Communication*
en
Xlib

- **Envoie direct d'évènements**
- **Propriétés**
- **Exemples : gestionnaire de fenêtre, copier/coller, sélection**

* Le contenu de ce cours est inspiré du cours de "Programmation d'interfaces en X Window" du Professeur Jean Berstel

Envoie direct d'évènements

- **Tout client peut envoyer un évènement à une autre fenêtre, et tout client qui a sélectionné le bon type d'événement sur la fenêtre ou sur ses antécédents le reçoit :**

```
Status XSendEvent(dpy, fen, propager, masque, e);
Window  fen;       /* peut être aussi PointerWindow ou InputFocus, */
Bool    propager;
Mask    masque;
XEvent *e;
```

- **Si** `masque` **=** `0`**,** `e` **est envoyé au client qui a créée** `fen`**,**
- **Sinon Si** `propager` **=** `False`**,** `e` **est envoyé à chaque client ayant sélectionné dans** `fen` **un des masques d'événement présents dans** `masque`**,**
- **Sinon Si** `propager` **=** `True` **et si aucun client a sélectionné un des masques d'événement présents dans** `masque`**, le serveur remplace** `fen` **par sa mère et répète la recherche des clients,**

- **Si aucune fenêtre n'est trouvée ou si la fenêtre trouvée est un parent d'une fenêtre ayant l'entrée clavier (focus) et que `fen` = `InputFocus`, alors `e` est ignoré.**

- **Le serveur joue le rôle de facteur en mettant la champs `send_event` de l'événement `e` à True.**
- **Il existe l'événement de type `ClientMessage` (sans masque) adapté pour la communication entre deux clients qui se connaissent, il peut contenir 20 octets de donnée : (voir `editres`)**

```
typedef struct {
  int            type;
  unsigned long  serial;
  Bool           send_event;
  Display       *display;
  Window         window;
  Atom message   type;
  int            format;
  union {
    char  b[20];
    short s[10];
    long  l[5];
  }              data;
} XClientMessageEvent;
```

Propriétés

- **Pour communiquer, les clients ont besoin :**
  - **des atomes,**
  - **à chaque atome est associée une valeur (caractère, entier, chaîne de caractères, structure, etc, …) pour former un couple (atome, valeur) appelé "propriété",**
  - **chaque propriété est définie pour (ou accrochée à) une fenêtre de rendez-vous choisie par les clients intéressés (en général, c'est la fenêtre racine),**

- **Atome :**
  - **chaque atome est un entier positif,**
  - **un client peut utiliser un atome prédéfini, un atome défini par un autre client, ou un atome défini par lui même,**
  - **à partir d'une chaîne de caractères (nom symbolique de l'atome), le serveur créée (côté serveur) un atome à la demande d'un client et le garde jusqu'à la fin de la vie du serveur,**
  - **pour un nom symbolique donné, il existe un seul atome,**

- **les atomes prédéfinis : (constante C, sa valeur et son nom)**

Atom	**Valeur entière**	**Nom symbolique** (chaîne de caractères)
XA_PRIMARY	1	"PRIMARY"
XA_SECONDARY	2	"SECONDARY"
...		
XA_CUT_BUFFER0	9	"CUT_BUFFER0"
...		
XA_INTEGER	19	"INTEGER"
...		
XA_STRING	31	"STRING"
...		
XA_WM_NAME	39	"WM_NAME"
...		
XA_WM_NORMAL_HINTS	40	"WM_NORMAL_HINTS "
XA_WM_SIZE_HINTS	41	"WM_SIZE_HINTS"
...		

- **on trouve les atomes prédéfinis dans le fichier** `Xatom.h`**,**
- **comme il y a une bijection entre les atomes et leurs noms symboliques, on confond l'atome et son nom.**

```
#define XA_PRIMARY              ((Atom) 1)
#define XA_SECONDARY            ((Atom) 2)
 …
#define XA_CUT_BUFFER0          ((Atom) 9)
 …
#define XA_INTEGER              ((Atom) 19)
 …
#define XA_STRING               ((Atom) 31)
 …
#define XA_WM_NAME              ((Atom) 39)
 …
#define XA_WM_NORMAL_HINTS      ((Atom) 40)
#define XA_WM_SIZE_HINTS        ((Atom) 41)
 …
```

- `XA_PRIMARY`, `XA_SECONDARY` **utilisé dans le *copier/coller* "nouvelle manière",**
- `XA_CUT_BUFFER0`, `XA_CUT_BUFFER1`, **… encore utilisé dans le *copier/coller* "ancienne manière" d'Emacs ou de XTerm,**

- `XA_INTEGER`, `XA_STRING` **utilisés pour le type de la valeur d'une propriété,**
- **les atomes prédéfinis sont chargés avec leurs noms symboliques sur le serveur au lancement de ce dernier,**
- **pour créer un atome à partir d'une chaîne de caractères (nom symbolique de l'atome):**

```
Atom XA_ESSAI;
Bool ne_pas_creer;

XA_ESSAI = XInternAtom(dpy,"ESSAI", ne_pas_creer);
```

- **si l'atom de nom** `"ESSAI"` **existe,** `XInternAtom()` **retourne l'atome,**
- **sinon si** `ne_pas_creer` **=** `True` **alors** `XInternAtom()` **retourne** `None`**,**
- **sinon le serveur créer un nouveau atome de nom** `"ESSAI"` **et** `XInternAtom()` **retourne cet atome (l'entier unique correspondant),**

- **remarquer le choix (souhaité, mais non obligatoire) du nom en majuscule et de l'identificateur (en ajoutant le préfixe `XA_` sur le nom) représentant l'atome : `"STRING"` et `XA_STRING`.**
- **pour connaître le nom :**

```
char *nom;
Atom  un_atom;
nom = XGetAtomName (dpy, un_atom);
```

- **attention, si `un_atom` n'existe pas, `XGetAtomName()` provoque l'erreur `BadAtom`,**
- **penser à libérer le `nom` après son utilisation par `XFree()`,**
- **il existe une version plurielle :**

```
Status XInternAtoms(display, names, nb, ne_pas_creer, atoms_resultat);
char **names;
int    nb;
Bool   ne_pas_creer;
Atom  *atoms_resultat;   /* le client fournit le tableau d'atom */

Status XGetAtomNames(display, atoms, nb, names_return);
Atom  *atoms;
char **names_resultat;   /* le client fournit le tableau de char * */
```

- **Propriété :**
  - **c'est un couple (atome, valeur) défini pour (ou accroché à) une fenêtre. Comme on confond l'atome et son nom, il est usuel de parler de (nom, valeur) plutôt que (atome, valeur),**
  - **elle est créée côté serveur et gardée jusqu'à**
    - **sa destruction par un client quelconque,**
    - **la destruction de la fenêtre,**
    - **ou la fin de la vie du serveur,**
  - **elle peut être consultée par n'importe quel client,**
  - **elle est créée par un client quelconque, accrochée à une fenêtre quelconque et modifiable par un client quelconque,**
  - **un client quelconque peut recevoir l'événement `PropertyNotify` s'il a sélectionné le masque `PropertyChangeMask` sur la fenêtre concernée pour le changement de ses propriétés,**
  - **la valeur peut être un caractère, un entier, une chaîne de caractères, une structure, etc. A voir comme un ensemble d'octets en C.**

- **la nature de la valeur est donc indiquée**
  - **par un type (c'est aussi un atome) pour les clients,**
    - **prédéfini, par exemple `XA_STRING` pour une chaîne de caractères (`char *`), `XA_ARC` pour une structure de type `XArc`, etc, ...),**
    - **défini par un client, par exemple `XA_ESSAI`,**
    - **laisse aux clients l'interpréter,**
  - **et par un format (8, 16 ou 32 bits) pour le serveur. Il indique au serveur comment stocker physiquement ou de restituer la valeur : par octet, par mot de 16 bits ou par mot de 32 bits.**
- **par exemple :**
  - **la propriété concernant le nom de la fenêtre principale d'un client que le gestionnaire de fenêtre consulte :**
    - **nom :** `XA_WM_NAME`**,**
    - **valeur : chaîne de caractères comme nom de la fenêtre pour être affiché dans la barre de titre,**
    - **type :** `XA_STRING`**,**
    - **format : 8 bits,**

- **la propriété concernant la configuration géométrique de la fenêtre principale d'un client que le gestionnaire de fenêtre consulte :**
  - **nom :** `XA_WM_NORMAL_HINTS`,
  - **valeur :** **structure** `XSizeHints` **(19 mots de 32bits),**
  - **type :** `XA_WM_SIZE_HINTS`,
  - **format :** **32 bits,**

- **convention de correspondance entre type prédéfini, structure C et format :**

Type	structure C	format
XA_ARC	XArc	16
XA_ATOM	Atom	32
XA_DRAWABLE	Drawable	32
XA_INTEGER	int	32
XA_STRING	char *	8
...		

- **pour changer une propriété :**

```
XChangeProperty(Display *dpy,
    Window          fen,            /* accrochee à fen       */
    Atom            prop,           /* nom de la propriete   */
    Atom            t,              /* type de la propriete  */
    int             format,         /* 8, 16 ou 32 (bits)    */
    int             mode,           /* substituer ou ajouter */
    unsigned char *donnees,         /* donnee                */
    int             taille);        /* nombre d'elements     */
```

avec mode **=** PropModeReplace, PropModePrePend **ou** PropModeAppend

- **par exemple :**

```
XChangeProperty(dpy, fen, XA_A, XA_STRING, 8, PropModeReplace,
     "Hello!", 1+strlen("Hello!");

XArc a = {0, 0, 100, 200, 0, 23040};
XChangeProperty(dpy, fen, XA_B, XA_ARC, 16, PropModePrePend,
                (unsigned char *) &a, 6);

Window x;
XChangeProperty(dpy, fen, XA_C, XA_ DRAWABLE, 32,
                PropModeAppend, (unsigned char *) &x, 1);
```

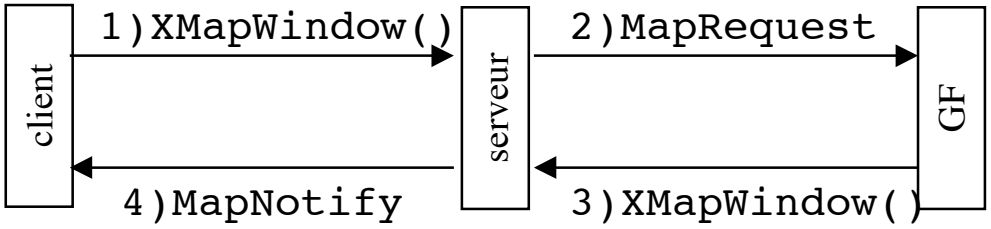
- **pour récupérer la valeur d'une propriété :**

```
int XGetWindowProperty(Display *dpy,
   Window           fen,
   Atom             nom_propriete,
   long            *decalage,
   long            *longueur,
   Bool            *supprimer_prop_apres_lecture,
   Atom            *type_souhaite_requete,
   Atom            *type_effectif_retour,
   int             *format_effectif_retour,
   unsigned long   *nb_lus_retour,
   unsigned long   *nb_octets_restants_retour,
   unsigned char **donnees_retournees);
```

`decalage` **du début de la valeur, compté par mot de 32 bits.**
`longueur` **de la valeur à lire, compté par mot de 32 bits.**
`type_souhaite_requete` **=** `AnyPropertyType` **si on n'est pas sûr ou que l'on s'en fiche.**
`nb_lus_retour` **nombre d'éléments en 8, 16 ou 32 bits contenus dans** `donnees_retournees`**.**
`donnees_retournees` **terminées toujours avec un octet contenant 0 en plus.**

Exemples

- **Gestionnaire de fenêtre : (GF)**
  - **un client comme un autre,**
  - **mais sélectionne en plus** `SubstructureRedirectMask` **et/ou** `ButtonPressMask` **sur la fenêtre racine,**
  - **un autre client (un autre GF) qui tente de sélectionner un des deux masques sur la même fenêtre est rejeté avec l'erreur** `BadRequest`**.**
  - **en sélectionnant le** `SubstructureRedirectMask` **sur la fenêtre mère (avec** `Sub`**), le GF reçoit les évènements** `CirculateRequest`**,** `ConfigureRequest` **et** `MapRequest` **concernant ses filles.**
  - **au moment où l'on veut afficher (Map) une fenêtre fille et que son attribut** `override_redirect` **est** `False` **(voir** `XSetWindowAttributes`**), le GF reçoit l'événement** `MapRequest`**,**



- **à la réception de l'événement `MapRequest` de la fenêtre principale `fen`, le GF consulte les propriétés de `fen` de nom `XA_WM_xxx` pour honorer les configurations demandées et décore `fen`.**
- **Il existe des utilitaires pour créer ou modifier ces propriétés :**

```
char       *nom;
XSizeHints  xsh;
XStoreName(dpy,fen,nom);
XSetWMNormalSizeHints(dpy, fen, &xsh);
```

- **Copier/Coller, version ancienne : (emacs, xterm, et quelques autres)**
  - **8 propriétés accrochées à la fenêtre racine de nom `XA_CUT_BUFFER0,..., XA_CUT_BUFFER7`,**
  - **tout client intéressé par le changement de ces propriétés sur la fenêtre racine sélectionne sur cette fenêtre le masque `PropertyChangeMask`. Lorsqu'il y a un changement de propriété, le client est informé.**
  - **après avoir sélectionné avec la souris un bout de texte, et/ou après avoir fait un copier ou couper, un client A a du texte à proposer, modifie la valeur de la propriété de nom `XA_CUT_BUFFER0` (ou `XA_CUT_BUFFERn`) avec son texte et distingue graphiquement son bout de texte à l'écran,**

- **Pour modifier la valeur de la propriété de nom `XA_CUT_BUFFER0`,**

```
XStoreBytes(dpy, octets, nombre)
char *octets;
```

- **Pour modifier la valeur de la propriété de nom `XA_CUT_BUFFERn`,**

```
XStoreBuffer(dpy, octets, nombre, n)
```

- **le client Y qui a proposé du texte précédemment reçoit un `PropertyNotify`, vérifie que c'est le bon nom de la propriété et banalise graphiquement son bout de texte à l'écran.**
- **en faisant un coller, un client B consulte la valeur de la propriété et ajoute le contenu du texte proposé.**

```
char *octets;
int   nombre;
octets = XFetchBytes(dpy, &nombre);
/* octets = XFetchBuffer(dpy, &nombre, n); */
...
XFree(octets);
```

- **Avantages :**
  - **simple à mettre en œuvre,**
  - **propriété qui reste même après la disparition du client qui l'a modifiée.**
- **Inconvénients :**
  - **propriété (qui peut être de très grande taille) bloque la mémoire pour un hypothétique client intéressé,**
  - **pas de choix sur le format : un seul format proposé par le client qui l'a modifiée, c'est à prendre ou à laisser.**

❑ **Copier/Coller, version sélection :**

- **les clients s'ignorent,**
- **zéro stock côté serveur si les clients respectent bien les règles de l'offre et de la demande des données,**
- **le demandeur peut proposer un format de donnée à celui qui offre, ce dernier informe en réponse le demandeur qu'il peut ou non honorer ce format.**

- **Sélections : l'offre et la demande**
  - **une sélection est caractérisée par**
    - **un nom (atome) connu des clients intéressés, en général** `XA_PRIMARY` **pour du texte, il caractérise la sélection,**
    - **un client, à travers une fenêtre choisie (généralement la fenêtre racine), déclaré propriétaire de l'offre de cette sélection ou plutôt, c'est la fenêtre qui est la propriétaire,**

- **Celui qui offre :**
  - **le client qui veut proposer une donnée se fait connaître en devenant propriétaire d'une sélection choisie**

```
XSetSelectionOwner(Display *dpy,
    Atom   nom_de_la_selection,
    Window fen_proprietaire,
    Time   a_partir_de_date);

XSetSelectionOwner(dpy, XA_PRIMARY, fen, CurrentTime);
```

- **le client, ex-propriétaire de cette sélection, reçoit juste après un évènement de type `SelectionClear` (sans masque), il vérifie que c'est bien la bonne fenêtre et bon nom (atome), il comprend alors qu'il n'est plus propriétaire (et banalise graphiquement son texte).**

```
typedef struct {
        int             type;
        unsigned long   serial;
        Bool            send_event;
        Display        *display;
        Window          window;    /* qui vient de perdre */
        Atom            selection; /* la selection        */
        Time            time;
} XSelectionClearEvent;
```

- **si l'ex-propriétaire est le même client, même avec une fenêtre différente, il ne recevra pas cet événement,**
- **un seul propriétaire à la fois par sélection,**
- **mais un client peut être propriétaire de plusieurs sélections à la fois,**
- **pour connaître la fenêtre propriétaire d'une sélection en donnant son nom**

```
fen = XGetSelectionOwner(dpy, XA_PRIMARY);
```

- **Celui qui demande :**
  - **le client formule sa demande sans connaître celui qui offre (ça peut être lui même), en donnant simplement le nom de la sélection, mais le serveur le connaît,**

```
XConvertSelection(Display *dpy,
     Atom    la_selection,
     Atom    type_demande,
     Atom    nom_de_propriete_a_mettre_les_donnees,
     Window  fen_de_RDV,
     Time    date_a_partir);

   XConvertSelection(dpy,
     XA_PRIMARY, XA_STRING, prop, fen, CurrentTime);
```

  - **il est important que *fen* soit une fenêtre créée par le client demandeur (voir la suite),**

- **Le serveur :**
  - **connaît le propriétaire de la sélection, lui envoie un événement de type `SelectionRequest` (sans masque)**

```
typedef struct {
        int             type;
        unsigned long   serial;
        Bool            send_event;
        Display        *display;
        Window          owner;      fenetre proprietaire de la sélection
        Window          requestor;  fenêtre de RDV du demandeur
        Atom            selection;  en général XA_PRIMARY ou XA_SECONDARY
        Atom            target;     type demandé
        Atom            property;   nom de propriété à mettre les données
        Time            time;
} XSelectionRequestEvent;
```

- **Celui qui offre :**
  - **en convertissant si possible la donnée au type (donc format) demandé**

```
switch (evmt.type)
  case SelectionRequest :
   if (evmt.xselectionrequest.selection == la_selection &&
       evmt.xselectionrequest.target == XA_STRING)
        XChangeProperty(dpy,
          evmt.xselectionrequest.requestor,
          evmt.xselectionrequest.property, XA_STRING,
          8, PropModeReplace, donnees, 1+strlen(donnee));
```

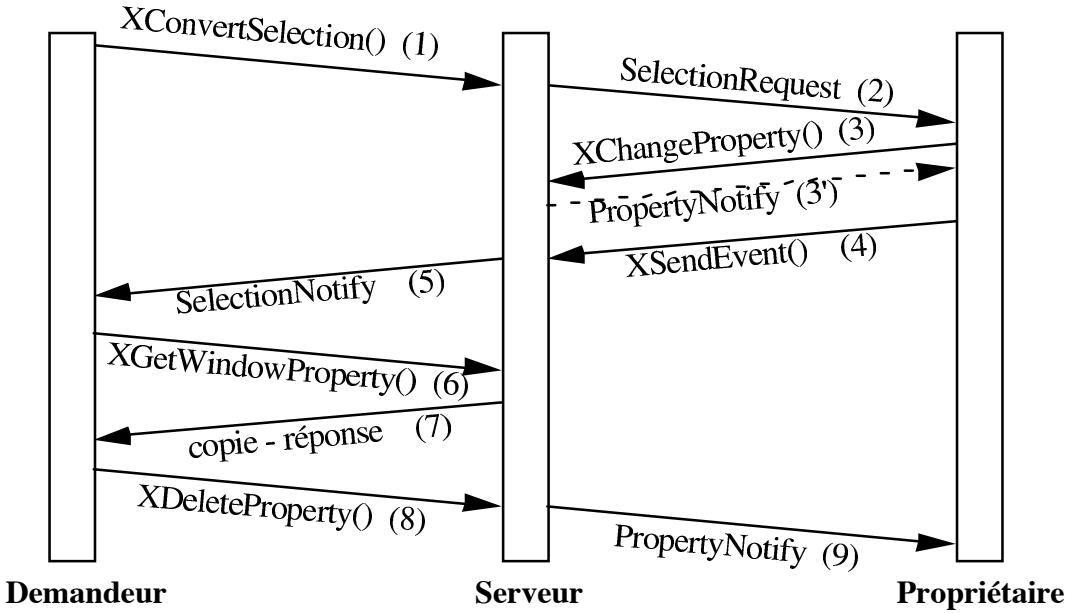
- **envoie ensuite par `XSendEVENT()` un événement de type SelectionNotify à la fenêtre du demandeur (il l'a créée),**

```
typedef struct {
        int    type;
        Bool   send_event;
        Window requestor;
        Atom   selection;
        Atom   target;       type demandé
        Atom   property;     None pour convertion impossible
        Time   time;
} XSelectionEvent;

XEvent send;
  send.type                  = SelectionNotify;
  send.xselection.requestor = evmt.xselectionrequest.requestor;
  send.xselection.selection = evmt.xselectionrequest.selection;
  send.xselection.target    = evmt.xselectionrequest.target;
  send.xselection.time      = evmt.xselectionrequest.time;
  send.xselection.property  = evmt.xselectionrequest.property; ou None

XSendEvent(dpy, send.xselection.requestor, False, 0, &send);
```

- **Retour à celui qui demande :**
  - **reçoit l'événement** `SelectionNotify`**,**
  - **récupère les données,**
  - **libère la place en le détruisant, le serveur émet alors un événement de type** `PropertyNotify` **à ceux qui ont sélectionné** `PropertyChangeMask` **sur le fenêtre du RDV (donc à celui qui a offert ces données si il l'a sélectionné).**

```
switch(evmt.type) {
case SelectionNotify :
  if (evmt.xselection.selection == la_selection) {
     if (evmt.xselection.property == None);    /* c'est raté */
     else {
        XGetWindowProperty(dpy,
          fen_de_RDV, XA_STRING, 0, 8192, False, propD,
          &quel_type, &quel_format, &lus, &restants, &donnees);
        ...
        XFree(donnees);
        XDeleteProperty(dpy, fenD, propD);
    }
    break;
```

- **En résumé :**


XConvertSelection() (1)
SelectionRequest (2)
XChangeProperty() (3)
PropertyNotify (3')
XSendEvent() (4)
SelectionNotify (5)
XGetWindowProperty() (6)
copie - réponse (7)
XDeleteProperty() (8)
PropertyNotify (9)
Demandeur
Serveur
Propriétaire