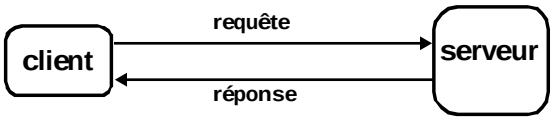
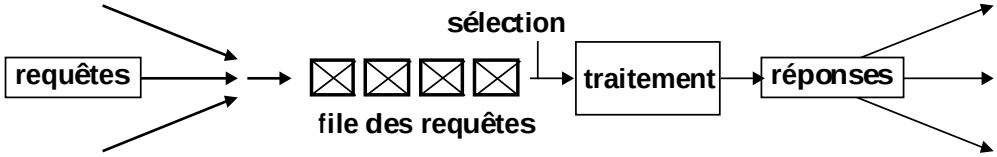
Modèle Client-Serveur
Partage du serveur entre clients

- Un serveur peut servir plusieurs clients
- Vu d'un client particulier

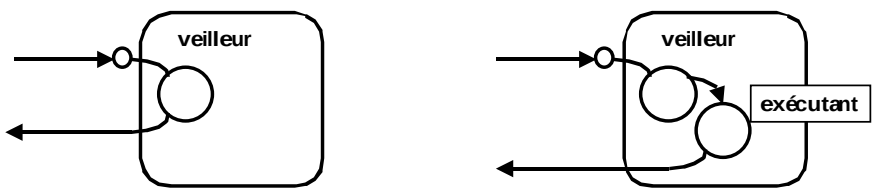


- Vu du serveur
  - Gestion des requêtes (priorité)
  - Exécution du service (séquentiel, concurrent)
  - Mémorisation ou non de l'état du client



Réalisation de l'appel de procédure à distance

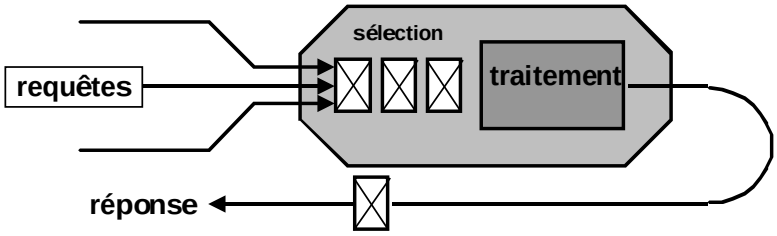
- Gestion de processus côté serveur
  - Sur le site serveur (site appelé) la procédure distante est exécutée par un processus
  - Plusieurs schémas possibles
    - Dans tous les cas, un processus "veilleur" attend derrière une porte spécifiée (le numéro de la porte dépend du service appelé)
    - L'appel se traduit par un message envoyé au veilleur, contenant le nom de la procédure appelée et les paramètres.
      - ou bien le veilleur exécute lui-même la procédure et renvoie les résultats à au processus client
      - ou bien le veilleur crée un nouveau processus exécutant (lourd ou léger) pour exécuter la procédure ; c'est ce processus qui renverra les résultats



Gestion des processus dans le serveur (1)

- **Processus serveur unique (exécution des requêtes par le veilleur)**

```
while (true) {
    receive (client_id, message) ;
    extract (message, service_id, params) ;
    do_service[service_id] (params, results) ;
    send (client_id, results) ;
} ;
```



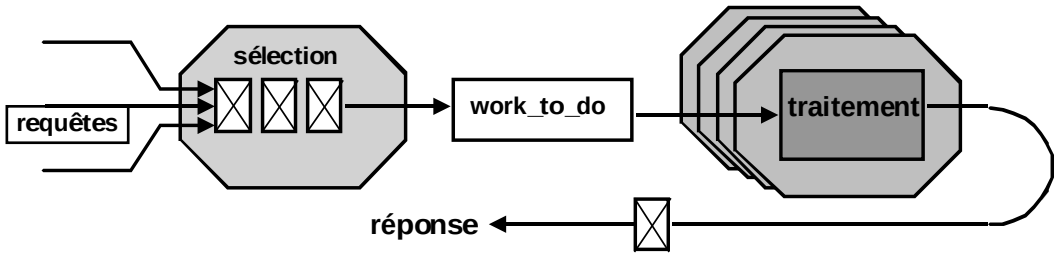
Gestion des processus dans le serveur (2)

Processus veilleur

```
while (true) {
      receive (client_id, message)
      extract (message, service_id,
                params) ;
      work_to_do.put (client_id,
          service_id, params) ;
};
```

Pool d'exécutants

```
while (true) {
      work_to_do.get (client_id,
          service_id, params) ;
      do_service{service_id]
          (params, results)
      send (client_id, results)
      };
```



Gestion des processus dans le serveur (3)

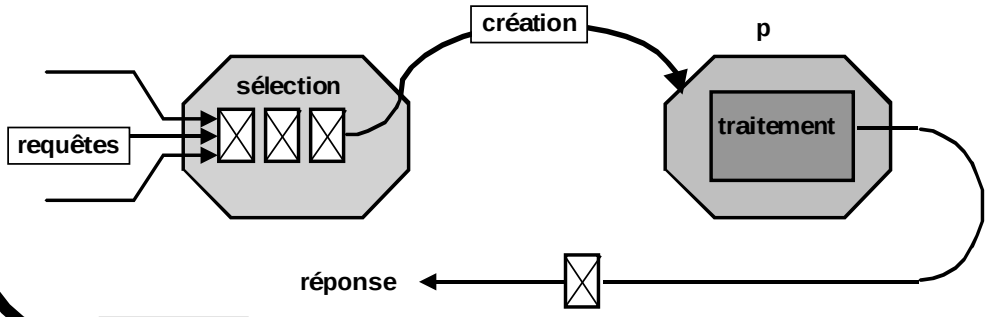
Processus veilleur

```
while (true) {
        receive (client_id, message)
        extract (message, service_id,
                    params) ;
        p=create_thread (client_id,
            service_id, params) ;
};
```

Création dynamique des exécutants

programme de p

```
do_service{service_id]
    (params, results)
send (client_id, results)
exit ;
```



Réalisation de l'appel de procédure à distance

- Problèmes d'identification et de désignation
  - le client doit pouvoir désigner le serveur (pour lui envoyer la requête)
  - le serveur doit pouvoir désigner le client (pour lui renvoyer les résultats)
- Solutions possibles
  - Désignation du serveur par le client
    - le client connaît (par convention générale ou par accord direct) le site et le numéro du porte du serveur - le problème est résolu
    - le client ne connaît qu'un nom symbolique du service
      - dans ce cas, on passe par un service de désignation, qui fournit le couple (site, numéro de porte) corrspondant au nom symbolique du service
      - il faut connaître directement le site et le numéro de porte du service de désignation
  - Désignation du client par le serveur
    - dans la requête, le client indique un site et un numéro de porte vers lesquels renvoyer la réponse
- Dans la pratique
  - On utilise des noms symboliques, et les opérations de désignation sont invisibles aux utilisateurs

Réalisation de l'appel de procédure à distance en Tcl-DP (1)

- Introduction à Tcl-DP
  - Tcl-DP est un logiciel ("middleware") qui réalise une extension répartie de DP
  - Services fournis aux utilisateurs : appel de procédure à distance et objets répartis
    - également : utilisation directe du protocole TCP (nous n'en parlons pas)
- Les opérations pour l'appel de procédure à distance (RPC)
  - Phases préliminaires
    - côté serveur, le lancement d'un processus veilleur qui attend derrière une porte spécifiée, sur le site serveur
      - dp_MakeRPCServer numéro_de_porte
    - côté client, une phase de connexion préalable du client au serveur dans laquelle se déroule l'identification mutuelle (connaissance de la désignation)
      - dp_MakeRPCClient machine numéro_de_porte
        - résultat de cette commande :un "descripteur" dont la valeur doit être conservée ; on écrit donc en général :
        - set serveur [dp_MakeRPCClient machine numéro_de_porte]
    - Plusieurs clients peuvent se connecter à un même serveur
  - Une fois ces deux opérations réalisées, le client peut appeler le serveur en utilisant l'identification mise en place
    - dp_RPC descripteur_de_serveur commande

Réalisation de l'appel de procédure à distance en Tcl-DP (2)

- Lancement du veilleur
  - dp_MakeRPCServer numéro_de_porte # exécuté chez le serveur
  - Le numéro de porte doit être choisi (>1024) ; si 0, c'est le système qui alloue un numéro
- Connexion du client au serveur
  - set serveur [dp_MakeRPCClient machine numéro_de_porte] # exécuté chez le client
  - machine = site du serveur (nom symbolique ou adresse IP) ; numéro_de_porte est celui avec lequel le veilleur a été lancé, il doit être connu (convention entre serveur et client)
  - La variable serveur reçoit un descripteur du serveur (les détails n'ont pas à être connus de l'utilisateur)
  - Attention : le veilleur doit avoir été lancé avant toute connexion de client
- Appel de procédure proprement dit (nécessite connexion préalable)
  - dp_RPC $serveur commande # exécuté chez le client
    - la commande est exécutée dans l'espace du serveur (les variables qui y figurent sont totalement invisibles au client)
    - la commande peut être toute procédure déclarée dans l'espace du serveur
    - effet de bord : une variable dp_rpcFile (dans l'espace du serveur) reçoit le descripteur du client (symétrie entre client et serveur, chacun peut maintenant appeler l'autre sans connexion, avec dp_RPC)

Rappels sur la programmation par objets

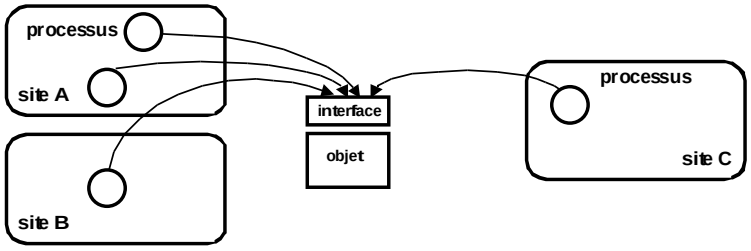
- **Motivations**
  - Améliorer la structuration et la réutilisabilité des programmes
    - facilité de compréhension et de modification, constructions génériques réutilisables
- **Notions de base**
  - Encapsulation
    - Un objet "encapsule" un état (ensemble de données), accessibles uniquement au moyen d'un ensemble de fonctions (méthodes) qui constituent l'interface de l'objet
    - L'interface définit tout ce qui est nécessaire à l'utilisation de l'objet ; on peut remplacer une réalisation par une autre, à condition de respecter l'interface
  - Classes et instances
    - Un objet est un exemplaire ("instance") d'une classe ; les instances d'une classe ont la même interface mais diffèrent par la valeur de l'état
    - Les instances sont créées et détruites dynamiquement
  - Héritage
    - Dans beaucoup de cas, on peut définir une classe à partir d'une autre, par spécialisation, enrichissement, surcharge. La nouvelle classe hérite de l'ancienne. Intérêt : réutilisation de l'effort de conception et de la réalisation, meilleure organisation du logiciel
  - Polymorphisme
    - Dans certains cas, des méthodes acceptent des paramètres de plusieurs types et ont un comportement différent selon le type

Objets répartis

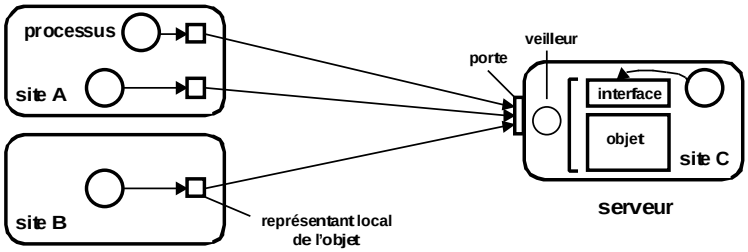
- **Motivations**
  - Fournir un mode d'organisation d'applications réparties privilégiant le partage d'informations réparties sur plusieurs sites entre des utilisateurs eux-mêmes répartis
- **Principes**
  - On définit un ensemble d'objets répartis, utilisables via leurs méthodes, avec les conditions d'accès suivantes
    - "transparence" de localisation : un objet est désigné par un nom logique indépendant de sa localisation physique (la localisation peut changer sans que le nom change)
    - "transparence" d'accès : on accède à un objet distant exactement de la même manière qu'à un objet local
- **Utilisations possibles**
  - Applications mettant en œuvre des données partagées réparties que l'on veut rendre globalement accessibles
    - édition coopérative
    - ingénierie coopérative (conception assistée par ordinateur avec partage de données)
    - autres formes de travail coopératif (télé-enseignement, télé-médecine)
    - documentation

Réalisation des objets répartis en Tcl-DP

- **Principe**
  - Schéma idéal (organisation symétrique)



  - Schéma réel (organisation autour d'un serveur))



Réalisation des objets répartis en Tcl-DP

- **Organisation générale**
  - Pour chaque objet à répartir, on choisit un serveur responsable de cet objet, et on définit l'organisation de la classe de l'objet (composants de l'état, méthodes)
  - Sur chaque serveur : on crée le ou le objet(s) et on les initialise (on fixe la valeur initiale de l'état)
  - Sur chaque client (utilisateur) d'un objet : on fait appel au serveur de l'objet pour le "distribuer" au client. Cette opération crée un représentant local de l'objet sur le site du client. Pour le client, tout se passe comme si le représentant "était" l'objet
  - (facultativement) Chez le serveur et chez les clients : mise en place de "déclencheurs" : mécanisme qui associe une action à chaque modification de l'état d'un objet. Utile par exemple pour réafficher un objet à chaque modification
- **Structure (de la classe) d'un objet**
  - L'état d'un objet est un ensemble de champs (*slots*). Chaque champ a un nom et une valeur
  - Chaque objet possède un ensemble de méthodes. Il y a trois méthodes obligatoires (tout objet doit les posséder)
    - `configure` : accès à l'état de l'objet
    - `slot-value` : lecture de la valeur d'un champ
    - `destroy` : détruit l'objet (instance)
  - La création d'une instance est une procédure qui s'applique à une classe

Réalisation des objets répartis en Tcl-DP

- **Primitives**
  - `dp_objectCreateProc class object` -- créé une instance
  - `dp_objectSlotSet object slot value` -- définit un champ dans un objet et l'initialise
  - `dp_objectSlots object` -- renvoie la liste des champs de l'objet
  - `dp_objectConfigure class object ?args?` -- configure un objet pour pouvoir y accéder
    en particulier définit quels sont les champs de l'objet et fixe leur valeur initiale
  - `dp_distributeObject object processes makeObject`
    distribue un objet (créé au préalable dans un serveur) à un ensemble de processus
    `processes` = liste des descripteurs des processus
    `makeObject` = la procédure de création associée à la classe de l'objet

    cette procédure est exécutée par le serveur, à la demande des clients
  - `dp_setf object slot value` -- affecte une valeur à un champ d'un objet (et la distribue automatiquement)
  - `dp_getf objet slot` -- renvoie la valeur d'un champ d'un objet
  - `dp_SetTrigger when object slot trigger_list` -- met en place un déclencheur
  - … encore quelques autres primitives

Étapes de l'utilisation d'un objet réparti (simplifié)

Écrire le programme de la classe de l'objet, comprenant la procédure de création (qui définit les champs de l'objet et les initialise) et les différentes méthodes de l'objet
on utilise pour cela `dp_ObjectCreateProc`, `dp_objectConfigure`, et les procédures d'accès aux champs
Écrire les programmes du client et du serveur, et les programmes auxiliaires (graphiques, etc.)

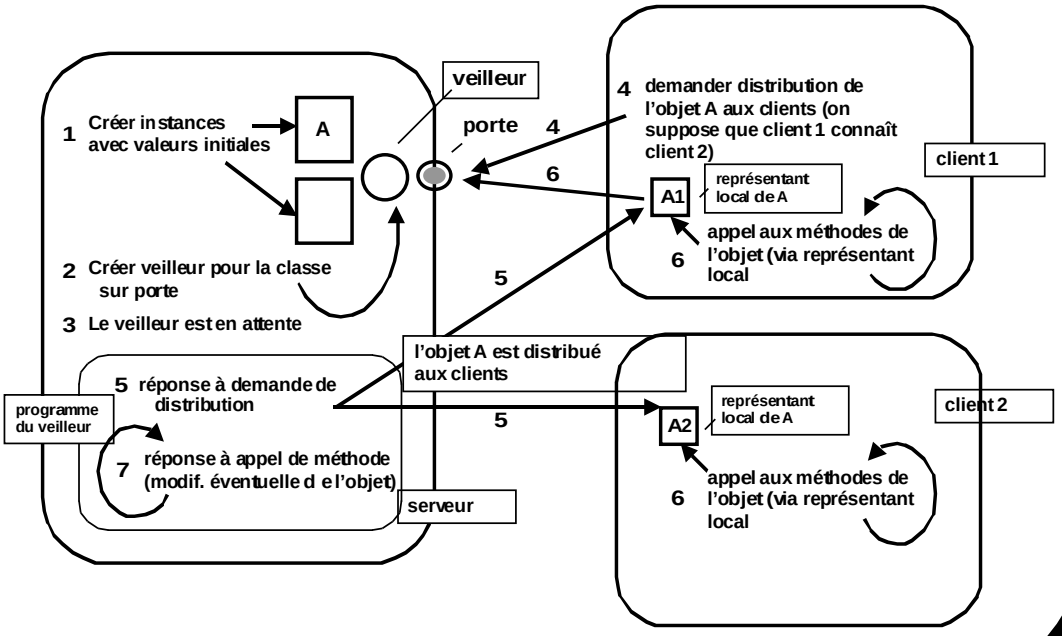
Sur le site serveur de l'objet :
créer autant d'instances de l'objet que nécessaire initialement
créer le processus serveur pour la classe de l'objet (comme un veilleur de RPC, avec une porte spécifiée)
placer si nécessaire les déclencheurs associés aux changements d'état

Sur chacun des sites clients de l'objet :
appeler le serveur (par `dp_RPC`, sur la porte convenue) et lui demander de distribuer chacune des instances à ce client (le serveur connaît l'identité du client après le RPC)
placer si nécessaire les déclencheurs associés aux changements d'état

Après ces étapes préliminaires, on peut utiliser indifféremment les objets sur n'importe quel site (serveur ou client) au moyen de leurs méthodes. Les réactions aux changements d'état sont exécutées grâce aux déclencheurs
Dans beaucoup d'applications, l'appel des méthodes est lui-même associé (via `bind`) à des opérations graphiques

Voir exemples en TP

Étapes de l'utilisation d'un objet réparti



Résumé de la séance 10

- **Applications client-serveur**
  - applications client-serveur utilisant l'appel de procédure à distance; principe, mise en œuvre dans Tcl, exemples
  - introduction aux applications client-serveur à base d'objets partagés ; principe, mise en œuvre dans Tcl, exemples
- **Plan de la suite**
  - applications client-serveur sur l'Internet
    - Service de noms : DNS (séance 11)
    - World Wide Web : fonctionnnement et utilisations (séances 11 et 12)