COMMUNICATION AVEC L ‘ APPLICATION

Handlers d'événements X

```
XtAddEventHandler (widget, eventmask, nonmaskable,
                                    handler, client_data)
    Widget              widget;
    unsigned long       eventmask;
    Boolean             nonmaskable;
    void                (*handler) ( ) ;
    XtPointer           client_data;
```

- Enregistre la fonction *handler* comme traitant les événements X indiqués dans eventmask pour le widget.

- Le handler est prototypé de la façon suivante:

```
void handler (w, client_data, event)
    Widget      w;
    XtPointer   client_data;
    XEvent*     event;
```

Handlers d'événements X: Exemple

```
#include <Xm/Xm.h>
#include <Xm/Label.h>

static void quit (Widget w, XtPointer client_data, XEvent *event)
{
        exit(0);
}

main (argc, argv) char **argv;
{
        Widget              appshell, w;
        XtAppContext        app;

        appshell = XtVaAppInitialize (&app, "Test", NULL, 0, &argc, argv, NULL, NULL);
        w = XtVaCreateManagedWidget ("Cliquez", xmLabelWidgetClass, appshell, NULL);
        XtAddEventHandler (w, ButtonPressMask, False, quit, NULL);
        XtRealizeWidget (appshell);
        XtAppMainLoop (app);
}
```

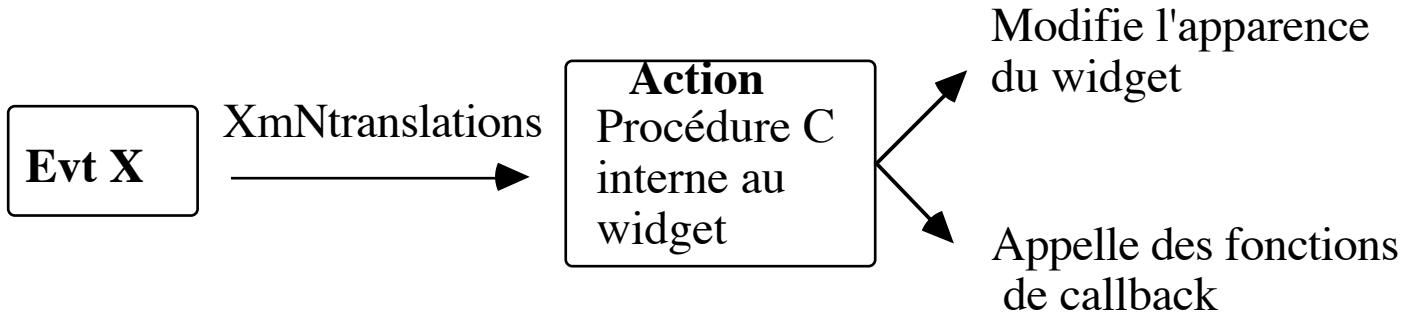
Masques de sélection d’événements (cf. doc Xlib)

Les principaux masques de sélection sont les suivants (mais il y en a beaucoup d’autres) :

ButtonPressMask	sélectionne l'enfoncement d’un bouton de souris
ButtonReleaseMask	sélectionne le relâchement d’un bouton de souris
PointerMotionMask	sélectionne les mouvements de souris
ButtonMotionMask	sélectionne les mouvements souris bouton enfoncé
Button<i>MotionMask	sélectionne les mouvements de souris bouton<i> enfoncé
KeyPressMask	sélectionne l'enfoncement d'une touche du clavier
KeyReleaseMask	sélectionne le relâchement d'une touche
ExposureMask	sélectionne les besoins de réaffichage

Traitement des événements d'entrée

Les événements X (clics, déplacements de souris, frappe au clavier) sont codifiés sous forme d'actions à l'aide d'une table de translation; une procédure C interne à la classe du widget implémente une action. Cette procédure appelle les fonctions de la liste de callbacks associée à l'action et modifie éventuellement l'apparence du widget.



Exemple de table de translation:

```
<Btn1Down>:        Arm () \n\
<Btn1Up>:          Disarm () Activate ()
```

Les listes de callbacks

XtAddCallback (Widget widget, String callback_name,
XtCallbackProc procedure, XtPointer client_data)

- Rajoute la fonction *procedure* à la liste de callbacks *callback_name* pour le widget.

Les fonctions d'une liste de callbacks sont appelées quand l'utilisateur exécute des actions spécifiques.

- Par exemple le PushButton possède (entre autres) la liste de callbacks XmNactivateCallback déclenchée normalement quand l'utilisateur clique puis relâche la souris à l'intérieur du bouton.

- Toute procédure de callback a un protopype du genre :

void procedure (Widget widget, XtPointer client_data, XtPointer call_data)

L'argument *call_data* est un pointeur sur une structure définie dans la documentation sur le widget (rubrique callbacks). Dans le cas d'un PushButton, c'est un pointeur sur une XmPushButtonCallbackStruct. Pour éviter les warnings du compilateur, on devra prototyper correctement les procédures oubien faire les casts appropriés.

Les listes de callback: Exemple

```
#include <Xm/Xm.h>
#include <Xm/PushB.h>

static void quit (Widget w, XtPointer client_data, XmPushButtonCallbackStruct * call_data)
{
      printf( "%s\n",  (char *) client_data );
      exit(0);
}

main (argc, argv) char ** argv;
{     Widget                top, w;
      XtAppContext      app;

      top = XtVaAppInitialize (&app, "Test", NULL, 0,&argc, argv, NULL, NULL);
      w = XtCreateManagedWidget ("Clickez-moi", xmPushButtonWidgetClass, top,NULL,0);
      XtAddCallback (w, XmNactivateCallback, quit, (XtPointer) "Adieu");
      XtRealizeWidget (top);
      XtAppMainLoop (app);
}
```

Retirer des fonctions de callback

```
XtRemoveCallback (widget, callback_name, procedure, client_data)
        Widget          widget;
        String          callback_name;
        XtCallbackProc  procedure;
        XtPointer       client_data;
```

- Retire la fonction procedure de la liste de callback *callback_name* pour le widget. La fonction n'est retirée de la liste de callback que si le client_data est identique à celui fourni lors de l'enregistrement avec XtAddCallback.

Table de Translation

- La table de translation indique par quelle action les événements X doivent être traités dans un widget.

- Chaque classe de widget possède une table de translation.

- Chaque instance peut aussi avoir sa propre table de translation. Initialement une instance utilise la table de sa classe.

Une table de translation est implémentée à l'aide d'une chaîne de caractères ASCII associant une liste d'actions à une suite d'événements X.

Exemple:

```
char translation [ ] =
    "<Btn1Down>:    Arm () \n\
    <Btn1Up>:       Disarm () Activate ()";
```

Table de Translation (suite)

XtTranslations XtParseTranslationTable (trans_string)
String trans_string;

- Compile une table de translation.

XtOverrideTranslations (w, translation);
Widget w;
XtTranslations translation;

- Rajoute des translations à la table de translation du widget en privilégiant les nouvelles translations.

XtAugmentTranslations (w, translation)
Widget w;
XtTranslations translation;

- Rajoute des translations à la table de translation du widget en privilégiant les anciennes translations.

Table de translation: dans un fichier de ressources

Dans un fichier de ressources:

```
test*XmPushButton.translations:        #override\n\
                                       <Key>A:              Arm() \n\
                                       <Btn1Up>:            Activate()

test*XmPushButton.translations:        #augment\n\
                                       Ctrl<Key>A:                Arm() \n\
                                       <Btn1Up>:                  Activate()

test*XmPushButton.translations:        #replace\n\
                              <Btn2Down>A:            Arm() \n\
                              <Btn2Up>:               Disarm() Activate()
```

Table de translation: dans un argument fallbacks

```
String fallbacks [ ] = {   "Test*XmPushButton.translations:            #override\\n\
        <Key>A:            Arm() \\n\
        <Btn1Up>:          Activate()",
                           "Test*XmPushButton.background:      red",
                           "Test*label.labelString: une ligne multiple \\n\
alignee a droite \\n\
dans un label",
                           "*label.alignment:  ALIGNMENT_END",
                           "*.fontList:     10x20",   NULL };


    Widget              appshell;
    XtAppContext        app;

    appshell = XtVaAppInitialize (&app, "Test", NULL, 0, &argc, argv, fallbacks, NULL);
```

Autres sources d'entrées

```
XtInputId       XtAppAddInput (appContext, fd, condition, procedure, client_data)
        XtAppContext            appContext;
        int                     fd;
        XtPointer               condition;
        XtInputCallbackProc     proc;
        XtPointer               client_data;
```

- Enregistre la procédure *procedure* qui sera appelée quand la condition sera effective pour le file descriptor *fd*.

- La condition peut être :

XtInputReadMask	lecture sur fd
XtInputWriteMask	écriture sur fd

Autres sources d'entrées (suite)

void InputProc (client_data, fdPtr, idPtr)

```
XtPointer      client_data;
int *          fdPtr;
XtInputId *    idPtr;
```

- La procédure est appelée avec une copie de *client_data* , un pointeur vers le file descriptor de la source, et un pointeur vers l'identificateur retourné par le *XtAppAddInput* correspondant (i.e. correspondant à son enregistrement comme procédure à déclencher).

void XtRemoveInput (id)

```
XtInputId;
```

- Retire la procédure associée à l'identificateur *id*.

Autres sources d'entrées: Exemple

```
void getInput (XtPointer client_data, int * fdPtr, XtInputId *idPtr) {
        char    buffer[BUFSIZ];
        int     nbytes;

        if ((nbytes = read (*fdPtr, buffer, BUFSIZ - 1)) == 0) {
            XtRemoveInput (*idPtr);
            return;
        }
        buffer [nbytes]= 0;
        printf ("input:%s", buffer);
}

main (argc, argv) char** argv; {
    ...
    XtAppAddInput(app, fileno (stdin), XtInputReadMask,
                                getInput, NULL);
    XtAppMainLoop (app);
}
```

Les timers

```
XtIntervalId    XtAppAddTimeOut (appContext, interval, proc, client_data)
        XtAppContext            appContext;
        unsigned long           interval;
        XtTimerCallbackProc     proc;
        XtPointer               client_data;
```

- Enregistre la procédure *proc* qui sera appelée quand *interval* millisecondes se seront écoulées.

- La procédure *proc* a le prototype suivant:
  void proc (XtPointer client_data, XtIntervalId *idPtr)

- Si on souhaite une exécution répétée, la procédure doit « réarmer » le timer

Les timers (exemple)

```
void  timerProc (XtPointer pushb, XtIntervalId *idPtr)
{
      static int  seconds = 1;   /* déclaré static pour mémoriser la valeur de l'appel précédent */
      Pixmap          pixmap;

      seconds++;
      if (seconds % 2)     pixmap = LightOnPixmap;
      else                 pixmap = LightOffPixmap;
      XtVaSetValues ( (Widget) pushb, XmNlabelPixmap, pixmap, NULL);
      if (seconds < 60)    XtAppAddTimeOut (app, 1000, timerProc, pushb);
      else                 seconds = 1;
}

timerActivateCallback (Widget pb, XtPointer clientd, XmPushButtonCallbackStruct * calld)
{     /* callback à ajouter à la activate callbacks list d'un pushButton */
      XtAppAddTimeOut (app, 1000, timerProc, pb);
}
```

SHELL & POPUPS

Plusieurs fenêtres principales

```
Widget  XtVaAppCreateShell (name, dummy, shell_class,
                                    dpy, ..., NULL)

        String          name;
        WidgetClass     shell_class;
        Display*        dpy;
```

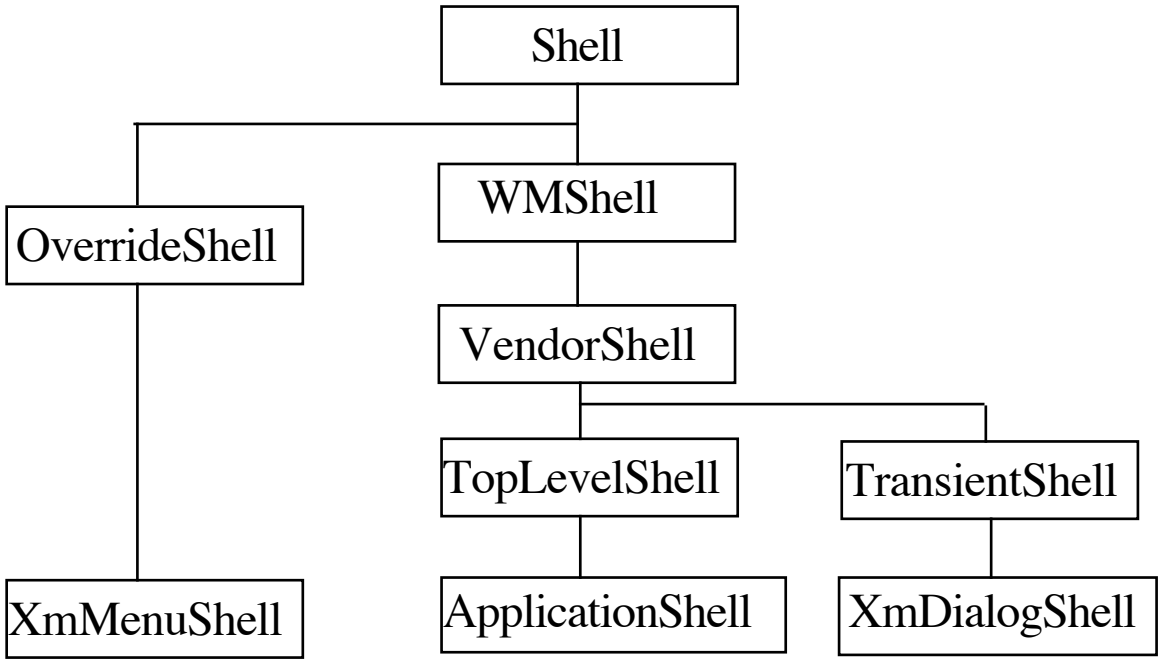
- Crée une instance d'un shell widget de type *shell_class*.

Par exemple:

applicationShellWidgetClass
topLevelShellWidgetClass

- *name* est le nom du widget pour les ressources.

Hiérarchie des widgets Shell



Plusieurs fenêtres principales: Exemple

```
/* Crée une fenêtre toplevel avec un bouton dedans */
make_toplevel_window (dpy) Display* dpy; {
        Widget          toplevel;
        toplevel= XtVaAppCreateShell ("Test", NULL,
                                topLevelShellWidgetClass, dpy, NULL);
        (void) XtVaCreateManagedWidget("bouton",
                                xmPushButtonWidgetClass, toplevel, NULL);
        XtRealizeWidget (toplevel);
}

main (argc, argv) char** argv;
{
        Widget appshell= XtVaAppInitialize (&app, "Essai", NULL, 0,
                                                &argc, argv, NULL, NULL);
        make_toplevel_window (XtDisplay (appshell));
        make_toplevel_window (XtDisplay (appshell));
        XtAppMainLoop(app);
}
```

Les shells popup

Widget XtVaCreatePopupShell (name, class, parent, ..., NULL)

```
String      name;
WidgetClass class;
Widget      parent;
```

- Crée une instance d'un popup shell widget de type *class*.

Par exemple:

transientShellWidgetClass

- *name* est le nom du widget dans la base de ressources.

Les popups shells sont principalement utilisés pour afficher des boîtes de dialogues. A la différence des autres widgets, la fenêtre X associée au popup shell n'est pas une sous-fenêtre de celle du widget parent.

Les shells popup: Apparition

typedef enum {XtGrabNone, XtGrabNonExclusive, XtGrabExclusive } XtGrabKind;

XtPopup (Widget shell, XtGrabKind grab_mode)

Si une application fait apparaître un popup qui lui-même en fait apparaître un autre, alors l'ensemble s'appelle une cascade de popups. Cette situation se rencontre avec les sous-menus. Selon la valeur de grab_mode les événements d'entrées seront traités de différentes façons:

XtGrabNone:
Les événements sont envoyés au widget où se trouve la souris.

XtGrabNonexclusive:
Les événements ne sont envoyés qu'aux widgets de la cascade de popups.

XtGrabExclusive:
Les événements ne sont envoyés qu'au dernier widget de la cascade de popups.

Les shells popup: Disparition

XtPopDown (shell)
    Widget   shell;

- Fait disparaître le popup shell.

- Les événements d’entrée sont traités à nouveau de façon normale.

Les shells popup: Exemple

```
static Widget        dialogShell;

static void done (Widget w, XtPointer dialog, XtPointer call_data)
{
      XtPopDown ( (Widget) dialog);
}

void InitDialog (Widget w)  {
      Widget          ok;

      dialogShell = XtVaCreatePopupShell ("dialog", transientShellWidgetClass, w, NULL);
      ok = XtVaCreateManagedWidget ("Ok", xmPushButtonWidgetClass, dialogShell, NULL);
      XtAddCallback (ok, XmNactivateCallback, done, dialogShell);
}

PopDialogBox ()
{
      XtPopup (dialogShell, XtGrabNonExclusive);
}
```

Les boîtes de dialogue MOTIF

- La toolkit MOTIF propose des fonctions spécifiques pour la création de boîtes de dialogue. Ces fonctions créent le popup shell ainsi que les sous-widgets nécessaires.

Exemple:

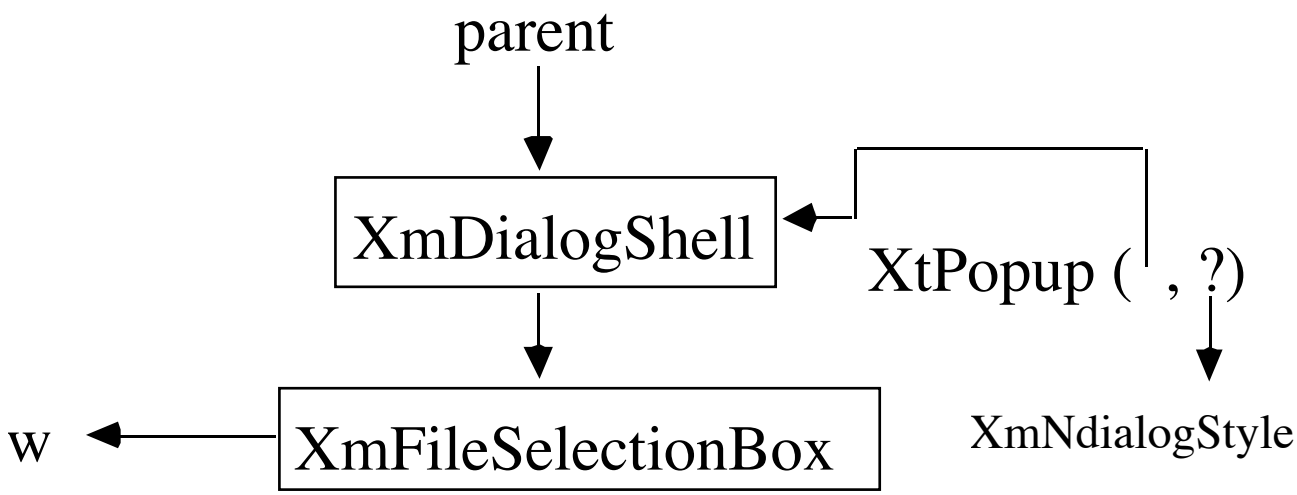
```
Widget XmCreateFileSelectionDialog (parent, name, arglist, argcnt)
                                    Widget      parent;
                                    String      name;
                                    Arg*        arglist;
                                    int         argcnt;
```

- Crée un FileSelectionBox dans un DialogShell ayant ce parent.

- Pour faire apparaître la boîte: XtManageChild (widget)

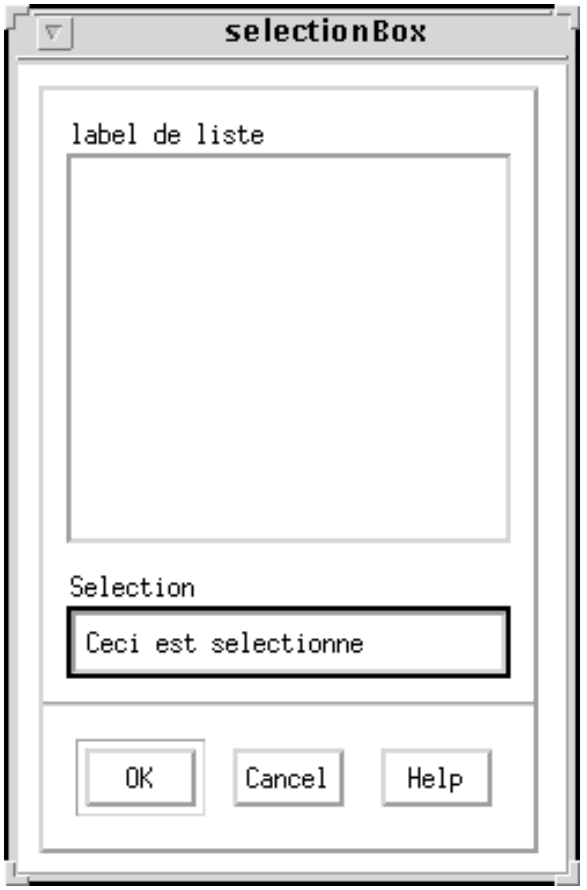
- Pour la faire disparaître: XtUnmanageChild (widget)

Les boîtes de dialogue MOTIF: principe

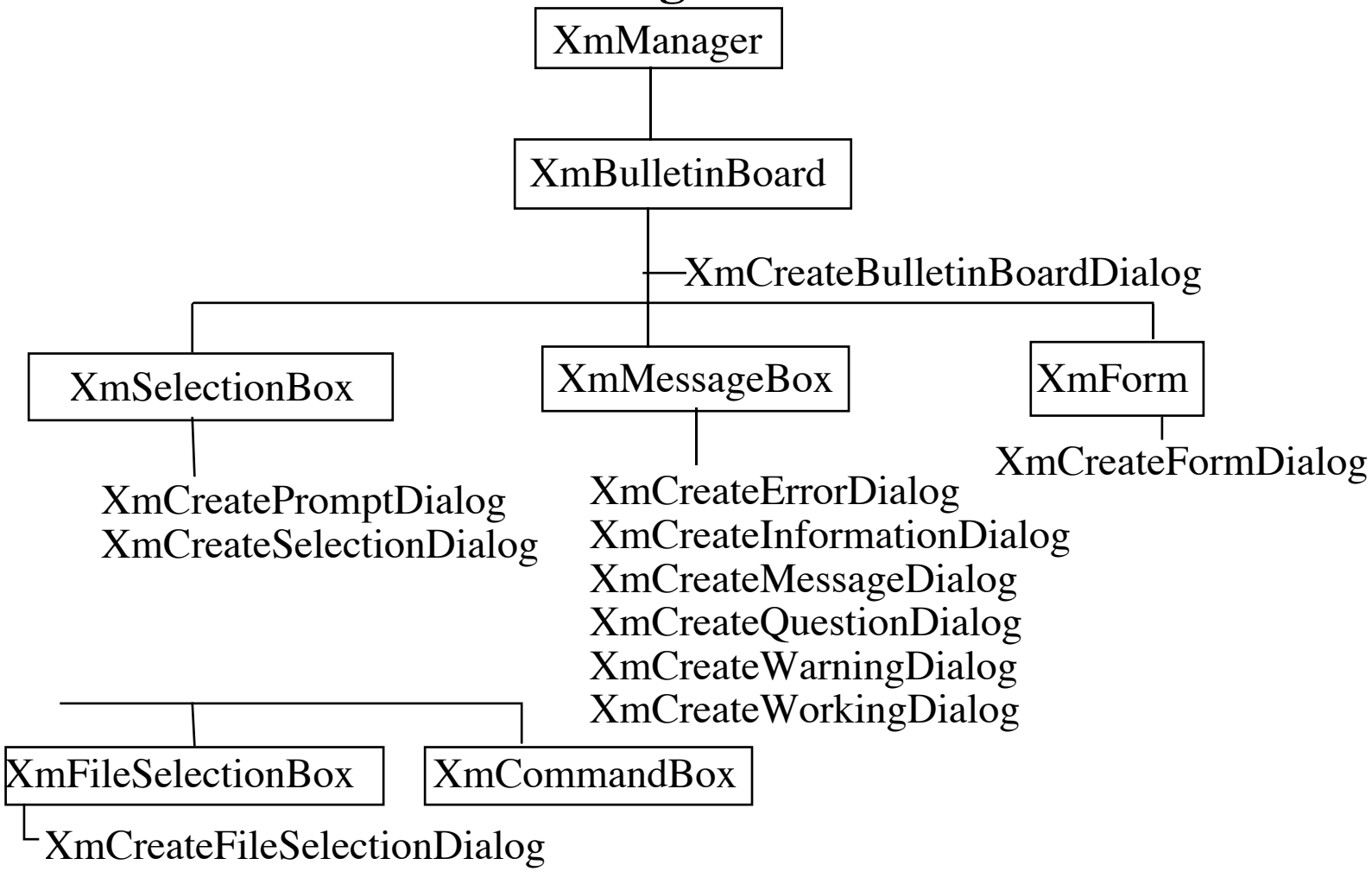


FileSelectionBox

MessageBox et SelectionBox



Hiérarchie des boîtes de dialogue MOTIF



Les boîtes de dialogue MOTIF: configuration

Les fonctions qui précèdent configurent les boîtes de dialogues pour un usage ou un autre.

Pour modifier les boîtes on pourra utiliser :

- Widget Xm<NomDeBoite>GetChild(boite, nom_de_widget)

  exemples: XmMessageBoxGetChild, XmFileSelectionBoxGetChild,
  XmSelectionBoxGetChild.

- Une fois un widget fils récupéré, on pourra le griser ou le faire disparaître

  avec XtSetSensitive(widget, False) ou XtUnmanageChild(widget),

  ou lui donner le focus initial dans la boîte, etc.

NAVIGATION CLAVIER

Navigation Clavier

- Le guide de style MOTIF spécifie que toute application doit pouvoir fonctionner sans utiliser la souris et précise comment un utilisateur peut interagir avec une application en utilisant uniquement le clavier

- Ces spécifications sont regroupées sous le nom de *Keyboard Traversal*. Ce modèle de navigation répartit l'arborescence des widgets en deux couches: la première contient des groupes de tabulation *(TAB groups)*, et la seconde les éléments de ces groupes

- Pour se déplacer d'un groupe de tabulation à l'autre, on utilise la touche TAB, et SHIFT-TAB pour un déplacement inverse

- Pour se déplacer à l'intérieur d'un même groupe de tabulation, on utilise les flèches

Navigation Clavier (suite)

- Par défaut, les widgets dont XmManager est une super classe, les XmList et les XmText sont initialisés comme groupe de tabulation

- Les widgets OutputOnly (XmLabel) sont exclus du mécanisme de navigation

- Les autres widgets (XmPushButton, XmToggleButton) sont initialisés comme membre d'un groupe de tabulation

Modification des groupes de tabulation

- La ressource XmNnavigationType permet de contrôler si un widget est un groupe de tabulation ou est un élément d'un tel groupe

- Cette ressource peut prendre les valeurs suivantes:
  XmTAB_GROUP : le widget est un groupe de tabulation
  XmNONE : le widget est élément d'un groupe de tabulation

- La ressource XmNtraversalOn de type Boolean, permet d'exclure (False) ou d'inclure (True) un widget du mécanisme de navigation

- Exemple:

  Test*XmPushButton*navigationType: TAB_GROUP
  Test*XmList*traversalOn: False

Focus du clavier

- MOTIF supporte deux différents modèles de gestion du focus du clavier:

XmPOINTER: le widget qui contient le pointeur (la souris) reçoit les événements clavier

XmEXPLICIT: l'utilisateur doit cliquer sur un widget pour indiquer que ce widget doit recevoir les événements clavier. On dit qu'il a le focus. Une fois que le focus est mis explicitement, le widget continue de recevoir les événements clavier même si le pointeur sort de ce widget

- Le modèle de gestion du focus clavier utilisé est contrôlé par la ressource XmNkeyboardFocusPolicy définie dans VendorShell. Cette ressource peut prendre l'une des deux valeurs suivantes:

XmPOINTER ou XmEXPLICIT

Focus du clavier (suite)

- Il est possible de rediriger le focus du clavier lors de la navigation en utilisant la fonction XmProcessTraversal (Widget widget, int direction)

L'argument direction indique où le focus d'entrée doit être déplacé par rapport au widget ; il peut prendre l'une des valeurs suivantes:

XmTRAVERSE_CURRENT
XmTRAVERSE_NEXT
XmTRAVERSE_PREV
XmTRAVERSE_HOME
XmTRAVERSE_UP
XmTRAVERSE_DOWN
XmTRAVERSE_LEFT
XmTRAVERSE_RIGHT
XmTRAVERSE_NEXT_TAB_GROUP
XmTRAVERSE_PREV_TAB_GROUP

- On peut aussi affecter la ressource XmNinitialFocus à un fils de XmManager