Les Fenêtres

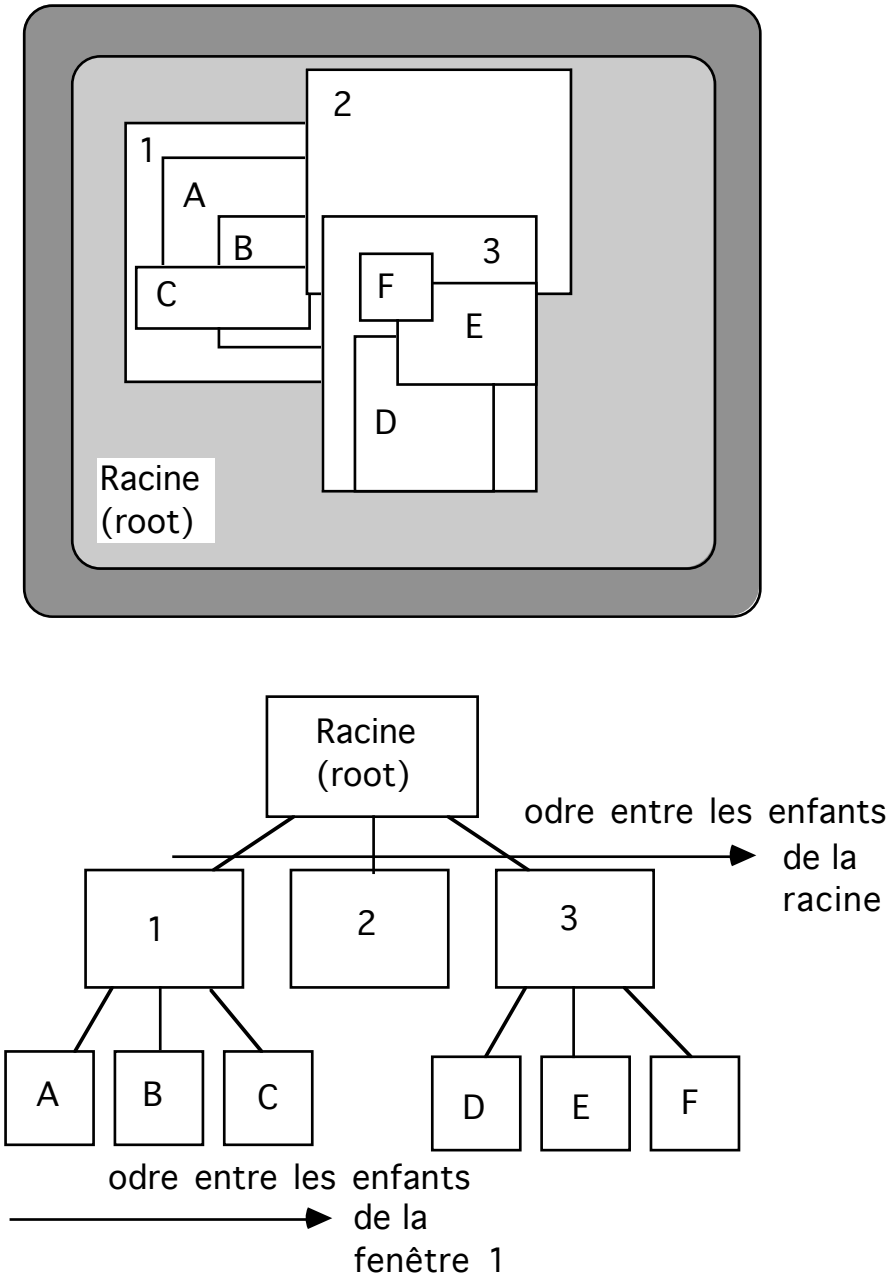

fig3. *La double hiérarchie des fenêtres à l'écran*

Les fenêtres

Les fenêtres sont la base du système X-Window. Ce sont les objets qui permettront de récupérer les événements provenant des dispositifs d'entrée et l'on dessinera *a priori* à l'intérieur de leur cadre. Elles sont hiérarchisées en fenêtre parent et fenêtres filles. Toutes les fenêtres ont un ancêtre commun: la fenêtre racine ou **root window**. Une fenêtre mère détermine un cadre (*Clip*) en dehors duquel les filles restent invisibles. Entre soeurs, les fenêtres sont susceptibles de se recouvrir partiellement et sont rangées selon un certain ordre d'empilement (cf. figure 3).

Les fenêtres possèdent un certain nombre d'attributs parmi lesquels des attributs de type géométrique[5]: position dans le repère de la fenêtre parent, largeur, hauteur, etc.,

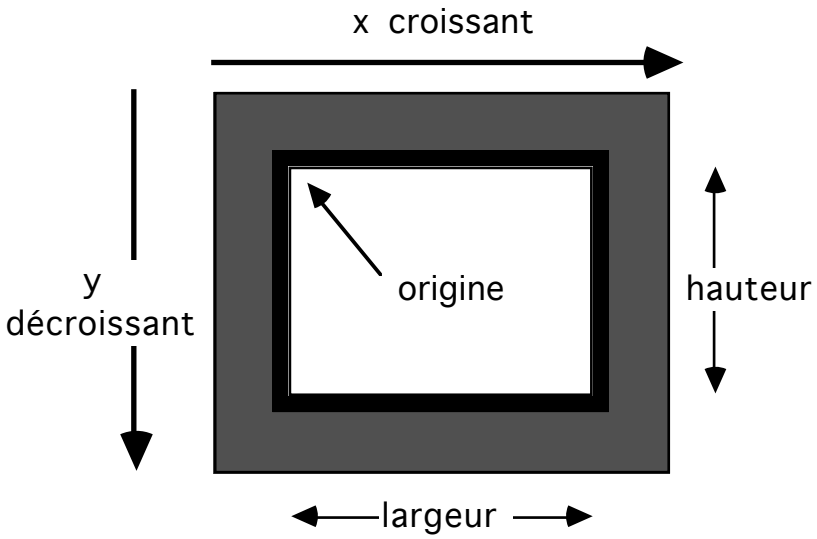

fig4. *Repérage géométrique des fenêtres*

et d'autres attributs plus spécifiques à leurs propriétés d'affichage: taille et couleur de bord, couleur du fond, gravité du dessin et des sous-fenêtres, table de couleurs, curseur, etc.

---

5  Attention, en présence d'un window manager, les attributs géométriques demandés à la création peuvent ne pas être pris en compte. Par exemple les attributs de position sont fréquemment ignorés pour laisser l'utilisateur positionner interactivement la fenêtre à l'écran.

Mais il y a en réalité deux classes de fenêtres, les fenêtres de classe **InputOutput** et les fenêtres de classe **InputOnly**. Les fenêtres de classe InputOnly sont des fenêtres dans lesquelles on ne peut pas dessiner. Le terme d'Input désigne ici les "entrées" de type clavier ou souris alors que le terme d'Output désigne les "sorties" textes ou graphiques, i.e. les dessins pouvant apparaître dans le cadre de la fenêtre. Une fenêtre InputOnly est une fenêtre qui n'admet aucun dessin c'est pourquoi nous utiliserons parfois le terme de fenêtre transparente.


En particulier, on veillera à ne pas spécifier d'attributs graphiques pour les fenêtres InputOnly, car cela génère des erreurs d'exécution. *Ces fenêtres n'ont ni couleur de bord, ni fond, ni épaisseur de bord*. On peut cependant leur associer un curseur particulier (curseur signifie ici écho visuel de l'emplacement virtuel de la souris à l'écran) ce qui les rendra apparentes à l'utilisateur.

L'intérêt des fenêtres transparentes est qu'elles permettent de délimiter des zones sensibles à moindre coût, car elles occupent moins de place en mémoire.

Création des fenêtres

La création des fenêtres peut s'effectuer avec l'une ou l'autre des deux fonctions suivantes:

```
XCreateSimpleWindow(dpy, parent,x, y, width, height,
                    border_width,   border_color,
                    background_color)
```

ou

```
XCreateWindow (dpy, parent, x, y,width, height,
                    border_width,
                    depth, class, visual,
                    valuemask, xswinattributes)
```

XCreateSimpleWindow permet de créer rapidement des fenêtres de classe InputOutput ayant une couleur de fond et une couleur de bord. On peut ensuite spécifier les événements que l'on souhaite recevoir dans cette fenêtre par un appel à **XSelectInput**(dpy, window, event_mask).
Par exemple:

```
win = XCreateSimpleWindow(dpy, root, x, y, width,
                          height, 3, whitepixel,
                          blackpixel);
```

```
XSelectInput(dpy, win, ButtonPressMask |
                            ButtonReleaseMask);
```

XCreateWindow permet de créer des fenêtres de n'importe quelle classe - à la restriction près qu'une fenêtre InputOutput ne peut être fille que d'une fenêtre InputOutput. **XCreateWindow** requiert l'initialisation préalable d'une structure de type **XSetWindowAttributes** contenant les attributs de la fenêtre à créer. On passe alors l'adresse de la structure en argument ainsi qu'un masque précisant les champs à prendre en considération dans la structure. Ainsi par exemple:

```
XSetWindowAttributes xswa;
Window win;

xswa.event_mask= ButtonPressMask | ButtonReleaseMask;
xswa.background_pixel = BlackPixel(dpy,screen);
win = XCreateWindow (dpy, root,10,100,
                          width,height,2,
          CopyFromParent, InputOutput, CopyFromParent,
          CWEventMask |  CWBackPixel, &xswa);
```

Il existe autant de masques prédéfinis que de champs pour cette structure, tous les masques étant de la forme **CW**AttributName, où AttributName est formé avec des majuscules pour séparer les mots figurant en minuscules dans la structure XSetWindowAttributes. Ainsi par exemple le masque **CWOverrideRedirect** correspond à l'attribut **override_redirect** et le masque **CWEventMask** à l'attribut **event_mask**. Il y a quelques exceptions cependant: **CWBackPimap** et **CWBackpixel** correspondent aux attributs background_pixmap et background_pixel et **CWDontPropagate** à l'attribut do_not_propagate_mask.

On peut donc initialiser l'attribut *event_mask* d'une fenêtre à la création en utilisant la fonction XCreateWindow, ce qui évite un appel ultérieur à XSelectInput. Cependant, cette procédure est plus lourde d'emploi que son homologue XCreateSimpleWindow car elle nécessite l'utilisation (déclaration et initialisation) d'une structure d'affectation d'attributs.


Dans la pratique, XCreateWindow est souvent source d'erreurs, car il faut faire attention à bien faire coïncider les champs initialisés de la structure d'attributs et le masque passé en argument. Il est recommandé de l'utiliser cependant dès que l'on veut instancier plusieurs attributs car elle économise les

échanges avec le serveur. En outre c'est la seule procédure permettant de créer des fenêtres transparentes de classe InputOnly.

```
typedef struct {
    Pixmap background_pixmap; /* Pixmap de fond,
                              None, ou ParentRelative */
    unsigned long background_pixel; /* couleur de fond */
    Pixmap border_pixmap;     /* bord de la fenetre */
    unsigned long border_pixel; /* couleur du bord */
    int bit_gravity; /* où redessiner le contenu */
    int win_gravity; /* où redessiner la sous-fenêtre */
    int backing_store;/* NotUseful, WhenMapped, Always */
    unsigned long backing_planes;/* plans à sauvegarder */
    unsigned long backing_pixel; /* couleur à utiliser
                                    pour restaurer */
    Bool save_under; /* doit-on sauver dessous (pop-up) */
    long event_mask; /* ens. des événements à recevoir */
    long do_not_propagate_mask; /* événements à bloquer */
    Bool override_redirect;/* pour empêcher la redirection
                              au window manager */
    Colormap colormap; /* table de couleurs associée */
    Cursor cursor;/* curseur courant spécifié (ou None) */
} XSetWindowAttributes;
```

fig5. La structure de données XSetWindowAttributes pour spécifier les attributs d'une fenêtre

```
#define CWBackPixmap          (1L<<0)
#define CWBackPixel           (1L<<1)
#define CWBorderPixmap        (1L<<2)
#define CWBorderPixel         (1L<<3)
#define CWBitGravity          (1L<<4)
#define CWWinGravity          (1L<<5)
#define CWBackingStore        (1L<<6)
#define CWBackingPlanes       (1L<<7)
#define CWBackingPixel        (1L<<8)
#define CWOverrideRedirect    (1L<<9)
#define CWSaveUnder           (1L<<10)
#define CWEventMask           (1L<<11)
#define CWDontPropagate       (1L<<12)
#define CWColormap            (1L<<13)
#define CWCursor              (1L<<14)
```

fig6. Masques d'attributs des fenêtres

```
#define CWX             (1<<0)
#define CWY             (1<<1)
#define CWWidth         (1<<2)
#define CWHeight        (1<<3)
#define CWBorderWidth   (1<<4)
#define CWSibling       (1<<5)
#define CWStackMode     (1<<6)
```

fig7. Masques d'attributs pour la configuration des fenêtres

Attributs des fenêtres

Les attributs figurant dans les structures de type **XSetWindowAttributes** (utilisée par XCreateWindow) sont modifiables par la fonction **XChangeWindowAttributes**. Cette fonction prend elle aussi en argument l'adresse d'une structure XSetWindowAttributes et un masque d'attributs permettant de l'interpréter. Les attributs affectables via cette structure sont les suivants:

- ***background_pixmap***: permet d'indiquer un motif (Pixmap) pour paver le fond de la fenêtre ou indique celui du parent.
- ***background_pixel***: indique la couleur du fond; c'est une alternative à un motif pour paver le fond.
- ***border_pixmap***: contient un éventuel motif pour le bord de la fenêtre.
- ***border_pixel***: indique une couleur de bord; c'est une alternative à un pavage de motif du bord.
- ***bit_gravity***: indique où le serveur devra repeindre le contenu de la fenêtre dans le cas d'un changement de taille de la fenêtre.
- ***win_gravity***: indique où replacer la sous-fenêtre dans le cas d'un changement de taille de la fenêtre parent.
- ***backing_store***: indique s'il y a lieu de sauvegarder le contenu de la fenêtre. (Attention: cette facilité n'est pas toujours implantée).
- ***backing_planes***: indique quels plans de couleurs doivent être sauvegardés si la capacité du serveur l'autorise. (Pas nécessairement implanté).
- ***backing_pixel***: indique la couleur à restaurer sur les plans sauvegardés. (Pas nécessairement implanté).
- ***save_under***: indique s'il y a lieu de sauvegarder le Pixmap situé sous la fenêtre[6]. (Pas nécessairement implanté).

[6] Cas d'un pop-up menu.

event_mask: ensemble des événements sélectionnés par l'application, i.e. les événements concernant la fenêtre et devant être envoyés par le serveur à l'application (client) lorsqu'ils se produisent.
do_not_propagate_mask: ensemble des événements que l'on ne souhaite pas voir propagés aux ancêtres de cette fenêtre.
override_redirect: valeur booléenne indiquant que les requêtes de modifications géométriques concernant cette fenêtre ne doivent pas être redirigées vers le window manager. Par défaut, cette valeur est à False pour les fenêtres filles de la racine et à True pour les autres.
colormap: table de couleurs associée à la fenêtre (valeurs d'intensité RGB associées à chaque couleur).
cursor: curseur (écho de la position de la souris à l'écran) dans cette fenêtre. C'est le serveur qui sera chargé de son affichage.

Les autres attributs des fenêtres sont les attributs géométriques. On les obtiendra par la fonction **XGetGeometry**(dpy, d, &root, &x, &y, &width, &height, &border_width, &depth). On peut également récupérer toutes les valeurs des attributs d'une fenêtre grâce à la fonction **XGetWindowAttributes**(dpy,win, &win_att) qui utilise une structure d'attributs plus grande que XSetWindowAttributes et dont le type est **XWindowAttributes** (cf. figure). Cependant, on prendra garde :

1. A ne pas confondre les deux types de structures XSetWindowAttributes (pour affectation) et XWindowAttributes (pour récupération).
2. A ne pas utiliser XGetWindowAttributes et XGetGeometry sans nécessité car ces procédures sont coûteuses. De manière générale, on essayera de récupérer les informations concernant la géométrie des fenêtres au travers des événements.

Les attributs géométriques sont normalement initialisés à la création des fenêtres mais sont en général modifiés par les requêtes interceptées par le window manager. Ainsi **XConfigureWindow** et les diverses fonctions:

XMoveWindow (dpy,win,x,y)
XResizeWindow (dpy,win,width,height)
XMoveResizeWindow (dpy,win,x,y,width, height)

risquent de surprendre le débutant qui les utiliserait sur des fenêtres filles de la racine. En effet, la plupart des window manager reparentent les fenêtres par une fenêtre englobante contenant des décorations diverses. Les coordonnées indiquées par ces dernières fonctions étant relatives à la fenêtre parent, seront donc interprétées relativement à cette nouvelle fenêtre englobante, et non pas relativement à la racine, comme le croit a priori le programmeur.

```
typedef struct {
     int x, y;           /* position relative à la mère */
     int width, height;  /* largeur, hauteur (en pixels) */
     int border_width;   /* taille du bord (en sus) */
     int depth;          /* profondeur des Pixmaps
                            (traitement couleur) */
     Visual *visual;     /* structure visuelle associée
                         (dépend du type de la machine) */
     Window root;        /* fenêtre racine */
     int class;          /* InputOutput ou InputOnly*/
     int bit_gravity;    /* une valeur de bit_gravity */
     int win_gravity;    /* une valeur de window gravity */
     int backing_store;/* NotUseful, WhenMapped, Always */
     unsigned long backing_planes; /* plans à préserver */
     unsigned long backing_pixel;  /* couleur à utiliser
                                      pour restaurer */
     Bool save_under; /* pour sauvegarder (pop-up menu)
                         ce qui est dessous*/
     Colormap colormap;  /* table de couleurs associée */
     Bool map_installed; /*  si  une  table  de  couleur  est
                            installée */
     int map_state;/*IsUnmapped IsUnviewable IsViewable */
     long all_event_masks;/* ens. des événements
               sélectionnés par toutes les applications */
     long  your_event_mask;/*  événements  sélectionnés  par
                         l'application */
     long do_not_propagate_mask;/* ens. des événements
                                   à ne pas propager */
     Bool override_redirect;  /* pour empêcher la
          redirection des requêtes sur les fenêtres
          au window manager */
     Screen *screen;     /* l'écran contenant la fenêtre */
} XWindowAttributes;
```

fig8. La structure XWindowAttributes,
pour récupérer les attributs d'une fenêtre

Signalons encore:

XRaiseWindow (dpy,win)
XLowerWindow (dpy, win)
XTranslateCoordinates (dpy,swin,dwin,sx,sy,&dx,&dy,child)

XQueryTree(dpy, win, &root, &parent, children, nchildren).

De même, de nombreuses procédures permettent de modifier directement un attribut donné (ce qui évite un appel à XChangeWindowAttributes). Toutes ces procédures commencent par XSetWindow. Ainsi par exemple:

XSetWindowBackground(dpy,win, pixel)
XSetWindowBackgroundPixmap(dpy,win, pixmap)
XSetWindowBorder(dpy,win, pixel)
XSetWindowBorderPixmap(dpy,win, pixmap), etc.

Fonds de fenêtres: la manipulation de "Pixmap"

Les points de l'écran peuvent être regroupés dans des structures (Pixmap) permettant de spécifier la couleur (pixel) de chaque point. Le terme **Pixmap** s'oppose à **Bitmap** - un bitmap représentant un simple rectangle de points à valeur dans {0,1} - alors qu'un Pixmap représente un rectangle de points à valeur dans $\{0,1\}^N$.

0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
1	1	1	0	1	1	1	0
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
1	1	0	1	1	1	0	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	0	1	1	1	0
1	1	0	1	1	1	0	1

fig9. *Un Bitmap*

En effet, chaque couleur (pixel) est codée par un entier - lui même codable sur N bits. Un pixel est donc identifiable à un N-uplet de $\{0,1\}^N$. Pour définir une zone rectangulaire de points colorés, on a donc besoin de N plans de bitmap. C'est pourquoi un Pixmap est constitué de plusieurs plans de bitmaps dont le nombre N - la profondeur du Pixmap (*depth*) - représente le nombre de bits

nécessaires pour coder une couleur (en général un multiple de 8, cf. le chapitre sur la couleur).

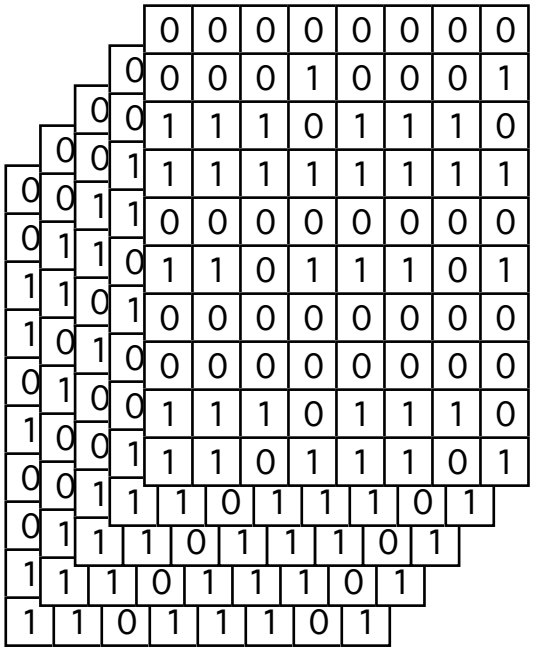

fig10. *Un Pixmap*

On peut associer un **Pixmap** au fond d'une fenêtre ou au bord d'une fenêtre via les attributs background_pixmap et border_pixmap. C'est une alternative à la spécification d'une couleur uniforme (attributs background_pixel et border_pixel modifiés par XCreateSimpleWindow) qui permet de réaliser le pavage d'un motif coloré comme fond de fenêtre ou comme bordure. On utilisera également des Pixmaps pour préciesr des contextes graphiques (utilisés par les primitives de remplissage comme XFillRectangle ou XFillArc). On se servira aussi de Pixmaps pour mémoriser une portion d'écran.

Il existe un logiciel standard (appelé *bitmap*) qui permet de créer interactivement des données au "format bitmap" - lesquelles données pourront être transformées en Pixmaps grâce aux fonctions **XCreatePixmapFromBitmapData** ou **XReadBitmapFile**. On peut en effet convenir que tous les points caractérisés par 1 dans un rectangle bitmap donné se trouvent en réalité affectés avec une couleur donnée (foreground_color par exemple) alors que ceux marqués d'un 0 se trouvent affectés d'une autre couleur (background_color). Un Bitmap rectangulaire permet ainsi de définir

un Pixmap[7]. Si l'on a créé un fichier "file.bm" à l'aide de l'utilitaire *bitmap* on pourra l'inclure dans son programme par #include, puis utiliser les fonctions **XCreatePixmapFromBitmapData**(dpy, d, data, width, height, fg, bg, depth) ou **XCreatePixmapFromData**(dpy, d, data, width, height). L'alternative sera de lire directement le bitmap dans le fichier "file.bm" en passant ce nom de fichier en argument à la fonction **XReadBitmapFile**(dpy, d, filename, &widh, &height, &pixmap, &xhot, &yhot)[8].


Remarques:

1. Ces fonctions de création requièrent un argument de type Drawable (i.e. une fenêtre de classe InputOutput ou un Pixmap) dans le but de fixer certains paramètres nécessaires pour la création du Pixmap. On leur passera en général la fenêtre racine.

2. Pour pouvoir utiliser les constantes de retour de ces fonctions spécifiées par la documentation MIT, on incluera le fichier <X11/Xutil.h >.

3. Un problème pour les écrans couleur avec la fonction `XReadBitmapFile` est que cette fonction créé une structure Pixmap de profondeur 1 (un seul plan), c'est-à-dire un bitmap, lequel n'est pas utilisable directement comme fond de fenêtre pour un écran couleur, car sa profondeur n'est pas celle de l'écran. Pour palier à ce problème, on créera au préalable un Pixmap de la bonne profondeur (`DefaultDepth`(…)) avec la fonction `XCreatePixmap`, puis on effectuera la copie (selon un plan avec `XCopyPlane`) du bitmap récupéré avec XReadBeatMapFile dans ce nouveau Pixmap.

Signalons également pour la manipulation de Pixmap les fonctions
XCreatePixmap(dpy, d, width, height, depth)
XFreePixmap(dpy, pixmap) et
XWriteBitmapFile(dpy, filename, pixmap, widh, height,
xhot,yhot).

---

[7] Dans les fonctions de création de curseurs à partir de Bitmap on utilise parallèlement un deuxième Bitmap appelé masque pour délimiter une zone non rectangulaire. Les bits à 1 du masque servent alors à indiquer quels sont les points pertinents, i.e. à prendre en considération, dans un autre Bitmap.

[8] Les coordonnées xhot et yhot sont les coordonnées d'un point de référence associables à des données en format bitmap. Ces coordonnées sont définissables interactivement avec le logiciel bitmap et sont utilisées pour définir les curseurs.

Affichage des fenêtres


Les fenêtres créées ne sont pas pour autant présentes à l'écran. Pour les afficher, on fait appel à la procédure

XMapWindow (dpy, win)

Une fenêtre ainsi affichée peut en outre ne pas être visible si elle se trouve masquée par une autre.

L'appel à XMapWindow aura généralement lieu en début de programme, *après* création et spécification des événements à récupérer sur la fenêtre. Mais on peut également demander l'affichage d'une fenêtre quand un événement particulier se produit dans une autre fenêtre (cas d'un menu déroulant par exemple).


Pour éviter des problèmes de retard dans l'affichage, on prendra soin de demander l'afficher des sous-fenêtres *avant* celui de leur mère. On obtient alors au final l'affichage simultané des deux. Pour cela on utilisera la fonction **XMapSubwindows** (dpy, parent_win) avant l'appel à XMapWindow. L'affichage des sous-fenêtres ne peut de toute façon se produire que si la fenêtre mère est affichée (car cette dernière sert de cadre d'affichage).

Inversement, on peut supprimer momentanément de l'affichage une fenêtre visible à l'écran en faisant un appel à **XUnmapWindow**[9](dpy,win). Il existe également une fonction de destruction de fenêtre, **XDestroyWindow** (dpy,win), qui permet de libérer entièrement les ressources du serveur allouées pour cette fenêtre. Il faudra cependant être sûr qu'aucun événement concernant cette fenêtre n'est encore accessible car sinon, un accès au champ window de cet événement produira une erreur.

Signalons également:

XMapRaised (dpy,win)
XUnmapSubwindows (dpy, parent_win)
XDestroySubwindows (dpy, parent_win)

[9] C'est ce qui se produit lorsque l'on "incônifie" une fenêtre. Pour chaque fenêtre, le window manager créé une fenêtre duale plus petite servant à représenter la fenêtre sous forme iconique. L'opération consiste alors à faire un appel à UnmapWindow sur la fenêtre d'origine suivi d'un appel à XMapWindow sur son icône.

Utilisation de contextes d'association

Les tables d'association ou contextes d'association (type **XContext**) fournissent un moyen très efficace pour associer des données à une fenêtre. Un gestionnaire de contexte (context manager) a en effet été défini de manière interne pour l'implémentation de la librairie, dans le but d'associer efficacement des données aux fenêtres. Les fonctions ainsi développées font partie de la librairie et sont accessibles au programmeur. Ces fonctions sont les suivantes:

XContext **XUniqueContext** ()
XSaveContext (dpy, win, context, pdata)
XFindContext (dpy, win, context, &pdata)
XDeleteContext (dpy, win, context)

Elles permettent d'associer à une fenêtre une structure de données de l'utilisateur relativement à un certain contexte. Ce type d'association est très utile car, comme nous allons le voir dans le chapitre suivant, le contrôle du flot d'un programme X est entièrement dirigé par l'envoi des événements. Ce mode de contrôle est donc a priori antinomique de celui de la programmation par objets. Le système X-Window présente cependant les fenêtres comme les objets de base du système et on aurait souhaité pouvoir programmer dans le style de la programmation par objets.

Grâce aux contextes d'association, on va pouvoir récupérer un style de programmation par objets. En effet, un événement contient toujours l'indication de la fenêtre dans laquelle l'événement s'est produit. On peut donc récupérer cette fenêtre à partir d'un événement, et, à partir d'elle, retrouver n'importe quel type de données qu'on lui aura associé grâce aux contextes d'association.

On va ainsi pouvoir considérer les fenêtres comme des objets comportant un certain nombre d'"attributs" ou "méthodes" (puisque le langage C permet de stocker des adresses de procédures) - attributs modifiables, et fonctions susceptibles d'être déclenchées quand certains types d'événements se produisent. Les contextes d'association permettent ainsi un traitement plus générique des fenêtres et proposent une architecture générale aux programmes d'interfaces. On peut en effet définir différent types de fenêtres relativement à des comportements attendus en réponse à la capture d'un événement. Cette architecture permet une plus grande modularité et évite le stockage d'informations redondantes (cf. l'exemple du pop-up menu vu en T.P.).