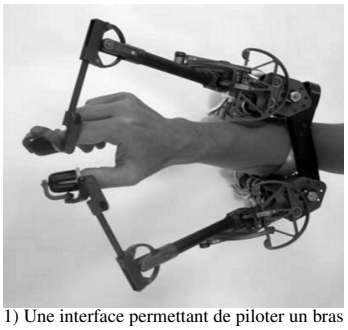
Université de la Rochelle
INFO-12605C - IHM - TD1 : Introduction

© B. Besserer, R. Péteri

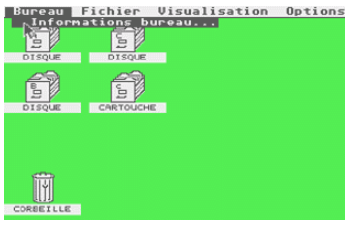
Année universitaire 2016-2017

1 Les interfaces

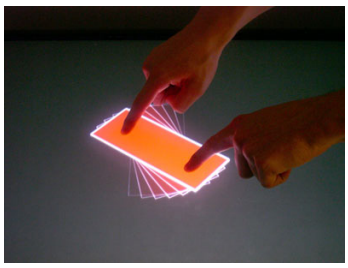
1. Examinez les interfaces ci-dessous et affectez chaque interface à son type (repondre 1=X, 2=Y, etc..) :
(A)Une interface haptique ; (B)Une interface tactile ; (C)Une interface de type WIMP



1) Une interface permettant de piloter un bras de robot pour saisir un objet et de resentir la préhension de cet objet.



2) L'interface d'un ordinateur ATARI (vers 1988-1990)



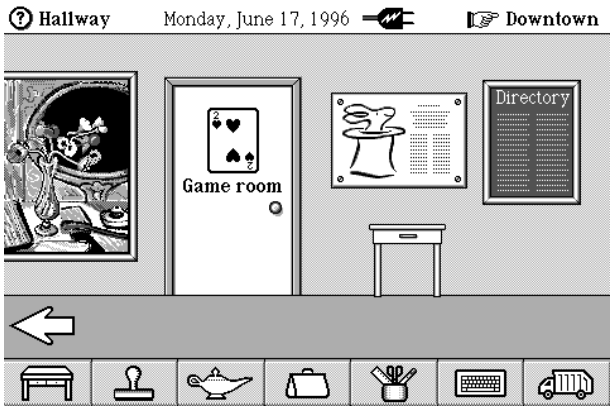
3) Une interface permettant de détecter sur une surface la position de multiples doigts

Figure 1 = A : interface haptique. L'haptique désigne la science du toucher, par analogie avec acoustique ou optique (provient du grecque âpto qui veut dire toucher). Au sens strict, l'haptique englobe le toucher et les phénomènes kinesthésiques, i.e. la perception du corps dans l'environnement. Certaines interfaces haptiques à retour de force sont utilisées pour reconstituer, à l'aide de moteurs qui impriment des mouvements en sens contraires, certaines sensations physiques liées à l'action se déroulant sur un écran. Le « bras haptique » permet d'aller plus loin en offrant la possibilité de s'immerger dans la manipulation d'objets virtuels dans les trois dimensions de l'espace.

Figure 2 = C : Interface WIMP (Windows, Icon, Menu, Pointer)

Figure 3 = B : Interface tactile, qui peut se toucher (avec un doigt, un stylet)

2. Observez la copie d'écran ci-contre ? Qu'essaye t'on de réaliser en adoptant cette symbolique et ces graphismes ? Quel est le nom utilisé pour décrire ce mimétisme avec un environnement connu ?



Elements de correction : *Le mot clé "métaphore" doit être trouvé par les étudiants.*
Cette copie d'écran montre une tentative de métaphore pour utiliser un ordinateur. On utilise l'espace familier du couloir du lieu de travail pour permettre d'accéder à diverses applications, comme le calendrier, l'annuaire téléphonique de la société, un espace de jeux, probablement une galerie d'images. D'autres actions ne sont pas décrites par la métaphore et sont représenté sous forme d'icones (d'ailleurs, la cohérence des icones avec la métaphore choisie est douteuse !). On sait

que la métaphore la plus utilisée est le "bureau" (desktop), avec le presse-papier, la corbeille, le copier-coller, etc...

3. Un laboratoire met au point le système suivant : Un joueur joue aux échecs ; l'échiquier est en verre (avec un verre fumé pour les cases sombres) et les pièces du jeu d'échecs disposent de marques de reconnaissance sous la base. Une caméra filme l'échiquier par en-dessous et transmets l'information à l'ordinateur. On ne considère pas la façon dont l'ordinateur peut déplacer ses pièces, mais on admets que c'est possible.

(a) De quel type d'interface s'agit-il ?

Interface tangible

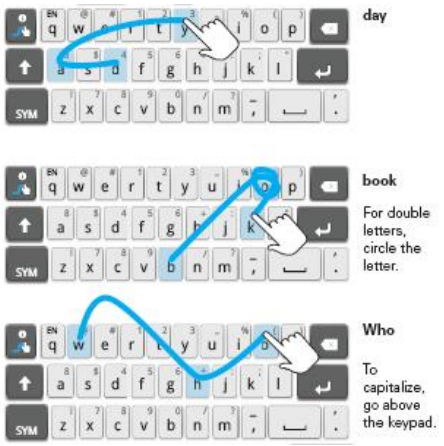
(b) Proposez un (des) mécanisme(s) de feedback informant le joueur que son action a été prise en compte.

Un feedback sonore n'est pas forcement adapté au jeu d'échec, ou le calme est de mise. On peut utiliser le plateau transparent pour un léger battement lumineux (une LED de couleur placée sous le plateau) lorsque l'ordinateur à enregistré la position de la pièce.

4. **Un peu de reflexion...** Dans la plupart des interfaces, **les dispositifs de pointage servent essentiellement à spécifier des positions**. Dans un mouvement de cliquer-glisser (*drag & drop*) par exemple, **seules la position initiale et la position finale** sont interprétées actuellement.
En exploitant **la dynamique du mouvement**, les techniques d'interaction gestuelles permettraient d'enrichir la sémantique des actions élémentaires de l'utilisateur.

(a) Donner un ou des exemple(s)

Elements de correction :
- Barrer, gommer ou effacer un élément en faisant le geste correspondant (comme si l'on barre avec un stylo ou l'on efface avec une gomme) à la souris ou sur une surface tactile.
- Ecriture intelligente sur smartphone ou tablette : swype (Motorola)

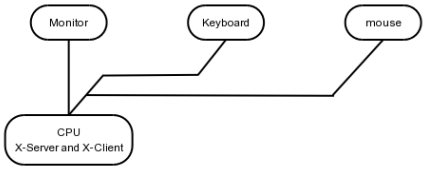


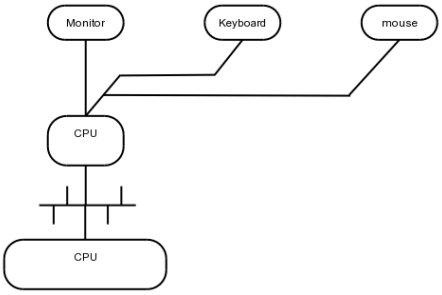
(b) Comment sont baptisés ces nouvelles interfaces ou modalités d'interaction.

Elements de correction : *Post-WIMP (du moins pour la modalité d'interaction du "gommage")*

2 X window

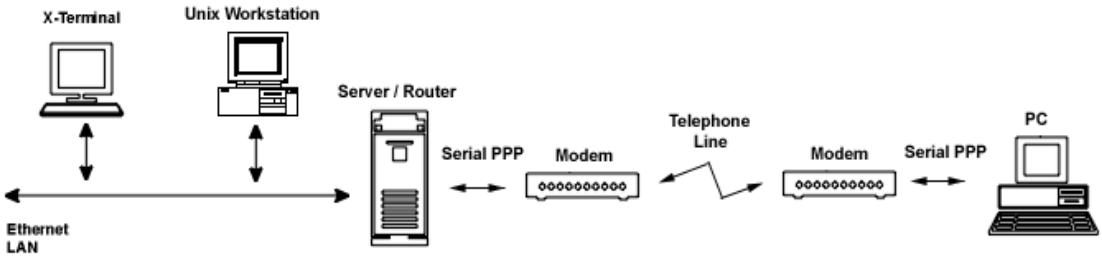
1. Ci-dessous, à gauche, la configuration classique d'une installation X11/XWindow (le cas d'un PC sous linux, par exemple). Dans ce le cas, le serveur X et le Client X s'exécutent sur la même machine. A droite, le client X est distant du serveur X, les machines (CPU) étant reliées par un reseau. Recopiez ce dessin en indiquant clairement ou s'execute le serveur X et ou s'execute le client X.





Le serveur X s'exécute sur la machine connectée aux périphériques. Le client X peut s'exécuter sur la même machine que le serveur, ou bien sur une machine distante (connexion réseau entre les deux machines).

2. Considérons la configuration informatique ci-dessous. La station de travail (Unix Workstation) exécute des logiciels utilisant, pour l'aspect IHM, le système X-Window. Répondez aux questions ci-dessous :
   (a) Le PC sous Windows 7+, peut, avec l'usage d'un logiciel adéquat (par exemple Xming), afficher la partie IHM de l'application qui s'exécute sur la station de travail ? ☐ oui ☐ non
   (b) Le cas échéant, le PC sera alors considéré comme un serveur X ? ☐ oui ☐ non
   (c) Le PC peut afficher les applications qui s'exécutent sur le terminal X ? ☐ oui ☐ non
   (d) Un terminal X a besoin d'un disque dur ? ☐ oui ☐ non
   (e) Quel impact aura la liaison modem qui est forcement "lente" entre le PC et l'application X ?



   (a) Pas de problèmes, il existe plusieurs programme permettant de créer un serveur X sur une plateforme PC/Windows (Xming, x-win32 ou cygwin/X). C'est évidemment encore plus facile sous linux. Pour OSX, il existe aussi des solutions (XQuartz)
   (b) Effectivement le PC gère les périphériques et transmet les événements, il est donc le serveur-X
   (c) Non, un terminal X est une console graphique qui ne peut exécuter que la partie "Serveur X", servant d'entrée et d'affichage pour des programmes (X-application ou X-client) distants, mais ne pouvant pas exécuter un programme localement.
   (d) Non, la gestion du serveur demande des ressources minimales, même pas un OS sous-jacent) et seul un firmware (boot depuis une EPROM ou boot depuis le réseau) est nécéssaire.
   (e) Même si la liaison est lente, l'asynchronisme de la communication permettra de construire l'image... lentement

3. Précisez si les actions suivantes sont du ressort du gestionnaire de fenetre (Window manager, comme OSF/MOTIF, KDE ou GNOME). Si non, elles sont sont du ressort de l'application X elle-même.
   (a) empiler les fenêtres lorsqu'une nouvelle fenêtre apparaît ☐ oui ☐ non
   (b) icônifier une application ☐ oui ☐ non

(c) gérer le "focus", par ex. déterminer vers quelle fenêtre sont dirigées les frappes du clavier ☐ oui ☐ non

(d) redessiner le contenu de la fenêtre lorsqu'elle redevient visible ☐ oui ☐ non

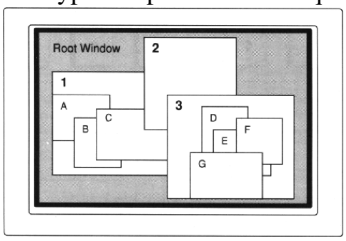
(e) changer la forme du pointeur de souris lorsque celui-ci entre dans une fenêtre spécifique ☐ oui ☐ non

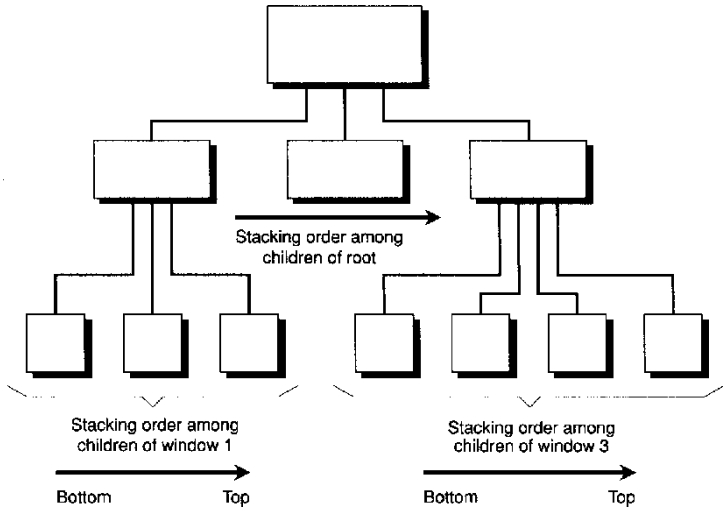
— *empiler les fenêtres lorsqu'une nouvelle fenêtre apparaît ⇒ OUI WINDOW MANAGER*
— *icônifier une applications ⇒ OUI WINDOW MANAGER*
— *gérer le "focus", c'est-à-dire de déterminer qu'elle est la fenêtre vers laquelle sont dirigées les frappes du clavier ⇒ OUI WINDOW MANAGER*
— *redessiner le contenu de la fenêtre lorsqu'elle redevient visible ⇒ NON APPLICATION*
— *changer la forme du pointeur de souris lorsque le pointeur entre dans une fenêtre specifique ⇒ NON APPLICATION*

Un gestionnaire de fenêtres fournit aux clients :
— *une certaine apparance, par la décoration qu'il leur donne,*
— *et des possibilités supplémentaires, par un éventuel menu de fenêtre et par l'icônification.*
De plus, un gestionnaire dispose en général d'un ou plusieurs menu de fond ("root menus") permettant de lancer de nouveaux clients ou d'en tuer, ou de relancer le gestionnaire.
Enfin, le gestionnaire définit
— *des raccourcis clavier,*
— *et un comportement de transition, qui décrit son comportement lorsque le curseur de la souris passe d'une fenêtre à une autre. Ceci concerne par exemple le comportement du focus du clavier :*
  — *"clicktotype" (cliquer dans la fenetre pour avoir le focus)*
  — *"followcursor" (le focus est attribué à la fenetre contenant le pointeur)*

La plupart de ces comportements sont régis par des ressources, ou des fichiers particuliers. L'usage d'un gestionnaire de fenêtres est recommandé, mais n'est pas indispensable.

4. Ci-dessous la configuration d'un écran d'un serveur X :
a) Trouver la hiérarchie (remplir la figure représentant l'arborescence)
b) Quels sont descendants (child windows ou subwindows) de la fenêtre 3. Quels sont les ancestrors (parent windows) de la fenêtre 3. Quelle est l'utilité de conserver une hiérarchisation ?
c) En général, lorsqu'une fenêtre redevient visible, peut-elle recevoir plusieurs événement de type Expose ? Dans quelles conditions ?





Evenements de type "expose" : *Il suffit qu'une fenêtre soit occultée par plusieurs autres. Lorsque celle-ci passe à l'avant-plan, l'application correspondant à cette fenêtre reçoit autant d'événement de type expose que de zones qui redeviennent visibles.*
L'application qui "gère" la fenêtre A reçoit 2 événements de type expose, et l'application qui gère la fenêtre B reçoit 1 evenement de type expose. En fait, un événement de type expose communique à l'application les coordonnées de la zone à rafraîchir. Dans le cas exposé, la zone à rafraîchir dans la fenêtre A n'est pas une zone simple (c'est à dire rectangulaire) et ne rentre donc pas dans le schéma : `xexpose.x, xexpose.y, xexpose.width, xexpose.height` *(voir annexe). La zone complexe est donc fragmenté en 2 zones rectangulaires simples.*
D'une façon générale :
— *Si une fenêtre, lorsqu'elle devient visible, reçoit plus d'un événement de type expose, il vaut mieux redessiner toute la fenêtre.*
— *Si une fenêtre, lorsqu'elle devient visible, reçoit 1 seul événement de type expose, il devient judicieux de ne redessiner que cette zone, surtout si l'on travaille avec des images Bitmap. Si le contenu de la fenêtre est dessiné avec des primitives graphiques, il vaut mieux redessiner toute la fenêtre.*

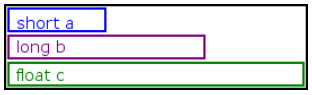
3 Les événements XEvent (Préparation au TP)

Xlib définit 33 évènements (XEvent). Chaque évènement signale une interaction (clavier, souris) ou un changement d'état (au sens large) du display. Ces événements sont envoyés par le serveur X aux clients. Les événements se présentent sous la forme d'une structure (regroupés en une union de structures) contenant **le type de l'événement** (toujours le premier champ de la structure) et un ensemble d'informations pertinentes concernant l'événement (comme le *timestamp*, la fenêtre à laquelle se rapporte l'évènement, les coordonnées du pointeur, ...). Voir les annexes pour l'union XEvent et en guise d'exemple la structure `xbutton`, `xkey` et la structure `xexpose`.

NOTE : *C'est le moment de réviser le type* `union` *du C :*
Voici comment on déclare une union ; exactement de la même manière que pour une structure :

```
union
{
    short a;
    long b;
    float c;
} mon_union;
```

Voici un rapide schéma montrant à quoi cette union correspondra en mémoire :



Le type `float` *étant, des trois types que nous avons groupé au sein de notre union,* ***celui qui occupe le plus de place en mémoire****, une variable du type de cette union* ***occupera en mémoire la taille d'un*** `float`*, dans ce cas. Ensuite, si nous adressons une variable du type de cette union par le champ de type short, nous n'utiliserons qu'une partie de l'espace mémoire occupé par cette variable, puisqu'un short est moins gros en mémoire qu'un float.*
Pour ce qui est de l'utilisation, il en va exactement de la même manière que pour les structures. Par exemple :

```
mon_union.a = 10;
mon_union.b = 70900;
mon_union.c = 3.14;
```

(En travaillant avec l'union que nous avons déclarée juste au dessus)
En ce qui concerne X11, XEvent est une union de tous les events possibles :

```
typedef union _XEvent {
   int                     type;
   XAnyEvent               xany;
   XKeyEvent               xkey;
   XButtonEvent            xbutton;
   XMotionEvent            xmotion;
   ...
}  XEvent ;
```

Chaque type d'événement est une structure particulière, mais le premier élément de la structure est TOUJOURS le type. Ainsi, il est possible de lire ce champ sans savoir exactement quelle est la structure que l'on manipule, pour pouvoir effectuer un typage pour la suite des opérations.

```
typedef struct {
  int type;              /* le type de l'événement */
  unsigned long serial;  /* dernière requête traitée */
  Bool send_event;       /* vrai si envoyé par XSendEvent */
  Display *display;      /* où l'événement s'est produit */
  Window window;         /* fenêtre recevant l'événement */
  Window root;           /* racine d'origine */
  Window subwindow;      /* sous-fenêtre dans laquelle l'ev. s'est produit*/
  Time time;             /* heure de l'événement */
  int x_root, y_root;    /* position relative à root */
  unsigned int state;    /* état (masque) boutons + modifieurs*/
  unsigned int button;   /* bouton ayant provoqué l'ev. */
  Bool same_screen;      /* si le pointeur est toujours sur l'écran */
} XButtonEvent;

typedef XButtonEvent XButtonPressedEvent;
typedef XButtonEvent XButtonReleaseEvent;
```

Si report est déclaré du type `XEvent`, il suffit de consulter le champ `report.type` pour connaitre le type de l'événement. Une structure de contrôle de type `switch ... case` permet un traitement en fonction de l'événement. Si l'événement est du type `XButtonEven`, le type dans l'union est `xbutton`, et la lecture des cordonnées du clic se fera via `report.xbutton.x` et `report.xbutton.y` Soit une boucle de gestion d'événements élémentaire et les informations sur les champs essentiels des événements les plus courants : si `report` a été déclaré de type `XEvent` (c'est donc une union), il faut donc **d'abord** en identifier le type (`switch(report.type)`). S'il s'agit d'une action sur la souris, alors l'accès aux champs des structures s'écrit :

```
report.xbutton.x
report.xbutton.y
report.xbutton.button
report.xbutton.state
```

en utilisant la description des structures données précédemment, complétez le code pour :

1. Afficher sur la console [1] la position X et Y du pointeur de souris lors du clic.

```
while(1)
{
  XNextEvent(display, &report)
  switch(report.type}
  {
    case Expose:
      break;
    case ButtonPress:
      fprintf(stderr,"Position du clic : %d, %d",report.xbutton.x,report.xbutton.y);
      break;
    default:
      break;
   }
```

2. Afficher sur la console le déplacement relatif de la souris (déplacement en X et en Y par rapport au dernier passage dans la boucle d'événement). Le type d'événement est XMotionEvent (xmotion dans l'union).

   Cet exercice est important. Comme il s'agit d'une boucle dans laquelle on passe pour chaque événement, et si cette boucle est dans une fonction, une variable locale sera forcement détruite lors de la sortie de la fonction.

   Pour pouvoir réutiliser les valeurs entre deux événements, il faut des variables globales ou une storage class de type `static`.

```
// variables globales ou static
int _oldX, _oldY;
while(1)
{
  XNextEvent(display, &report)
  switch(report.type}
  {
    case Expose:
      break;
    case MotionNotify:
      fprintf(stderr, "Deplacement en x : %d et en y : %d",
          report.xmotion.x - oldX,report.xmotion.y - oldY);
      oldX = report.xmotion.x;
      oldY = report.xmotion.y;
      break;
    default:
      break;
  }
```

3. Afficher sur la console les coordonnées de la portion de la fenêtre qui doit être redessinée après réception d'un événement de type expose. Y-a-t'il dans la structure xexpose une donnée membre permettant de savoir si l'événement de type expose est isolé ou non ? Dans quels cas y a t'il génération d'événements multiples ?

```
while(1)
{
  XNextEvent(display, &report)
  switch(report.type}
  {
    case Expose:
      if(report.xexpose.count !=0)
        {
        do
          XNextEvent(display,&report);
        while(report.xexpose.count != 0);
        printf("Plusieurs expose, on redessine toute la fenetre");
        /* inserer code dessin */
        }
      else
        printf("Un seul expose, on redessine la zone %d,%d,%d,%",
```

1. On part du principe que l'application a été démarré depuis une console, l'utilisation de `printf` est donc valide

```
                report.xexpose.x, report.xexpose.y,
                report.xexpose.width, report.xexpose.height);
      break;
    case ButtonPress:
      break;
    default:
      break;
  }
```

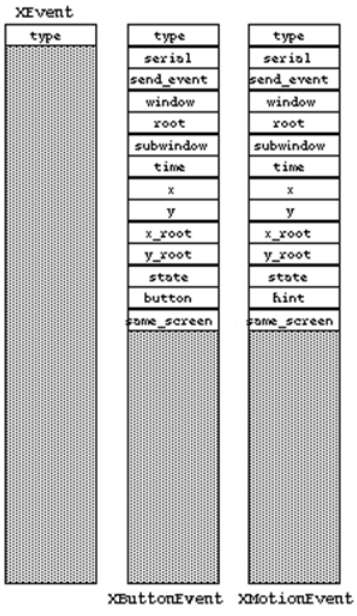
4. Si l'on utilise Qt sur une plateforme linux, Qt utilise les mécanismes sous-jacents de X11 pour la gestion de l'IHM graphique. On peut, entre autre, récupérer les événements transmis à la fenêtre de l'application construite avec Qt.

   Pour cela, on dérive une classe de la classe de base `QApplication`, et l'on implémente la méthode `x11EventFilter` (`qDebug()` permet d'écrire vers la console ou vers la fenêtre "output" de l'IDE si l'application est démarrée depuis QtCreator)

```
class XApplication: public QApplication
{
   public:
      XApplication (int & argc, char **argv): QApplication (argc , argv) { }

   protected:
      bool x11EventFilter (XEvent *e)
      {
         qDebug() << "X11 Event: " << e->type;
         return QApplication::x11EventFilter(e);   // appel de la méthode de la classe de base
      }
};
```

   Sur la base ce ce code, affichez les touches du clavier appuyées en même temps qu'un clic (*modifier keys*, genre shift-clic, control-clic). On souhaite quitter l'application sur un CTRL-clic.




```
typedef struct {
  int type; /* ButtonPress or ButtonRelease */
  unsigned long serial; /* # of last request processed by server */
  Bool send_event; /* true if this came from a SendEvent request */
  Display *display; /* Display the event was read from */
  Window window; /* ``event'' window it is reported relative to */
  Window root; /* root window that the event occurred on */
  Window subwindow; /* child window */
  Time time; /* milliseconds */
  int x, y; /* pointer x, y coordinates in event window */
  int x_root, y_root; /* coordinates relative to root */
  unsigned int state; /* key or button mask */
  unsigned int button; /* detail */
  Bool same_screen; /* same screen flag */
} XButtonEvent;
typedef XButtonEvent XButtonPressedEvent;
typedef XButtonEvent XButtonReleasedEvent;
```

Dans le fichier `Xlib.h` :

```
...
Button1Mask  (1<<8)
Button2Mask  (1<<9)
Button3Mask  (1<<10)
Button4Mask  (1<<11)
Button5Mask  (1<<12)
ShiftMask  (1<<0)
LockMask  (1<<1)
ControlMask  (1<<2)
...
```

C'est le serveur X qui détecte l'appui d'une touche "de modification". Bien sur, lorsqu'on frappe la touche CTRL avant de cliquer, un événement de type XKeyEvent est bien transmis. Mais l'application peut bien l'ignorer. Lors du clic souris, l'événement XButtonEvent continent, dans son élément `state` les indications nécessaire pour connaître la ou les touches appuyées lors du clic.

```
while(1)
{
  XNextEvent(display, &report)
  switch(report.type}
  {
    case Expose:
      break;
    case ButtonPress:
      {
      if(report.xbutton.state & ControlMask}
        exit(0);
      else if(report.xbutton.state & ShiftMask)
        printf("Touche SHIFT enfonce");
      else
        printf("Pas de touche de modification enfonce");
      default:
        break;
  }
```