# TOOLKIT MOTIF

# Principes de base des toolkits

• Encapsulation

- Les widgets sont des objets opaques. On n'accède pas à leur code mais on connaît leur comportement car ils instancient une classe.

• Réutilisation facile du code de l'interface d'une application.

- Différentes toolkits utilisent la même bibliothèque intrinsics Xt.

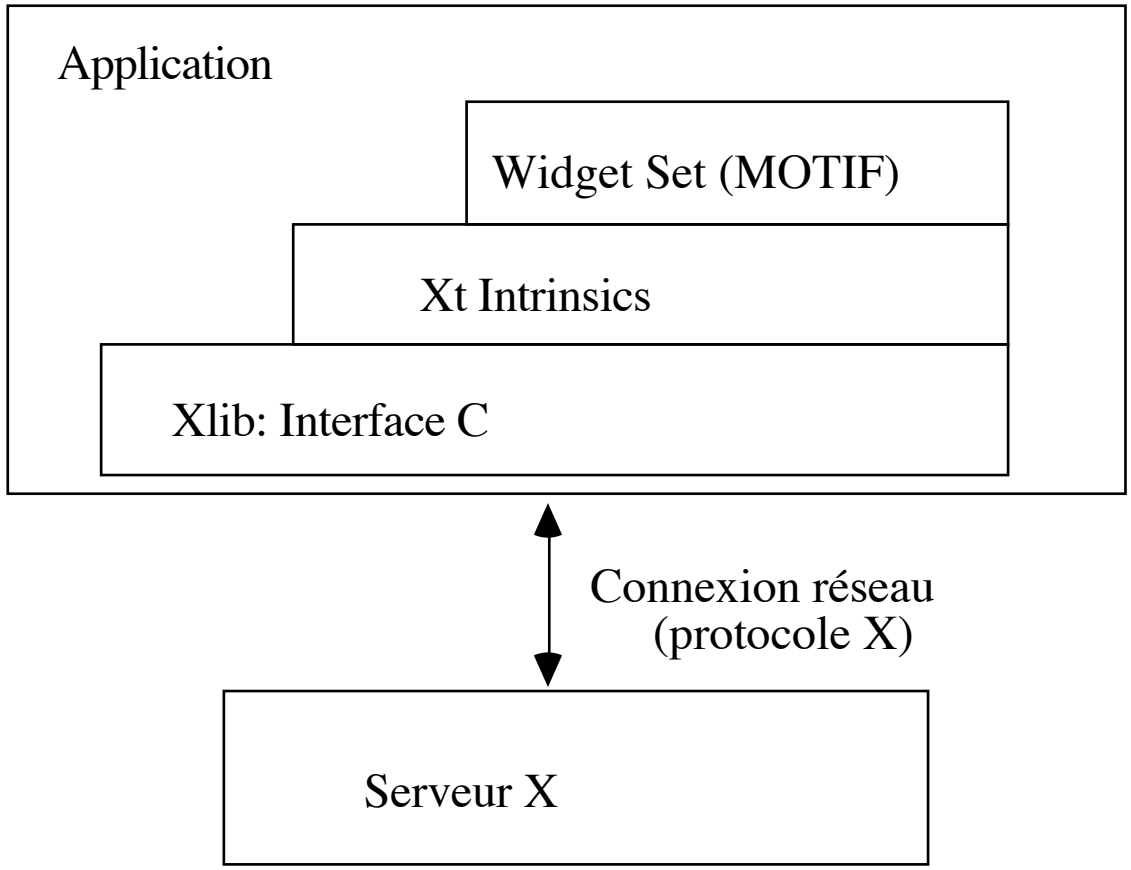
• Séparation nette entre une application et son interface utilisateur.

- Une même application peut avoir deux interfaces différentes.

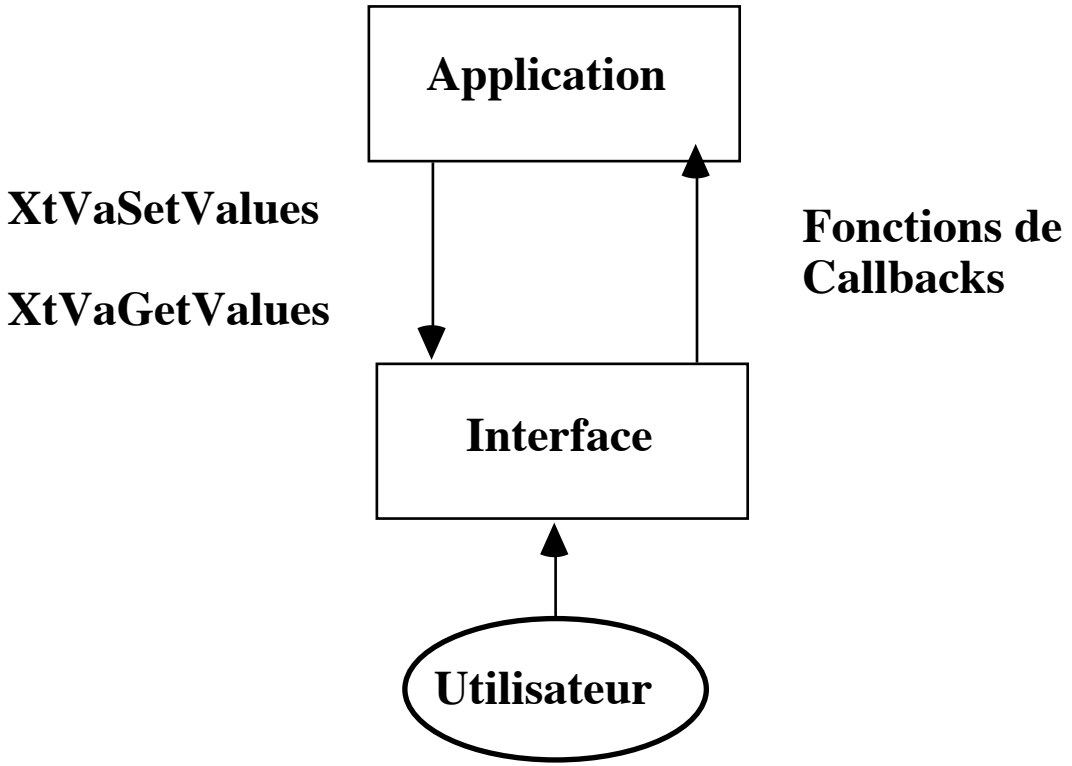
• L'interface peut être modelée au goût de chaque utilisateur.

- Des fichiers de ressources permettent de modifier les widgets, sans recompilation du code.

# Les différentes couches de bibliothèques



# Communication Application / Interface



# Xt et Widgets

Une toolkit utilise deux librairies :

- La bibliothèque INTRINSICS.
  Elle implémente le mécanisme d'objets graphiques.
  Elle définit des méta-classes d'objets et fournit les fonctions nécessaires pour l'implémentation de classes d'objets.
  Elle est commune aux différentes toolkits.

- Un ensemble de classes instanciables (les classes de widgets)
  C'est l'ensemble des objets graphiques proposés par un fournisseur de toolkit.
  - Les Athena widgets (MIT)
  - Les HP widgets (Hewlett Packard)
  - OLIT (OpenLook Intrinsic Toolkit - SUN)

# Toolkit MOTIF

La toolkit MOTIF (Open Software Foundation) a pour spécificité qu'elle

- Implémente les GADGETS (widgets sans fenêtre X associée).
- Implémente des chaînes composées qui permettent d'avoir des messages dans différentes fontes.
- Permet d'utiliser différentes unités pour indiquer la taille des fenêtres (pixels, millimètres, pouces, ...).
- Propose un window manager (mwm) et un style d'interaction.

# MOTIF

On utilise le terme MOTIF pour désigner plusieurs composantes de l'interface utilisateur standard MOTIF :

- L'ensemble des widgets MOTIF basé sur les XtIntrinsics. (La toolkit MOTIF définie par la librairie Xm)

- Le guide de style MOTIF qui contient les règles et recommandations pour avoir une cohérence entre les différentes applications.

- Le window manager MOTIF : mwm, (ou son *look* en environnement CDE).

- UIL (User Interface Language). Ce langage offre une alternative au langage C pour implanter les objets de la toolkit MOTIF.

# Widgets MOTIF

Ce sont les objets qui seront dans l'espace de l'interface graphique utilisateur de l'application. La toolkit MOTIF offre différentes catégories de widgets :

- Widgets d'affichage ou primitifs (widgets ou gadgets)
  → boutons, étiquettes, éditeurs de textes, ascenseurs, etc.

- Widgets conteneurs qui gèrent les positions d'autres widgets
  → cadres, formulaires, etc.

- Boîtes de dialogues

- Menus
  (pop-up, barre de menus, menus en cascade, menus à option)

# Widgets MOTIF

widgets = objets gérés dans l'application

fenêtre (window) = objets gérés dans le serveur

**Etats des widgets**

- créé – structure interne initialisée dans l'application
- réalisé – une fenêtre est créé dans le serveur
- géré – (managed) positionné sur l'écran

# Conventions Xlib

Les fonctions : XCreateWindow(…)
XDrawString(…)

Les macros : DisplayWidth(…)
ButtonPressMask

# Conventions Xt

Les fonctions et macros: XtCreateWidget(…)
XtSetArg(…)

Les chaînes nommées:

```
#define   XtNWidth          "width"
#define   XtCBackground     "Background"
#define   XtRInt            "Int"
```

# Conventions MOTIF

Les fonctions : XmTextGetString(…)
Les constantes : XmDO_NOTHING

Les chaînes nommées : #define XmNlabelString "labelString"
#define XmNwidth "width"

Les noms de classes de widgets :
XmRowColumn
XmPushButton

Attention : les pointeurs sur les classes de widgets commencent par un x minuscule
xmRowColumnWidgetClass
xmPushButtonWidgetClass

# Modèle de programmation

En général une application se déroule en cinq étapes:

1. Initialiser la toolkit: connexion au serveur X, analyse des options de la ligne de commande, création d'un widget principal.

2. Créer les différents widgets nécessaires à l'interface.

3. Enregistrer les callbacks (fonctions de l'application qui sont appelées automatiquement sur les widgets quand l'utilisateur fait certaines actions).

4. Créer la fenêtre X associée à chaque widget.

5. Entrer dans une boucle d'événements (attendre le prochain événement X, puis le traiter, etc.).

# Exemple

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

main (argc, argv) char ** argv;
{
      Widget                appShell;
      XtAppContext      app;
                        /* 1. initialiser la toolkit et créer le toplevel widget */
      appShell= XtVaAppInitialize (&app,"Test3", NULL, 0, &argc, argv, NULL, NULL);
                        /* 2. Créer un bouton poussoir */
      (void) XtVaCreateManagedWidget ("Poussez-moi", xmPushButtonWidgetClass,
                                                appShell, NULL);
      XtRealizeWidget (appShell);       /* 4. Créer toutes les fenêtres X */
      XtAppMainLoop (app); /* 5. Boucle d'événements */
}
```

# Les ressources

Les ressources (valeurs des champs) des widgets sont affectables à partir de fichiers chargeables dans une base de ressources du serveur lue à l'initialisation de la toolkit. Les ressources des widgets (couleurs, positions, etc.) peuvent ainsi être modifiées sans recompilation du code.

! fichier.rdb chargeable avec la commande unix `xrdb`

```
Test3*XmPushButton.labelString:        OK
Test3*XmPushButton.fontList:           10x20
Test3*background:                      skyblue
```

```
% xrdb  fichier.rdb
```

# Autre exemple

```
#include <Xm/Xm.h>          /* nécessaire pour utiliser la librairie MOTIF */
#include <Xm/Label.h>       /* nécessaire pour utiliser un XmLabel  */
#include <stdlib.h>
#include <stdio.h>

main (argc, argv) char ** argv;
{
      Widget              shell, msg;
      XtAppContext        app;
      XmString            xmstr;

      shell = XtAppInitialize (&app,"Memo", NULL, 0, &argc, argv, NULL, NULL, 0);
      If (argc != 2) {
            fprintf(stderr, "Usage:  %s message-string\n", argv[0]);
            exit(1);
      }
```

```
        /*  créer une chaîne composée de type XmString */
    xmstr = XmStringCreateLocalized (argv[1]);

        /*  créer un widget Label */
    msg = XtVaCreateManagedWidget( "message", xmLabelWidgetClass, shell,
                                    XmNlabelString, xmstr,
                                    NULL);

        /*  libérer la chaîne */
    XmStringFree(xmstr);

        /*  créer les fenêtres et entrer dans la boucle d'événements */
    XtRealizeWidget(shell);
    XtAppMainLoop(app);
}
```

# Compilation

% gcc -g -I/usr/openwin/share/include -I/usr/dt/include -c Button.c
→ créer un fichier Button.o

% setenv LD_LIBRARY_PATH /usr/dt/lib:/usr/openwin/lib

% gcc -g -o Button Button.o -lXm -lXt -lX11

=> Dans le fichier .cshrc

setenv LD_LIBRARY_PATH [$…:]/usr/dt/lib:/usr/openwin/lib
setenv MANPATH [$…:]/usr/man:/usr/share/man:/usr/dt/man:/usr/openwin/man

# Fichier Makefile

```
CFLAGS = -I/usr/dt/include -I/usr/openwin/share/include

$(X):   $(X).o
        gcc  -g  -o  $(X)  $(X).o  -lXm  -lXt  -lX11

$(X).o: $(X).c
        gcc  -g  $(CFLAGS)  -c  $(X).c
```

Attention: l'ordre des librairies (-lXm –lXt –lX11) est très important.

# FONCTIONS DE BASE

# Fonctions de base: Initialisation

Widget XtVaAppInitialize (&app, class, options, noptions, &argc, argv, fallback, ... , NULL)

```
XtAppContext        app;
String              class;
XrmOptionDesRec*    options;
int                 noptions;
int                 argc;
char**              argv;
String              fallback;
```

- Initialise la toolkit.
- Fait la connexion au serveur X.
- Interprète les options de la ligne de commande.
- Crée un ApplicationShell widget (fenêtre principale). *class* est le nom de l'instance du widget principal dans la base de ressources.
- L'ellipse ... indique une liste à longueur variable de paramètres spécifiques du widget créé (par exemple sa taille ou sa position) sous la forme de paire : nom, valeur, etc., et termine par NULL (fonction Variadique).

# Initialisation

Widget XtVaAppInitialize (&app, class, options, noptions, &argc, argv, fallback, ... , NULL)

On verra plus loin l'algorithme utilisé pour charger les ressources. Initialement, un fichier

de ressources peut se trouver dans /usr/lib/X11/app-defaults/class.

Si ce fichier n'existe pas, l'argument fallback est interprété.

exemple:

```
String fallback [ ] = {     "Memo*fontList:    10x20", "*background:  white",
                            "Memo*XmPushButton.labelString:    OK",  NULL};

Widget                      top;
XtAppContext                app;

top = XtVaAppInitialize ( &app, "Memo", NULL, 0, &argc, argv, fallback, NULL);
```

# Fonctions de base: Création des widgets

Widget XtVaCreateWidget (name, widget_class, parent, ..., NULL)
Widget XtVaCreateManagedWidget (name, widget_class, parent, ..., NULL)

String name;
WidgetClass widget_class;
Widget parent;

- Créent une instance d'un widget de type *widget_class*.
- *name* est le nom du widget dans la base de ressources.
- *parent* est le widget parent (hiérarchie qui suit celles des fenêtres associées).
- Ces deux fonctions dont variadiques : l'ellipse ... indique une liste de paramètres spécifiques du widget créé (par exemple sa taille ou sa position) donnée sous la forme de paire nom, valeur, etc., et qui termine par NULL.

# Fonctions de base MOTIF

La toolkit MOTIF propose des fonctions spéciales de création de widgets. Ces fonctions utilisent un tableau d'arguments au lieu d'un nombre variable d'arguments. Elles ne positionnent pas le widget créé (voir plus loin XtManageChild).

Par exemple pour créer une instance d'un widget de classe PushButton:

```
Widget XmCreatePushButton (parent, name, args, num_args)
        String          name;
        Widget          parent;
        Arg*            args;
        int             num_args;
```

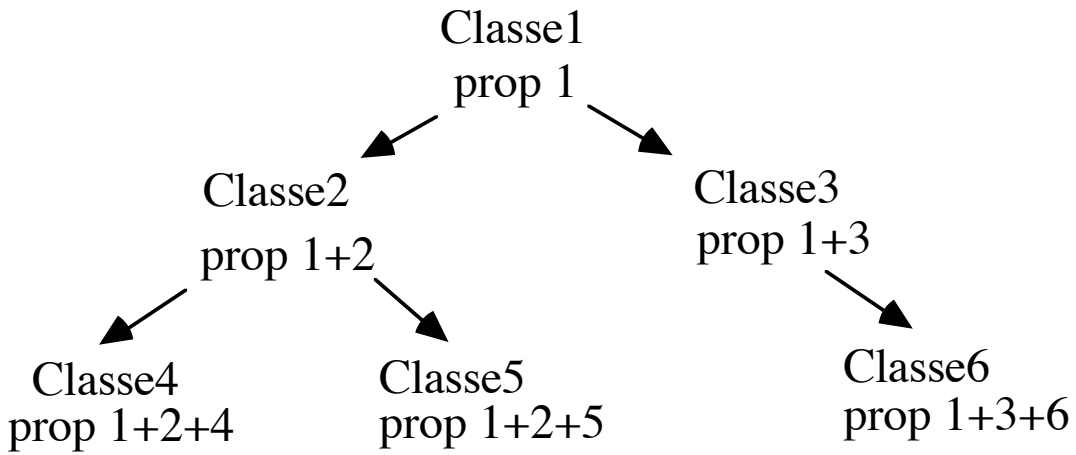
Pour créer une instance d'un gadget de type PushButton:

```
Widget XmCreatePushButtonGadget (parent, name, args, num_args)
        String          name;
        Widget          parent;
        Arg*            args;
        int             num_args;
```

# Héritage

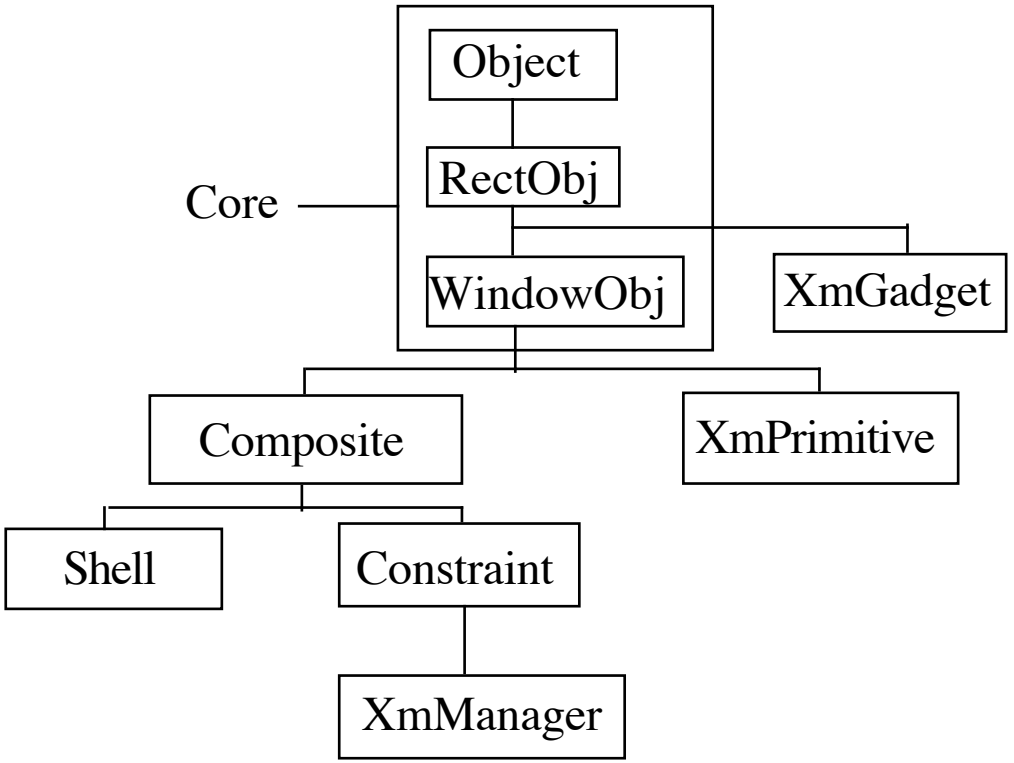
- Hiérarchie des classes
- Chaque sous-classe hérite des champs (données ou procédures) de sa super classe

Arbre d'héritage

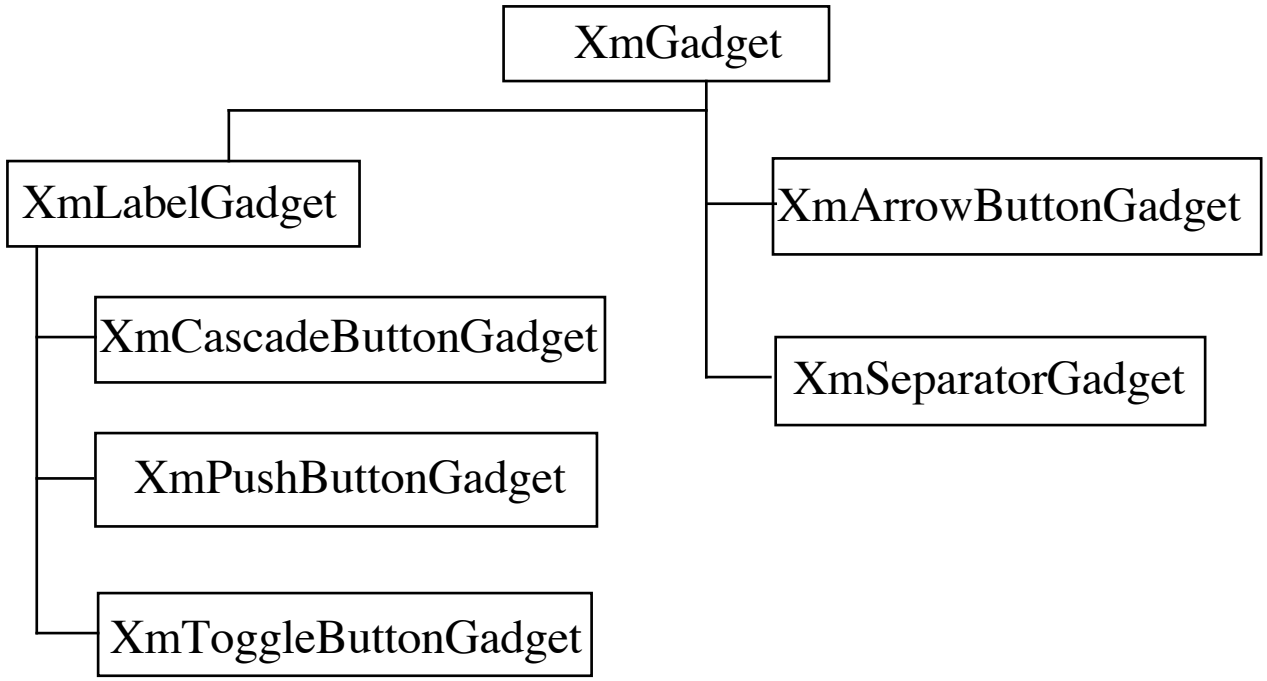



Hiérarchie: on va du plus général au particulier

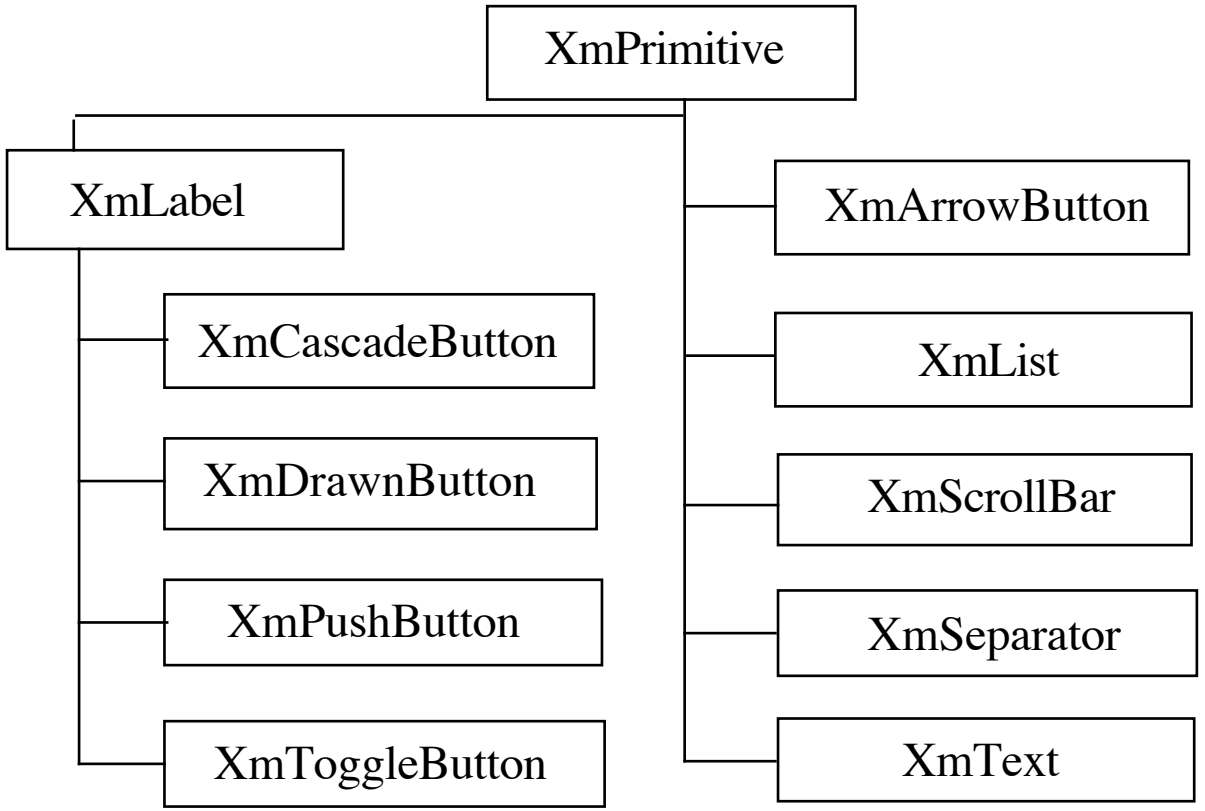
# Arbre d'héritage des classes de widgets Xt et MOTIF



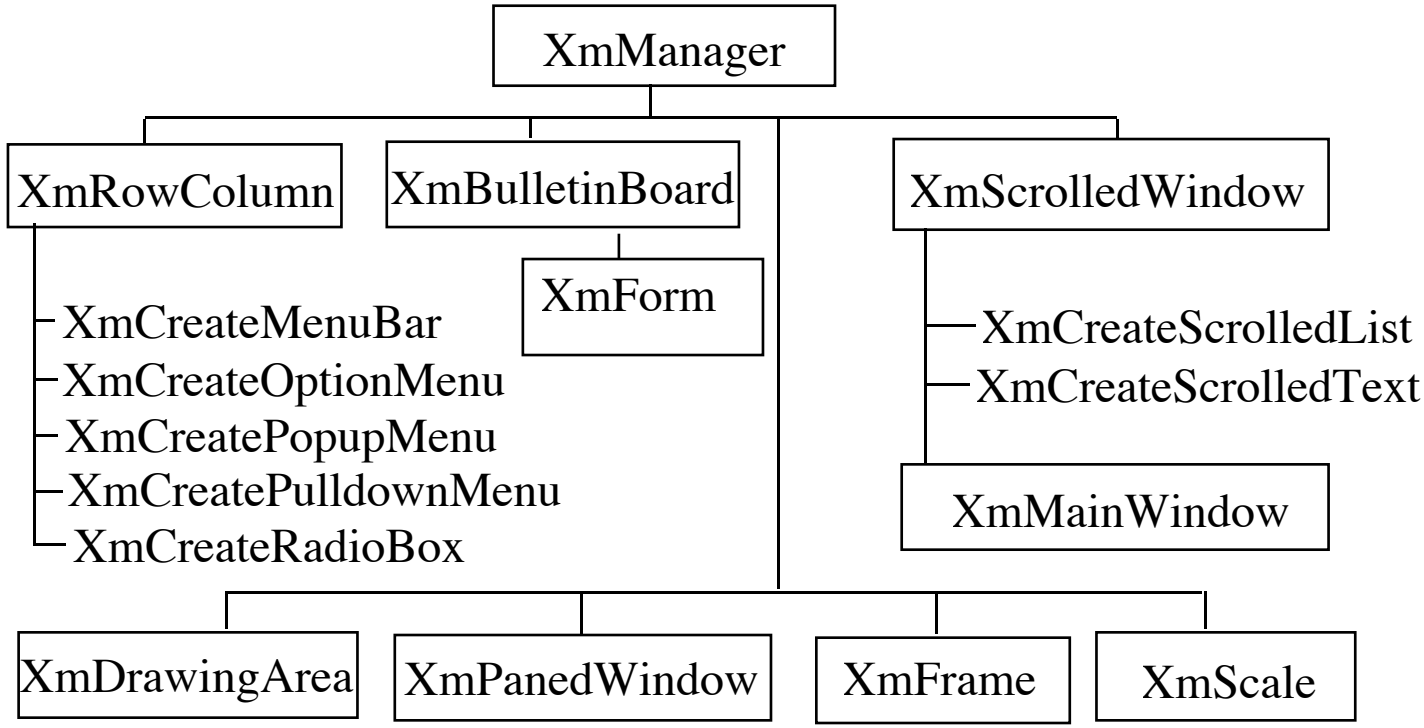
# Arborescence des Gadgets MOTIF



# Les widgets primitifs de la toolkit MOTIF



# Les widgets composites de la toolkit MOTIF



# Fonctions de base: Destruction des widgets

XtDestroyWidget (widget)
    Widget widget;

- Détruit le widget ainsi que toutes les ressources X associées (fenêtre, contextes graphiques, etc.).

# Fonctions de base: Afficher un widget

XtManageChild (widget)  Widget widget;

- Indique au parent du widget d'en gérer l'affichage (par exemple de l'aligner avec d'autres, ou bien d'adapter sa taille).
- Fait apparaître le widget.

XtUnmanageChild (widget) Widget widget;

- Indique au parent du widget de ne plus le gérer.
- Fait disparaître le widget.

Widget XtVaCreateManagedWidget (name, widget_class, parent, ..., NULL)

- Crée un widget en demandant au widget parent de le gérer.

# Fonctions de base: Créer les fenêtres

XtRealizeWidget (widget)
    Widget widget;

- Crée la fenêtre X associée au widget.
- Si le widget possède des descendants, cette opération est appliquée récursivement aux descendants.

Boolean XtIsRealized (widget)
        Widget widget;

- Retourne vrai si la fenêtre X du widget existe

# Fonctions de base: Boucle principale

XtAppMainLoop (app)

- Boucle d'attente et de traitement des événements X.
- Peut être écrite sous la forme:

```
XtAppMainLoop (app)
        XtAppContextapp;

{
        XEvent          event;
        for (; ; ) {
                XtAppNextEvent (app,&event);
                XtDispatchEvent (&event);
        }
}
```

# Fonctions de base: Evénements

XtAppNextEvent (app, event_ptr)

```
XtAppContext *     app;
XEvent *           event_ptr;
```

- Récupère le prochain événement X sur les widgets.

XtDispatchEvent (event_ptr)

```
XEvent * event_ptr;
```

- Appelle le gestionnaire (handler) de l'événement pour le widget qui est associé à la fenêtre où a eu lieu cet événement.

# Boucle d'événements : autres contrôles en attente

XtInputMask XtAppPending (app)
    XtAppContext app;

- Retourne un masque de bits indiquant s'il y a quelque chose en attente dans la file d'entrée. Ce masque est constitué des valeurs suivantes:

| | |
|---|---|
| XtIMXEvent | événement en attente de traitement |
| XtIMTimer | timer en attente |
| XtIMAlternateInput | autres entrées (fd) |

- Retourne 0 si la file est vide.

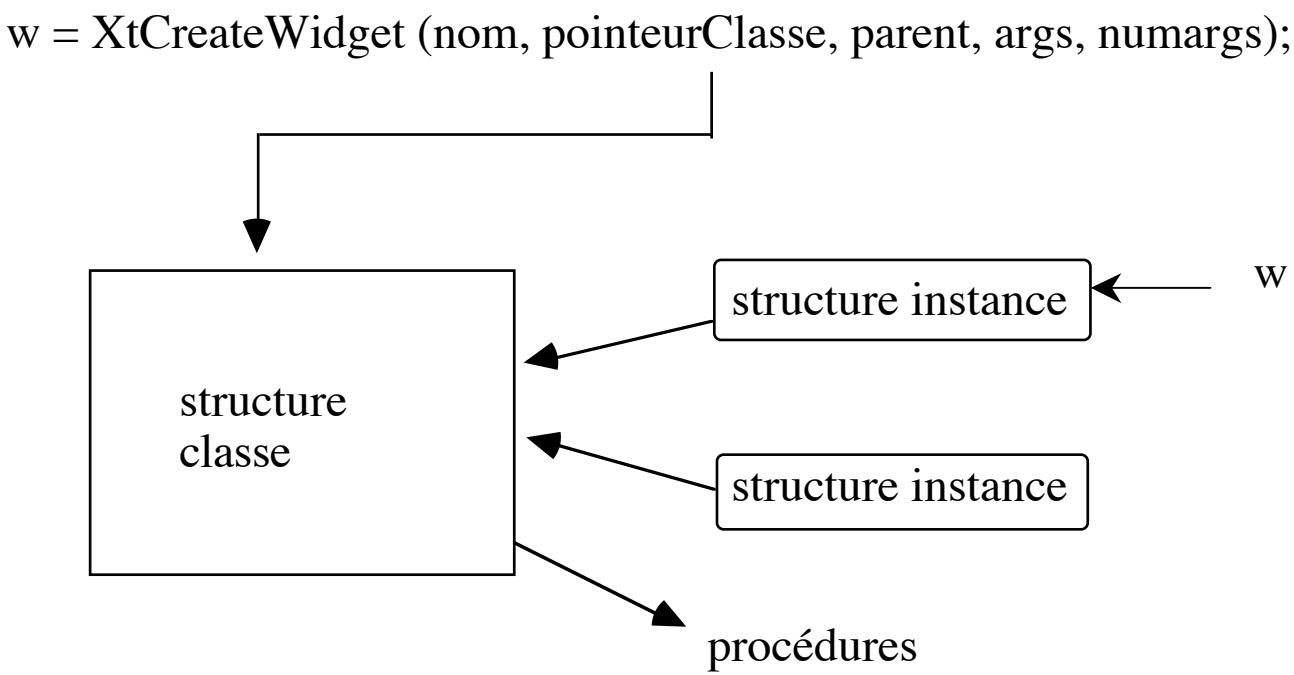
# Boucle d'événements non bloquante

Une boucle non bloquante traitant les événements présents dans la file d'entrée.

```
MyLoop ()  {
        XEvent  e;

        while (XtAppPending (app) {
                XtAppNextEvent (app, &e);
                XtDispatchEvent (&e);
        }
}
```

# Structure de données



# Arbre des instances de widget

L’interface graphique est une hiérarchie d’instances de widgets contenues les unes dans les autres.

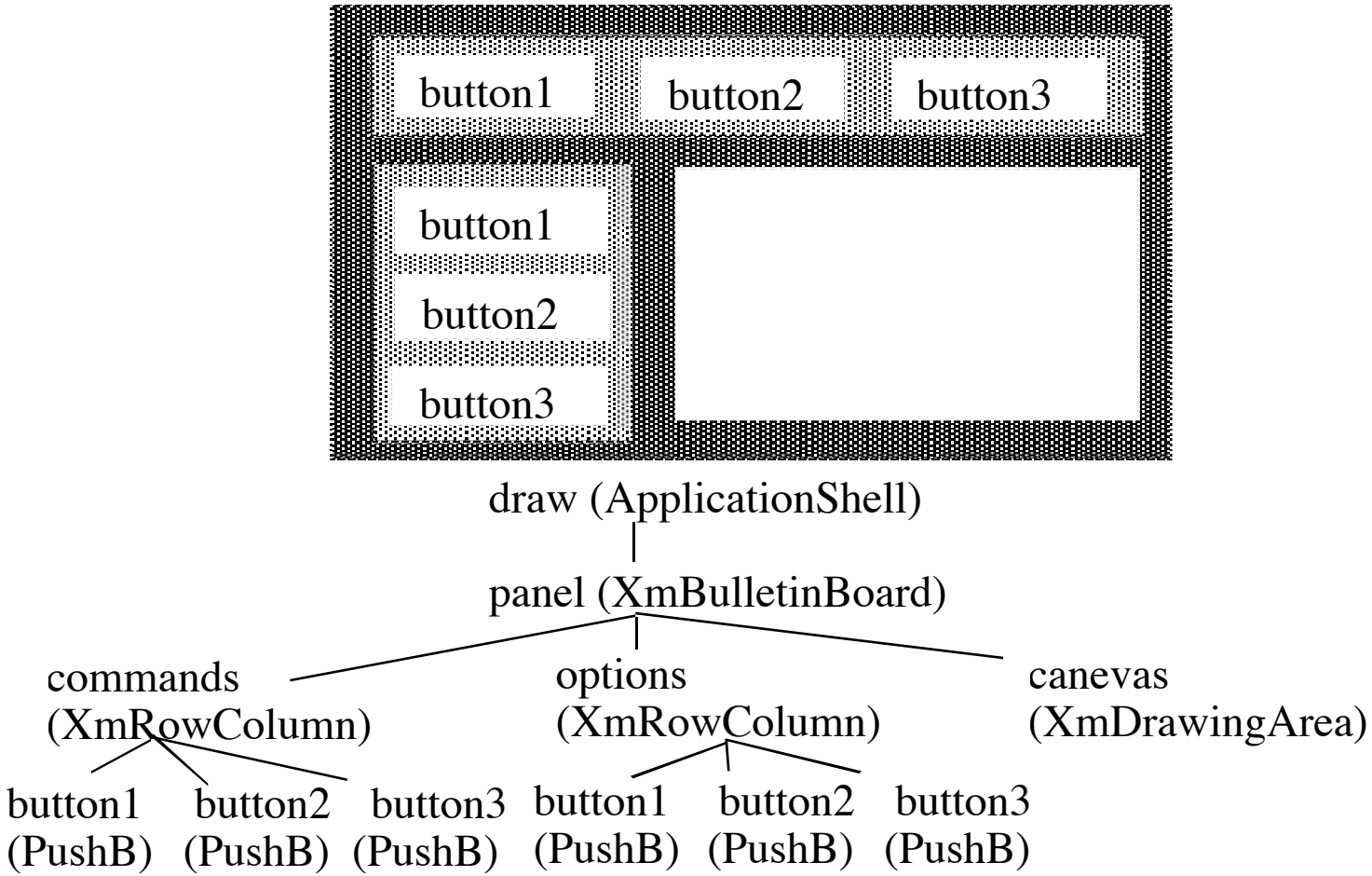
=> Arbre des instances

C’est l’arbre de filiation des instances de widgets

Principe:

- Créer d’abord le widget de plus haut niveau : le père. (Il ne peut avoir qu'un seul fils).
- Créer ensuite le fils du père (en général un Manager: il pourra ainsi avoir plusieurs fils).
- Puis les fils du fils du père, etc., etc.
- Cet arbre est utilisé dans les fichiers de ressources pour désigner des widgets particuliers.

# Exemple: XmRowColumn



# Exemple: XmForm