Introduction

Le système X-Window a été développé au M.I.T. dans le cadre du projet Athena. Parmi les auteurs d'origine figurent Robert Scheifler et Jim Gettys mais il y a eu en réalité de nombreuses autres contributions (différentes universités américaines ainsi que la plupart des grandes sociétés constructeurs de matériel informatique ont participé au développement de ce projet, par exemple Sun, Apollo, Tektronix, Digital et IBM, etc.). Le système réalisé aujourd'hui est donc le produit du travail de plusieurs dizaines de personnes. Il a été développé dans le but de réaliser une plate-forme portable pour la programmation graphique. Les constructeurs ont la charge d'implanter les fonctions de base du serveur X, mais la librairie d'interface Xlib ne varie pas d'une machine à l'autre. Aujourd'hui X sert de base dans les systèmes Unix pour l'architecture d'outils de générations d'interface plus sophistiqués comme les *toolkits* Motif ou Openlook qui permettent la création d'objets interactifs comme les boutons, les ascenseurs, les barres de menus etc. Ces *toolkits* définissent des styles d'interfaces différents mais sont construites par dessus la librairie Xlib.

L'utilisation des *toolkits* résout un certains nombre de problèmes concernant la programmation des interfaces graphiques sous Unix, mais dès que l'on souhaite réellement faire un logiciel comportant autre chose que des boutons, des menus et des boîtes de dialogues, une bonne connaissance de la librairie graphique est nécessaire. La librairie Xlib est donc devenue en réalité incontournable pour toute personne désirant comprendre sérieusement le domaine des logiciels graphiques. La version que nous présentons ici est la Release 4. Elle est plus réduite que les versions ultérieures, mais permet de comprendre l'essentiel des principes.

Les différentes sections de ce polycopié rappellent brièvement les notions abordées dans le cours : 1. modèle Client/Serveur, 2. fenêtres, 3. événements, 4. contexte graphique et fonctions de dessins. Ces sections coïncident généralement avec une section de la documentation du MIT mais ne sauraient en dispenser totalement la lecture. Au contraire le but de ce polycopié est plutôt d'introduire agréablement à la lecture de la documentation anglaise, lecture qui pourra alors s'effectuer au coup par coup, selon les besoins en programmation de chacun.

Je me suis attachée pour cela à fournir rapidement la liste des fonctions et des structures de données essentielles de la librairie. J'ai pris en outre le parti d'insister sur les erreurs classiques des débutants. Les erreurs ou confusions à ne pas commettre seront pointées par l'icône  .

Le nom des principales fonctions de la librairie à connaître figure en caractère gras. Le type des arguments de ces fonctions n'est pas toujours précisé car il alourdirait considérablement la lecture. Cependant il est facile de consulter l'index de la documentation M.I.T. pour y retrouver (en gras dans l'index) la page correspondant à la définition de la fonction. En outre, pour faciliter une compréhension rapide, certains noms d'arguments indiqueront en réalité conventionnellement leurs types. Ainsi par exemple, une variable nommée display aura nécessairement le type Display *, de même une variable de nom screen, aura le type Screen, etc. Ainsi, conventionnellement, on admettra dans la suite les déclarations implicites suivantes :

```
Display*    display, dpy;
Screen      screen;
int         screen_num ;
Window      win, root, w;
GC          gc;
XContext    context;
XEvent      event, ev;
```

Modèle Client/Serveur

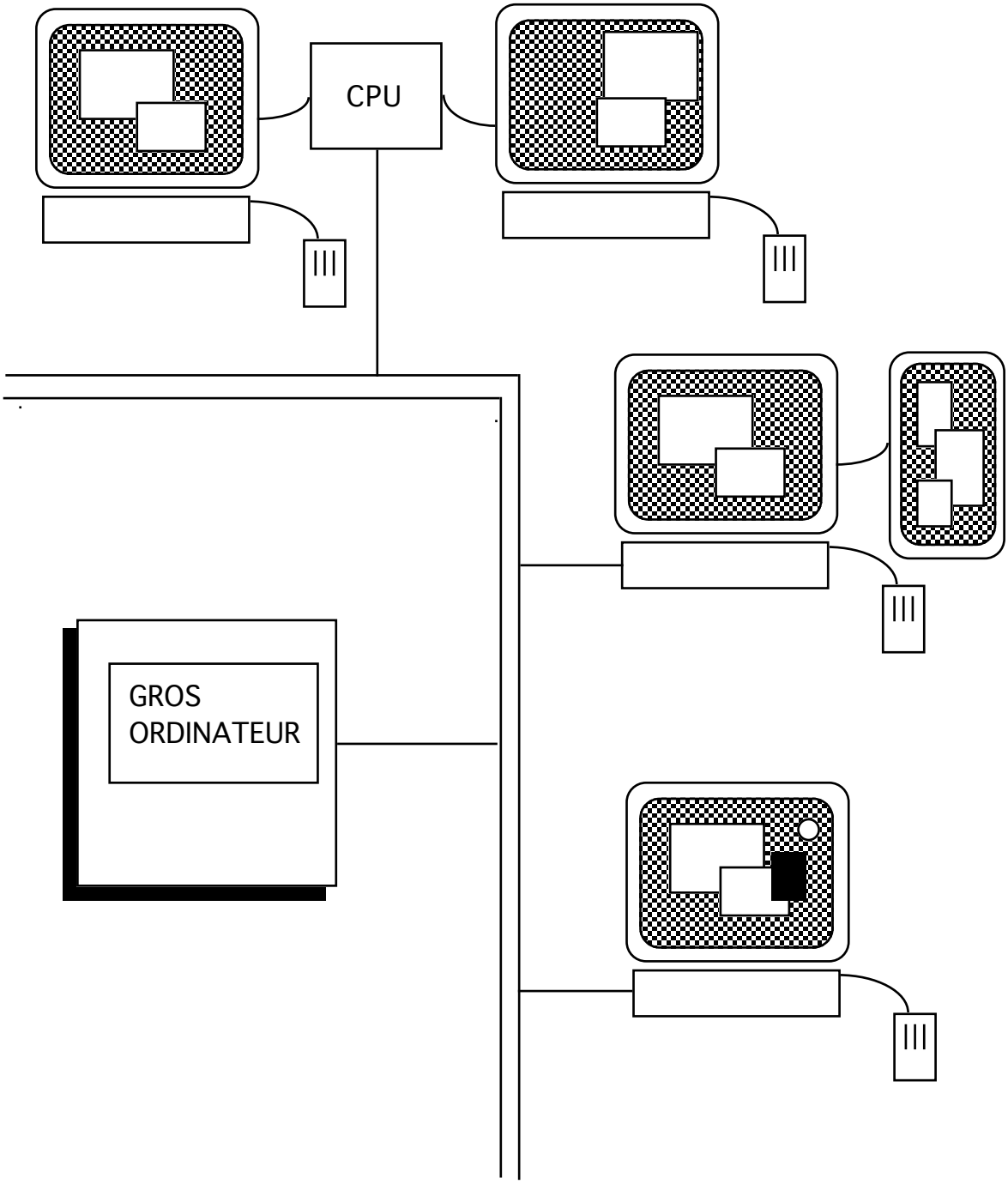

fig1. *Un Réseau de machines utilisant X-Window*

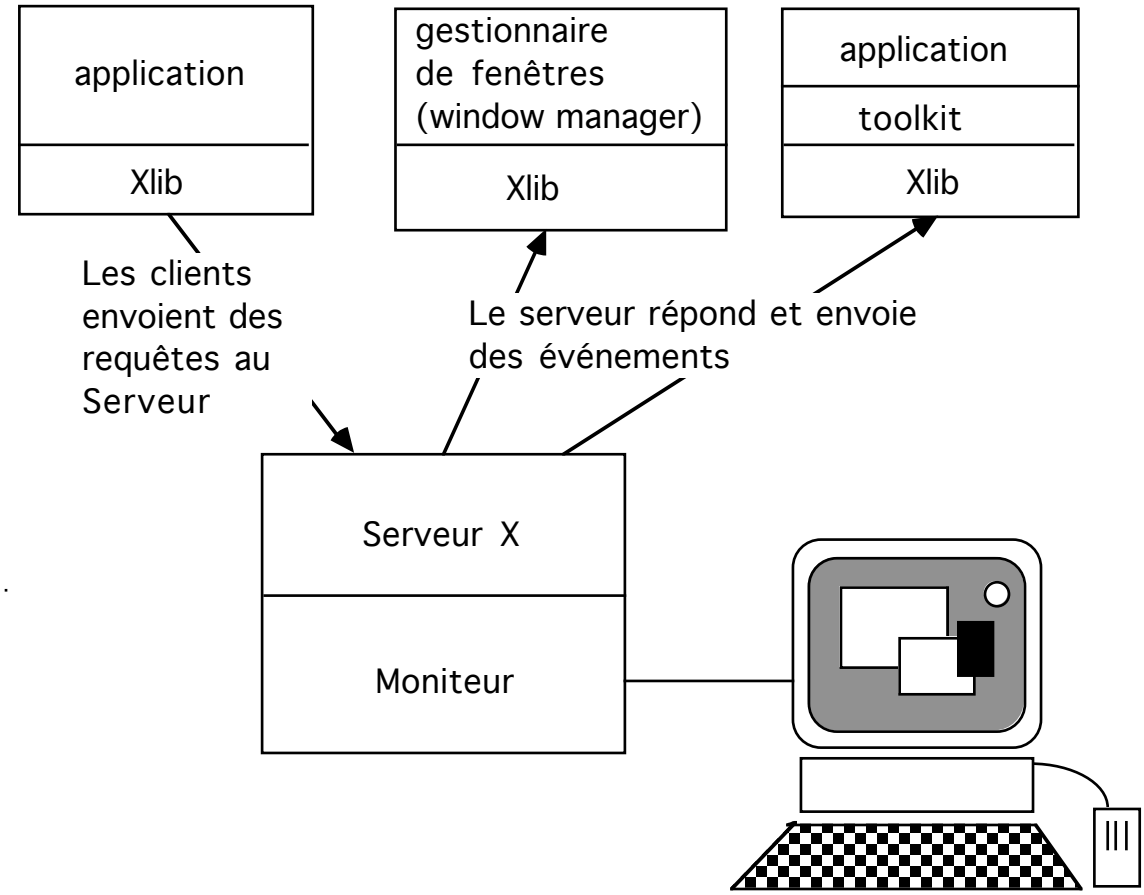

fig2. *L'architecture Client/Serveur*

Modèle Client/Serveur

La librairie de fonction Xlib est une librairie de requêtes faites auprès d'un serveur, le serveur X. L'exécution d'un programme utilisant cette librairie nécessite le lancement préliminaire du serveur (commande **xinit**) à moins qu'on ne se trouve directement connecté à un terminal spécialisé (terminal X).

Le serveur est en charge d'administrer des ressources partagées entre plusieurs programmes s'affichant sur le même dispositif d'affichage (le display) mais s'exécutant sur différentes machines d'un réseau. Ces ressources partagées sont typiquement l'écran, le clavier, la souris, mais aussi des structures plus abstraites comme des palettes de couleurs ou des fontes. Plusieurs programmes (et même plusieurs utilisateurs) peuvent effectuer des requêtes auprès du même serveur X. On appelle habituellement ces programmes des clients car le modèle de communication sous-jacent est le modèle "client/serveur" (métaphore du restaurant ou du bistrot).

Les clients communiquent avec le serveur au moyen des appels à la Xlib et inversement, le serveur envoie des informations (sur demande des clients) au travers d'unités appelées **événements**. Les événements sont produits par des actions de l'utilisateur sur les dispositifs d'entrée (souris, clavier), mais également par des actions produites par l'ensemble des programmes (affichages graphiques, destructions de fenêtres, modification de ressources partagées, appel à une fonction donnée). Du fait de cette architecture Client/Serveur, le contrôle du flot d'un programme d'interface est centré autour des événements envoyés par le serveur. Une section complète sera consacrée à la description les différents événements.


Dans un modèle Client/Serveur la communication est **asynchrone**. Les programmes communiquent avec le serveur en faisant appel aux **requêtes** (fonctions de la librairie). Mais ces requêtes ne sont, comme le mot l'indique, que des *demandes* et leur exécution en tant que fonctions C du programme client peut consister:

- en l'empilement local de la requête
  (l'envoi sera effectué plus tard lors d'un appel à une requête plus impérative)
- en l'envoi pour traitement de la requête au serveur
  (mais sans attendre le résultat du traitement)
- en l'envoi et traitement effectif de la requête par le serveur
  (cas général des fonctions ramenant des valeurs)

En tout état de cause, le serveur garantit que le traitement des requêtes sera effectué dans l'ordre des appels du client, mais le traitement de ces requêtes peut s'effectuer de manière asynchrone (par rapport au reste du programme); c'est-à-dire que si l'on fait par exemple dans le programme un appel à une procédure d'affichage, suivi immédiatement d'un appel à une fonction C pour lire un caractère, il se peut qu'on lise le caractère avant d'avoir effectivement obtenu l'affichage de la fenêtre à l'écran. Pour remédier aux problèmes de synchronisation, on dispose de deux fonctions **XFlush** et **XSynchronize. XFlush** envoie les requêtes au serveur et **XSynchronise** permet d'attendre dans l'application que les requêtes aient été traitées par le serveur avant de passer à l'instruction suivante. On utilisera principalement cette fonction pour rechercher des erreurs délicates (bugs).

Connexion au serveur

La connexion au serveur est réalisée en début de programme grâce à l'appel à la fonction **XOpenDisplay**(string). Cette fonction prend en argument une chaîne de caractères permettant d'identifier symboliquement un dispositif d'affichage, i.e. un certain nombre de périphériques (écrans, clavier et souris) sur lesquels porteront la communication avec le serveur. Le serveur retourne alors une variable de type pointeur sur Display, permettant d'identifier la connexion du client et certains éléments caractéristiques du dispositif d'affichage:

```
#include <X11/Xlib.h>
...
Display *dpy;
...
dpy = XOpenDisplay(NULL);
```

Appelée avec une chaîne de caractère vide, cette fonction consulte la variable d'environnement DISPLAY, qu'il suffit donc d'affecter convenablement au préalable au niveau du shell:

```
% setenv DISPLAY sun3:0.0
```

La variable *dpy* du programme ainsi initialisée sera passée en argument à la plupart des requêtes pour permettre au serveur d'identifier la connexion.

On peut également appeler XOpenDisplay avec une chaîne de caractères de la forme "hostname:server_number.screen_number". Cette chaîne pourra être lue par exemple comme option de lancement du programme. Ainsi le programme xterm permettant de lancer la commande

```
% xterm -display sun2:0.0
```

utilise un appel à XOpenDisplay("sun2:0.0") qui lance l'exécution du programme sur le dispositif d'affichage par défaut de la machine sun2.

La déconnexion sera réalisée au moyen de la requête **XCloseDisplay**(dpy)[1].

Initialisations diverses (à partir du display)

Pour obtenir du serveur les valeurs de différents attributs nécessaires à la bonne définition des fenêtres, on utilise généralement des macros fonctions (cf. le premier chapitre de la documentation M.I.T. concernant le *display*). Ainsi, on utilisera fréquemment :

```
int screen_num;
Window root;
unsigned long whitepixel, blackpixel;
GC gc;
               root = DefaultRootWindow(dpy);
               screen_num = DefaultScreen(dpy);
               blackpixel= BlackPixel(dpy,screen_num);
               whitepixel= WhitePixel(dpy,screen_num);
               gc = DefaultGC(dpy,screen_num);
```

Signalons aussi la présence des macros suivantes:

```
DisplayWidth(dpy,screen_num)
DisplayHeight(dpy,screen_num)
DefaultVisual(dpy,screen_num)
DefaultDepth(dpy,screen_num)
DefaultColormap(dpy,screen_num)
```

Un client particulier: le window manager

Le modèle sous-jacent au système de fenêtrage X a prévu la présence d'un client particulier, le gestionnaire de fenêtre (ou *window manager*), qui rend bien des services mais peut également troubler le programmeur débutant dans sa communication avec le serveur.

Le window manager offre interactivement à l'utilisateur le service de déplacer les fenêtres, de changer leur taille, etc., en rajoutant souvent une décoration qui lui est propre et qui permet ainsi d'uniformiser l'apparence des fenêtres

[1] Attention, cette requête doit être la dernière requête du programme, sinon on a toute chance de référencer encore la variable dpy avec cette dernière non correctement instanciée.

principales des différentes applications tournant à l'écran. Ainsi la manière dont les fenêtres sont déplacées, fermées, etc. se trouve indépendante des applications. L'intérêt du gestionnaire de fenêtre est évident : il uniformise l'ergonomie de la manipulation des fenêtres à l'écran, ce qui soulage d'autant l'utilisateur, et il évite également au programmeur la tâche de définir certaines opérations[2].

Le window manager est donc un client privilégié jouant un certain rôle dans l'organisation de l'écran. Il est important de noter cependant qu'il ne s'intéresse *qu'aux fenêtres principales*, le plus haut dans la hiérarchie des fenêtres, i.e celles qui sont directement rattachées à la racine. Il impose sa politique en ce qui concerne leur taille, position, couleur et épaisseur de bord, etc. Ainsi, un mécanisme interne par défaut[3] prévoit que les requêtes portant sur la création et la modification de la géométrie des fenêtres soient en fait interceptées par le window manager.

Les débutants pourront être surpris par la conduite de leur programme (s'ils cherchent par exemple simplement à déplacer une fenêtre principale de leur programme via un appel à XMoveWindow), car le résultat peut dépendre du window manager en présence[4]. Ainsi, un programme client donné n'aura pas le même comportement selon le contexte (ici les autres clients en présence). Cette situation peu courante en programmation traditionnelle ne fait que souligner cet aspect incontournable de la programmation sous X-Window: **les programmes partagent des ressources**. Il ne faut donc plus penser un programme comme définissant un processus isolé, mais comme le lancement d'un processus dans un contexte. Connaître la librairie graphique Xlib c'est en réalité connaître les fonctions de la librairie et le modèle sous-jacent.

---

[2] La contrepartie à payer pour le programmeur va être que les fenêtres pourront être modifiées en dehors du contrôle de son programme et donc qu'il faudra que l'application prévoit de s'informer des modifications pouvant survenir sur ses fenêtres afin d'agir en conséquence (comme par exemple en réafficher une partie).

[3] Bien qu'il soit possible en principe de se libérer de ce mécanisme (via l'attribut *override_redirect* des fenêtres), on ne le fera a priori que dans certains cas bien particuliers (menu déroulant par exemple).

[4] Beaucoup de window manager "reparentent" les fenêtres, ce qui a pour effet de modifier le repère par rapport auquel on situe la fenêtre. Au lieu d'être déplacée relativement au fond d'écran (parent initial supposé), la fenêtre se trouve déplacée relativement à la fenêtre englobante créée par le window manager pour afficher ces décorations (nouveau parent).