeud courant, on garde les éléments "uv"
   2-  à partir de l'ensemble résultat, on garde les éléments "td"
   3-  à partir de chaque noeud "td" de l'ensemble résultat précedent,
       on garde tous les éléments "groupe-td", on a donc maintenant
       un node-set contenant des tous les éléments "groupe-td" de
       tous les "td" de toutes les "uv"
   4-  pour chaque "groupe-td" du node-set produit en 3, on garde
       les éléments "horaire"
```

Formes courtes des chemins de localisation

Ces formes courtes étant employées dans la quasi totalité des exemples de codes sources XSLT disponibles, il est indispensable de les connaitre.

```
Forme longue      Forme courte
child::nom        nom
child::*          *
attribute::nom    @nom
attribute::*      @*
[position()=x]    [x]
self::node()      .
parent::node()    ..
/descendant-or-self::node()/   //

Exemple :
     bloc[3]/figure[@type='gif'][1]
     sélectionner les "figure" ayant un attribut type='gif' qui sont
     aussi le premier ([1]) enfant du troisième ([3]) élément "bloc"
     du noeud contexte.
```

21.6 XSLT : eXtensible Markup Language Transform

Introduction : XSLT est un langage de transformation de documents XSL. C'est un langage déclaratif, qui utilise des **motifs** ("pattern"). Un "programme" XSLT va rechercher la **concordance des motifs** ("pattern matching") avec les données d'un document XML. Puis, quand une concordance est trouvée, il va appliquer au motif du document un **modèle de transformation** ("template"). Les deux modèles de transformation fondamentaus de XSLT étant **xsl :value-of** et **xsl :apply-templates**.

Structure d'un document XSLT : un programme XSLT est d'abord un fichier XML dont le domaine nominal est **"http ://www.w3.org/1999/XSL/Transform"**, abrévié généralement en **xsl :**, et dont la racine est l'élément **xsl :stylesheet**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
..........
</xsl:stylesheet>
```

Un élément XSLT est un élément XML du domaine nominal "xsl :", c'est-à-dire une balise du type **<xsl :nom_balise...>**. Une instruction XSLT est un élément XSLT. Les éléments fils direct de la racine **<xsl :stylesheet ..>** sont dits "instructions de premier niveau".

Régles de transformation : XSLT est un langage déclaratif ("à la prolog") dans lequel on décrit des règles de transformation à appliquer quand le cas se présente lors de la lecture du fichier source XML à traiter. En général, l'ordre dans lequel ces règles sont énoncées dans le fichier XSLT n'a pas d'influence sur le résultat.

Le processeur XSL parcours l'arbre du document XML d'entrée et applique à certains des éléments de cet arbre une règle de transformation choisie parmi l'ensemble des règles constituant le programme XSLT.

Une règle est composée de deux parties :

- **un motif :** ("pattern"), exprimé en **XPath**, il dit si on doit traiter l'élément courant ou pas ;
- **un modèles de transformation :** ("template"), qui, si le motif correspond, indique **par quoi** remplacer l'élément courant

```
Forme d'une règle XSLT :
    <xsl:template match="..pattern..">
       <!-- modèle de transformation -->
       ....
    </xsl:template>
```

Exemple de programme XSLT : l'exemple ci-dessous s'applique au fichier **uv.xml** donné plus haut.

```
<?xml version="1.0" encoding="ISO-8859-1"?><!-- uv-8.xsl -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output  method="text" encoding="ISO-8859-1" />

<xsl:template match="/">
    <xsl:text>&#10;Infos sur l'UV :</xsl:text>
    <xsl:apply-templates/>
```

```
</xsl:template>

<xsl:template match="enseignants">
    <xsl:text>Les enseignants de l'UV :</xsl:text>
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="un-enseignant">
    <xsl:text>Enseignant </xsl:text>
    <xsl:value-of select="./nom" />
    <xsl:text> (</xsl:text>
    <xsl:value-of select="./statut" />
    <xsl:text>).</xsl:text>
</xsl:template>

<xsl:template match="cours">
    <xsl:text>Le cours :</xsl:text>
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="td">
    <xsl:text>Les TDs :</xsl:text>
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="groupe-td">
    <xsl:text>TD : </xsl:text>
    <xsl:value-of select="./horaire/jour" />
    <xsl:value-of select="./horaire/heure" />
    <xsl:text>, enseignant : </xsl:text>
    <xsl:value-of select="./enseignant/nom" />
    <xsl:text>.</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

L'exécution par la commande **$ xsltproc uv-8.xsl uv.xml** donne le résultat suivant :

```
Infos sur l'UV :
  Les enseignants de l'UV :
      Enseignant Jean-Marc Philippe (UTC).
      Enseignant Pierre Arthur (extérieur).

  Le cours :

       mercredi
       14h15


       Jean-Marc Philippe


  Les TDs :
```

```
TD : lundi08h00, enseignant : Jean-Marc Philippe.
TD : mardi10h15, enseignant : Pierre Arthur.
```

Dans cet exemple, les règles de transformation produisent du texte composé d'un mélange de valeurs incluses dans le fichier XSLT et de valeur extraites du fichier XML qui est le document source.

Principe de fonctionnement d'un processuer XSLT

Écrire un "programme" XSLT revient à prévoir le comportement du processeur XSLT face à la feuille de style XSLT que l'on écrit.

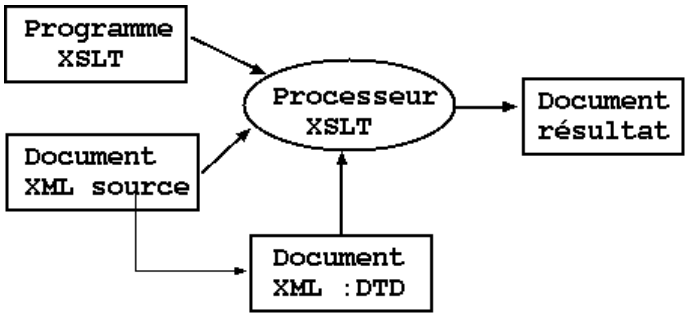


FIG. 213: Fonctionnement de XSLT

Le processeur XSLT commence par construire l'arbre XML à partir du fichier source XML qu'on lui donne à traiter. Rappelons que les deux représentations (arbre et fichier XML) sont équivalentes.

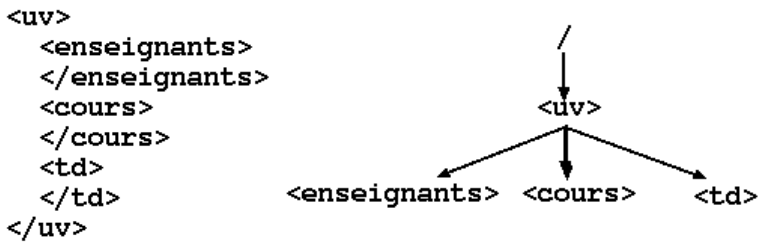


FIG. 214: Document XML et arbre associé

Le processeur XSLT opère des transformations sur l'arbre source pour l'arbre résultat. La sérialisation de l'arbre résultat produit le document XML résultat. Le processeur XSLT ne modifie jamais l'arbre source : il ne peut que construire un nouvel arbre. La conséquence est que les seules opérations utiles consistent à greffer un nouveau morceau à l'arbre résultat en cours de construction. Du point de vue des documents XML, la greffe d'un morceau à un arbre revient à concaténer un morceau de document à un document en construction.

Or il est beaucoup plus facile de **décrire** des manipulations de documents. C'est pourquoi, les transformations XSLT sont souvent décrites en termes de construction de document à partir de de fragments de documents.

Traitement du document source XML

Au départ, on constitue une liste de noeuds contenant uniquement la racine du document source XML. On traite cette liste de noeuds sources initiale.

Le traitement d'une liste de noeuds sources consiste à traiter chaque noeud de la liste dans l'ordre où il apparait dans la liste. Le traitement de chaque noeud produit un fragment de document résultat. Le document résultat est la concaténation, dans l'ordre, de ces fragments.

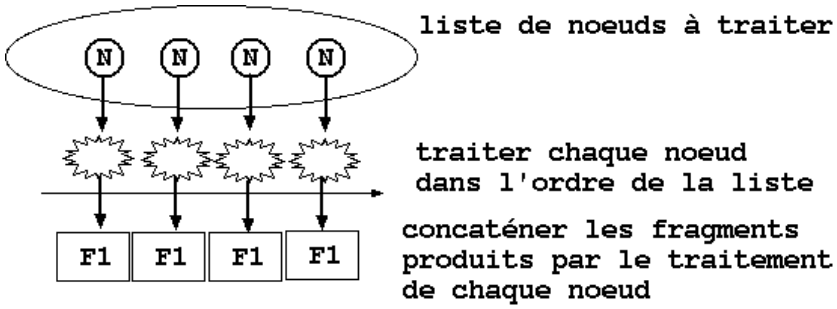


FIG. 215: Traitement sur un nœud

Traitement d'un noeud source

Traiter un noeud consiste à chercher dans la feuille de style XSLT, la règle dont le motif (pattern) correspond ("match") au noeud traité et à appliquer cette règle avec pour noeud courant le noeud en cours de traitement et pour liste courante la liste de noeuds source.

Si aucune règle explicite n'est trouvée, une règle par défaut s'applique. Si plusieurs règles sont trouvées, un méccanisme de calcul de priorité permet d'un sélectionner une et une seule (sinon, le programme XSLT est incorrect).

Appliquer une règle consiste à instancier le modèle de transformation contenu dans la règle. Ceci produit un fragment de document qui est le résultat du traitement du noeud source en cours.

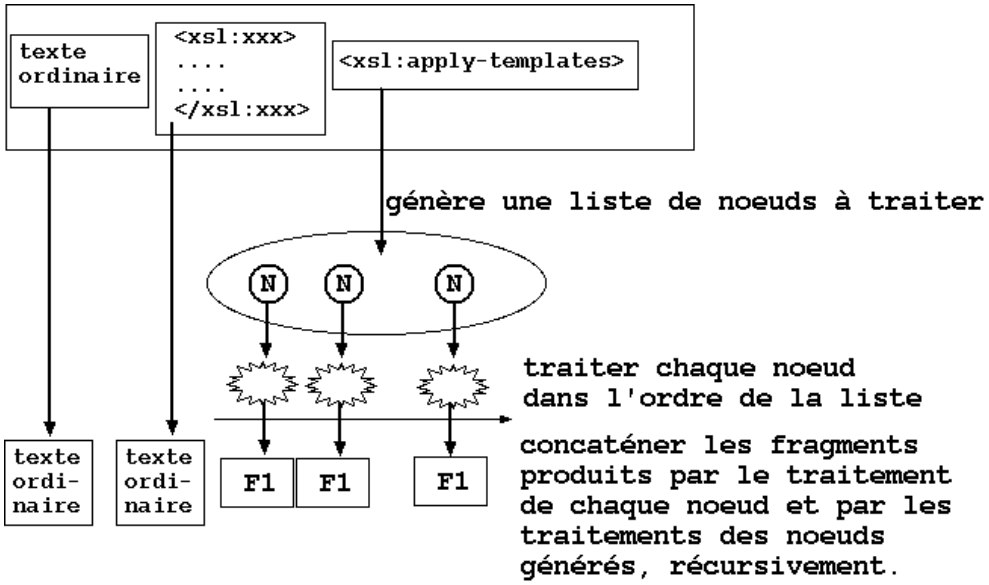


FIG. 216: Traitement d'un document

L'instanciation d'un modèle consiste à produire le fragment résultat. Pour cela, tout de qui est du texte brut ou du texte XML, en dehors du domaine nominal XSLT (autrement dit, en dehors des balises <xsl :xxx>), dans le modèle est recopié tel quel. Le reste est formé d'éléments XSLT (i.e., le texte compris dans les balises <xsl :xxx> ... </xsl :xxx>). Ces éléments XSLT sont remplacés par leur valeur. Un élément XSLT de ce type s'appelle une **instruction XSLT**.

L'évaluation de l'une des deux instruction XSLT <xsl :apply-templates> et <xsl :for-each> produit une **nouvelle liste de noeuds sources** qui est récursivement traitée.

Motifs ("patterns")

C'est la valeur de l'attribut **match** dans la définition de la règle XSLT. Cet attribut doit représenter une **expression XPath**. Cette expression va désigner un ensemble de noeuds de l'arbre source.

Un motif utilisant un chemin de localisation relatif (c'est-à-dire qui ne commence pas par un /), ne peut être évalué que par rapport à une noeud contexte. Avec certains noeuds contexte, un motif peut donner un ensemble vide.

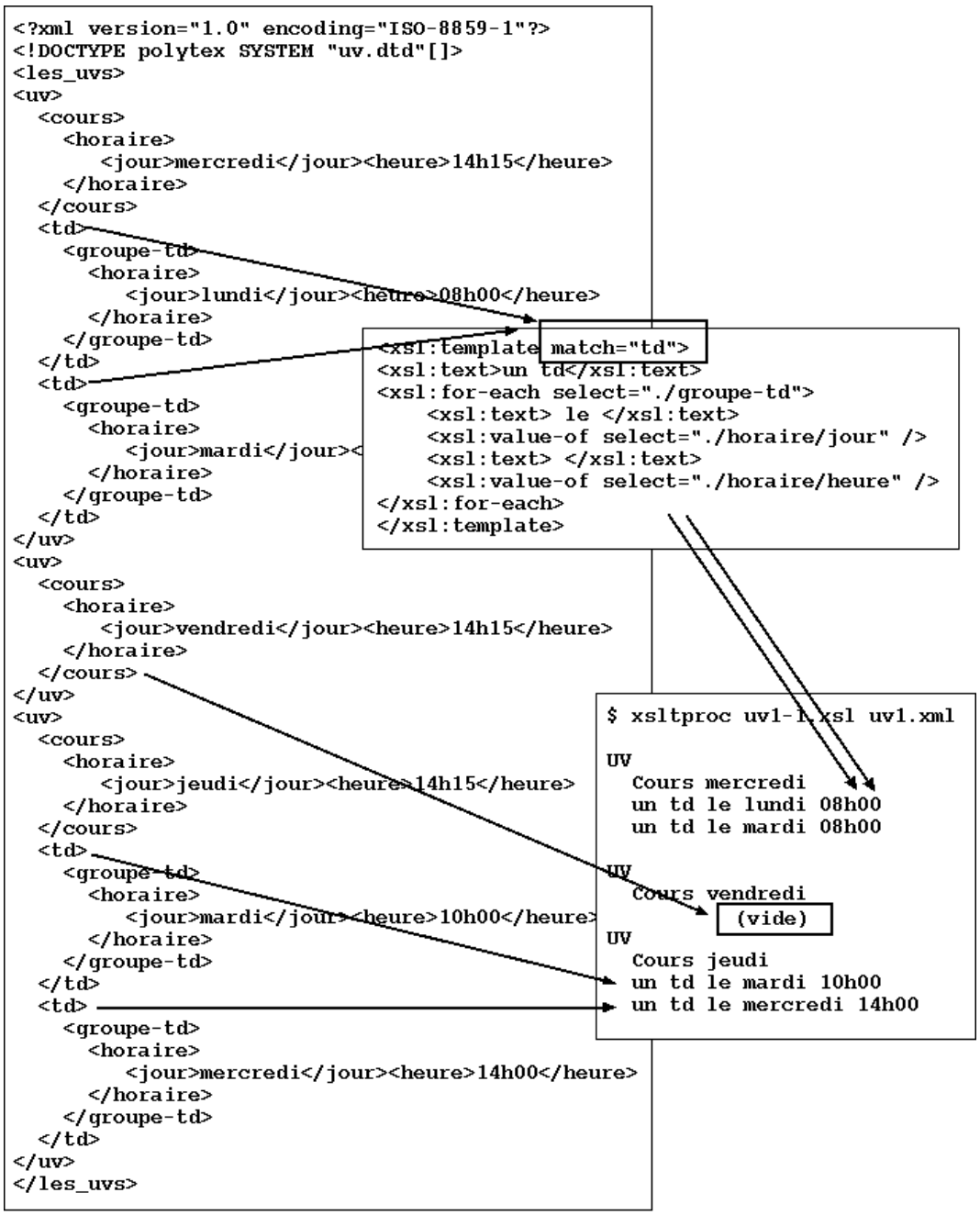


FIG. 217: Traitement d'un document

Dans l'exemple ci-dessus, le motif "td" s'applique successivement à chaque noeud courant. Quand ce noeud est une "uv" contenant des "td", il produit un ensemble contenant tous les noeuds "td" de cette "uv", et pour chaque noeud de l'ensemble, il applique la règle associée au motif.

Le but d'un motif est de désigner un noeud ou un groupe de noeud sur lesquels on désire appliquer une certaine règle.

Dans l'exemple, lorsqu'on applique le motif "td" à un noeud contexte "uv", il y a concordance du motif pour chaque noeud "td" fils direct de "uv". La concordance de motif n'est examinée par le processeur xslt que sur l'axe "child" (sinon, il faudrait examiner tout l'arbre à chaque fois). Ceci n'est pas une limitation. En effet, le but d'un motif est de désigner un noeud ou un groupe de noeud sur lesquels on désire appliquer une certaine règle. L'intérêt d'un motif est son pouvoir discriminant. Une règle qui s'appliquerait à trop de noeuds de l'arbre ne serait pas très utile.

Une conséquence de cette limitation à l'axe "child" est l'interdiction dans un motif de l'utilisation de "./" ou "../" puisque ceci permettrait d'étendre la recherche à tout l'arbre.

Remarque : si au cours du processus de traitement plusieurs règles sont éligubles, c'est la plus spécifique qui aura priorité.

21.7 Instruction XSLT : xsl :value-of

Exemple : une instruction "value-of" située dans un modèle de transformation, sera remplacée par la valeur textuelle de ce qui est désigné dans son attribut "select".

```
<xsl:template match=" motif ">
  <xsl:text>.. du texte ..</xsl:text>
  <xsl:value-of select="... chemin de localisation ..." />
</xsl:template>
```

Ainsi, dans l'exemple précédant, <xsl :value-of select="./horaire/jour" /> est remplacé, quand le motif s'applique au noeud courant, par la valeur textuelle du noeud <jour>.

Remarque : en fait, ce qui est désigné par un chemin de localisation est un "node-set". Si ce node-set comporte plusieurs noeuds, sa valeur textuelle est celle du noeud source qui arrive en premier dans l'ordre de lecture du document. Ainsi, si on modifie l'exemple "xml09" plus haut en mettant deux noeuds <jour> dans un groupe-td/horaire, **seul le premier** sera utilisé par <xsl :value-of select="./horaire/jour" />

```
      <horaire>
         <jour>dimanche</jour><heure>08h00</heure>
         <jour>lundi</jour><heure>08h00</heure>
      </horaire>
$ xsltproc uv1-1.xsl uv2.xml
produisant :
  un td le dimanche 08h00
ou
  un td le lundi 08h00

suivant l'ordre des noeuds <jour>.
```

21.8 Instruction XSLT : xsl :apply-templates

Fonction : une instruction "apply-templates" est remplacée par le fragment de document qui résulte du **traitement de la liste des enfants** du noeud courant. Ces éléments enfants sont pris **dans l'ordre de lecture** du document source.

Par exemple :

```
<xsl:template match="/">
    <xsl:apply-templates/>
</xsl:template>
```

Cette règle va prendre un à un tous les enfants du noeud racine et leur appliquer le traitement, c'est-à-dire, chercher pour chacun si une des autres règles s'applique (concordance de motif). Dans l'exemple "xml09" plus haut, elle applique les transformations à tous les enfants "<uv>" de la racine "<les_uvs>".

Si, dans le traitement d'un noeud, aucune règle du programme XSLT n'est applicable (aucune concordance de motif avec le noeud courant), alors une règle par défaut s'applique. La règle par défaut dépend de la nature du noeud à traiter.

Règle par défaut pour la racine ou un élément de l'arbre :
On reporte le traitement sur les enfants du noeud courant.

```
<xsl:template match="/|*">
    <xsl:apply-templates/>
</xsl:template>
```

Règle par défaut pour un noeud de type text ou attribute :
On ajoute un fragment de document en prélevant la valeur du noeud courant.

```
<xsl:template match="text()|attribute()::*">
    <xsl:value-of select="." />
</xsl:template>
```

L'application récursive de ces deux règles par défaut explique qu'un programme XSLT vide de règle, appliqué sur un fichier XML produise la suite des textes contenus comme valeur des différents noeuds.

Remarque : ceci peut avoir des conséquences en apparence bizarres quand le fichier source xml contient des erreurs. Une erreur courante étant une faute de frappe dans une balise, conduisant à la présence d'un noeud qui ne correspond à aucun motif. On peut alors essayer la méthode suivante pour trouver l'erreur : introduire dans le programme XSLT une règle "attrape-tout" qui imprime le nom de chaque élément sur lequel elle est appelée.

```
<xsl:template match="*">
  élément traité : balise <xsl:value-of select="local-name(.)" /> =
  <xsl:apply-templates/>
</xsl:template>
```

Appliqué au fichier uv.xml, ceci va produire :

```
$ xsltproc uv-0a.xsl uv.xml
<?xml version="1.0" encoding="ISO-8859-1"?>

  élément traité : balise uv =

  élément traité : balise enseignants =

  élément traité : balise un-enseignant =

  élément traité : balise nom =
  Jean-Marc Philippe

  élément traité : balise statut =
  UTC
etc...
```

<xsl :apply-templates select="..."/>

L'attribut optionnel select permet de modifier la liste des noeuds à traiter. Au lieu de prendre tous les enfants directs de ".", la valeur de select (un chemin de localisation) est calculée, ce qui donne un node-set dont les noeuds sont pris dans l'ordre de lecture du document source xml. Il est prudent de se limiter dans le select à un sous-ensemble des descendants du noeud courant, même si ce n'est pas imposé, sinon on risque d'introduire une récursion infinie dans le traitement.

21.9 Instruction XSLT : xsl :for-each

Fonction : une instruction "for-each" va, (et c'est la seule avec l'instruction "apply-templates"), lors de l'instanciation du modèle qui la contient, **provoquer la création d'une nouvelle liste** de noeuds, traitée récursivement. On a déja vu un exemple d'utilisation dans "uv-3.xsl", "uv-4.xsl", "xml08" et "xml09".

Instruction typique :

```
<xsl:template match="..motif..">
     <xsl:for-each select="...">
          ... texte ou instructions XSLT ...
     </xsl:for-each>
</xsl:template>
```

L'instruction "for-each" construit une nouvelle liste de noeuds d'après la valeur de son attribut "select" et lance un traitement sur cette liste. La différence avec l'instruction "apply-templates" est que la règle à appliquer à chaque noeud de la liste n'est pas cherchée dans l'ensemble du programme XSLT, mais on applique à chaque noeud de la liste le même modèle de transformation défini entre les balises <xsl :for-each ..> et </xsl :for-each>.

21.10 Instruction XSLT : xsl :sort

Fonction : une instruction "sort" est une instruction de tri qui ne s'emploie que comme complément à "xsl :apply-templates" ou "xsl :for-each" pour traiter les membres du node-set produit par ces deux instructions dans un ordre différent de celui du document source xml. Elle ne peut apparaitre qu'en "deuxième niveau", dans le modèle de transformation d'un "apply-templates"ou d'un "for-each".

Instruction typique :

```
<xsl:template match="..motif..">
     <xsl:for-each select="...">
         <xsl:sort select=".." order=".." case-order=".."
           lang=".." data-type=".." />
         ... texte ou instructions XSLT ...
     </xsl:for-each>
</xsl:template>
```

"xsl :sort" est toujours vide, et elle doit se trouver placer avant le modèle de transformation de la balise "apply-templates"ou "for-each" associée.

Attribut select : défini la clé de tri, chaîne de caractère à extraire de l'élément en cours. La valeur par défaut est la valeur textuelle du noeud courant.

Attribut order : ordre du tri (ascending ou descending), par défaut ascendant.

Attribut case-order : upper-first ou lower-first. La valeur par défaut dépend de la langue utilisée.

Attribut lang : un des codes de langues définis par XML. La valeur par défaut dépend de l'environnement de traitement ("locale" sous unix).

Attribut data-type : la nature de la clé : text ou number (défaut = text). Remarque : un type "date" aurait été très utile mais n'a pas été encore défini par la norme.

Tri à clés multiples :

```
<xsl:template match="..motif..">
     <xsl:for-each select="...">
         <xsl:sort select="nom" />
         <xsl:sort select="prénom" />
         ... texte ou instructions XSLT ...
     </xsl:for-each>
</xsl:template>
```

21.11 Instruction XSLT : xsl :copy-of

Fonction : une instruction "copy-of" est instanciée sous la forme d'une copie conforme des éléments sélectionnés par son attribut select=".. chemin de localisation ..".

Instruction typique :

```
<xsl:template match="..motif..">
     ... texte ou instructions XSLT ...
     <xsl:copy-of select="..." :>
     ... texte ou instructions XSLT ...
</xsl:template>
```

Si le résultat du select est une valeur terminale (texte, booléen, nombre), l'effet de "copy-of" est le même que celui de "value-of". Mais, si le résultat est un node-set, alors le node-set est "sérialisé" : celci produit un fragment de document constitué de la concaténation des fragments produits en sérialisant chacun des noeuds du node-set, pris dans l'ordre de lecture du document source xml.

En terme d'arbre XML, "copy-of" provoque un duplicata récursif d'un sous-arbre.

21.12 Instructions de programmation XSLT

Introduction : les instructions de programmation XSLT permettent :
- conditionner l'instanciation d'un modèle à une expression booléenne : **xsl :if, xsl :choose**.
- manipuler des variables ou des paramètres : **xsl :variable, xsl :param**.
- manipuler des modèles nommés ("named templates") : **xsl :template name="nom" et xsl :call-template**.

C'est tout !
- pas de boucle
- pas d'évolution de la valeur d'une variable ; donc pas de compteur, pas d'incrémentation, pas d'effet de bord, pas de break, pas de return, ...
- pas de tableaux, pas de struct, pas de class, pas de listes, pas de hash-tables, ...

Et pourtant : XSLT est un langage "Turing-complet" !

En effet : les modèles nommés peuvent être appelés récursivement, on peut tester une condition, on peut créer des pseudo-tableaux en construisant un arbre que l'on convertit en chaine de caractère et en l'explorant avec des fonctions prédéfinies de XPath telles que "substring", "substring-after", "substring-before". Or il a été montré que l'on pouvait tout programmer en ne disposant que des tableaux, de la récursion et des tests.

Bien entendu, c'est inutilisable dans la pratique.

Mais : la plupart des moteurs XSLT disponibles possèdent une extension permettant de convertir un arbre "RTF" (Result Tree Fragment) qui est normalement "write only" en un node-set sur lequel on peut appliquer n'importe quelle expression XPath et donc en extraire nominativement une information individuelle. La version suivante de la norme (XSLT 2.0) devrait intégrer cette extension en unifiant les notions de "RTF" et de "node-set".

Ceci dit, la programmation XSLT reste difficile car c'est une programmation fonctionnelle pure étrangère à tout ce qui est familier. De plus le langage est très "spartiate" et la bibliothèque de fonctions peu développée. XSLT n'est pas conçu pour faire de l'algorithmique, mais pour faire des transformations d'arbres XML.

Il faut donc voir les quelques fonctions de programmation comme des compléments à la transformation d'arbres XML.

Exemple d'instruction :

```
<xsl:template match="..motif..">
    <xsl:value-of select="." />
    <xsl:if test="not(position() = last()">
        <xsl:text>,</xsl:text>
    </xsl-if>
</xsl:template>
==> ajoute une "," après chaque élément sauf le dernier.
```

21.13 Instructions xsl :if et xsl :choose

Fonction : ces instructions permettent une instanciation conditionnelle en fonction de la valeur d'une expression booléenne.

Instruction typique xsl :if :

```
<xsl:template match="..motif..">
    <xsl:if test=" .. expression ..">
         ... texte ou instructions XSLT ...
    </xsl-if>
</xsl:template>
```

L'expression XPath, argument de l'attribut "test=" est évalue et transformée en booléen : il sera vrai si c'est un nombre non nul, une chaîne non nulle ou un node-set non vide.

Il n'y a **pas** de "else". Si on veut une alternative, il faut utiliser xsl :choose.

Instruction typique xsl :choose :

```
<xsl:template match="..motif..">
    <xsl:choose>
      <xsl:when test=" .. expression ..">
           ... texte ou instructions XSLT ...
      </xsl-when>
      <xsl:when test=" .. expression ..">
           ... texte ou instructions XSLT ...
           autant de when que l'on veut, mais au moins un
      </xsl-when>
      <xsl:otherwise>
           ... texte ou instructions XSLT ...
      </xsl:otherwise>
    <xsl:choose>
</xsl:template>
```

L'expression XPath, argument de l'attribut "test=" des différents "when" sont évaluées en séquence. Dès que l'une est vraie, le modèle associé est instancié, et on "sort" du choose. Si aucune n'est vraie, et qu'il existe une clause "otherwise", son modèle est instancié. Si aucune expression de "when" n'est vraie et qu'il n'y a pas de "otherwise", aucun modèle n'est instancié. Si plusieurs conditions sont vraies, seule la première évaluée à vrai verra son modèle instancié.

21.14 Instruction xsl :variable

Fonction : une instruction "xsl :variable" permet de **créer** une association nom/valeur. Une fois crée on **ne peut pas changer** la valeur d'une variable. La seule instruction permettant d'affecter une valeur à une variable est la déclaration de la variable, déclaration qui ne peut être faite qu'une fois.

Instruction typique :

```
<xsl:template match="..motif..">
     ... texte ou instructions XSLT ...
     <xsl:variable name="mavar" select=".. expression XPath .." />
     <xsl:choose>
        <xsl:when test="$mavar &lt;=1" >
    <xsl:text>mavar est inférieure ou égale à 1</xsl:text>
</xsl:when>
<xsl:otherwise>
    <xsl:text>mavar est supérieure à 1</xsl:text>
        </xsl:otherwise>
     </xsl:choose>
     ... texte ou instructions XSLT ...
</xsl:template>
```

La variable "mavar" est définie au début d'un modèle de transformation et utilisée dans le modèle par la syntaxe "$mavar".

Exemple : le programme XSLT ci-dessous :

```
<?xml version="1.0" encoding="ISO-8859-1"?><!-- uv1-2.xsl -->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output  method="text" indent="no" encoding="ISO-8859-1" />

<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="uv">
    <xsl:text>UV</xsl:text>
    <xsl:variable name="nbretd" select="count(./td)" />
    <xsl:choose>
        <xsl:when test="$nbretd &lt;=1" >
            <xsl:text>- au plus un td -</xsl:text>
        </xsl:when>
        <xsl:otherwise>
            <xsl:text>- plusieurs tds -</xsl:text>
        </xsl:otherwise>
    </xsl:choose>
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="cours">
    <xsl:text>Cours </xsl:text>
    <xsl:value-of select="./horaire/jour" />
```

```
</xsl:template>

<xsl:template match="td">
<xsl:text>un td</xsl:text>
<xsl:for-each select="./groupe-td">
    <xsl:text> le </xsl:text>
    <xsl:value-of select="./horaire/jour" />
    <xsl:text> </xsl:text>
    <xsl:value-of select="./horaire/heure" />
</xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

Appliqué au fichier uv1.xsl, produit le résultat ci-dessous :

```
$ xsltproc uv1-2.xsl uv1.xml

UV- plusieurs tds -
  Cours mercredi
  un td le lundi 08h00
  un td le mardi 08h00

UV- au plus un td -
  Cours vendredi

UV- plusieurs tds -
  Cours jeudi
  un td le mardi 10h00
  un td le mercredi 14h00
```

21.15 Instructions xsl :param

Fonction : une instruction "xsl :param" ressemble beaucoup à "xsl :variable". La différence est que la valeur donnée par l'attribut select lors de la déclaration est la **valeur par défaut** du paramètre. Cette valeur par défaut pourra être remplacée soit lors de l'appel du processeur XSLT par une valeur donnée sur la ligne de commande, soit lors de l'appel d'un template nommé (call-template).

Instruction typique :

```
<xsl:template match="..motif..">
     ... texte ou instructions XSLT ...
     <xsl:param name="monparam" select="'default_val'" />
     <xsl:choose>
        <xsl:when test="$mavar != val" >
    <xsl:text>monparam non égal à val</xsl:text>
</xsl:when>
<xsl:otherwise>
    <xsl:text>monparam égal à val</xsl:text>
```

```
        </xsl:otherwise>
     </xsl:choose>
     ... texte ou instructions XSLT ...
</xsl:template>
```

21.16 Instructions xsl :call-template et xsl :template name="nom"

Fonction : une instruction <xsl :template name="nom"> est un modèle nommé destiné à factoriser une série de transformations qui sont utilisées à l'identique dans plusieurs règles du programme XSLT.

Forme de l'instruction :

```
Une règle :
<xsl:template match="..motif..">
     ... texte ou instructions XSLT ...
</xsl:template>
Un modèle nommé :
<xsl:template name="nom_du_modèle">
     ... texte ou instructions XSLT ...
</xsl:template>
```

Un modèle nommé peut comporter des arguments formels qui seront remplis par des instructions <xsl :with-param name="par_i" select="..."> dans l'appel au modèle nommé par <xsl :call-template ..>

Instruction typique :

```
<xsl:template match="..motif..">
     ... texte ou instructions XSLT ...
     <xsl:call-template name="exemple">
         <xsl:with-param name="par_1" select="...">
         <xsl:with-param name="par_2" select="...">
     </xsl:call-template>
</xsl:template>

<xsl:template name="exemple">
     <xsl:param name="par_1" />
     <xsl:param name="par_2" />
     ... texte ou instructions XSLT ...
     ... utilise un paramètre par : $par_1
     ... exemple :
     <xsl:value-of select="$par_1" />
</xsl:template>
```

21.17 Instructions de création <xsl :text>

Fonction : une instruction <xsl :text> permet d'insérer du texte litéral dans le document de sortie. On en a déja vu de nombreux exemples.

Un élément <xsl :text> ne doit pas contenir de balises. Il recopie le texte de son modèle de transformation dans le document résultat. L'utilisation systématique de balises <xsl :text>, à l'exclusion de texte inséré dans les modèles de transformation en dehors des balises permet de mieux contrôler le format de sortie, en particulier les caractères "blanc" et "à la ligne" (&#10 ;).

21.18 Création de texte XML par un élément source littéral

Fonction : plutôt que de sortir un fichier texte, on peut vouloir produire un autre document XML. On va alors insérer les balises XML à copier dans le document résultat dans les modèles XSL. On pourra donner des contenus à ces balises avec des éléments calculés extraits du fichier source par des instructions XSL.

Exemple uv-9.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?><!-- uv-9.xsl -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output  method="text" encoding="ISO-8859-1" />

<xsl:template match="/">
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="enseignants">
</xsl:template>
<xsl:template match="un-enseignant">
</xsl:template>
<xsl:template match="cours">
</xsl:template>

<xsl:template match="td">
    <xsl:text>Les TDs :</xsl:text>
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="groupe-td">
    <xsl:text>TD : </xsl:text>
    <xsl:value-of select="./horaire/jour" />
    <xsl:value-of select="./horaire/heure" />
    <xsl:text>, enseignant : </xsl:text>
    <xsl:value-of select="./enseignant/nom" />
    <xsl:text>.</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

L'exemple uv-9.xsl appliqué à uv.xml produit :

```
  Les TDs :
    TD : lundi08h00, enseignant : Jean-Marc Philippe.
    TD : mardi10h15, enseignant : Pierre Arthur.
```

L'exemple uv-9x.xsl ci-dessous est modifié pour produire un fichier XML calculé à l'aide du document source :

```
<?xml version="1.0" encoding="ISO-8859-1"?><!-- uv-9x.xsl -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output  method="xml" encoding="ISO-8859-1" />

<xsl:template match="/">
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="enseignants">
</xsl:template>
<xsl:template match="un-enseignant">
</xsl:template>
<xsl:template match="cours">
</xsl:template>

<xsl:template match="td">
  <td>
  <titre>
    <xsl:text>Les TDs :</xsl:text>
  </titre>
    <xsl:apply-templates/>
  </td>
</xsl:template>

<xsl:template match="groupe-td">
  <untd>
    <jour>
       <xsl:value-of select="./horaire/jour" />
    </jour>
    <heure>
       <xsl:value-of select="./horaire/heure" />
    </heure>
    <xsl:text>&#10;          </xsl:text>
    <enseignant>
       <xsl:value-of select="./enseignant/nom" />
    </enseignant>
  </untd>
</xsl:template>

</xsl:stylesheet>
```

À noter : le changement de méthode de sortie par <xsl :output method="xml" encoding="ISO-8859-1" />.

uv-9.xsl appliqué à uv.xml produit :

```
  <td><titre>Les TDs :</titre>
    <untd><jour>lundi</jour><heure>08h00</heure>
          <enseignant>Jean-Marc Philippe</enseignant></untd>
    <untd><jour>mardi</jour><heure>10h15</heure>
          <enseignant>Pierre Arthur</enseignant></untd>
  </td>
```

Remarque : toutes les balises (<td>, <jour>, ... insérées directement dans le programme XSL aurait pu être remplaçées par des instructions <xsl :element name="td"> .. <xsl :element> , <xsl :element name="jour"> .. <xsl :element>, etc...

Bien que plus compliqué, cette méthode a un avantage, en effet, le contenu de l'attribut name=".." de l'instruction <xsl :element> peut être **calculé** en écrivant <xsl :element name="{$var}">.

Ici les {} qui entourent la variable $var ont pour rôle d'indiquer au processeur XSLT que le contenu de name="" n'est pas une valeur litérale comme d'habitude, mais une valeur à calculer, c'est-à-dire qu'il doit remplacer $var par sa valeur et non pas insérer la chaîne de 4caractères $,v,a,r !

Ceci s'appelle une valeur différée d'attribut ("Attribute Value Template").

Complément : l'instruction <xsl :attribute> permet d'ajouter un attribut et sa valeur **dans** une balise XML que l'on construit dans le document résultat.

```
Ainsi :
    <enseignant>
      <xsl:attribute name="Nom">
         <xsl:value-of select="./enseignant/nom" />
      </xsl:attribute>
    </enseignant>
Produira :
    <enseignant Nom="Jean-Marc Philippe" />
```

21.19 Autres instructions XSLT

XSLT comporte un certain nombre d'instructions qui n'ont pas été montrées car elles dépassent le cadre d'une introduction.

- **xsl :message**
- **xsl :key**
- **xsl :attribute-set**
- **xsl :copy**
- **xsl :comment**
- **xsl :processing-instruction**
- **xsl :number**
- **xsl :include**
- **xsl :import**
- **xsl :apply-imports**

22 SR03 2004 - Cours Architectures Internet - XMLRPC

22.1 XML-RPC : appel RPC utilisant XML

Introduction : What is XML-RPC ?

It's a spec and a set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet.

It's remote procedure calling using **HTTP as the transport** and **XML as the encoding**. XML-RPC is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned.

An XML-RPC message is an HTTP-POST request. The body of the request is in XML. A procedure executes on the server and the value it returns is also formatted in XML.

Procedure parameters can be scalars, numbers, strings, dates, etc. ; and can also be complex record and list structures.

Example of an XML-RPC request :

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181


<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param> <value><i4>41</i4></value> </param>
  </params>
</methodCall>

La réponse sera :
---------------

HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param> <value><string>South Dakota</string></value> </param>
  </params>
</methodResponse>
```

Scalar <value>s

<i4> or <int>	four-byte signed integer	-12
<boolean>	0 (false) or 1 (true)	1
<string>	ASCII string	hello world
<double>	double-precision signed	-12.214
<dateTime.iso8601>	date/time	19980717T14 :08 :55
<base64>	base64-encoded binary	eW91IGNhbid0IHJlYW=

<struct>s

```
A <struct> contains <member>s and each <member> contains a <name> and a <value>.
<struct>
  <member>
    <name>lowerBound</name>
    <value><i4>18</i4></value>
  </member>
  <member>
    <name>upperBound</name>
    <value><i4>139</i4></value>
  </member>
</struct>

<struct>s can be recursive, any <value> may contain a <struct> or any other type,
including an <array>
```

<array>s

```
A value can also be of type <array>.
An <array> contains a single <data> element,
which can contain any number of <value>s.
<array>
<data>
<value><i4>12</i4></value>
<value><string>Egypt</string></value>
<value><boolean>0</boolean></value>
<value><i4>-31</i4></value>
</data>
</array>

You can mix types as the example above illustrates.

<arrays>s can be recursive, any value may contain an <array>
or any other type, including a <struct>
```

La spécification peut être trouvée à : **http ://www.xmlrpc.com/spec**

Les implémentations sont nombreuses :
http ://www.xmlrpc.com/directory/1568/implementations

Entre autres : **AOLserver, Apache, C/C++, Cold Fusion, Delphi/Kylix, Eiffel, Flash, Java, JavaScript, KDE, Macintosh OS X, Mozilla, Perl, PHP, Python, Rebol, Scheme, Tcl, WebObjects, Zope.**

22.2 XML-RPC : Installation

Installation de apache-xmlrpc

```
copier xmlrpc-1.1-src.tar.gz (161141 octets)
décompresser puis compiler pour fabriquer xmlrpc-1.1/bin/xmlrpc-1.1.jar
    ==> il y a besoin de "ant"

copier jakarta-ant-1.4.1-bin.tar.gz (1868369 octets)
décompresser :
    ls ./jakarta-ant-1.4.1/lib/*
    ./jakarta-ant-1.4.1/lib/README
    ./jakarta-ant-1.4.1/lib/ant.jar
    ./jakarta-ant-1.4.1/lib/crimson.jar
    ./jakarta-ant-1.4.1/lib/jaxp.jar
mettre ant dans le classpath :
    setenv CLASSPATH .:
      /usr/uvs/lo33/lo33/jakarta-ant-1.4.1/lib/ant.jar:
      /usr/uvs/lo33/lo33/jakarta-ant-1.4.1/lib/crimson.jar:
      /usr/uvs/lo33/lo33/jakarta-ant-1.4.1/lib/jaxp.jar
définir JAVA_HOME : setenv JAVA_HOME /usr/java
compiler xml-rpc :
    cd ./xmlrpc-1.1
    ~/jakarta-ant-1.4.1/bin/ant
    Buildfile: build.xml
    ...........
    compile:
     [javac] Compiling 32 source files to ./xmlrpc-1.1/bin/classes
    jar:
     [jar] Building jar: ./xmlrpc-1.1/bin/xmlrpc-1.1.jar
     [jar] Building jar: ./xmlrpc-1.1/bin/xmlrpc-1.1-applet.jar
    BUILD SUCCESSFUL

définir le jar xml-rpc dans le classpath :
    setenv CLASSPATH .:./xmlrpc-1.1/bin/xmlrpc-1.1.jar
compiler le source utilisant xml-rpc :
    javac JavaServer.java
exécuter :
    java JavaServer
```

Installation de PHP-XML-RPC

```
copier xmlrpc1_02.tar.gz (66547 octets)
décompresser : il y a des exemples et deux bibliothèques :
    -rw-r--r--  1 lo33   http   29181 Nov 29 14:43 xmlrpc.inc
    -rw-r--r--  1 lo33   http    9641 Nov 29 14:40 xmlrpcs.inc
dans les clients PHP :
     include 'xmlrpc.inc';
dans les serveurs PHP :
     include 'xmlrpc.inc';
     include 'xmlrpcs.inc';
clients et serveurs PHP sont appelés depuis un navigateur :
     http://www4.utc.fr/~lo33/xmlrpc/php/clidiff.php
```

22.3 XML-RPC : exemples

Premier exemple : serveur java standalone, client java

Compilation du serveur et du client :

```
setenv CLASSPATH .:./xmlrpc-1.1/bin/xmlrpc-1.1.jar
javac JavaServer.java
javac JavaClient.java

lancer le serveur :     >java JavaServer

et dans un aute xterm : >java JavaClient
                        Sum: 8, Difference: 2
```

Code source du serveur

```
import java.util.Hashtable;
import org.apache.xmlrpc.*;

public class JavaServer {
    public JavaServer () {
// Our handler is a regular Java object. It can have a
// constructor and member variables in the ordinary fashion.
// Public methods will be exposed to XML-RPC clients.
    }
    public Hashtable sumAndDifference (int x, int y) {
Hashtable result = new Hashtable();
result.put("sum", new Integer(x + y));
result.put("difference", new Integer(x - y));
return result;
    }
    public static void main (String [] args) {
try {
    // Invoke me as <http://localhost:8080/RPC2>.
    WebServer server = new WebServer(8080);
    server.addHandler("sample", new JavaServer());

} catch (Exception exception) {
    System.err.println("JavaServer: " + exception.toString());
}
    }
}
```

Code source du client

```
import java.util.Vector;
import java.util.Hashtable;
import org.apache.xmlrpc.*;

public class JavaClient {

    // The location of our server.
    private final static String server_url =
// "http://xmlrpc-c.sourceforge.net/api/sample.php";
"http://sunserv.utc:8080/RPC2";

    public static void main (String [] args) {
try {
    // Create an object to represent our server.
    XmlRpcClient server = new XmlRpcClient(server_url);

    // Build our parameter list.
    Vector params = new Vector();
    params.addElement(new Integer(5));
    params.addElement(new Integer(3));

    // Call the server, and get our result.
    Hashtable result =
(Hashtable) server.execute("sample.sumAndDifference", params);
    int sum = ((Integer) result.get("sum")).intValue();
    int difference = ((Integer) result.get("difference")).intValue();

    // Print out our result.
    System.out.println("Sum: " + Integer.toString(sum) +
       ", Difference: " +
       Integer.toString(difference));

} catch (XmlRpcException exception) {
    System.err.println("JavaClient: XML-RPC Fault #" +
       Integer.toString(exception.code) + ": " +
       exception.toString());
} catch (Exception exception) {
    System.err.println("JavaClient: " + exception.toString());
}
    }
}
```

Appel du serveur RPC Java depuis un script PHP

```
Entrer dans le navigateur :
http://www4.utc.fr/~lo33/xmlrpc/php/clidiff.php

Réponse :

XML-RPC PHP Demo

Sum: 8, Difference: 2
```

Source du client PHP

```
<html>
<head>
<title>XML-RPC PHP Demo</title>
</head>
<body>
<h1>XML-RPC PHP Demo</h1>

<?php
include 'xmlrpc.inc';

// Make an object to represent our server.
//$server = new xmlrpc_client('/api/sample.php',
//     'xmlrpc-c.sourceforge.net', 80);
$server = new xmlrpc_client('/RPC2',
  'sunserv.utc', 8080);

// Send a message to the server.
$message = new xmlrpcmsg('sample.sumAndDifference',
 array(new xmlrpcval(5, 'int'),
       new xmlrpcval(3, 'int')));
$result = $server->send($message);

// Process the response.
if (!$result) {
    print "<p>Could not connect to HTTP server.</p>";
} elseif ($result->faultCode()) {
    print "<p>XML-RPC Fault #" . $result->faultCode() . ": " .
$result->faultString();
} else {
    $struct = $result->value();
    $sumval = $struct->structmem('sum');
    $sum = $sumval->scalarval();
    $differenceval = $struct->structmem('difference');
    $difference = $differenceval->scalarval();
    print "<p>Sum: " . htmlentities($sum) .
", Difference: " . htmlentities($difference) . "</p>";
}
?>
</body></html>
```

Appel du serveur PHP depuis un script PHP

Code source du serveur PHP (fichier serdiff.php)

```
<?php
  include 'xmlrpc.inc';
  include 'xmlrpcs.inc';

  function sumAndDifference ($params) {

      // Parse our parameters.
      $xval = $params->getParam(0);
      $x = $xval->scalarval();
      $yval = $params->getParam(1);
      $y = $yval->scalarval();

      // Build our response.
      $struct = array('sum' => new xmlrpcval($x + $y, 'int'),
      'difference' => new xmlrpcval($x - $y, 'int'));
      return new xmlrpcresp(new xmlrpcval($struct, 'struct'));
  }

  // Declare our signature and provide some documentation.
  // (The PHP server supports remote introspection. Nifty!)
  $sumAndDifference_sig = array(array('struct', 'int', 'int'));
  $sumAndDifference_doc = 'Add and subtract two numbers';

  new xmlrpc_server(array('serdiff.sumAndDifference' =>
  array('function' => 'sumAndDifference',
'signature' => $sumAndDifference_sig,
'docstring' => $sumAndDifference_doc)));
?>
```

Code source du client PHP (fichier clidiff.php)

```
Il est le même que celui qui appelle le serveur Java (voir plus haut),
avec les modifications suivantes :
Les lignes :
    $server = new xmlrpc_client('/RPC2', 'sunserv.utc', 8080);
    $message = new xmlrpcmsg('sample.sumAndDifference',
sont remplacées par :
    $server = new xmlrpc_client('/~lo33/xmlrpc/php/serdiff.php',
                                'sunserv.utc', 80);
    $message = new xmlrpcmsg('serdiff.sumAndDifference',
```

Appel du serveur PHP depuis un client Java

Code source du client Java

```
Il est le même que celui qui appelle le serveur Java (voir plus haut),
avec les modifications suivantes :
Les lignes :
    public class JavaClient {
```

```
    "http://sunserv.utc:8080/RPC2";
    (Hashtable) server.execute("sample.sumAndDifference", params);
sont remplacées par :
    public class JavaClientPHP {
    "http://sunserv.utc:80/~lo33/xmlrpc/php/serdiff.php";
    (Hashtable) server.execute("serdiff.sumAndDifference", params);
```

SR03 2004 - Cours Architectures Internet - XMLRPC
Fin du chapitre.

SR03- Ch.23 - Les serveurs d'applications

Michel.Vayssade@utc.fr -- Université de Technologie de Compiègne

---

Les serveurs d'applications

Différence entre objets métiers, classes et composants ?

- **objets métiers:** issu de la phase de modélisation (souvent UML)
- **classe:** c'est l'implémentation dans un langage d'un objet
- **composant:** code binaire conforme à un modèle (COM ou EJB), regroupe plusieurs classes et services techniques.

==> seul le composant à une réalité physique lorsqu'il est instancié dans la mémoire du serveur d'application.

Serveurs d'applications : innovation vraie ou concept marketing ?

les deux !

issus des expériences de développement d'applications transactionnelles à fort taux d'accès dans un contexte Internet.

objectif : développer une approche visant à réduire le plus possible la part de code "technique" (communication, sécurité, transaction, persistance, accès aux bases de données).

Différence entre serveur d'application et serveur web ?

À l'origine serveur de pages html statiques, le serveur web a été étendu par l'ajout de scripts côté serveur : CGI, ASP, PHP, servlets, JSP. Il est désormais capable de fournir des services applicatifs.

Un serveur d'application est un **conteneur** pour applications construites sur le modèle à trois niveaux "3 tiers" (présentation, logique métier, stockage de données).

L'évolution conduit à une fusion entre les deux technologies.

Situation de COM et EJB par rapport à Corba ?

Les serveurs d'applications offrent une solution globale de mise en production d'applications à base de composants.

Corba 2 se situe une **couche en dessous**. Il sert de socle technique pour l'implémentation de certains EJBs, mais de façon transparente.

Corba n'est **pas** un modèle de composants mais une spécification de **communication d'objets distribués**, plus quelques services de "*plomberie*" (nommage, évènements,...).

Corba 3 devrait introduire un modèle de composants indépendant du langage. Il est à craindre qu'il n'arrive trop tard pour s'imposer.

L'utilisation d'un serveur d'application fait-elle

changer l'infrastructure existante ?

Heureusement non. C'est même le contraire : beaucoup de serveurs d'application ont été installés pour "consolider" une vision globale d'un ensemble existant hétérogène.

Les serveurs d'applications

Le modèle d'architecture à trois niveaux

À l'origine le but du modèle à trois niveaux (en anglais "three-tier"), était de distinguer 3 niveaux physiques : le poste client, la machine de traitement et la machine de gestion de base de données.

Le but étant d'optimiser la répartition de la charge de travail entre les postes de travail et le serveur de données en introduisant un intermédiaire chargé de passer les requêtes et les réponses.

Par extension le modèle a désigné une architecture logique séparant les **fonctions** de **présentation**, de **traitement** et de **données**.

Aujourd'hui les objectifs de ce type d'architecture sont :

- améliorer la vitesse de développement de nouvelles application par la réutilisation de **"briques logicielles"** existantes,
- autoriser l'accès aux applications par des canaus variés : intranet, Internet (personnel itinérant), WAP, postes de travail variés,
- réutiliser l'existant ("legacy applications" : applications héritées,
- permettre une évolution de l'un des trois niveaux de façon relativement indépendante;

Différents modèles de répartition

Il y a plusieurs façon de répartir les différentes couches de traitements entre les 3 niveaux.

moniteurs de traitements proposant aux clients des "services" sous forme de procédures stockées. On aboutit alors à un modèle client-serveur de traitements.

e ce type :

distribués

Conséquences de la distribution :

- distribution des données ==> complexité de la gestion de la donnée de référence
- distribution ==> complexité de mise à jour et synchronisation
- distribution ==> impossibilité d'une mise à jour transactionnelle quand l'un des éléments du système peut ne pas être présent (cas des application "B2B" (business to business) ou l'échelle (réseaux distants, voire l'Internet), ne permet plus de garantir la disponibilité de tous les éléments

Conséquences de l'utilisation de middlewares propriétaires :

- chaque base de donnée à son propre middleware
- si une application utilise plusieurs bases de données, de

constructeurs différents (cas fréquent), il faut les installer tous sur les postes avec une gestion cauchemardesque des chaines (version d'OS, DLLs, versions des applications, etc ...)

Et les objets distribués apparurent !

- moyen astucieux de distribuer données ET traitements,
- moyen d'y accéder de façon normalisée
- deux standards :
  - COM/DCOM
  - CORBA

MAIS :

- complexité des protocoles,
- interdépendances créées entre les objets,
- complexité de la distribution des annuaires décrivant les **composants**, leur **localisation** et leurs **interfaces**.

Cette complexité devrait être prise en compte par des ateliers de développement.
Malheureusement ceux-ci sont arrivés tard, sont complexes à utiliser et manquent de maturité.

Finalement, l'application du concept de distribution viendra du web qui a détourné l'intérêt des architectes.

Il semble se dégager un consensus pour estimer que les technologies utilisées sur les postes de travail seront issues du web.

Les technologie web

Le succès du web est dû d'abord à sa simplicité et à son efficacité pour la diffusion de documents.

Toutefois l'ajout d'une intelligence applicative se fera dans la douleur : CGI, puis extensions propriétaires aux serveurs : ASP, PHP, JSP (Java Server Pages). Le but étant de faciliter le plus possible la génération automatique des pages.

L'échec du "network computer"

Le succès du web a conduit certains (SUN, Oracle, IBM) a essayer de remplaçer le poste de travail par un "super navigateur" chargé de toute l'interaction avec l'utilisateur.

Mais plusieurs erreurs ont été commises :

- manque de standard, chacun "ramant" dans son coin pour imposer sa solution,
- n'avoir pas apporté de réponse à l'existant (bureautique et applications client-serveur qu'il fallait continuer à utiliser, puisqu'on ne pouvait pas tout jeter d'un coup),
- la lenteur de téléchargement et d'exécution des applications java sur ces "pizza box" à empêché toute utilisation professionnelle :
  - les réseaux de l'époque insuffisants (ethernet 10 partagé),
  - les interfaces réseaux des serveurs saturant et le logiciel réseau (la pile IP) faisant goulôt d'étranglement dans les serveurs,
  - les CPUs des "NC" étaient trop faibles pour compenser le surcoût de l'interprétation de Java, et les compilateurs JIT ne parvenaient alors que jusqu'à un rapport 5 au mieux.

Du mainframe en mode texte au mainframe en mode graphique
=> retour à la case départ !

On a vu successivement :

- les grappes de terminaux alphanumériques 3270 sur des mainframes IBM
- la connexion de PCs en mode "fat client"--"serveur de données"
- le déferlement du web pour la diffusion d'information
- les balbutiements de la logique applicative intégrée au web
- le retour au pragmatisme marqué par une **consolidation** des serveurs, qui en fait le retour à **une logique centralisée**.

C'est le besoin d'offres de service sur des canaux de distribution hétérogène qui a (semble-t-il définitivement) imposé cette solution qui simplifie les modèles de distribution au profit d'une architecture multi-niveaux (3 ou 4), séparant **présentation**, **logique métier** et **persistance des données**.

Restait à mettre en oeuvre ce modèle de façon économique (en temps de développement et de maintenance).

C'est le rôle dévolu aux **serveurs d'application**.

Alors tout le monde est d'accord ?

Oui, ... mais :

- les concepts ne sont pas triviaux,
- les outils ne sont pas matures,
- le niveau d'expertise pour mettre en oeuvre un serveur d'application est élevé,
- les bénéfices de l'utilisation de ces technologies ne sont perceptibles qu'à moyen terme (coût du ticket d'entrée en termes d'acquisition de compétences).

Les architectures multi-niveaux à base de composants hébergés dans des serveurs d'applications sont une tendance lourde du marché.

Les architectures transactionnelles

Le transactionnel provient des mainframes IBM des années 70, où des ***milliers*** de terminaux alphanumériques (en mode texte) sont connectés à une machine centrale.

Le terme ***transaction*** désigne alors un ***ensemble d'écrans*** qui doivent s'enchainer pour conduire à une modificatio durable de la base de données. Ces transactions sont bien sûr mono-moniteur et mono-base (CICS ou IMS sur DB2).

Le modèle DTP

Un modèle transactionnel a été proposé par l'**X/Open** : le modèle DTP : Distributed Transaction Service, processus mettant en oeuvre des ordinateurs en réseau.

Le modèle propose :

- une API permettant d'invoquer le moniteur transactionnel (TM) : l'interface **TX**,
- une API utilisée par le moniteur transationnel pour invoquer les Ressource Manager (RM) : l'interface **XA**. ces RM sont les sources de données ou les traitements distribués.

Ces interfaces permettent de mettre en oeuvre le protocole de validation à deux phases (**two phases commit**).

Le Transation Manager la cohérence : propriétés **ACID**:

- **Atomicité**: toutes les mises à jour ou aucune,
- **Cohérence**: on passe d'un état cohérent à un autre état cohérent,
- **Isolation**: les résultats ne deviennent visibles aux autre processus qu'une fois la transaction validée,
- **Durabilité**: une fois la transaction validée, le système garantit que les modifications sont durables, même en cas de panne.

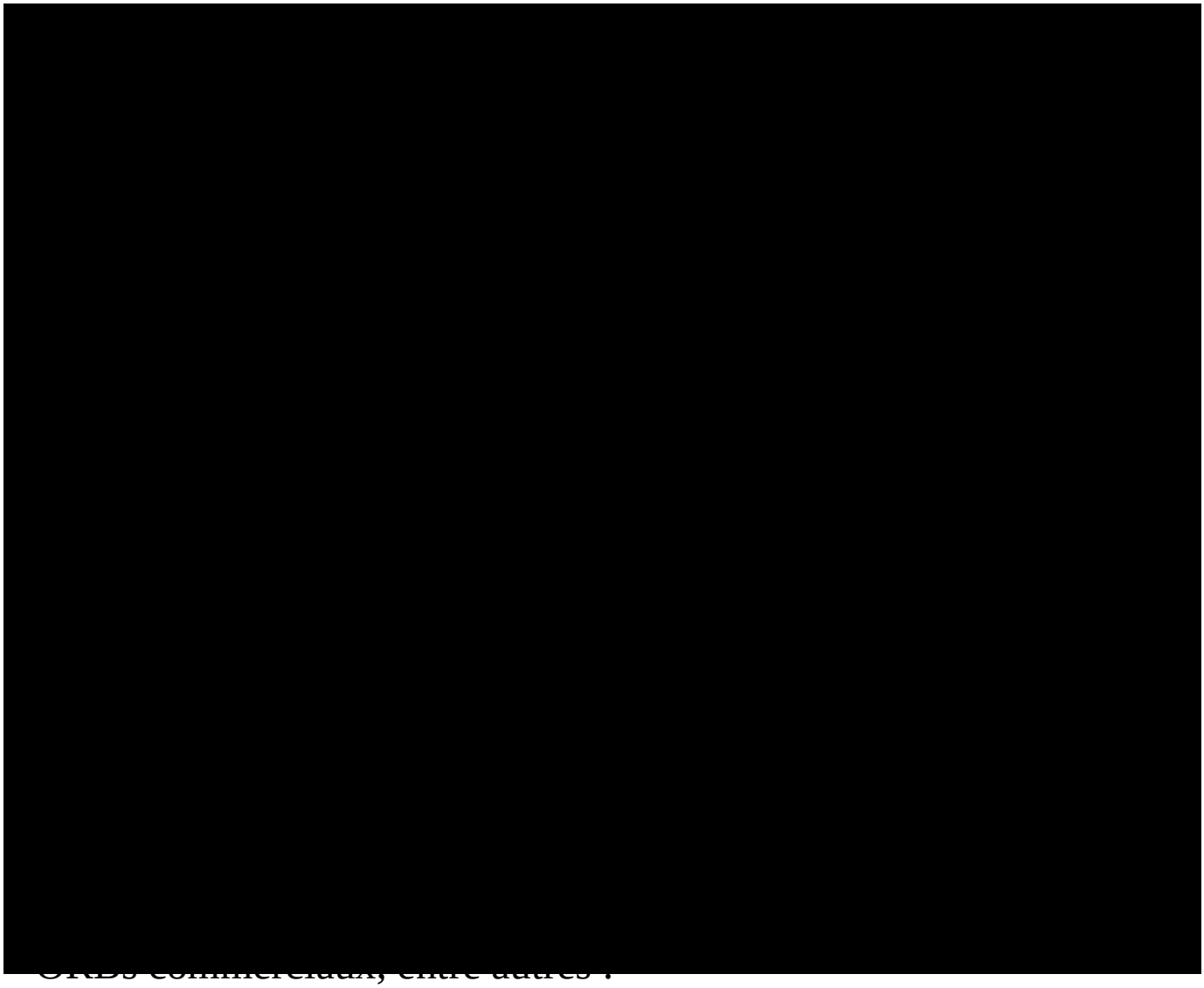
Les **TM** sont mis en oeuvre dans des moniteurs transactionnels tels que Tuxedo, les **RM** se trouvent dans les principaux SGBD/R (Sybase, Oracle, DB2) et dans certains MOM (Middleware Orientés Messages) tel que MQSeries (IBM).

L'interopérabilité des TM et des RM est assurée par le support de **XA**.

Limite : le modèle DTP repose sur une **hypothèse très contraigante** : la transaction doit être **courte**. En cas de panne du TM après le début de la phase 1, la correction du protocole de validation n'est pas garantie.

Objets et transactions

En plus de l'interopérabilité (qui est le point fort de CORBA: langages hétérogènes, machines hétérogènes, fournisseurs d'ORBs différents), l'OMG a dû intégrer la notion de de transaction. On parle alors **d'objet transactionnel**.



ORBs commerciaux, entre autres !

- Orbix OTM d'Iona,
- Visibroker ITS d'Inprise-Borland.

Même si les concepts de transactions sont simples à comprendre, ils **sont toujours compliqués à mettre en oeuvre**. Ils font partie des **services de base attendus** des serveurs d'applications.

Le poste de travail

Les serveurs d'applications permettent la mise en oeuvre de différents types d'architectures :

- client-serveur traditionnel Win32,
- applications Java,
- applications Win32 émulées,
- applications web.

L'application client-serveur traditionnel Windows

- Le client est une appli win (.exe), **développée avec un outil visuel** (**L4G**) (Power Builder, Visual Basic, Delphi, ...).

- Elle utilise des composants via le réseau par l'intermédiaire d'un **middleware** (logiciel d'intermédiation): **DCOM** ou **CORBA/IIOP/RMI**.
- Ceci implique la présence de **proxies**: passerelle entre l'application et le middleware, qui doit être fournie par l'éditeur du L4G.
- **Nota:** les proxies dont on parle ici sont les **stub et skeletons** des appels RPC et Corba.

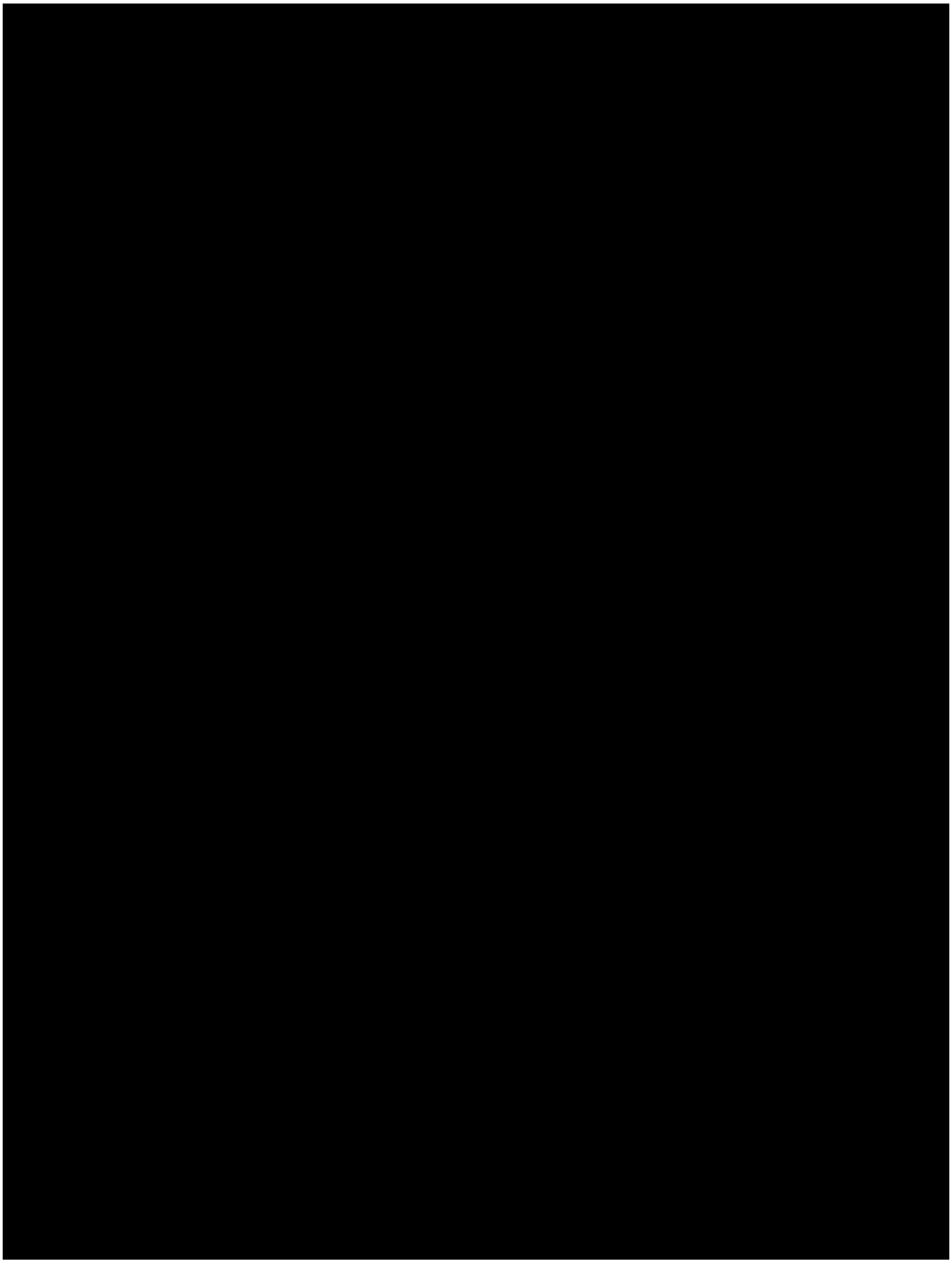
Avantages

Les ateliers L4G sont des produits matures et stabilisés.
Ils permettent de développer rapidement des interfaces graphiques.

Inconvénients

- Le démarrage à distance de composants serveurs nécessitent la présence **sur le poste client** de bibliothèques DCOM ou IIOP
- Les proxies (composants d'accès) doivent être redéployés sur chaque client si l'interface est modifiée.
- DCOM impose de les enregistrer dans la base de registres. Il faut aussi y enregistrer le **nom de la machine serveur**.
- L'utilisation de DCOM ou IIOP s'accomode mal des **coupes feu (firewalls)**, il faut alors **encapsuler** DCOM ou IIOP dans HTTP (**HTTP tunneling**).
- Ce type de client n'a de sens que lorsque la configuration des postes clients est totalement maîtrisée.
- Déployer du COM/DCOM reste un casse-tête : versions d'OS, services packs, clés de Registres, versions de DCOM, ...

Application ou applets Java



Avantages

- Utilisation "simple" puisque Corba spécifie les proxies en Java.
- Java garanti "en théorie" la portabilité.
- Le mécanisme **"d'auto-déploiement"** à travers un navigateur est simple et désormais performant (par les conteneurs **.JAR**).
- Les ateliers Java ont ratrapé les L4G traditionnels (JBuilder, Visual Café, Visual Age,...)

Inconvénients

- L'utilisation des fonctions évoluées d'interface homme-machine nécessite la présence d'un "runtime" récent (Java 2).
- Ceci nécessite la maîtrise des postes clients à cause de petites différences dans l'implémentation des JVM;
- Les applications Java restent plus lentes au démarrage.
- La promesse de Java *"write once, run everywhere"* reste une promesse. Ainsi une même appli Java essayée sur Netscape, Internet Explorer, JRE-Solaris, Linux, Mac verra des différences, de "petites" à "rédhibitoires" (polices, alignements, refraichissements, raccourcis,...).

Émulation Windows

- MetaFrame de Citrix, couplé avec la version Windows Terminal Server (WTS) permet d'exécuter des applications windows sur un serveur NT en déportant l'affichage sur le poste client (comme en X11, 20 ans après).
- Prérequis : installér un client WTS ou Cytrix sur le poste client (clients Citrix existent pour w9x, wnt, w2k, Mac et les principaux unix).
- Le client Citrix utilise le protocole **ICA** au-dessus de TCP/IP et est **peu gourmant** (30 kilobits/sec par client).

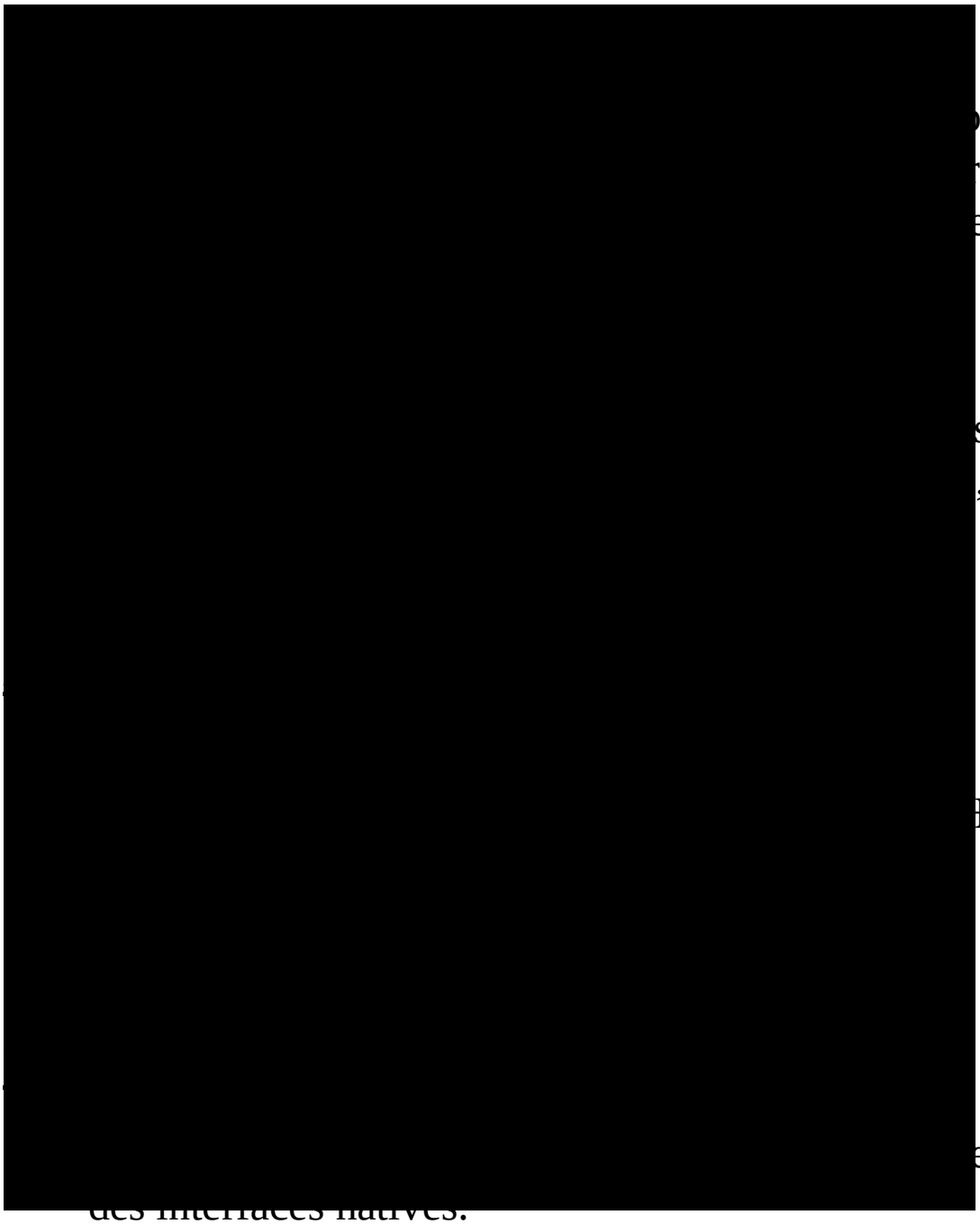
Avantages

- Accès aux applis windows sans les coûts de déploiement.
- Utilisation à distance sur lignes à faible débit.
- Accès aux applis windows sur postes unix.

Inconvénients

- Coût : une licence Citrix + une WNT **par poste client**
- Certaines applis ne fonctionnent pas en mode multi-utilisateur, ou bien doivent être modifiées.

Application web



lications
HTTP, et
erelle (JSP,

s le poste
, qui se

HTML et

e que celle

- Cet inconvénient se résorbera au fur et à mesure de la généralisation du DHTML et des CSS.

Miser sur les technologies issues du web est un bon pari.

Les composants

Qu'est-ce qu'un composant ?

Le composant est un **objet** qui adhère à un **modèle**, c'est-à-dire qu'il supporte un ensemble prédèfini de **méthodes**.
Ces méthodes (choisies par l'auteur du modèle de composant et non pas l'auteur du composant) permettent : • d'appeler (d'instancier puis d'exécuter) le composant: un client extérieur au serveur d'application provoque le chargement du composant dans le serveur puis son exécution au sein du serveur d'applications. • d'être construit par un atelier de type L4G grâce à des méthodes **d'introspection**.

Les méthodes disponibles dans le composant sont regroupées sous forme **d'interfaces** (on retrouve l'**IDL**).
Le serveur d'application doit supporter le transactionnel, la sécurité, la persistance, l'accès concurrent, etc ... sans ajout de code spécifique à ces fonctions dans le composant.
Ceci diminue la complexité de mise en oeuvre d'une architecture à trois niveaux.

Les modèles de composants

- COM/DCOM, issu de OLE, appelé ActiveX,
- les EJB (Enterprise JavaBean), d'origine SUN et IBM,
- Zope serveur d'applications Python,
- ? Corba 3, si des implémentations arrivent à temps.

Les deux principaux sont **COM** et **EJB**.

Le modèle COM

Les composants COM peuvent être en C++, VB, ...développés le plus souvent avec un outil (Delphi, PowerBuilder, ...).
==>Le choix de COM impose *de facto* la plate-forme WNT sur le serveur intermédiaire.

Le modèle EJB

C'est un environnement d'exécution adapté à des objets écrits en Java.

La spécification de cet environnement, faite par SUN et IBM s'appelle **Enterprise Java Beans**.

C'est une architecture en couches:

- l'EJB server contient l'EJB Container et lui fournit des services de bas niveau,
- L'EJB Container est l'environnement dans lequel s'exécute l'EJB (le Beans).

- les clients se connectent à une représentation du Bean fournie par le conteneur qui route les requêtes vers le Bean.

L'EJB Container fournit des services aux Beans :

- support du mode transactionnel,
- gestion des instances multiples, pool d'instances, cache, ...
- persistance (à partir de version 2.0),
- sécurité (par le biais d'ACLs au niveau du Bean ou des méthodes du Bean),
- (en option) une administration des Beans et une gestion des versions.

Le Bean

Le fournisseur d'un Bean doit fournir une **home interface (EJBHome)** et une **remote interface**.

La séquence d'appel ressemble à la suivante :

- le client appelle JNDI (service de nommage) et récupère une **référence** sur un objet implémentant EJBHome.

- le client appelle une méthode de EJBHome pour obtenir une **référence** sur l'interface Remote (qui dérive de EJBObject),

On trouve deux types de Beans :

- les **Session Beans** créés en réponse à **une** requête **d'un** client, ils peuvent être **sans état** ou **avec ét&t**, c'est-à-dire garder trace d'un appel à l'autre de valeurs dans leurs variables memebres;
- les **Entity Beans** ont vocation à "vivre" longtemps et à être partagés par différents clients. Leur persistance peut être gérée par le conteneur ou par eux-mêmes. Ils peuvent être réenrants ou non.

Déploiement des Beans

Un serveur EJB s'appuie le plus souvent sur un **moteur Corba** qui fournit les services de bas niveau (localisation d'objets, gestion d'évènements, appel de méthodes distantes, ...). Mais ce n'est pas obligatoire.
Ainsi, EJB encapsule les aspects Corba de bas niveau : IDL, BOA, POA, ...

Le serveur d'applications

Les services de base du moteur de composants

Un serveur d'applications doit disposer de 4 services de base :

- l'accès aux composants,
- l'optimisation de l'accès aux ressources locales et distantes,
- la gestion transactionnelle,
- la répartition dynamique de charge.

L'accès aux composants

- permettre aux clients de **localiser**, **instancier** et **utiliser** les composants,
- un service de nommage (JNDI : Java Naming and Directory Interface, pour les EJBs, interface générique d'accès aux annuaires). L'implémentation peut utiliser LDAP, un fichier, une base de données, le DNS ...
- après localisation et obtention d'une référeence, le client va utiliser l'objet à travers son interface EJBObject en passant par RMI ou IIOP.
- dans le cas de COM, l'annuaire est **distribué** dans la base de registre des clients, ce qui complique beaucoup l'utilisation sauf dans le cas d'une application web/COM ou seuls quelques serveurs participent à l'accès DCOM.

L'optimisation de l'accès aux ressources locales et distantes

Les composants sont de **gros consommateurs** de ressources (mémoire, UC, E/S, réseau). L'optimisation des ressources va porter sur :

- en local, gestion des threads et de la mémoire : mécanismes de pré-instanciation, et de pools d'objets;
- interaction entre composants : sémaphores, IPCs, ...
- vers l'extérieur : connexion aux bases de données (pool de connexion, ne pas réouvrir une session de base de données chaque fois);

La gestion transactionnelle

La gestion de transaction peut être faite par programme (dans le code du composant) ou de façon déclarative. Le serveur se charge du reste : début de transaction (start), validation (commit) ou annulation (rollback).

La répartition dynamique de charge ou *fail over*

Un composant doit pouvoir être déployé sur plusieurs serveurs afin d'augmenter la capacité à monter en charge (scalability) et la robustesse.

Les serveurs EJB les plus performants disposent de **services de routage des requêtes** vers les machines les moins chargées d'une grappe (algorithmes simples de round-robin ou plus complexes avec poids sur la puissance et la charge).

Dans le cas de Beans de session, il est préférable de router toutes les requêtes d'un même client sur la même machine afin d'éviter une migration des données internes du Bean qui serait plus coûteuse que la répartition de charge. On utilise alors l'adresse IP du client pour l'orienter vers la même machine.

L'administration et l'exploitation

Dépendant du produit utilisé. Devrait proposer :

- packaging et distribution
- inventaire et gestion de versions (existe rarement)
- outils de surveillance, ordonnancement, sauvegarde

Sur ces points la plupart des produits manquent de maturité.

Sécurité

Elle se décompose en trois niveaux :

- **authentification** (qui se connecte ?) : centralisation sur un annuaire LDAP
- **contrôle d'accès** (untel as-t-il le droit de ?) : il est souhaitable de pouvoir gérer des autorisations avec une granularité au niveau des méthodes des objets (et pas seulement des objets eux-mêmes);
- **intégrité et confidentialité** (utilisation de SSL pour les échanges HTTP et IIOP avec des clés DES ou RC4 de 128 bits)

SR03- Ch.23 - Les serveurs d'applications – Fin du chapitre.

Michel.Vayssade@utc.fr -- Université de Technologie de Compiègne

---

SR03 2004 - Cours Architectures Internet - Introduction à J2EE

24 SR03 2004 - Architectures Internet - Introduction à J2EE

24.1 Introduction à J2EE

Les EJB (Enterprise Java Beans) sont une architecture de composants destinée à créer des applications distribuées, multi-tiers et extensibles ("scalables"). De plus, elle rend possible la création de serveurs d'application extensibles dynamiquement (on peut ajouter des fonctions sans arrêter le serveur, sans relinker, sans rebooter).

Les EJBs ont été introduits avec la publication de la spécification 1.0 en mars 1998. De nombreuses compagnies proposent des produits conformes aus spécifications EJB : IBM, BEA, Oracle, Tandem, Sybase, Visigenic, Iona, ...

24.2 Historique du Client/Serveur

Le "bon vieux temps"

Au début était le "mainframe". L'état de l'art en matière d'informatique dans les années 60 consistait en grosses machines, très chères, utilisées pour les opérations de gestion : paie, comptabilité, facturation, ...

Tout le monde était content : les vendeurs qui gagnaient beaucoup d'argent sans se fatiguer, les acheteurs qui se sentaient important en signant des gros chèques, les gestionnaires grands prêtres et gardiens du temple ou trônait LA machine, tous ... sauf peut-être les utilisateurs.

Les miniordinateurs en temps partagé des années 70 ont rendu les machines plus accessibles, mais les traitements restaient centralisés.

Les premiers micro ordinateurs, dans les années 80, ont transformé le paysage monolithique de l'informatique d'entreprise en des centaines de petits îlots de traitement et de stockage d'informations, devenant très vite incohérents les uns par rapport aux autres.

Le Client/Serveur à la rescousse

L'architecture client/serveur a longtemps été la seule réponse à la double contrainte du besoin de **centraliser le contrôle** des données (pour assurer leur cohérence et leur intégrité) et du besoin de **disséminer l'accès** à ces données. Mais les nombreux défauts du client/serveur traditionnel ont conduit à le faire évoluer : on est d'abord passé du client/serveur 2 tiers au client-serveur 3 tiers, qui permet de donner de la souplesse au système et facilite l'interopérabilité entre systèmes d'origines différentes.

... et l'objet vint

Puis le client/serveur a évolué vers les techniques de programmation objet. CORBA est une architecture permettant à des objets qui s'exécutent dans des applications différentes de s'appeler les uns les autres. Des applications héritées (legacy applications) peuvent être encapsulées dans un service CORBA et ainsi, interopérer avec des applications nouvelles.

Les EJBs, qui ont été concus pour être compatibles avec CORBA, sont une autre façon de faire inter-opérer des objets.

24.3 Les EJBs et les serveurs d'application extensibles

L'idée de base derrière les EJBs est de fournir un cadre permettant de **brancher des composants dans un serveur existant** de façon à étendre les fonctionalités du serveur.

Il faut bien noter que les EJBs ont très peu à voir avec les simples JavaBeans. La spécification des EJBs est **entièrement** différente de celle des JavaBeans.

La spécification des EJBs indique comment les EJBs interagissent avec le serveur et avec les clients. Elle indique aussi comment les EJBs assurent la compatibilité avec CORBA, et définit précisément les responsabilités de chaque composant du système.

Les objectifs des EJBs

- rendre facile la création d'applications par les développeurs. Les EJBs libèrent le développeur de la nécessité de gérer dans le détail les threads, les transcactions, la répartition de charge, la persistance, etc ... Tout en permettant, s'il en est besoin, d'utiliser des APIs de "bas niveau" pour adapter le comportement par défaut du système pour une application particulière.
- la spécification définit la structure du système de fonctionnement des EJBs ("framework") et définit les contrats entre chaque élément de cette structure. Créer un EJB est un travail **très** différent de la création d'un serveur d'EJBs.
- de même que les JavaBeans créés par différents outils (du genre Delphi de Borland) peuvent être associés en une seule application, des EJBs créés par des outils de différents vendeurs peuvent interopérer et être combinés pour produire une application. Les EJBs sont un moyen **standard** de créer des applications client/serveur en Java.
- les EJBs sont compatibles avec les autres APIs Java, peuvent interopérer avec des applications non-Java et avec CORBA.

24.4 Le fonctionnement d'un système client/serveur à base d'EJBs

Un système client/serveur à base d'EJBs est composé de trois éléments : le composant EJB, le conteneur d'EJBs et l'objet EJB ("EJB component", "EJB container", "EJB object").

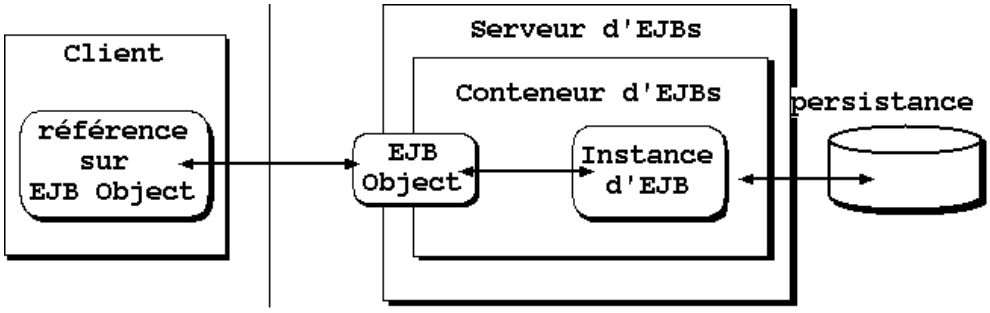


FIG. 218: Les éléments du système EJB

Le composant EJB ("EJB component")

Un composant EJB s'exécute à l'intérieur d'un conteneur d'EJBs qui, lui-même, s'exécute à l'intérieur d'un serveur d'EJBs.

Tout serveur qui peut héberger un conteneur d'EJBs et lui fournir les services définis par la spécification peut devenir un serveur d'EJBs.

Ceci a permis à de nombreux vendeurs d'étendre les fonctionnalités de leurs serveurs pour les transformer en serveurs d'EJBs.

Un composant EJB est une classe Java qui implémente la logique de l'application (la logique métier ("business logic")). Toutes les autres classes du système EJB fournissent des services aux composants ou supportent les accès par les clients.

Le conteneur d'EJBs ("EJB container")

Le conteneur est le support d'exécution du composant. Il lui fournit des services tels que la gestion des transactions, des versions, la mobilité, l'extensibilité, la persistance et la sécurité.

Par exemple, si le composant décide qu'une transaction en cours doit être avortée, il le signale, selon l'API prévue, à son conteneur, et c'est le conteneur qui va se charger d'effectuer tous les "rollbacks" nécessaires. Typiquement, de multiples instances de composants EJB coexistent dans un seul conteneur d'EJBs.

Le conteneur est l'endroit ou "vit" le beans. Il lui fournit un environnement sûr, extensible et transactionnel dans lequel le bean peut opérer. Le conteneur est invisible aux clients : il n'y a pas d'API des conteneurs pour les clients.

Quand un bean est instancié sur un serveur (i.e. dans un conteneur), le conteneur fournit deux implémentations : une implémentation de **l'interface EJBHome** (l'interface d'accueil) pour le bean et une implémentation de **l'interface remote** du bean. Le conteneur est aussi responsable d'enregistrer l'interface EJBHome auprès de JNDI.

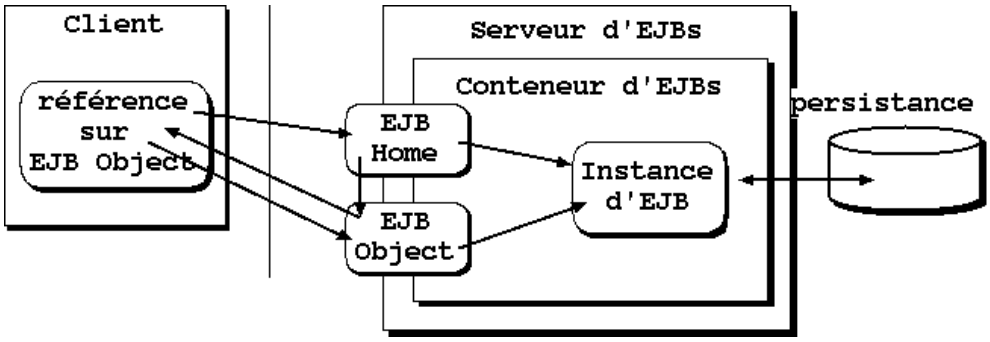


FIG. 219: Architecture des EJBs

L'interface remote, qui joue le rôle du "skeleton" dans CORBA, est générée automatiquement quand le bean est installé sur le serveur. L'implémentation de l'interface remote est "l'objet EJB". Cet objet EJB expose toutes les méthodes publiques du bean.

Le composant EJB lui-même n'implémente pas l'interface remote bien qu'il dispose d'une implémentation de toutes les fonctions présentes dans cette interface, avec les mêmes signatures.

L'objet EJB ("EJB object")

Les clients exécutent des méthodes des EJBs distants par le biais de l'objet EJB.

L'objet EJB implémente l'interface remote du composant EJB. Cette interface remote, liste les méthodes de la logique métier qui sont accessibles aux clients. Les méthodes du composant font le travail réel, utile, pour lequel le composant a été créé.

Les composants EJBs et les objets EJBs sont des classes séparées, bien que, vues de l'extérieur, elles semblent identiques. C'est simplement qu'elles implémentent **non pas** la même interface, mais plus subtilement la **même liste de méthodes** (avec les mêmes signatures). Les objets EJBs font cela en implémentant l'interface distante (remote interface) du composant EJB, alors que le composant EJB se "contente" d'implémenter ces méthodes sans pour cela implémenter (au sens Java, avec le mot clé "implements") l'interface remote.

Le composant s'exécute sur le serveur dans le conteneur et implémente la logique métier, alors que l'objet client s'exécute sur le client et fait des appels à distance **vers les méthodes de l'objet EJB** qui renvoie ces appels vers les méthodes du composant..

L'objet EJB est **"créé" par le conteneur** lors de l'installation du composant EJB sur le serveur : le programmeur de l'EJB n'écrit **que** le source du composant et les deux interfaces (distante et d'accueil). Le conteneur utilise les informations trouvées dans la description de l'interface et l'instrospection Java sur le code du composant EJB pour créer le code implémentant la classe "EJB object" qui va servir d'intermédiaire entre les clients et le composant EJB.

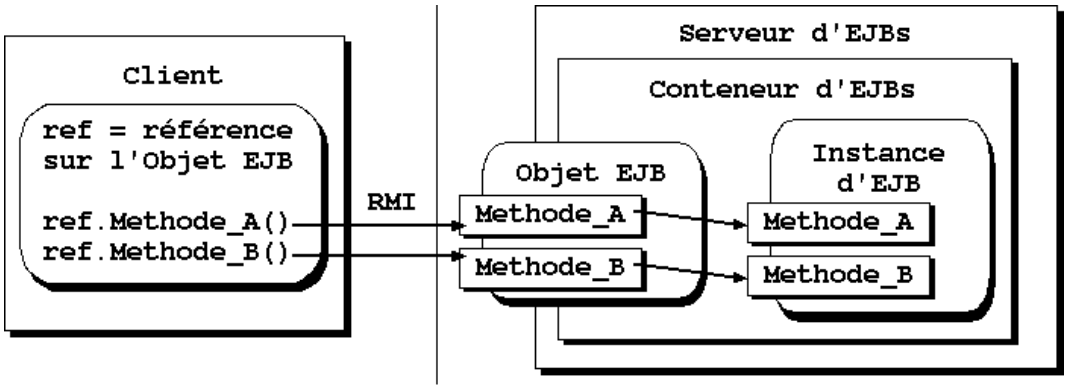


FIG. 220: L'objet EJB transfère les appels au composant EJB

Le client va récupérer (par l'intermédiaire de l'interface d'accueil) une référence distante sur l'objet EJB. Cette référence lui permettra de faire des appels RMI des méthodes disponibles de l'objet référencé. Les méthodes de l'objet EJB dont la référence RMI a été passée au client ont été construites automatiquement par le conteneur pour être compatibles avec les appels par RMI, et transmettre l'appel à la méthode de même nom du composant EJB. Le client n'a jamais d'accès direct au composant.

Analogie : si on imagine que le composant est un magnétoscope, alors l'objet EJB représente **le récepteur de la télécommande** du composant et la référence de l'objet EJB passée au client est la télécommande.

Quand on appuie sur un bouton de la télécommande (quanf on appelle une méthode de la référence distante), un signal est envoyé au récepteur sur le magnétoscope (un appel RMI est envoyé à l'objet EJB). Le récepteur actionne le même moteur que le bouton correspondant dans le magnétoscope : le résultat est une **action** dans le magnétoscope (l'objet EJB appelle la méthode de même nom du composant, celle-ci contient le code effectif qui fait l'action).

24.5 Cycle de vie des EJBs

En plus de l'interface remote, un composant EJB définit une **home interface**, ou interface d'accueil, qui permet de retrouver le composant.

L'objet EJB est **généré automatiquement** par les outils de compilation des EJBs, à l'image des fichiers sources générés par les compilateurs IDL de CORBA. Cet outil de génération est fourni avec le conteneur d'EJB. L'objet EJB joue le rôle du "skeleton" dans CORBA alors que la référence distante fournie au client représente le "stub".

Pour le client, l'objet EJB ressemble exactement à un objet du domaine d'application, mais il s'agit juste d'un "déguisement" ou d'un remplacant de l'EJB réel qui, lui, s'exécute dans le conteneur. Quand le client appelle à distance une méthode de l'objet EJB, celle-ci appelle à son tour la même méthode de l'objet "composant EJB", avec les mêmes arguments.

Ceci est le concept central de fonctionnement d'un système client/serveur à base d'EJBs.

L'intérêt de cette étape intermédiaire est de séparer le comportement des clients de la gestion des instances d'EJBs par le conteneur.

Un EJB est composé d'un fichier "jar" contenant la classe du composant EJB, les 2 fichiers sources décrivant l'interface "remote" et l'interface "Home" et d'un fichier XML appelé **descripteur de déploiement**.

Le descripteur de déploiement décrit le type de bean (session, entité, avec état, sans état, persistant ou non) et les services qu'il va utiliser (sécurité, transactions), ainsi que les noms du bean et des classes que le conteneur doit créer pour "Home", "remote" et "bean".

Un programme client contacte un serveur et demande que le serveur crèe un EJB pour effectuer du traitement de données pour le compte du client. Le serveur répond en créant une instance de composant EJB, et **en renvoyant au client un mandataire ("proxy")** de cet EJB sous la forme d'une référence distante vers l'objet EJB. L'objet EJB a la même interface que l'EJB réel, mais ses méthodes ne font que s'exécuter suite à une invocation à distance ("RMI") depuis le client pour renvoyer l'appel sur les méthodes correspondantes de l'EJB réel.

Le client utilise alors "l'objet EJB" comme si c'était l'objet réel, sans avoir besoin de savoir que le vrai travail est effectué à distance dans la mémoire du serveur d'EJBs.

Chaque classe de composant EJB possède une interface d'accueil ("home interface"). Cette interface définit des méthodes pour créer, initialiser, détruire, et (dans le cas des beans entité) trouver des instances de l'EJB sur le serveur.

L'interface "home" d'un EJB est un contrat entre l'EJB et son conteneur. Ce contrat indique comment on construit, détruit et retrouve l'EJB.

L'interface "EJB home" étends (hérite de) javax.ejb.EJBHome qui définit les fonctions de base pour une interface "home". Toutes les méthodes de cette interface doivent être compatibles avec RMI.

L'interface "EJB home" définit aussi une ou plusieurs méthodes "create" (distinguées par leur signature). La valeur de retour de ces méthodes "create" est l'interface remote du composant EJB.

Quand un client veut créer un bean EJB sur le serveur, il appelle d'abord le service JNDI (Java Naming Directory Interface) qui lui permets de localiser une ressource par son nom.

Une fois récupérée le référence sur un objet "Home interface", le client appelle une des méthodes "create" de cet objet pour provoquer la création d'une instance de l'EJB sur le serveur. La méthode de l'interface home retournée au client fait un appel RMI sur le conteneur de l'EJB, qui, à son tour, crée l'objet instance d'EJB et renvoie au client **une référence sur l'objet EJB**. Le client peut alors appeler (par RMI) les méthodes de l'objet EJB, chacune d'elle contenant en fait uniquement le code qui "renvoie" l'appel sur la méhode correspondante du composant EJB instancié sur le serveur.

Une interface EJB remote peut définir les méthodes qu'elle veut, avec la seule restriction que tous les arguments d'appel et les valeurs de retour soient compatibles avec RMI et que chaque méthode contienne une gestion de l'exception java.rmi.RemoteException.

24.6 Les deux types d'EJB : beans de session et beans entités

Pour construire un EJB, on commence par implémenter les méthodes "métier" : par exemple pour un bean "compte_bancaire" on créera une méthode "débit", une méthode "crédit" et une méthode "solde".

Il faut aussi implémenter une des deux interfaces SessionBean ou EntityBean. Ces interfaces gèrent les interactions du bean avec le conteneur et sont invisibles aux clients.

Les beans de session ("session beans")

Un bean de **session** est une instance d'EJB associée à un client **unique**. Les beans de sessions sont généralement non persistants (bien qu'ils puissent l'être) et peuvent éventuellement participer à des transactions.

En particulier, les beans de session ne survivent pas aux crashs du serveur.

Exemple de bean de session : un bean, qui, dans un serveur web, est associé à un client et conserve la "trace" du chemin suivi par le client à travers le site.

Quand le client quitte le site, ou après un temps d'inactivité, ce bean de session est détruit.

Les beans entités ("entity beans")

Un bean entité représente de l'information stockée de façon permanente dans une base de données. Un bean entité représente un accès aux données et donne accès à ces données à de multiples clients.

Les beans entités survivent aux crashs de serveurs, car lorsque le serveur reboote, il peut reconstituer le bean à partir des données stockées de façon permanente.

Dans une base de données relationnelle, un bean entité pourrait représenter une rangée d'une table, dans une base de données objet, il représenterait un objet de la base.

Pour résumer, les beans de session représentent les échanges entre le client et le serveur et les beans entités les données stockées de façon permanente.

Chaque EJB a un identifiant unique : pour un EJB entité, cet identifiant joue le rôle d'une clé primaire dans une base de données. Pour un bean de session, l'identifiant joue le rôle de la référence IOR dans CORBA : une valeur unique permettant de retrouver et utiliser l'EJB.

24.7 Les EJBs : conclusion partielle

Le but fondamental d'une technologie de serveur de composants est de répondre à la question : comment créer un serveur dont les méthodes sont extensibles à l'exécution ?

Autrement dit : Comment ajouter au serveur une fonctionnalité, puis utiliser aussitôt les clients correspondant, sans que l'on ai eu besoin de recompiler le serveur, ni même de le redémarrer ?

La réponse des EJBs est de créer le client en faisant appel à des fonctions définies dans une interface distante, les fonctions de cette interface distante étant implémentées sur le client sous

forme d'appels RMI vers un objet du serveur préalablement instancié, qui lui implémente des méthodes de même signature (même nom, mêmes arguments) qui réalisent le travail effféctif.

Le serveur d'EJBs utilise JNDI pour publier la liste des interfaces disponibles. Le client utilise JNDI pour localiser l'interface home, puis celle-ci pour créer et ensuite utiliser le composant EJB.

C'est cette association tardive ("late binding") entre un nom de composant et la référence permettant de l'utiliser qui rend le système EJB extensible à l'exécution, et pas seulement après reboote ou recompilation.

L'architecture des EJBs, en utilisant la plateforme Java, garanti l'indépendance du code écrit dans les méthodes des EJBs et, grâce au respect des spécifications des EJBs, permet d'écrire des composants qui s'exécuteront sur n'importe quel serveur d'EJBs, même de fournisseurs différents ("write once, run anywhere").

L'une des applications les plus difficiles à écrire et mettre au point sont les applications transactionnelles et distribuées. Une fonction clé du système des EJBs est le support des transactions distribuées.

Le système EJB permet d'écrire des applications qui accèdent des bases de données multiples, distribuées sur plusieurs machines et accédées à travers plusieurs serveurs d'EJBs.

Une autre fonction clé du système d'EJB est qu'il est neutre vis-à-vis du protocole de communication entre objets. Ainsi un serveur d'application EJB peut invoquer d'autres objets soit par RMI (pure Java) soit par IIOP (CORBA).

Des conteneurs spécialisés peuvent faire apparaitre des applications existantes non-java comme des Beans, permettant au développeur d'EJBs d'accéder à ces applications sans en apprendre leur spécificités.

Un des buts de conception des EJBs a été de permettre une simplification de l'écriture d'applications distribuées, en transformant des fonctions habituellement codées manuellement par de simple déclarations de propriétés des EJBs. Par exemple, la mise en place de la sécurité ou l'usage des transactions peut se faire simplement par une déclaration dans le composant lui-même.

De plus, les EJBs permettent de développer les deux principaux modèles d'application :

1. **session :** le client débute une session avec un objet qui réagit comme une application, exécutant un certain travail pour le compte du client, travail pouvant inclure une ou plusieurs transactions, et des interrogations de type SQL,
2. **entité :** le client accède un objet du serveur qui représente une entité de la base de donnée ; toutes les modifications sur l'objet sont stockées de façon permanente dans la base de données ; ce modèle est plus souvent utilisé avec des bases de données objet.

24.8 Le processus de développement avec les EJBs

Prenons l'exemple d'un caddie électronique ("shopping cart") pour acheter des objets ou des services sur un site web.

1. On écrit les méthodes contenant la "logique métier" avec un outil de développement tel que JBuilder, Java Workshop, Café, ... Après la compilation, le Bean est empaqueté (packaged) dans un fichier de type EJB JAR. Ce fichier est un fichier JAR ordinaire avec en plus un **descripteur de déploiement** ("DeploymentDescriptor") sous la forme d'une instance de classe sérialisée qui contient les informations sur la sécurité, le comportement du bean vis-à-vis des ransactions, etc ...

2. On installe le Bean sur un serveur en utilisant un **outil de déploiement** fournit par le vendeur.

On commence par écrire l'interface remote du bean :

```
public interface ShoppingCart extends javax.ejb.EJBObject {
  boolean addItem(int itemNumber) throws java.rmi.RemoteException;
  boolean purchase() throws java.rmi.RemoteException;  }
```

Puis on écrit la classe du composant EJB (le bean), avec l'implémentation des méthodes métiers et d'une méthode d'initialisation dont le nom est obligatoirement ejbCreate(). Cette méthode ejbCreate() sera appelée par la méthode create() de l'interface d'accueil. Mais il faudra d'abord obtenir une référence sur un objet implémentant cette interface d'accueil (ici CartHome).

```
public class ShoppingCartEJB implements SessionBean {
   public boolean addItem (int itemNumber) { // code métier }
   public boolean purchase() {  // code métier  }
   public ejbCreate (String accountName, String account) {
     // code d'initialisation de l'objet EJB  }    }
```

On remarquera que la classe de l'EJB implémente SessionBean mais **pas** ShoppingCart. La classe EJB possède toutefois une méthode publique pour chacune de celles déclarées dans l'interface. Elle possède en plus au moins une méthode "ejbCreate". C'est l'EJBObject, généré par le conteneur lors de l'installation du bean qui implémente l'interface remote ShoppingCart.

Comme les beans de session n'implémentent pas la persistance automatique, des accès explicites aux bases de données doivent être codés, si besoin, dans les méthodes métier.

Un client commence par chercher l'EJBHome en utilisant JNDI. Il obtient ainsi la racine de la hiérarchie de nommage JNDI. L'appel à InitialContext() pourra comporter un argument "properties" donnant l'adresse du serveur J2EE et le port sur lequel il écoute. Ensuite le client appelle la fonction de recherche du bean :

```
Context ctx = new InitialContext();
Object cartref = ctx.lookup("nom du bean");
```

L'interface d'accueil (Home) est définie de la façon suivante :

```
public interface CartHome extends javax.ejb.EJBHome {
   Cart create(int itemNumber) throws java.rmi.RemoteException; }
```

Ensuite, il faut transtyper la référence vers l'objet Home à l'aide d'une méthode "narrow". L'utilisation de narrow (qui n'est pas obligatoire si on a un client java communiquant avec le serveur par RMI) permet de rendre le client indépendant du protocole entre lui et le serveur.

```
CartHome home = (CartHome)javax.rmi.
    PortableRemoteObject.narrow(cartref, CartHome.class);
```

Maintenant le client peut alors provoquer la création d'une instance d'objet EJB puis appeler les méthodes déclarées dans l'interface distante :

```
cart = home.create();
cart.addItem( n );  cart.purchase();
```

SR03 2004 - Cours Architectures Internet - Serveur d'application ZOPE

25 SR03 2004 - Cours Architectures Internet - Serveur d'application ZOPE

25.1 Exemple de hoix d'une technologie de serveur

Exemple de choix d'un serveur pour un catalogue des UVs modifiable en ligne, depuis le web.

Solutions possibles

- **Serveur web + PHP + stockage persistant**
  - stockage sous forme de base de donnée
  - stockage sous forme de fichiers html ou XML
- **Serveur web + servlets et JSP + stockage**
  - stockage sous forme de base de donnée
  - stockage sous forme de fichiers XML
  - stockage sous forme d'objets java sérialisés
- **Serveur d'application**
  - J2EE (weblogic, websphere, iplanet, jakarta/tomcat)
  - PHP : SPIP
  - Zope

Choix d'une solution

- **solution (1) (PHP+XML) :** testée dans un projet étudiant ; fonctionnelle, mais avait un certain nombre de défauts la rendant difficile à utiliser telle quelle : la non prise en compte à la conception du caractère bilingue, obligeait à ré-écrire le code PHP et le code XSLT (c'est-à-dire presque tout !)
- **solution (2) (Java+J2EE) :** apparait comme largement "overkill" sur cette application, même avec un serveur "léger" tel que Jboss
- **solution (3) (Zope) :** apparaissait intéressante sous plusieurs aspects : solution légère facile à mettre en œuvre ; occasion de tester quelque chose de nouveau et intéressant avec de faibles risques, l'application étant petite open-source intégrable, si besoin, à l'existant connu (base de données SQL, serveur web, PHP)

25.2 Zope : qu'est ce que c'est ?

Zope est un serveur d'applications Internet :
- open-source,
- fonctionnel et fiable,
- utilisable seul comme serveur web ou en collaboration avec Apache,
- utilisé suffisamment : il y a une communauté active,
- développé par Zope Inc. qui l'a mis en open-source et vend du service (création de sites web dynamiques et e-commerce).

Fonctionnement d'un serveur Zope

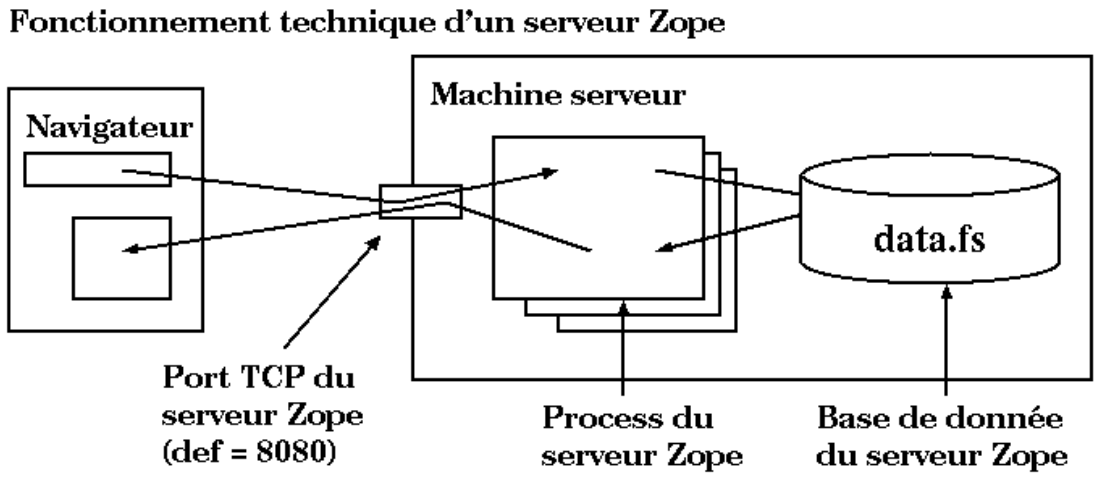


FIG. 221: Un serveur Zope

Toutes les requêtes se traduisent par l'exécution de code dans le serveur.

Ce code fabrique "à la volée" la page html renvoyée au client.

Zope utilise deux langages :

un langage de script ("**dtml**") : similaire à PHP dans son principe, un script dtml peut être "mélangé" à une page html et être exécuté à la suite d'une requête ; il va de même construire dynamiquement le code html renvoyé au client. Le client n'a aucun moyen de voir le source dtml.

le langage natif de Zope : **python**. Il est possible d'ajouter à un site Zope des scripts python, mais aussi de **créer de nouveaux objets Zope** qui seront des instances d'une classe python que l'on va écrire et intégrer dans le serveur Zope.

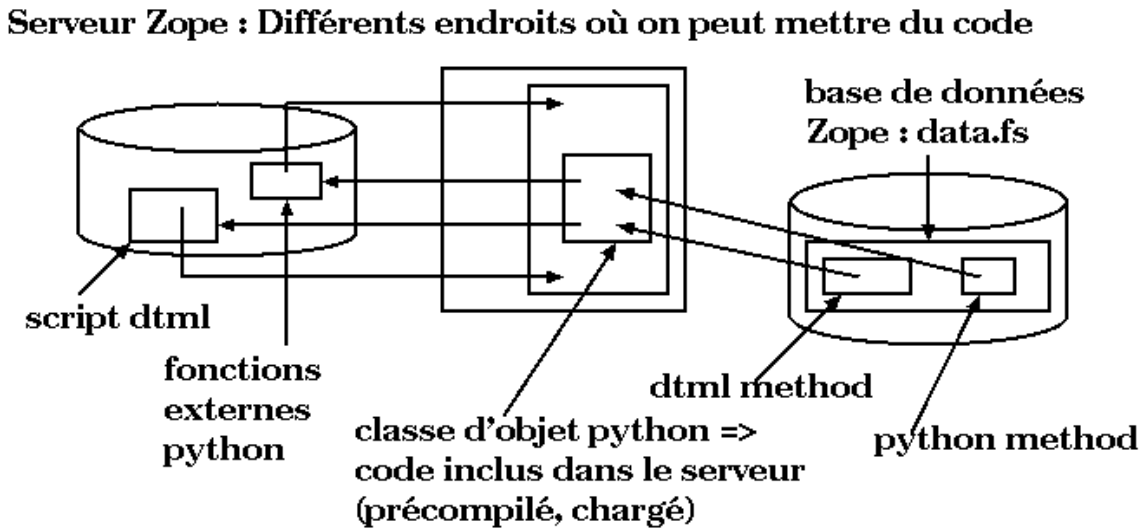


FIG. 222: Architecture d'un serveur Zope

En quoi Zope est-il un **serveur d'applications** ?

les process Zope (multithread) conservent le code applicatif en mémoire : il n'y a pas de surcoût lié à la création d'un process à chaque requête ;

bon, mais le mod-CGI et le mod-PHP de Apache fait la même chose ...

le code en mémoire peut garder un **état** => plus facile d'implémenter des sessions

le concepteur de l'application peut choisir de construire son site avec des pages entièrement statiques ou bien des pages entièrement dynamiques, avec tous les intermédiaires possibles.

Toutes les requêtes se traduisent par l'exécution de code dans le serveur.

Ce code fabrique "à la volée" la page html renvoyée au client.

Le serveur Zope gère une collection d'objets organisés selon une hiérarchie arborescente comme un ensemble de répertoires et de fichiers.

Parmi ces objets, on trouve :

– des répertoires
– des scripts dtml
– des scripts python
– des objets applicatifs

25.3 Fonctionnement d'un serveur Zope : La hiérarchie d'objets

Faire une requête sur un serveur Zope revient le plus souvent à "appeler" un des objets de la hiérarchie. L'objet va alors exécuter sa méthode "view", ce qui a pour effet de se faire voir (montrer le rendu de l'objet) dans le navigateur du client.

Tout objet a une méthode "view" : s'il n'en définit pas, il hérite de celle de son objet parent.

Plus exactement, plutôt que d'héritage (réservé aux instances d'objets python) ; on parle, sous Zope, **d'acquisition**. Si un objet ne définit pas la méthode "view", le serveur va remonter la hiérarchie des objets à la recherche d'une telle méthode, et exécuter la première qu'il trouve sur l'objet.

Fonctionnement d'un serveur Zope : L'acquisition

L'acquisition est différente de l'héritage car elle est **dynamique**. Si on place une nouvelle méthode dans la hiérarchie, "plus près" de l'objet invoqué, c'est cette nouvelle méthode qui sera invoquée par le mécanisme d'acquisition, et non pas la méthode de la classe dont hérite l'objet par construction.

==> l'acquisition fonctionne lors de la recherche d'une méthode **dans la hiérarchie des objets et répertoires** et non pas seulement dans la hiérarchie d'héritage.

Quand un objet est situé dans un répertoire, les "objets-méthodes" de ce répertoire sont vus comme des méthodes de l'objet. On peut les invoquer, les appeler entre elles et passer des paramètres.

Stockage des données par Zope

Zope peut accéder aux bases de données classiques (MySQL, PostGreSQL, Oracle, Sybase, ..).

Mais il possède aussi sa propre base de données objets : il y stocke des objets python sérialisés.

Zope peut aussi accéder au système de fichiers de la machine hôte et manipuler des fichiers.

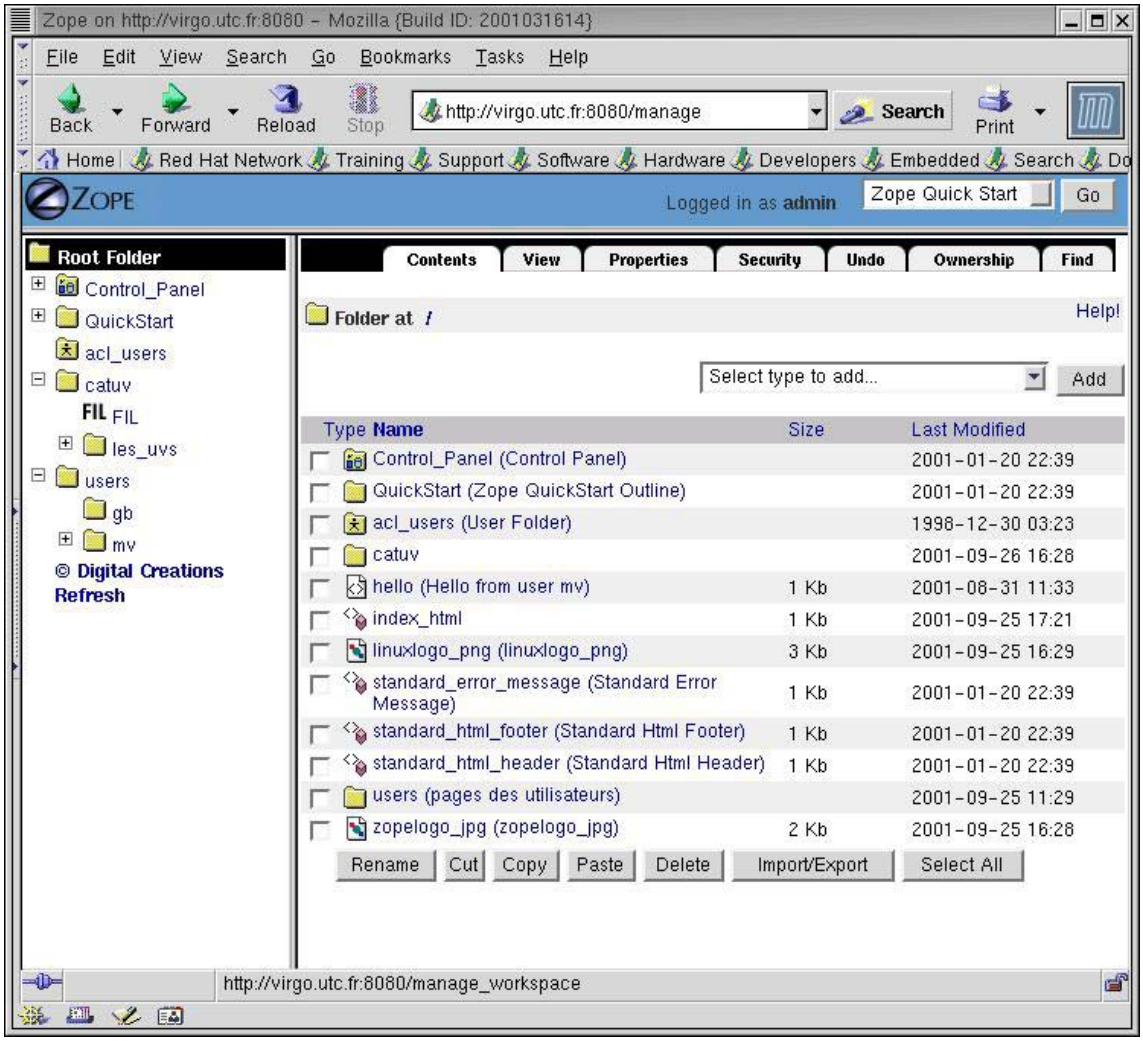


FIG. 223: La page d'accueil de gestion de Zope

Conclusion sur Zope

Zope est un vrai serveur d'application, comportant des caractéristiques intéressantes :
- puissance fonctionnelle
- développement rapide (une fois passée la courbe d'apprentissage)
- maintenance facile (le code python est facile à lire)
- s'intègre facilement à un existant
- gère les protections et la sécurité
- multi-plateforme (grâce à python)

26 SR03 2004 - Cours Architectures Internet - PVM

26.1 Introduction à PVM ("Parallel Virtual Machine")

PVM est un logiciel (programmes et bibliothèques) permettant d'utiliser un ensemble de machines reliées par un réseau IP comme des processeurs d'une machine virtuelle parallèle, pour des applications de calcul parallèle.

PVM est issu d'un projet du ORNL (Oak Ridge National Laboratory) en 1989. Les sources sont presque depuis le début du domaine public.

PVM est disponible sur pratiquement toutes les machines connues : tous les Unix, Linux, Windows, MAC-OS, Open-VMS, IBM-MVS, ...

Il est très facile à installer, il peut même être installé en mode utilisateur dans le répertoire privé d'un utilisateur.

PVM n'exige pour fonctionner entre deux machines que deux choses :

1. avoir été installé sur les deux machines,
2. que l'utilisateur puisse faire "rsh" d'une machine à l'autre.

PVM va permettre à l'utilisateur d'écrire des applications parallèles en provoquant l'exécution de programmes ("task") sur les machines distantes.

Grâce à l'empaquetage des données échangées par la bibliothèque XDR, PVM peut fonctionner entre des machines hétérogènes (constructeur différents, modèles mémoire différents). Par exemple, on pourra "enroler" dans une machine PVM des stations "sparc", des serveurs IBM Power-PC, des PC Intel sous Linux et d'autres sous Windows, des Macintosh, ... tout cela pour coopérer à une même application.

PVM est interfacé avec les deux principaux langages du monde du calcul scientifique intensif : C et Fortran.

Si PVM est installé sur une machine multiprocesseurs, il pourra utiliser le mémoire partagée pour faire communiquer plus efficacement entre elles les tâches allouées à chacun des processeurs.

26.2 Les principes de foncionnement de PVM

Chaque machine physique participant à la "machine PVM" sera identifiée par un numéro (un entier). PVM crèe une table de correspondance entre les numéros de processeur de la machine virtuelle et les adresses IP de la machine physique correspondante.

L'application est découpée (par le programmeur) en **tâches**. Une tâche PVM sera implémentée par un process. PVM va attribuer à chaque tâche un numéro d'identification unique : le **TID**.

Le TID est différent du numéro de processeur, car on peut décider de faire exécuter plusieurs tâches sur le même processeur. En fait, le TID est composé de la concaténation du Nř de processeur et du PID du process implémentant la tâche plus deux bits de contrôle "S" (démon PVM où tâche) et "G" (Group multicast)

En effet, il est possible de créer des groupes de tâches vers lesquelles on pourra faire du multicast.

Le mécanisme de communication de base de PVM est le passage de messages entre tâches ("message passing").

La bibliothèque PVM implémente des protocoles spécifiques, au-dessus de TCP et de UDP pour réaliser au mieux cette communication inter-tâches par messages.

26.3 L'implémentation de PVM

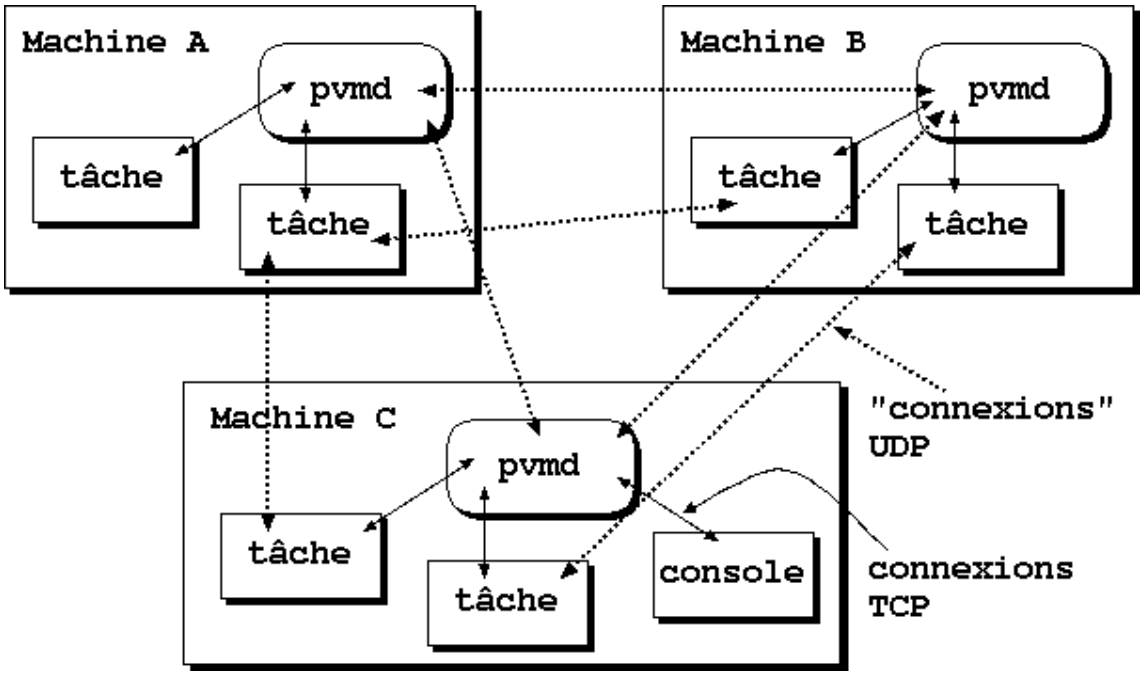


FIG. 224: Architecture de PVM

PVM est composé de trois parties :

1. le **démon PVM (pvmd)** qui doit être lancé sur chaque machine qui participe à la machine virtuelle ; ces démons écoutent les messages en provenance de la **console PVM** et font le routage des messages entre tâches ; ils coopèrent par l'intermédiaire de sockets UDP ;
2. la bibliothèque pvm (**libpvm**) contient des fonctions pour envoyer et recevoir des messages, ainsi que pour préparer le contenu des messages (appel à XDR) ;
3. une **console PVM**, tâche particulière, résultat de la commande **pvm** qui va servir à gérer le cycle de vie des tâches, des démons pvm et donc de la machine virtuelle PVM.

26.4 Configurer l'environnement et la machine virtuelle PVM

Configurer l'environnement

PVM a besoin de variables d'environnement indiquant l'endroit où est installée la bibliothèque. Comme ces variables d'environnement doivent être définies dans **tous** les process participant à une machine virtuelle, y compris ceux créés à distance par rsh, il est indispensable de

mettre cette définition dans un fichier exécuté par le système lors de la création d'un process. Ce fichier dépend du shell par défaut utilisé. Par exemple pour le shell tcsh ce sera soit **.tcshrc**, soit, s'il est absent **.cshrc** ; pour un shell bash, ce sera **.bashrc**.

On mettra dans ce shell une ligne du type **source /set_pvm_env** ou bien **. /set_pvm_env**.

Le fichier set_pvm_env contiendra par exemple :

```
# set up env var for pvm
setenv PVM_ROOT /usr/local/pvm/pvm3

if ($?PVM_ROOT) then
        setenv PVM_ARCH ‘$PVM_ROOT/lib/pvmgetarch‘

# uncomment the following line if you want the PVM
# executable directory to be added to your shell path.

       set path=($path $PVM_ROOT/bin/$PVM_ARCH)
       set path=($path $PVM_ROOT/lib/$PVM_ARCH)
       set path=($path $PVM_ROOT/lib)
       set path=($path $PVM_ROOT/include)

setenv MANPATH ‘printenv MANPATH‘:$PVM_ROOT/man

endif
```

Dans le cas d'un shell bash, les commandes "setenv VAR value" sont remplaçées par la syntaxe export VAR=value .

Afin d'éviter de permettre l'utilisation de rsh sans avoir à fournir un couple username/password à chaque création de process, on utilise, entre les machines servant de support à la machine virtuelle PVM, la fonction ".rhosts".

Si la machine "bravo" contient dans le fichier /.rhosts de l'utilisateur "sr03nn" les lignes suivantes :

```
echo
golf
kilo
```

alors, on pourra faire un **rsh bravo** depuis une de ces trois machines, sans avoir à donner de mot de passe. Un login shell sera créé sur bravo, sous le compte du user "sr03nn".

Configurer la machine virtuelle PVM

La configuration de la machine virtuelle PVM consiste à indiquer combien de "processeurs virtuels" on désire utiliser dans notre application, et sur quelles machines physiques vont être créés (sous forme de démons pvmd) ces processeurs virtuels.

Ceci se fait par un fichier "pvm_hostfile" qui a l'aspect suivant :

```
# fichier hostfile de configuration d’une machine virtuelle PVM
#
bravo ep=$HOME/pvm
echo  ep=$HOME/pvm
golf  ep=$HOME/pvm
kilo  ep=$HOME/pvm
```

Ici on crèe 4 processeurs virtuels. Dans chaque cas on défini la variable pvm "ep" qui indique **le chemin d'accès aux exécutables PVM**. C'est-à-dire, à quel endroit, le programme pvm lançé depuis une tâche sur un processeur virtuel devra chercher l'exécutable.

Dans l'exemple ci-dessus, tous les chemins sont identiques car on est dans le cas simple ou toutes les machines citées font partie du même ensemble NFS. Et donc, l'utilisateur "sr03nn" a le même répertoire "HOME" sur toutes les machines.

Mais ceci n'est pas obligatoire, on pourrait ajouter à la machine PVM une machine physique située ailleurs (éventuellement, même, sur un autre site géographique) et pour laquelle les noms de répertoires sont différents.

Par exemple l'utilisateur "sr03nn" pourrait installer PVM dans son compte "vega" et utiliser son username "jdupond" sur vega pour lancer une tâche PVM depuis sunserv, en ajoutant à son fichier "pvm_hostfile" une ligne du type :

```
vega.utc.fr lo=jdupond dx=pvm/pvmd ep=$HOME/pvmexe
```

Le fichier "pvm_hostfile" est un fichier de données qui va être passé en paramètre de la commande **pvm** qui lance la console pvm : > **pvm pvm_hostfile**.

```
77 sr03 sunserv:~/pvm> pvm pvm_hostfile
pvm> conf
3 hosts, 1 data format
                     HOST     DTID     ARCH   SPEED
                  sunserv    40000 SUN4SOL2    1000
                    bravo    80000 SUN4SOL2    1000
                     golf   100000 SUN4SOL2    1000
pvm>
```

On a maintenant une machine virtuelle constituée de 3 processeurs.

Attention ! Il a deux façons de sortir de la console pvm :

- **pvm> quit** sort de la console pvm **sans arrêter les démons** : la machine virtuelle reste en fonction.
- **pvm> halt** provoque l'arrêt de tous les démons avant de sortir de la console. C'est la commande à utiliser pour "nettoyer" la machine virtuelle.

Autre commandes utiles :

- **pvm> ps** : liste des process pvm en cours ;
- **pvm> kill numéro_process_pvm** : tuer un processus pvm ;
- **pvm> pstat numéro_process_pvm** : afficher le status d'un processus pvm ;