

X-I

Communication

- o Envoyer des évènements
- o Atomes
- o Propriétés
- o Les tampons du “couper-coller”
- o Gestion des propriétés
- o Sélections

```
ClientMessage  
PropertyNotify  
SelectionClear  
SelectionNotify  
SelectionRequest
```

X-I

Envoyer des évènements

- o **Tout client peut envoyer un évènement à une fenêtre, et tout client qui a sélectionné le bon type d'évènement peut le lire.**

```
XSendEvent(dpy, voisine, propager, masque, e);  
Window voisine;  
Bool propager;  
Mask masque;  
XEvent * e;
```

- o **La fenêtre voisine peut aussi être PointerWindow ou InputFocus.**
- o **Si propager est False, il n'y a pas propagation depuis la fenêtre voisine.**
- o **La requête est honorée par le serveur (jouant le rôle de facteur) en transmettant l'évènement sans modification, sauf qu'il l'oblitére en mettant le champ send_event à True.**
- o **Exemple : faire suivre, à la fenêtre voisine, un évènement KeyPress :**

```
switch (evmt.type) {...  
  case KeyPress :  
    XSendEvent(dpy, voisine, True, KeyPressMask, &evmt);  
    break;  
}
```

X-I

Les messages clients

```
typedef union {  
    ...  
    XClientMessageEvent xclient;  
    ...  
} XEvent;  
  
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;
```

- o **Il existe un évènement spécialement conçu pour servir à des besoins particuliers de transmission d'informations entre clients : ClientMessage.**
- o **Les données à transmettre peuvent occuper jusqu'à 20 octets (ou leur équivalent dans d'autres types).**
- o **Un tel évènement n'est pas masquable, et est généralement envoyé par SendEvent (le booléen send_event vaut alors True).**
- o **Exemples :**
 - | **L'utilitaire editres d'édition de ressources utilise ce mécanisme;**
 - | **Certaines émulations du système Windows™ utilisent ces évènements pour simuler les messages.**

X-I

Atomes

- Pour communiquer, les clients X doivent disposer d'un *vocabulaire*.
- Les *vocables* sont les *atomes*.
- Un *atome* est la représentation interne, par un entier long, d'une chaîne de caractères.
- Chaque atome prédéfini possède un *nom symbolique* (inclure `Xatom.h`).
 - | Tous ces noms commencent par `XA_`.
 - | Le nom symbolique se déduit simplement de la chaîne:
 - `XA_STRING` correspond à "STRING",
 - `XA_INTEGER` correspond à "INTEGER".
- Un client peut utiliser les atomes prédéfinis ou définir lui-même des atomes.
- Les atomes sont stockés dans le *serveur*. Les atomes prédéfinis sont chargés au moment du lancement du serveur.


X-I

Exemples

- o Les échanges entre gestionnaire de fenêtres et autres clients utilisent en particulier les atomes commençant par `XA_WM_`.
- o Le *couper-coller* “ancienne manière” utilise les 8 tampons `XA_CUT_BUFFER0` jusqu’à `XA_CUT_BUFFER7`. Ceci est utilisé dans `xterm` ou `emacs` (les nouvelles versions sont à jour).
- o Le *couper-coller* “nouvelle manière” utilise le mécanisme des *sélections* via les atomes `XA_PRIMARY` et accessoirement `XA_SECONDARY`. Ceci est utilisé dans les widgets de texte de Motif.
- o Motif utilise aussi `XA_CLIPBOARD` (à ne pas confondre avec l’utilitaire `xclipboard`).
- o Un utilitaire de conversion entre les deux “couper-coller” existe (`xcutsel`)

Quelques atomes prédéfinis, et leurs noms symboliques

```
#define XA_PRIMARY ((Atom) 1)
#define XA_SECONDARY ((Atom) 2)
#define XA_ARC ((Atom) 3)
#define XA_ATOM ((Atom) 4)
#define XA_CARDINAL ((Atom) 6)
#define XA_CUT_BUFFER0 ((Atom) 9)
#define XA_CUT_BUFFER7 ((Atom) 16)
#define XA_DRAWABLE ((Atom) 17)
#define XA_FONT ((Atom) 18)
#define XA_INTEGER ((Atom) 19)
#define XA_PIXMAP ((Atom) 20)
#define XA_POINT ((Atom) 21)
#define XA_RECTANGLE ((Atom) 22)
#define XA_RESOURCE_MANAGER ((Atom) 23)
#define XA_RGB_BLUE_MAP ((Atom) 26)
#define XA_RGB_RED_MAP ((Atom) 30)
#define XA_STRING ((Atom) 31)
#define XA_WINDOW ((Atom) 33)
#define XA_WM_HINTS ((Atom) 35)
#define XA_WM_TRANSIENT_FOR ((Atom) 68)
```

X-I

Création d'atomes

- L'utilisation d'un atome réservé permet à deux clients d'échanger des informations selon un code qui leur est propre. Mais, selon la philosophie X, il n'y a pas de "propriétaire" des atomes.
- Une application qui veut utiliser un atome réservé le définit à partir d'une chaîne de caractères.

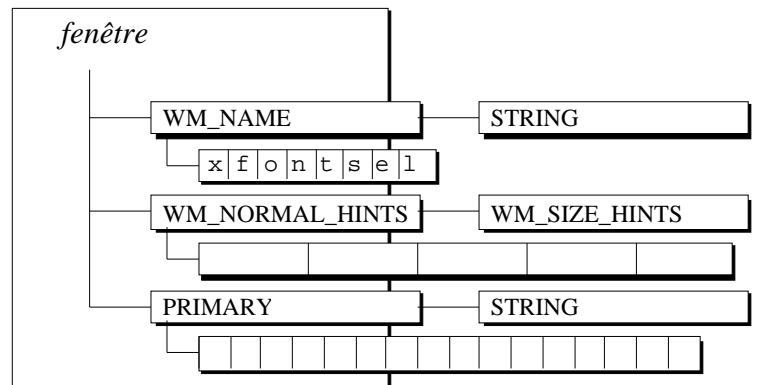
```
Atom A;  
Bool ne_pas_creer;  
A = XInternAtom(dpy, "rubis", ne_pas_creer);
```

- La fonction `XInternAtom` vérifie si un atome de ce nom littéral existe.
 - | S'il existe, elle renvoie l'atome;
 - | Sinon
 - si `ne_pas_creer` est `True`, elle renvoie `None`,
 - si `ne_pas_creer` est `False`, elle demande au serveur de créer l'atome et en renvoie le numéro.
- La fonction `XGetAtomName` prend en argument un atome, et retourne la chaîne de caractères associée : `nom = XGetAtomName(dpy, A)`
- Il existe (depuis X11R6) des versions plurielles de ces fonctions :
`XInternAtoms` et `XGetAtomNames`

X-I

Propriétés

- o Une *propriété* est un couple (nom, valeur) attaché à une fenêtre.
 - | Le *nom* de la propriété est un atome;
 - | La *valeur* est a priori quelconque, représentée dans un format rudimentaire (une chaîne d'octets)
 - | la propriété est dite *définie* pour la fenêtre.
- o Une propriété a en outre
 - | un *type*, qui est une indication sur la nature de la valeur associé
 - | un *format* (8, 16, 32) qui informe sur la représentation des données.



X-I

Propriétés des propriétés

- **Deux fenêtres peuvent avoir des propriétés de même nom et de valeur différentes.**
- **Les valeurs des propriétés sont conservées sur le serveur.**
- **Tout client peut créer des propriétés dans toute fenêtre et y écrire.**
- **Tout client peut lire la valeur contenue dans une propriété, et la détruire.**
- **Un client peut être informé d'un changement dans une propriété (par un évènement `PropertyNotify`).**

X-I

Exemple de propriétés : les tampons du “copier-coller”

- X prédéfinit 8 atomes de tampons, de nom `XA_CUT_BUFFER0`, . . . , `XA_CUT_BUFFER7` utilisés pour le “couper-coller”.
- L’échange entre clients se fait par des propriétés ayant ces noms, *attachées à la fenêtre racine* de l’écran.
- Il existe des fonctions utilitaires pour accéder aux tampons.
- Le mécanisme du “couper-coller” (ancienne manière) est implémenté comme suit :
 - ┆ le client A, lorsqu’il décide de “couper” ou de “copier” (i. e. quand il a fini la sélection), copie l’objet dans un tampon.
 - ┆ le client B, lorsqu’il décide de “coller”, récupère l’objet dans le tampon.
- Ce mécanisme est simple, mais pas très souple:
 - ┆ La copie de A vers le serveur se fait d’office, et non à la demande, d’où un possible encombrement du serveur pas des données qui ne servent pas;
 - ┆ Cette copie se fait dans le format connu de A, et non dans le format souhaité par B, d’où une limitation des échanges possibles.
- **Avantage** : les données copiées sont conservées même après la disparition éventuelle du client A.

X-I

Pour copier et coller

- o Copier dans le tampon 0:

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

Le tableau d'octets n'est pas nécessairement terminé par le caractère nul.

- o Copier dans un tampon numero:

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

- o Récupérer les données du tampon 0 :

```
char *octets;
octets = XFetchBytes(dpy, &nombre);
...
XFree(octets);
```

Retourne le nombre d'octets dans nombre et alloue lui-même la place nécessaire. C'est pourquoi elle doit être libérée ensuite.

- o Récupérer les données du tampon numero :

```
octets = XFetchBuffer(dpy, &nombre, numero);
```

X-I**Propriétés : formats et types**

- o Les propriétés ont des valeurs.
 - | le *format* indique la représentation physique. C'est toujours une suite d'octets (unsigned char), mais comptés en :
 - 8 bits (octets)
 - 16 bits (entiers courts)
 - 32 bits (entiers longs).
 - | le *type* définit le sens de la valeur.
- o Pour certaines propriétés prédéfinies, il existe des utilitaires pour ranger des valeurs ou les récupérer:

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

Ceci suppose une conversion des données (par l'utilitaire) selon un *type* convenu.
- o Comment convenir de types entre clients ? En les nommant dans le vocabulaire commun, donc par des atomes !

X-I

Exemples de types et formats

- o Les propriétés prédéfinies ont des *types prédéfinis*.
 - | Le type ne donne aucune indication sur la manière de stocker les données sur le serveur (suite d'octets) : c'est le rôle du format.
 - | Il sert à convenir de l'*interprétation* des données brutes entre les *clients*.
 - | Il est de la responsabilité des clients de fournir les données dans le format approprié.
- o Pour l'implémentation C de Xlib, il y a des correspondances naturelles entre les types décrits dans l'ICCCM et les structures C.
- o Exemples de conventions de correspondance, et formats:

XA_ARC	XArc	16
XA_ATOM	Atom	32
XA_DRAWABLE	Drawable	32
XA_FONT	Font	32
XA_INTEGER	int	16 ou 32
XA_STRING	char *	8
XA_WM_SIZE_HINTS	XSizeHints	32

X-I

Manipuler des propriétés

```
XChangeProperty(Display* dpy, Window fen,  
Atom prop, /* nom de la propriété */  
Atom t, /* type de la propriété */  
int format, /* 8, 16 ou 32 */  
int mode, /* substituer ou ajouter */  
unsigned char* donnees, /* data */  
int taille); /* nombre d'éléments */
```

o Exemples

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

```
XArc a = {0, 0, 100, 200, 0, 23040};
```

```
XChangeProperty(dpy, fen, B, XA_ARC, 16, PropModePrePend,  
(unsigned char *) &a, 6);
```

X-I

```
XChangeProperty(dpy, fen, prop, t, format, mode, donnees, taille)
```

- o Format :
 - | Les données sont stockées, dans le serveur, de manière non structurée.
 - | Trois formats sont possibles : 8 bits, 16 bits et 32 bits.
 - | Les conventions pour les types prédéfinis doivent être respectées pour permettre une lecture conforme.
- o Il y a conversion de type obligatoire des données à unsigned char *.
- o La taille est le nombre d'éléments dans le tableau de données.
- o Le mode peut être
 - PropModePrePend
 - PropModeAppend
 - PropModeReplace
- o On ne peut pas mélanger les types ou formats dans une propriété: si l'on ajoute des données, elles doivent être du même type et format que celles déjà présentes.
- o Exemple : une fonction CopierTexte(dpy, texte) qui copie texte dans le "cut-buffer 0", s'écrit :

```
XChangeProperty(dpy, DefaultRootWindow(dpy), XA_CUT_BUFFER0,  
XA_STRING, PropModeReplace, texte, 1+strlen(texte));
```

X-I

Récupérer les valeurs

- o La fonction de récupération est complexe car
 - | elle permet d'extraire une partie seulement de la valeur
 - | elle rend aussi la valeur de ce qui reste
 - | elle permet de supprimer la propriété après lecture

```
int XGetWindowProperty(Display * dpy, Window fen,
    Atom prop, /* le nom de la propriete */
    long decalage, /* indice de debut de lecture */
    long longueur, /* nombre d'entites a lire */
    Bool supprimer_prop_apres_lecture,
    Atom type_souhaite,
    Atom* type_effectif,
    int* format_effectif,
    unsigned long* nb_octets_lus,
    unsigned long* nb_octets_restants,
    unsigned char** donnees_retournees);
```

X-I

```
XGetWindowProperty(dpy, fen, prop, decalage, longueur,  
supprimer?, type_souhaite,  
&quel_type, &quel_format, &lus, &restants, &donnees);
```

Exemple:

```
Atom quel_type;  
int quel_format, lus, restants;  
char * donnees;
```

```
XGetWindowProperty(dpy, fen, prop, 0, 8192, False, AnyPropertyType,  
&quel_type, &quel_format, &lus, &restants, &donnees);
```

- o **La fonction de récupération permet d'être vague sur le type : AnyPropertyType convient.**
- o **On récupère en fait une tranche d'au plus longueur*quel_format/8 octets.**
- o **La suppression de la propriété provoque un évènement PropertyNotify pour la fenêtre fen.**
- o **donnees est un pointeur vers les données retournées. Xlib alloue lui-même la place nécessaire + un octet contenant NULL. Cet octet n'est pas compté dans la taille du résultat.**
- o **Ne pas oublier de libérer la place (XFree) après utilisation.**

X-I

Exemple

- o Une fonction `CollerTexte(dpy, &texte)` qui récupère le texte dans le "cut-buffer0" a la forme :

```
CollerTexte(dpy, &texte)
{
    Atom quel_type;
    int quel_format, lus, restants;

    XGetWindowProperty(dpy,DefaultRootWindow(dpy),
        XA_CUT_BUFFER0,0,8192,False,AnyPropertyType,
        &quel_type, &quel_format, &lus, &restants, &texte);
}
```

X-I

Le mécanisme des sélections

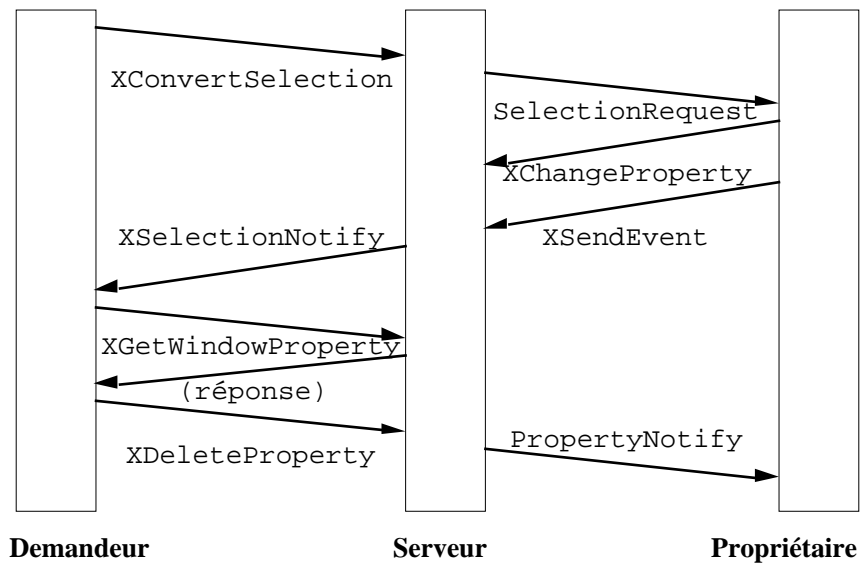
- Tout transfert entre clients doit passer par le serveur, car on ne peut supposer que les clients sont susceptibles de communiquer directement.
- Les *sélections* constituent le mécanisme principal recommandé par X pour l'échange de données entre clients.
- Aspects spécifiques:
 - | il n'y a pas de stockage intermédiaire dans un lieu "public" comme dans les "cut buffer", mais dans une propriété d'une fenêtre du demandeur (destinataire);
 - | le format des données transférées est précisé par le destinataire, et c'est l'expéditeur qui essaie de s'y conformer.
- La "sélection" :
 - | a un nom : un *atome* qui peut être prédéfini ou convenu, le plus souvent XA_PRIMARY.
 - | a un *propriétaire* : un client qui est aussi l'expéditeur, sur demande,
 - | est *attachée* à une fenêtre, fréquemment la fenêtre racine.
- La communication a lieu entre le *propriétaire* d'une donnée et un *demandeur* qui veut l'obtenir.

X-I

Principe du dialogue

- **Un client annonce (au serveur) qu’il détient des informations en se “rendant propriétaire de la sélection”:**
SetSelectionOwner[PRIMARY]
- **Le demandeur formule sa demande (au serveur), en indiquant où et comment (dans quelle propriété de quelle fenêtre, et sous quelle forme) il veut obtenir les informations:**
ConvertSelection[PRIMARY, où-et-comment]
- **Le propriétaire est informé de cette demande par le serveur; il essaie de faire le transfert :**
ChangeProperty[où-et-comment]
puis informe le demandeur (via le serveur) du succès ou de l’échec par
SendEvent[où]
- **Le demandeur, informé (via le serveur) du message du propriétaire, récupère ses données et détruit la propriété.**
- **Le propriétaire est informé de la destruction et sait ainsi que tout s’est bien passé...**

X-I

Etapes du dialogue

X-I

Le propriétaire

- o **Que le propriétaire se fasse connaître ! Il transmet au serveur des informations qui permettent de l'identifier :**
 - | **un atome de sélection** : l'atome `XA_PRIMARY` est recommandé (ICCCM!) pour des échanges entre clients qui ne se connaissent pas;
 - | **une fenêtre de rendez-vous** : cette fenêtre permet de vérifier que le serveur q bien enregistré la requête;
 - | **le moment** à partir duquel l'acte est à prendre en compte.
- o **La syntaxe est**

```
XSetSelectionOwner(Display* dpy,
    Atom atomS, /* atome de selection*/
    Window fenP, /* fenetre de rendez-vous */
    Time dateP); /* date de prise d'effet */
```
- o **Par exemple :**

```
XSetSelectionOwner(dpy, XA_PRIMARY, fenP, CurrentTime);
```
- o **Pour connaître le propriétaire de la sélection, on le demande au serveur :**

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

 - | **on reçoit le nom de la fenêtre de rendez-vous** (car les clients n'ont pas de "nom"!)
 - | **cette connaissance n'est pas requise pour l'échange; en revanche, cette primitive permet au propriétaire de vérifier** (par `fen==fenP`) que le serveur a enregistré la requête.

X-I**Changement de propriétaire**

- **Si un nouveau propriétaire (un autre client X) se déclare pour une sélection, l'ancien propriétaire perd ses droits (N.B. il y a un seule propriétaire par atome de sélection).**
- **L'ancien propriétaire reçoit un avis, sous forme d'un évènement SelectionClear (que l'on sélectionne par PropertyChangeMask).**
- **Si le même propriétaire change seulement de fenêtre de rendez-vous, il ne reçoit pas cet évènement. Il est donc utile, dans les applications, de mémoriser si l'on est toujours propriétaire.**

```
typedef struct {
    int type;
    unsigned long serial;
    Bool send_event;
    Display *display;
    Window window;
    Atom selection;
    Time time;
} XSelectionClearEvent;
```

X-I

Formuler la demande

- o **Que le demandeur formule sa demande ! Il indique où (dans quelle fenêtre, sous quelle propriété) et comment (dans quel type) il veut les obtenir :**

```
XConvertSelection(Display* dpy,  
    Atom  atomS,      /* atome de sélection */  
    Atom  typeD,      /* type cible (target) */  
    Atom  propD,      /* propriété du demandeur */  
    Window fenD,     /* fenêtre cible */  
    Time  dateD);    /* heure de la demande */
```

- | **typeD est l'atome dénotant le *type cible* (“target”) dans lequel le destinataire veut recevoir les données;**
 - | **propD est le nom (atome) de la propriété dans laquelle il convient de stocker les données;**
 - | **fenD est la fenêtre à laquelle est attachée la propriété;**
 - | **dateD est le moment à partir duquel la demande est valable.**
- o **Exemple :**

```
XConvertSelection(dpy, XA_PRIMARY, XA_STRING, propD, fenD,  
    CurrentTime);
```

X-I

Window owner	= fenP :	rendez-vous propriétaire
Window requestor	= fenD :	fenêtre du demandeur
Atom selection	= atomS :	atome de sélection
Atom target	= typeD :	type cible du demandeur
Atom property	= propD :	propriété du demandeur

Récapitulons

- o Les acteurs du mécanisme de sélection sont :
 - | l'atome de sélection : le plus souvent XA_PRIMARY
 - | la fenêtre du propriétaire : sert à identifier indirectement le propriétaire
 - | la fenêtre du demandeur : lieu ultime de dépôt des données
 - | la propriété du demandeur : nom sous lequel seront déposées les données
 - | le type cible : forme sous laquelle les données seront déposées
- o Dans l'échange de chaînes de caractères pour un copier-coller simple, on choisit :
 - | owner = requestor = DefaultRootWindow(dpy)
 - | selection = target = XA_PRIMARY
 - | target = XA_STRING

X-I**Le Serveur (ici facteur)**

- o **Le serveur reçoit la requête du demandeur, par exemple:**
`XConvertSelection(dpy, XA_PRIMARY, XA_STRING, propD, fenD, CurrentTime);`
- o **Il connaît le propriétaire, et lui envoie un évènement `SelectionRequest`.**
- o **En cas de problème (par exemple parce que le propriétaire a entre temps disparu), on passe à l'étape suivante.**
- o **Le propriétaire reçoit donc, s'il a choisi `PropertyChangeMask` sur une fenêtre, les informations requises pour**
 - | **transformer les données dans le type cible;**
 - | **les transférer dans la propriété demandée de la fenêtre demandée;**
 - | **d'informer le serveur du succès ou de l'échec**

```
typedef struct {
    int type;
    unsigned long serial;
    Bool send_event;
    Display *display;
    Window owner;
    Window requestor;
    Atom selection;
    Atom target;
    Atom property;
    Time time;
} XSelectionRequestEvent;
```

X-I

Au propriétaire de jouer

- o Le propriétaire essaie d'honorer la demande :
 - | en transformant éventuellement les données;
 - | en chargeant ces données dans la propriété `propD`, dans le format associé au type `typed`.
-

- o Exemple : on suppose `typed = XA_STRING`, donc le format est 8 :

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

X-I

- o **Le propriétaire informe le serveur (qui répercute l'information au demandeur) du résultat par un `SendEvent` d'un évènement `SelectionNotify`.**

```
typedef struct {
    int type;
    Bool send_event;
    Window requestor;
    Atom selection;
    Atom target;
    Atom property; /* ATOM or None */
    Time time;
} XSelectionEvent;
```

- o **Exemple**

```
XEvent sn;
sn.type = SelectionNotify;
sn.xselection.requestor = evmt.xselectionrequest.requestor;
sn.xselection.selection = evmt.xselectionrequest.selection;
sn.xselection.target = evmt.xselectionrequest.target;
sn.xselection.time = evmt.xselectionrequest.time;
```

si le propriétaire sait transformer, alors

```
sn.xselection.property = evmt.xselectionrequest.property;
```

sinon

```
sn.xselection.property = None;
```

Envoi par :

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

X-I

Retour au demandeur

- o Le demandeur est informé, par le serveur, de l'accomplissement de sa demande:
 - | il récupère ses données,
 - | libère la place,
 - | détruit éventuellement la propriété (ce qui informe le propriétaire).

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