Le système de fenêtres X Window

Xlib, Xt Intrinsics et Motif

Introduction

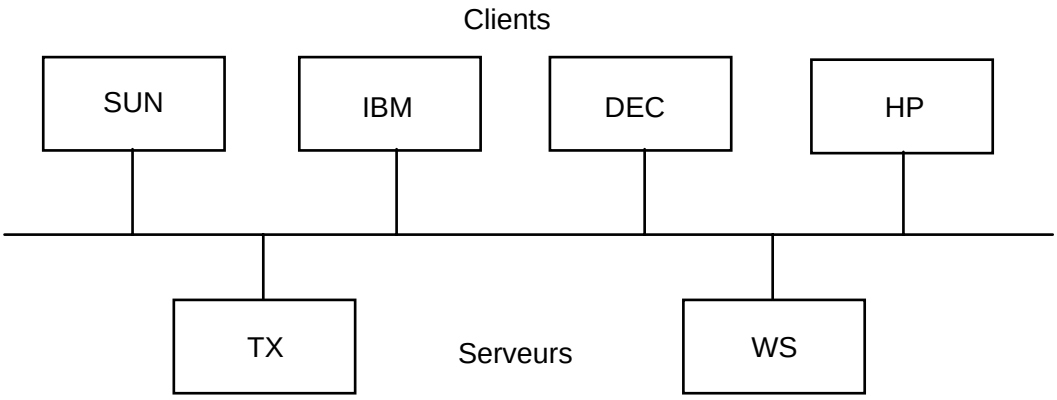
The X Window System, ou alors X Window, ou X tout court, mais surtout pas X-Windows.
X est le standard industriel pour les stations de travail. X a été développé au MIT avec l'aide financière de DEC et IBM (projet Athena). Le nom X, de mêmes que certaines notions de base de la conception dérivent des travaux effectués sur un système de fenêtres antérieur (dénommé W) et effectué à Stanford.
X10 .... X11R4, X11R5 et maintenant X11R6.
En 1987, une douzaine de constructeurs de stations de travail on adopté X comme standard pour leurs interfaces utilisateurs. Il existe aujourd'hui des terminaux spécifiques pour X, appelés terminaux X.
X se distingue d'autres systèmes (comme le Macintosh d'Apple, ou Windows de Microsoft) par le fait qu'il ne définit pas de style d'interface particulier. X est un ensemble de mécanismes indépendants des constructeurs, des systèmes d'exploitation, des styles d'interfaces.

1. Les buts du système X Window

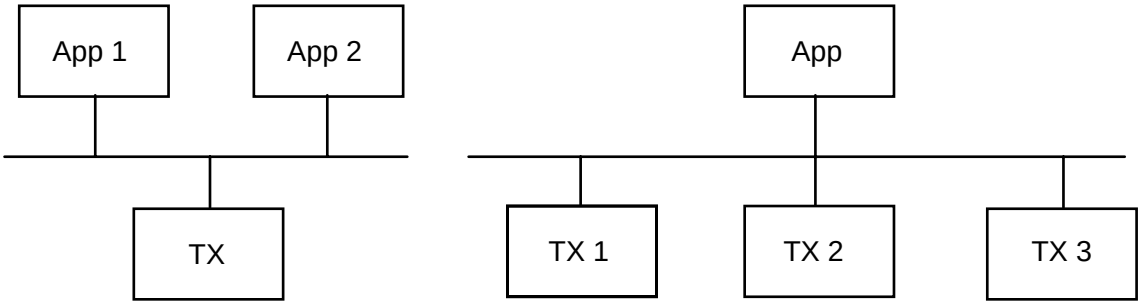
X offre un environnement pour les stations de travail, transparent du point de vue du réseau et totalement indépendant des constructeurs.

transparence réseau : une application qui s'exécute sur un CPU donné peut sortir des données sur n'importe quel écran, qu'il soit connecté au même CPU ou à un autre CPU.

Architecture Client / Serveur :



Grâce à la transparence réseau il est possible d'exécuter les applications sur le CPU le plus approprié (machine la moins chargée, binaire pour ce CPU, base de donnée centralisée, protection par machine, ...).
Il n'est pas nécessaire de modifier les applications pour utiliser un écran connecté à un autre CPU. Pour l'utilisateur, tout se passe comme si le terminal X était connecté à plusieurs ordinateurs simultanément. De même, une application X peut utiliser plusieurs écrans en même temps.



1.1. Indépendance et portabilité

Les applications X sont portables (en ce qui concerne l'interface), car elles ne se préoccupent pas des particularités (matérielles, langages et systèmes d'exploitation). Les écrans sont encapsulés par X. Les différences d'implémentation (ou de capacités) sont dissimulées par le protocole X. De ce fait, une application qui utilise le protocole X, et seulement le protocole X, peut utiliser n'importe quel terminal, de n'importe quel constructeur, sans recompilation (ni édition de liens). Bien entendu, il faut quand même compiler et linker la partie client X pour la combinaison particulière (CPU/OS) qui exécutera le programme. (Le protocole X ne rend pas les binaires compatibles entre eux !). X assure également la conversion entre les formats « big-endian » et « little-endian ».

1.2. Les dispositifs de sortie

X offre de nombreuses opérations pour manipuler (accéder) les écrans de type « bit-mapped », X n'est pas prévu pour les terminaux alpha-numériques. X permet de montrer, de manipuler et de capturer des images.

X organise les écrans en hiérarchie de fenêtres superposées. Chaque application utilise le nombre de fenêtres qui lui convient, elle peut les retailler, les bouger et les empiler les unes sur les autres.

X offre des mécanismes pour dessiner. Les opérations sont dites « immédiates ». Les stations de travail ne stockent pas les requêtes, elles les exécutent immédiatement. Les opérations de dessins sont orientées « bitmap ». Les applications adressent des pixels (adresses entières) au sein des fenêtres. Les applications ne sont pas concernées par la position des fenêtres sur l'écran.

X offre des mécanismes pour écrire du texte de haute qualité. Par exemple de l'émulation de terminal mais aussi du traitement de texte évolué.

X fonctionne avec une large gamme de terminaux graphiques. En couleur, en niveaux de gris et monochrome.

1.3. Les dispositifs d'entrée

X offre un ensemble de mécanismes pour la distribution des informations en provenance des utilisateurs vers les applications.

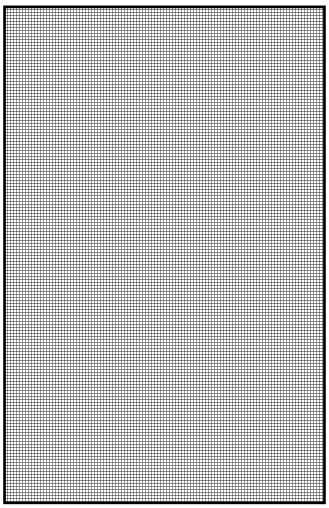
Les utilisateurs génèrent des informations d'entrée par l'intermédiaire des touches du clavier ou des boutons de la souris. Ces informations sont distribuées par le système X sous la forme d'événements.

De nombreux types d'événements permettent de déterminer les changements appliqués aux fenêtres et à la hiérarchie des fenêtres.

X supporte un grand nombre de claviers et de jeu de caractères.

1.3.1. La souris (tablette, boule, scanner, ...)

La souris est le dispositif d'entrée le plus populaire, il est utilisé à la fois pour désigner et pour sélectionner. L'utilisateur pointe une position sur l'écran en déplaçant une petite image dénommée « sprite » (mouse cursor), X utilise le terme de « pointer ». Les applications clientes peuvent changer la forme du sprite à volonté (exemple : pour indiquer une activité longue => la montre ou le sablier). Un point particulier (le « hotspot ») définit la position précise du sprite sur l'écran. Le sprite est dans une fenêtre lorsque le hotspot est dans cette fenêtre.

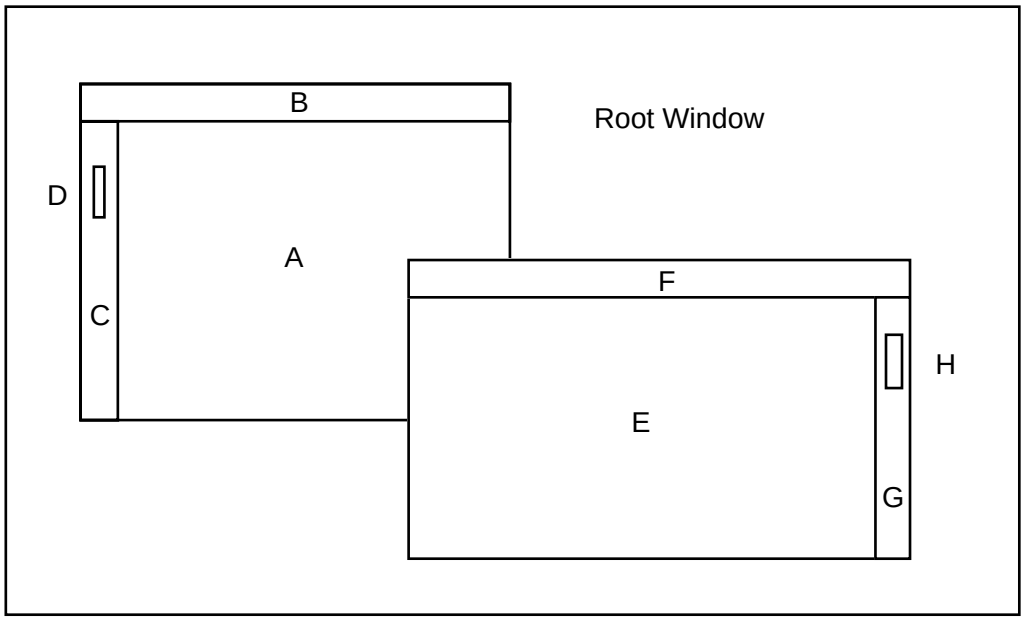


1.3.2. Le clavier

A chaque touche du clavier correspond un code d'identification. Le client peut traduire ce code en caractère ASCII s'il le désire. Le code est complètement indépendant des claviers ce qui rend les applications indépendantes des constructeurs.

1.4. Les fenêtres

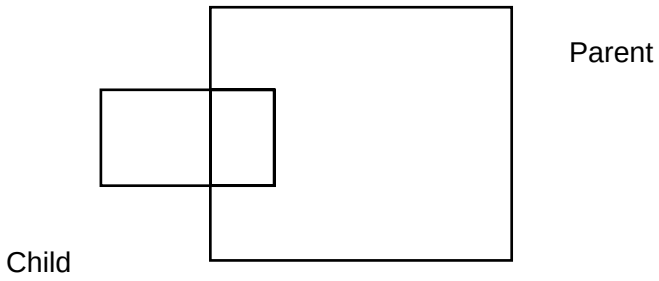
La fenêtre est le concept de base. Chaque fenêtre représente une zone rectangulaire. Contrairement à d'autres systèmes (Mac, Windows, Next, ...) une fenêtre X ne comporte aucune décoration (pas de barre de titre, d'ascenseur, ...). Une fenêtre X se présente comme un rectangle avec une couleur de fond ou un motif de fond, entourée d'une bordure.



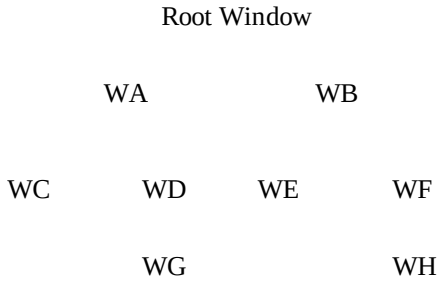
Le serveur X crée les fenêtres suite aux requêtes des clients. Le serveur encapsule toutes les informations utiles sur les fenêtres. Les applications clientes ne connaissent les fenêtres que par l'intermédiaire d'un identificateur. Les clients peuvent demander au serveur de modifier ou déplacer les fenêtres, sans pour autant être le propriétaire de la fenêtre. Une application A peut donc modifier une fenêtre créée par une application B. Cette fonctionnalité est utilisée par les gestionnaires de fenêtres par exemple.

1.4.1. La hiérarchie des fenêtres

X organise les fenêtres en hiérarchie (« window tree »). La fenêtre racine est dénommée la « root window ». Le serveur X créé une fenêtre racine pour chaque écran placé sous son contrôle. La root window occupe l'ensemble de l'écran, elle ne peut pas être redimensionnée ou déplacée. Chaque fenêtre (sauf la root window) a une fenêtre parente et peut avoir des fenêtres filles (sous-fenêtres). X impose peu de restriction sur la taille et la position des fenêtres, mais seule les parties de sous-fenêtres qui intersectent avec leur fenêtre parente sont visibles.

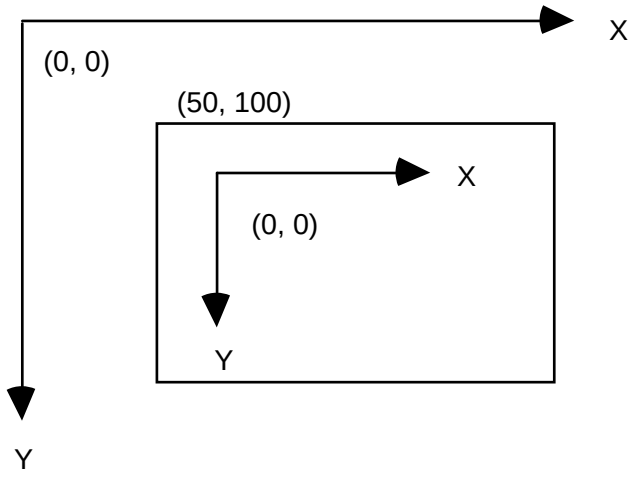


Il est possible d'empiler les fenêtres de même rang (« siblings »). La fenêtre du dessus cache alors les parties des fenêtres du dessous avec lesquelles elle intersecte. L'ordre d'empilement peut être altéré entre fenêtres de même rang. Du point de vue de l'utilisateur, les descendants d'une fenêtre font surface en même temps que la fenêtre parente.



1.4.2. Le système de coordonnées de X Window

Chaque fenêtre X possède son propre référentiel. Le point haut-gauche porte les coordonnées (0, 0). La position d'une fenêtre est toujours exprimée par rapport à la fenêtre parente.



Le référentiel de chaque fenêtre bouge avec la fenêtre, de sorte que les applications peuvent y placer des sous-fenêtres indépendamment de la position de la fenêtre parente.

1.4.3. Règles de visibilité des fenêtres

Les fenêtres ne sont pas nécessairement visible sur l'écran dès leur création. En effet, l'allocation des structures de données dans le serveur X, et l'invocation des procédures dépendant du matériel sont disjointes. Une fenêtre est rendue effectivement visible par l'opération « map ». En fait, une fenêtre - même « mappée » - peut ne pas être visible du fait des points suivants :

- ♣ la fenêtre est entièrement recouverte par une autre fenêtre
- ♦ un ancêtre de la fenêtre n'est pas mappé
- ♥ la fenêtre n'est pas disposée dans la zone visible d'un ancêtre.

1.4.4. Conservation du contenu des fenêtres

Comme les fenêtres peuvent être recouvertes en tout ou partie à tout moment, il faut être capable de préserver leur contenu afin de pouvoir le restaurer lorsque la fenêtre sera de nouveau visible. De nombreux systèmes de fenêtres enregistrent le contenu de l'écran sous la forme d'une bitmap ou « raster ». Cette approche demande une grande quantité de mémoire pour le serveur X dès lors que le nombre de fenêtres augmente.

Dans X, ce sont les applications (les clients) et non le serveur X, qui sont responsables pour la reconstitution du contenu des fenêtres. Chaque client doit donc être prêt, à tout moment, à redessiner le contenu des fenêtres qu'il utilise. Toutefois, certains serveurs X offrent la possibilité optionnelle d'enregistrer des rasters. Cette technique est connue sous l'appellation de « backing store », mais ne peut être considérée comme toujours disponible, et les clients doivent donc toujours être capables de reconstruire le contenu des fenêtres.

1.5. Les événements

Le serveur X communique avec ses clients par l'intermédiaire de messages asynchrones appelés événements. Les événements sont générés directement, ou indirectement, suite aux manipulations effectuées par les utilisateurs (frappe d'une touche du clavier, mouvement de la souris, click de bouton, ...). Le serveur génère également des événements pour informer les clients des changements relatifs aux fenêtres. Par exemple lorsque le contenu d'une fenêtre doit être restauré, le serveur génère un événement « expose ». X possède 33 types d'événements et offre un mécanisme qui permet aux clients de définir de nouveaux événements.

En plus des événements liés aux fenêtres, X permet de traiter des événements de type « chiens de garde » par la procédure `XtAddTimeOut (interval, proc, data)` , et également de s'abonner à des arrivées d'information sur des descripteurs de fichiers au moyen de la procédure `XtAddInput (source, condition, proc, client_data)`.

Le serveur envoie les événements dans une file associée à chaque client. Chaque événement regroupe le type de l'événement, l'identification de la fenêtre concernée et éventuellement des données spécifiques à chaque type d'événements.

La plupart des applications X sont contrôlées par les événements (« event-driven ») c'est-à-dire qu'elles sont conçues pour attendre des événements et les traiter séquentiellement. Ce modèle est particulièrement bien adapté pour les applications inter-actives, car il laisse le contrôle à l'usager.

Une application peut également envoyer un événement (à une autre application par exemple) au moyen de la procédure `XSendEvent (display, window, propagate, mask, &event)`.

Liste des événements de X :

```
ButtonPress, ButtonRelease, MotionNotify, MapNotify, EnterNotify, FocusIn,
LeaveNotify, FocusOut, Expose, GraphicsExpose, NoExpose, VisibilityNotify,
KeyPress, DestroyNotify, UnmapNotify, MapRequest, ReparentNotify, ResizeRequest,
ConfigureNotify, GravityNotify, CirculateNotify, CirculateRequest,
PropertyNotify, SelectionClear, KeymapNotify, SelectionNotify, ColormapNotify,
ClientMessage, MappingNotify, SelectionRequest, KeyRelease, ConfigureRequest,
CreateNotify
```

1.6. Le gestionnaire de fenêtres

Le gestionnaire de fenêtres (« window manager ») est une application particulière, qui permet à un utilisateur de contrôler la taille et la position des fenêtres sur l'écran. Un gestionnaire de fenêtre est un client comme un autre, mais il fait appel à certaines fonctionnalités du serveur X spécifiquement conçues pour la gestion des fenêtres. Par exemple, un GDF peut demander au serveur X de lui passer tous les événements liés à la structure des fenêtres. Si une application demande de « mapper » une fenêtre, et si un GDF à manifesté son intérêt pour ce type d'événement, le serveur X envoie un événement « MapRequest » au GDF, qui peut alors prendre toutes les actions qu'il jugera opportunes. Par exemple ajouter un cadre, un titre, des décorations ; ou encore réarranger les autres fenêtres (« tiling window manager »), mais aussi refuser de montrer la fenêtre.

La gestion des fenêtres est une tâche complexe qui affecte à la fois l'interaction entre les applications et l'usager et entre les applications elles-mêmes. Le documents ICCCM (Inter Client Communication Conventions Manual) définit le protocole que les GDF et les applications devraient suivre pour leurs interactions.

Les GDF les plus connus sont wm, uwm (Ultrix wm), awm (Ardent wm), twm (Tom's wm), rtl (tiled wm de Siemens Research Technology Laboratory), et surtout mwm (Motif wm).

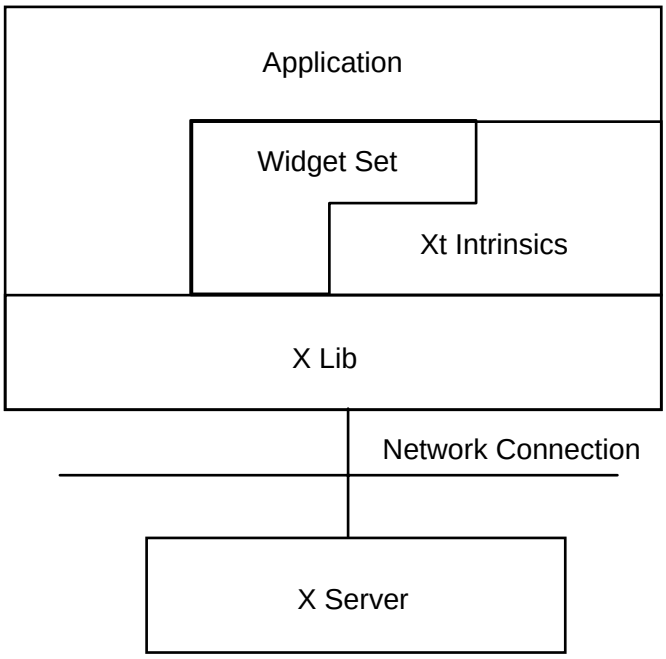
1.7. La programmation avec X Window

La définition formelle de X est donnée en termes du protocole de communication utilisé pour connecter les applications aux stations de travail. Ce protocole est décrit dans « X Window System Protocol, Version 11 » et est généralement disponible en machine dans la bande de distribution du M.I.T. sous « doc/Protocol/spec ». Ce protocole est difficile à lire (formel) et de fait, l'interface la plus populaire vers X est la « Xlib » écrite en C. Cette interface définit un ensemble de

fonctionnalités très complet pour accéder et contrôler les écrans, les fenêtres et les dispositifs d'entrée.

Nota : Xlib peut être utilisée en Ada par l'intermédiaire de « Bindings » (bibliothèque interfacée) ou par la Ada Xlib de Gary Barnes.

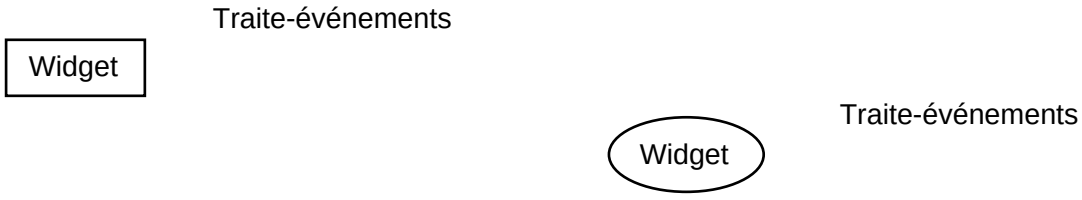
La programmation d'applications graphiques à partir de Xlib est difficile car le niveau d'abstraction de cette bibliothèque est très bas (« assembleur graphique »). Pour simplifier les développements des interfaces utilisateurs, des boîtes à outils on été construites à partir de la Xlib. Citons Interviews (Stanford), Andrew (Carnegie Mellon), Xray (HP), CLUE (Texas Instruments) et dans notre cas X Toolkit. X Toolkit se décompose en deux couches : Xt Intrisincs et les Widgets. En fait, Xt sert de base à de nombreux jeux de widgets aux fonctionnalités identiques mais à l'apparence différente. Motif définit un ensemble de widgets conçus pour être intégrés facilement avec Xt et Xlib de sorte que les clients peuvent (et souvent doivent) utiliser simultanément les trois niveaux de logiciel.



2. La programmation avec les boîtes à outils

Toutes les applications X sont normalement conduites par événements (event-driven). Toutefois, Xt étend ce modèle vers une programmation qui pourrait être qualifiée de *distributive* (« dispatch-driven »). Avec Xlib, les événements sont traités par une boucle qui contient une instruction à branchement multiple et qui effectue une action particulière suivant le type de l'événement. Cette boucle se complique notablement lorsque l'application contient de nombreuses fenêtres.

Afin de simplifier la boucle d'événements, les Intrinsics encapsulent le processus de distribution d'événements dans un mécanisme d'enregistrement de procédures de traitement d'événements, associés à chaque widget (« callbacks et event-handlers »).



Au lieu d'écrire une grande boucle qui contient toute la logique du programme, le programmeur se contente d'associer les traitements appropriés à chaque widgets. Cette approche simplifie l'écriture des programmes qui peuvent être rédigés dans un style plus déclaratif (Si A faire B).

2.1. Fonctionnalités de base de Xt

2.1.1. Initialisation des Intrinsics

Etablissement de la connexion avec le (ou les) serveurs X, et allocation des ressources.

```
XtInitialize (name, class, options, noptions, &argc, argv)
```

name : le nom du programme (« emacs »)

class : la catégorie du programme (« editor ») en fait « Emacs ».

La convention a évoluée et la classe est en fait le nom du programme, avec une majuscule. Le gestionnaire de ressource (défini plus loin) utilise ces deux informations pour déterminer les ressources utilisées par le programme.

XtInitialize renvoie un « TopLevelShell ».

2.1.2. Création des widgets

Les widgets sont des objets graphiques (construits à la main) qui regroupent des structures de données complexes avec les opérations applicables sur ces données.

Les widgets, à l'image des fenêtres X, sont regroupés dans une hiérarchie dénommée « WidgetTree ». La racine de chaque WidgetTree est toujours un « Shell Widget ». Les shells ont un et un seul enfant, ils servent d'enveloppe (de coquille) pour le widget enfant, et offrent l'interface avec le

GDF. La fenêtre X associée a un shell widget est fille de la root window. Les applications qui possèdent plusieurs fenêtres indépendantes doivent créer plusieurs TopLevelShells.

2.1.3. "Realization" des widgets

A sa création, le widget ne possède pas de fenêtre X associée. L'opération de « realization » consiste à créer l'ensemble des fenêtres X associées à un widget. `XtRealizeWidget (widget)` réalise également les widgets enfants du widget spécifié. Normalement, cette opération n'est appliquée qu'une fois, directement sur le TopLevelShell.

2.1.4. Enregistrements des " traites-événements " et des " callbacks "

2.1.4.1. Les traites-événements

Un traite-événement est une procédure invoquée par les Intrinsics lorsqu'un type particulier d'événement se produit au sein d'un widget. Le programmeur de widget peut traiter tout ou partie des événements X. Le programmeur d'application (le client du programmeur précédent) peut enregistrer des traites-événements additionnels à l'aide de la procédure

```
XtAddEventHandler (widget, eventmask, nonmaskable, handler, client_data)
```

La forme générale d'un traite-événement est comme suit :

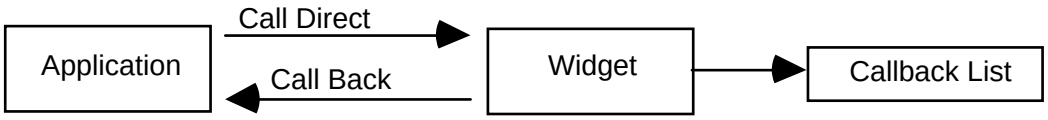
```
void handler (w, client_data, event)
      Widget      w;
      caddr_t     client_data;
      XEvent      *event;

caddr_t     =>    untyped pointer

ex: typedef caddr_t     char *
```

2.1.4.2. Les Callbacks

Les widgets peuvent également définir des points d'entrée, ou crochets, que les applications peuvent utiliser pour spécifier le traitement devant être effectué lors de l'occurrence d'une condition donnée (matérialisée par un événement). Ce mécanisme est connu sous l'appellation de « callback », car l'interface (le widget) effectue un appel (rappel) de procédure vers l'application, c'est-à-dire vers du code non connu lors de l'écriture du widget. Chaque widget possède une liste de callbacks pour un type d'action donné.



Exemple : tout widget possède `XtNdestroyCallback`

pour un bouton `XmNactivateCallback, XmNarmCallback, XmNdisarmCallback`

2.1.4.3. Différence entre un callback et un traite-événement

Le callback est plus abstrait, il n'est pas attaché à un événement mais à une action. D'autre part, l'utilisateur peut personnaliser l'action qui déclenche le callback grâce au « translation manager ».

événement => tripes du widget callback => interface du widget

2.1.5. La distribution des événements

La programmation par événements à partir de Xlib seule est fastidieuse car toute la logique du programme est concentrée dans la boucle principale. Xt simplifie considérablement la boucle principale, qui se résume en l'appel de la procédure `XtMainLoop ()`. La boucle principale contient le code suivant :

```
While (TRUE)
{
      XEvent event;
      XtNextEvent (& event);
      XtDispatchEvent (& event);
}
```

Chaque application possède une file d'événements remplie par le serveur X. `XtNextEvent` récupère un événement, puis `XtDispatchEvent` part à la recherche du traite événement correspondant.

Il est possible d'insérer un traitement supplémentaire dans la boucle d'événements, lorsque l'application désire prendre la main de temps en temps, par exemple pour effectuer des tâches non interactives. La fonction non bloquante `XtPending()` permet de savoir si la file d'événements est vide.

```
While (TRUE)
{
      if (XtPending ())
      {
            XtNextEvent (& event);
            XtDispatchEvent (& event);
      } else
            .....
      }
}
```

2.1.6. Le "Translation Manager"

Le Translation Manager de Xt permet à l'utilisateur de spécifier (sans reprogrammation de l'application) les associations entre les actions de l'usager et les opérations à déclencher. Voici par exemple, comment modifier l'action d'un bouton poussoir dans l'application « memo ».

Dans le fichier `.Xdefaults` :

```
memo*XmPushButton.translations:<key>q:ArmAndActivate()
```

De façon générale, une table de translation contient une liste d'expressions qui possèdent chacune une partie gauche et une partie droite, séparées par un double point. La partie de gauche spécifie l'action de l'usager et la partie de droite la procédure à exécuter.

```
*translations:<Btn1Down>, <Btn1Up>:ArmAndActivate()
*translations:Ctlr<Btn1Down>:ArmAndActivate()
```

Les applications peuvent ajouter leurs propres action en utilisant la procédure :

```
XtAddActions (actions, num_actions)
```

3. Le gestionnaire de ressources de X

Les ressources sont toutes les informations personnalisables par l'usager. Par convention, les applications X devraient permettre la personnalisation de la plupart des données utilisées, tout en offrant des valeurs par défaut raisonnables. En fait, l'utilisateur peut personnaliser les ressources s'il le désire, mais il n'est pas obligé de le faire. Notons que le programmeur doit faire un effort pour rendre l'application personnalisable, ce n'est pas complètement automatique.

3.1. Spécification des ressources

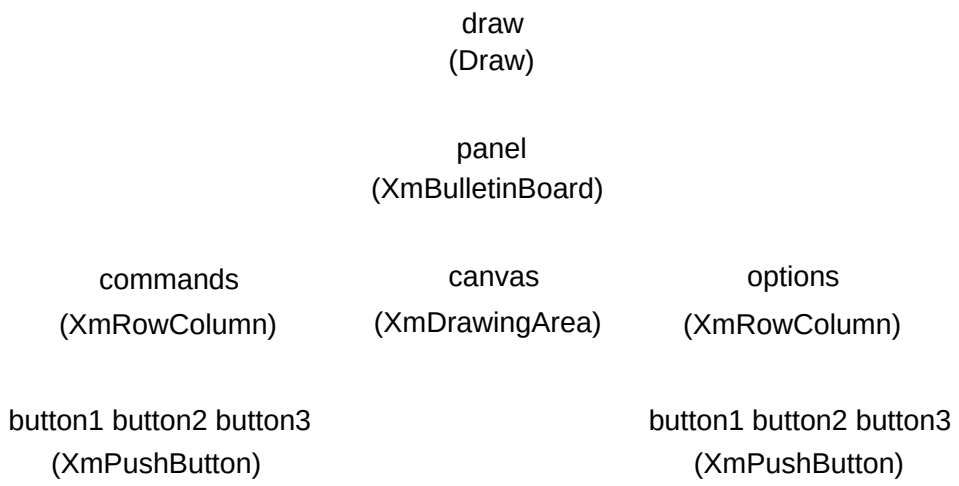
X maintient une base de données qui contient à la fois les préférences des utilisateurs et les valeurs par défaut. Les applications peuvent retrouver la valeur d'une ressource, durant leur exécution, en questionnant le gestionnaire de ressources. Le GDR se distingue des SGBD traditionnels, car il contient des informations générales sur les ressources, du genre « Tous les boutons sont rouges » ou « Tous les champs de saisie de texte sont long de 80 caractères ». Par contre les requêtes sont précises : « Couleur du bouton Quit de l'application mail ? ».

3.1.1. Noms et classes des ressources

Chaque ressource possède un nom et une classe. Par exemple : « emacs » et « Editor », « destroyCallback » et « Callback ». Les noms et les classes des ressources sont des chaînes de caractères, les valeurs les plus utilisées sont définies dans le fichier « StringDefs.h ». Il est possible d'ajouter de nouveaux noms et de nouvelles classes de ressources.

Par convention, le nom des ressources commence par une lettre minuscule, alors que le nom des classes commence par une majuscule. Fréquemment, les noms de ressource et de classe sont identiques, sauf pour la première lettre capitale. La nom de classe est fixée par le programmeur du widget, le nom de ressource (en fait de widget) est déterminé par le programmeur de l'application.

Une application récupère la valeur d'une ressource en spécifiant précisément la ressource désirée, sous la forme d'un chemin d'accès au travers de l'arbre de widgets (séparés par des points).



nom de ressource (un widget particulier) :

```
draw.panel.commands.button1.foreground: orangeRed
```

nom de classe (un ensemble de widgets) :

```
Draw.XmBulletinBoard.XmRowColumn.XmPushButton.Foreground: red
```

3.1.2. Algorithme de substitution de noms

La spécification complète des chemins d'accès est fastidieuse, pour cette raison le RM de X définit le caractère spécial `*` qui peut représenter un ou plusieurs noms de ressources ou de classes. Exemple :

```
draw*foreground: darkSlate
```

```
*XmPushButton.foreground: mediumAquamarine
```

Notons que l'on peut mélanger noms de ressources et noms de classes.

3.1.2.1. Règles de précédence

1_ Chaque nom de ressource ou de classe dans le chemin doit correspondre à une entrée dans la base de données.

2_ Le "`.`" est prioritaire par rapport au « `*` » car il est plus spécifique.

```
*commands.Background: green   *commands*Background: red
```

3_ Le nom de ressource est prioritaire par rapport au nom de classe.

4_ Les noms de ressource et de classe sont prioritaires par rapport au caractère de substitution « `*` ».

5_ Le RM compare les entrées de la gauche vers la droite, et les premiers éléments sont prioritaires par rapport aux suivants.

```
draw*XmBulletinBoard*foreground: green
```

est prioritaire par rapport à :

```
draw*XmPushButton*foreground: red
```

Conclusion : c'est un peu difficile lorsque le nombre de ressources est élevé ! Cela peut aussi prendre du temps lors du démarrage de l'application.

3.1.3. Chargement des ressources

Les ressources sont chargées lors du démarrage de l'application, dans l'ordre suivant :

```
1_    /usr/lib/X11/$LANGapp-defaults/<class>
2_    /usr/lib/X11/app-defaults/<class>  // si 1 échoue
3_    $XAPPLRESLANGPATH<class>
4_    $XAPPLRESDIR<class>          // si 3 échoue
5_    RESOURCE_MANAGER property
6_    $HOME/.Xdefaults             // si 5 n'existe pas
7_    $XENVIRONMENT
8_    $HOME/.Xdefaults-<hosts>     // si 7 n'existe pas
9_    la ligne de commande ...
```

3.1.4. Ressources et conventions

Les ressources définies par le programmeur sont prioritaires par rapport aux ressources définies par l'utilisateur. Parfois, il est difficile de savoir qui devrait positionner une ressource ; une bonne règle est de ne fixer que les ressources nécessaires à la bonne marche de l'application.

Exemple : ne pas permettre de transformer le label du bouton « delete » en « save ». Hélas, ce n'est pas si simple, car dans ce cas on ne peut plus personnaliser la langue. Une bonne approche consiste à livrer l'application avec un fichier de ressources que l'utilisateur peut modifier à sa guise.

Plus une application est personnalisable, plus elle est à même de satisfaire les goûts et les couleurs de l'utilisateur? Toutefois, il ne faut pas tomber dans le travers de forcer l'utilisateur à personnaliser. Par défaut, l'application doit comporter un assortiment de ressources raisonnablement positionnées.

4. Les couleurs

X propose un modèle qui dissimule la diversité des matériels d'affichage derrière une interface commune. Cette approche permet de construire des applications versatiles, qui fonctionnent de façon satisfaisantes sur des écrans noirs et blancs, en niveaux de gris, ou couleur.

X est conçu pour les environnements multitâches, c'est-à-dire des systèmes qui demandent le partage des écrans entre plusieurs applications. Cette contrainte complique considérablement le modèle des couleurs, car un écran donné ne peut afficher qu'un nombre limité de couleurs simultanément, et les différentes applications devront donc se partager les couleurs disponibles à un moment donné.

Une approche simple consiste à limiter le nombre total de couleurs disponibles. C'est la solution retenue sur des ordinateurs rudimentaires (choix entre 8 ou 16 couleurs prédéfinies).

Cette solution n'est pas satisfaisante pour construire un environnement graphique de qualité, c'est pourquoi X se distingue en proposant une technique qui permet à chaque application d'allouer autant de couleurs que nécessaire, tout en encourageant le partage de couleurs entre applications.

4.1. Les tables de couleurs (colormaps)

X utilise des « colormaps », c'est-à-dire des tableaux qui contiennent des couleurs. Les applications accèdent aux couleurs par l'intermédiaire d'un index dans la colormap. La colormap introduit une indirection entre l'index utlisé par les applications et la couleur affichée sur l'écran. Les écrans possèdent une profondeur, exprimée en nombre de bits par pixel (on parle également de plans d'écran). Un écran monochrome possède une profondeur de 1 bit. Le nombre de couleurs qu'un écran peut afficher simultanément est donnée par la relation couleurs = 2 plans.

Différentes applications peuvent partager la même table des couleurs. Par défaut, les fenêtres héritent de la table de leur parent et la plupart des applications peuvent utiliser la table de la root window. Cette colormap est appelée « default colormap ». Le serveur y alloue deux couleurs, le blanc et le noir ; les autres couleurs y seront rajoutées par les applications qui le désirent. Une application qui nécessite un grand nombre de couleurs peut créer une nouvelle colormap, différente de la colormap par défaut. La fonction

```
XInstallColormap (display, map)
```

permet de positionner la colormap par défaut, et donc d'en changer selon les besoins. Toutefois, lorsque seulement une application a besoin d'une colormap particulière, la fonction

```
XSetWindowColormap (display, window, cmap)
```

permet d'associer une colormap avec une fenêtre (et à ses enfants par héritage).

5. Les images

En plus des images visibles à l'écran, X offre des trois catégories d'images non visibles : les « pixmaps », les « bitmaps » et les « Ximages ».

5.1. Les pixmaps

Une pixmap est une zone de mémoire similaire à une région rectangulaire de l'écran, mais sans correspondance visible. Les pixmaps ont une profondeur exprimée en bits. La fonction

XCreatePixmap (display, drawable, width, height, depth)

créé une pixmap associée avec le même écran que le drawable spécifié. Les pixmaps servent principalement pour sauvegarder le contenu de fenêtres. Les pixmaps peuvent également être utilisées pour construire des images de fond à partir de petits motifs élémentaires (tiles de 16x16).

5.2. Les bitmaps

Les bitmaps sont des pixmaps avec une profondeur de 1. Elles peuvent être créées par la fonction

```
XCreateBitmapFromData (display, drawable, data, width, height)
```

Les données étant des séries de bits qui représentent la valeur de chaque pixel. L'éditeur `bitmap`, livré avec X, permet de construire des fichiers bitmaps directement exploitable par la fonction précédente.

5.3. Les Ximages

X offre un mécanisme de transfert d'images entre applications et le serveur X, par l'intermédiaire d'une structure de données `Ximage`. Cette structure sauvegarde les informations dans une représentation canonique, indépendante des particularités de chaque machine. L'allocation d'une Ximage s'effectue par la fonction

```
XCreateImage (display, visual, depth, format, offset, data, width, height, bitmap_pad, bytes_per_line)
```

les images peuvent également être extraites d'un drawable par la fonction

```
XGetImage (display, drawable, x, y, width, height, plane_mask, format)
```

5.4. Copie entre drawables

Le contenu d'un drawable (une fenêtre, une pixmap) peut être copié vers n'importe quel autre drawable de la même profondeur, par la fonction

```
XCopyArea (display, src, dest, gc, src_x, src_y, width, height, dest_x, dest_y)
```

La fonction suivante permet de copier entre drawable de profondeurs différentes

```
XCopyPlane (display, src, dest, gc, src_x, src_y, width, height, dest_x, dest_y, plane)
```

Le contexte graphique détermine l’avant-plan et l’arrière-plan du drawable destination.

6. Les contextes graphiques et les régions

Les applications graphiques nécessitent des informations de description comme la largeur des lignes, les couleurs d'avant-plan et d'arrière-plan, les fontes, les formes de remplissage, ...

6.1 Les contextes graphiques

Tous les attributs graphiques sont stockés par X dans une structure de données interne appelée « graphic context ». Une application peut créer puis manipuler plusieurs contextes graphiques.

```
XCreateGC (display, drawable, mask, &values)

XtGetGC (widget, mask, values)     // prévue pour le partage de
                  // GC entre applis
```

`values` est de type `XGCValues`, une structure qui contient une vingtaine de membres.

`mask` indique quel(s) membres contiennent une information utile. Un masque nul entraîne la création d'un GC avec des valeurs par défaut.

Les contextes graphiques sont alors passés en paramètres à toutes les opérations qui dessinent ou qui écrivent.

ex : `XDrawString (display, drawable, gc, x, y, str, length)`

6.2. Les régions

Les régions permettent de représenter et de manipuler des zones non-rectangulaires. Les régions sont surtout utilisées lors des opérations de rafraîchissement de l'écran, par exemple pour déterminer quelles parties d'une image doivent être reconstruites. Une région encapsule des données qui ne peuvent être manipulées que par l'intermédiaire d'opérations prédéfinies. Ces opérations s'exécutent localement dans le client, sans faire de requêtes au serveur X. Xlib fournit des primitives pour créer, comparer et détruire les régions.

Les régions sont utilisées par certains widgets qui exigent que les Intrinsics compressent les événements de type `expose`.

7. Le texte et les fontes

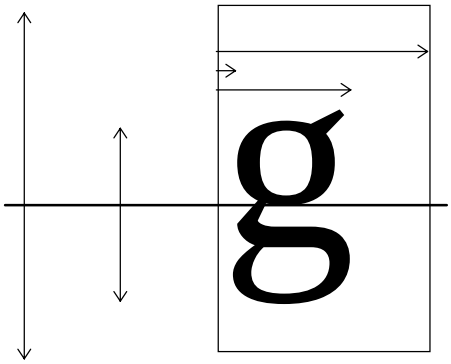
Xlib offre un ensemble de primitives pour dessiner du texte dans des fenêtres ou des pixmaps (zone de mémoire identique à une fenêtre, mais non visible à l'écran). Motif rajoute une abstraction dénommée « compound string » pour représenter du texte indépendamment des modes de représentation.

7.1. Les fontes

Une fonte est une collection de bitmaps rectangulaires, dénommées « glyphs ». Un glyph peut contenir n'importe quelle image, de sorte que certaines fontes ne contiennent pas des caractères

alphabétiques. Les fontes doivent être chargées pour être disponibles (une fonte consomme des ressources). La fonction `XLoadFont (display, font_name)` charge la fonte et retourne un identificateur de fonte. Chaque fonte est décrite par une structure de données `XFontStruct` qui peut être récupérée par la fonction `XQueryFont (display, font_ID)`. Cette structure contient :
l'identificateur de la fonte, la direction (gauche ou droite), l'encombrement du plus petit caractère, l'encombrement du plus gros caractère, la hauteur par rapport à la ligne de base, la profondeur par rapport à la ligne de base, et un tableau de `XCharStruct` qui décrit chaque caractère en détail. Cette structure contient entre autres les informations suivantes :

```
short lbearing;
short rbearing;
short width;
short ascent;
short descent;
```



Des fonctions de Xlib (comme XTextWidth ou XTextExtents) permettent d'obtenir des informations sur l'encombrement global de chaînes de caractères.

7.2. L'écriture du texte

X affiche le texte dans un « drawable » au moyen des fonctions suivantes :

```
XDrawString (display, drawable, gc, x, y, str, length) //foreground only
```

et

```
XDrawImageString (display, drawable, gc, x, y, str, length)
```

7.3. Les chaînes composites

Motif offre un mécanisme pour la représentation du texte de manière plus abstraite que la chaîne de caractère. Les chaînes composites séparent le contenu de la forme. Une chaîne composite est constituée de trois parties :

- ♥ le jeu de caractère (le nom de la fonte)
- ♠ la direction (par défaut de gauche à droite)

- le texte lui même.

Motif fournit de nombreuses fonctions pour manipuler les chaînes composites, comme :

```
XmStringCreate, XmStringLtoRCreate, XmStringFree, XmStringSeparatorCreate,
XmStringCompare, XmStringEmpty, XmStringConcat, XmStringLength
```

ou pour les afficher :

```
XmStringDraw et XmStringDrawImage.
```

8. La structure des widgets

Les Intrinsics définissent l'architecture de base des widgets. Le but est de faciliter l'intégration de widgets de différentes provenances au sein de la même application.

Les widgets sont organisés en classes, elles mêmes organisées en hiérarchie. Chaque widgets comporte deux composants : les informations de classe (« class record ») et les informations d'instance (« instance record »). Les Intrinsics sont codés en langage C de sorte que l'héritage doit être simulé à la main. Chaque widget est réalisé sous la forme d'une structure qui contient des données et des pointeurs vers les opérations. Les widgets d'une même classe partagent les données et les opérations de leur classe, et possèdent leur propre copie des données spécifiques à chaque instance.

La description de classe est allouée et initialisée statiquement, la partie propre aux instances est allouée à la demande, lors de l'exécution.

8.1. Les informations de classe

Les informations de classe (class record) regroupent les données et les opérations qui sont communes à tous les widgets (les objets) de la classe. Il s'agit essentiellement de données statiques comme par exemple le nom de la classe, l'apparence ou le comportement en général.

Tous les widgets héritent de la class `Core` qui définit entre autres les informations de classe suivantes :

le nom de la superclasse, le nom de la classe, la taille du widget, les actions, les ressources, la version des Intrinsics, ...

ainsi que des méthodes pour l'initialisation, la réalisation, le re-dimensionnement, la destruction, le rafraîchissement, ...

8.2. Les informations d'instances

Chaque widget contient des informations sur son état. Les widgets, qui héritent tous de la classe `Core`, contiennent entre autres les informations d'instances suivantes :

un pointeur vers self, un pointeur vers la classe, un pointeur vers le parent (celui qui le contient), un nom, l'identificateur de la fenêtre X associée, la position, la taille, ...

8.3. Encapsulation des données

Les applications qui utilisent les widgets ne voient pas leur implémentation. Les Intrinsics définissent une double définition pour chaque widget, une vision privée pour le constructeur du widget, et une vision public pour l'utilisateur. L'occultation d'information est réalisée en organisant l'implémentation du widget en différents fichiers. Chaque widgets est construit à partir de un ou plusieurs fichiers d'entête privés (« private header files »), un fichier d'entête public (« public header file »), et un ou plusieurs fichiers de source C. Le fichier d'entête privé contient toutes les définitions nécessaires pour la construction du widget, alors que le fichier public ne contient que les définitions nécessaires pour l'utilisation du widget.

8.4. Les widgets de Motif

```
Core..XmPrimitive..          XmArrowButton
                .            XmLabel....... XmCascadeButton
                .            .              XmDrawnButton
                .            .              XmPushButton
                .            .              XmToggleButton
                .            XmList
                .            XmSash
                .            XmScrollBar
                .            XmSeparator
                .            XmText
                .
                Composite.... Shell....OverrideShell...XmMenuShell
                              .        TransientShell..XmDialogShell
                              .
                              Constraint... XmManager...XmBulletinBoard.XmForm
                                                        .               XmMessageBox
                                                        .               XmSelectionBox
                                                        .               XmCommand
                                                        .               XmFileSelectionBox
                                                        XmDrawingArea
                                                        XmFrame
                                                        XmRowColumn
                                                        XmScrolledWindow.... XmMainWindow
                                                        XmScale
                                                        XmPanedWindow
```

Conclusion

X est un bon exemple de système de fenêtres, mais il devient de plus en plus complexe.

X est ouvert, extensible.

Par X on entend plutôt Xlib, Xt Intrinsics, Motif, ...

La programmation reste difficile, il faut s'aider de constructeurs d'interfaces.