État de l'art associé au projet tuteuré

Développement d'une interface graphique pour TacOS

Nicolas Hug <hug@etud.insa-toulouse.fr>
Julien Marchand <jmarchan@etud.insa-toulouse.fr>
Benoît Morgan <morgan@etud.insa-toulouse.fr>
Adrien Sulpice <sulpice@etud.insa-toulouse.fr>

4ème année Informatique

Résumé du projet

Il y a deux ans, dans le cadre des projets tutorés, un système d'exploitation 32 bits a été développé par des étudiants : TacOS. Nous souhaitions à notre tour contribuer à ce projet, et le développement d'une interface graphique nous a semblé intéressant. En effet, cet axe de développement permet d'adopter à la fois une approche bas niveau (conception de drivers, gestion des mecanismes propres au système d'exploitation), et une approche haut niveau (gestion des fenêtres, élaboration d'une API, etc.).

Cette étude bibliographique nous permettra de découvrir la structure et le fonctionnement de quelques interfaces graphiques existantes, afin de concevoir notre propre solution adaptée à TacOS. Dans une première partie, nous présenterons diverses généralités sur les interfaces graphiques, puis dans la seconde nous détaillerons l'architecture des systèmes existants, en mettant l'accent sur la solution libre la plus répandue : X Window System.

Table des matières

Chapitre 1

Généralités sur les interfaces graphiques

Dans cette partie, nous présenterons les concepts fondamentaux relatifs aux interfaces graphiques. Nous commencerons par une définition et un historique de l'évolution des interfaces graphiques des années 1960 à aujourd'hui. Nous détaillerons ensuite le paradigme WIMP, puis nous présenterons les différents composants d'une interface graphique. Enfin, nous étudierons les mécanismes permettant à l'utilisateur d'interagir avec l'interface.

1.1 Historique

1.1.1 Définitions

Un environnement graphique est "un ensemble de programmes qui fournit un cadre de travail simplifiant la manipulation des appareils informatiques à l'aide d'interfaces graphiques" [wik12].

Une interface graphique se définit quant à elle un comme "dispositif de dialogue homme-machine, dans lequel les objets à manipuler sont dessinés sous forme de pictogrammes à l'écran, que l'usager peut opérer en imitant la manipulation physique de ces objets avec un dispositif de pointage, le plus souvent une souris". Les interfaces graphiques sont communément appelées GUI, pour Graphical User Interface.

1.1.2 Événements majeurs

Le **SketchPad** (figure 1.1) est la première solution matérielle et logicielle graphique interactive. Présenté en 1963 et développé au sein du MIT, ce logiciel était destiné à dessiner des schémas techniques, en deux dimensions, sur écran à l'aide d'un stylo optique. Ce logiciel est considéré comme l'ancêtre des logiciels de conception assistée par ordinateur (CAO). C'est grâce à cette innovation que son concepteur, Ivan Sutherland, obtient en 1988 le prestigieux prix Turing.

FIGURE 1.1 – Démonstration du SketchPad [Ét06]

Douglas Carl Engelbart, chercheur à la *Stanford Research Institude*, invente la souris en 1968. C'est cette même année qu'il présente le premier environnement graphique fenêtré manipulable à l'aide d'une souris (figure 1.2). Sur ce système est notamment développé un logiciel de travail collaboratif, de traitement de texte mais aussi un des premiers systèmes hypertexte. Il est à noter que Douglas C. Engelbart est également un des pionniers de la recherche sur l'hypertexte.

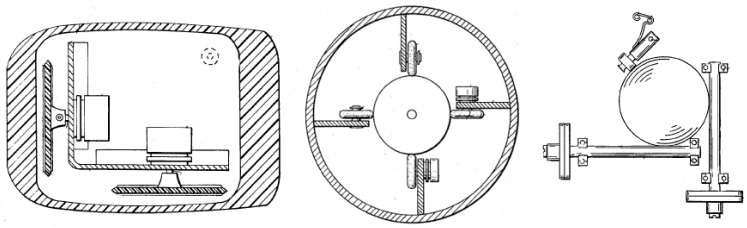

FIGURE 1.2 – Schéma technique de la première souris [Jer05]

Le **Xerox Alto** , conçu en 1973 au Xerox PARC (Palo Alto Research Center), est cité comme le premier ordinateur personnel même si d'autres machines disputent également ce titre. Il est en outre le premier à définir le concept de *bureau* que nous connaissons bien aujourd'hui (figure 1.3). Cet ordinateur n'est pourtant qu'un prototype n'ayant pas de but commercial. Son concepteur principal, Charles P. (Chuck) Thacker, a lui aussi reçu en 2009 le prix Turing pour ces travaux sur ce projet.

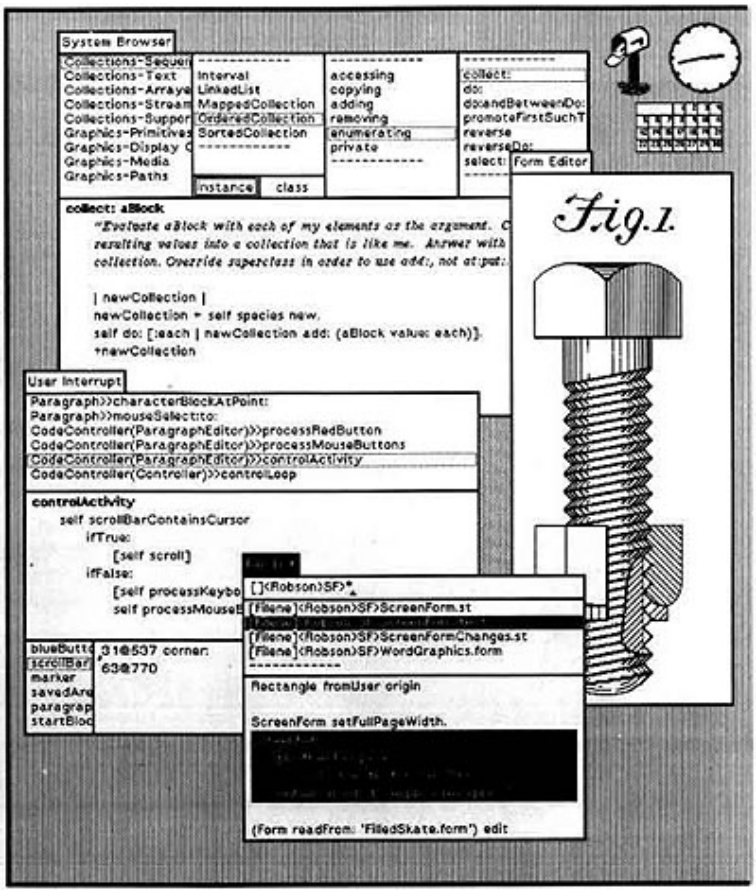

FIGURE 1.3 – Premier bureau [Mus98]

Dick Shoup en Avril 1973, propose une machine dotée d'une carte graphique couleur capable d'afficher des images d'une résolution de 640 pixels de largeur par 480 de hauteur le tout avec une profondeur de 256 couleurs mais aussi d'acquérir un signal vidéo. De plus il développe un logiciel de dessin et d'effet vidéos, **Superpaint**.

C'est en Avril 1981 que Xerox sort un produit étonnamment en avance sur son temps, le **Xerox Star** (figure 1.4). Cette machine reprend et améliore tous les concepts mis en place par son grand frère, le Xerox Alto au Xerox PARC. Elle innove sur trois grands axes des interfaces graphiques. Tout d'abord, elle utilise au maximum le *drag & drop* (*glisser/déposer* en français). De plus, elle utilise pour la première fois le concept de presse papier et de copier coller : on peut, simplement grâce à l'interface graphique, recopier du texte présent dans la mémoire et l'interface d'un programme vers un autre programme. Enfin, elle marque l'apparition de menus contextuels à la manière du plus célèbre Macintosh, qui, selon le programme ayant le focus, changent de contenu. Cette machine, trop en avance sur son temps et trop onéreuse (17000$ USD), n'aura aucun succès commercial.

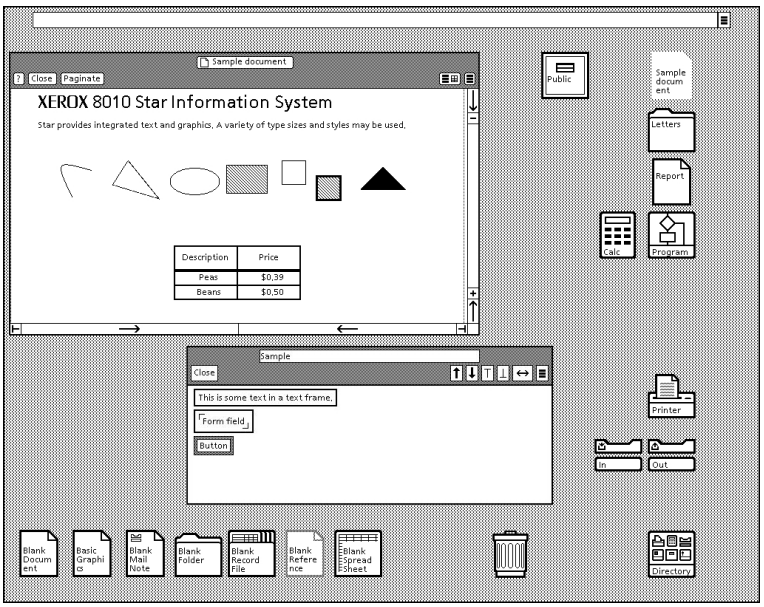

FIGURE 1.4 – Environnement de bureau du Xerox Star [Wic03]

En 1984, le MIT commence à développer un serveur graphique pour les machines Unix : X Window System. Ce système n'est pas une simple interface graphique, c'est également et surtout un serveur évolué capable de gérer plusieurs écrans locaux ou distants. Le système X Window sera détaillé dans la section 2.2.1.

1.2 WIMP

1.2.1 Introduction

WIMP est un paradigme informatique dont l'acronyme signifie **W**indows, **I**cons, **M**enus, **P**ointer. WIMP est une manière de concevoir les interfaces graphiques développée par Xerox et implémentée avec le Xerox Alto par des ingénieurs du PARC (cf section 1.1.2). Ce concept a ensuite été popularisé avec la sortie du **Macintosh** qui améliora la gestion des fenêtres et ajouta le concept de barres de menus. Les systèmes utilisant ces concepts sont aussi appelés **WIMPS**.

1.2.2 Structure

Une fenêtre (**W**indow) ne représente qu'un seul processus encapsulé et isolé s'exécutant en concurrence avec les autres processus de la même machine.

Les icônes (**I**cons) représentent des raccourcis pour exécuter des opérations sur le système comme par exemple exécuter un programme.

Un menu ou menu contextuel (**M**enu) est un système permettant d'exécuter des sous programmes, en relation avec le contexte courant pour le deuxième. Il pourront être textuels ou représentés l'aide d'icônes ou les deux.

Enfin, le pointeur (**P**ointer) est une marque précise sur l'espace de l'écran, toujours (ou presque) au premier plan. Il permet de manipuler les fenêtres, les menus et les icônes. Ce pointeur est déplacé avec la souris mais aussi avec les flèches de direction dans des systèmes plus anciens.

1.3 Éléments d'une GUI

Dans cette partie nous décrirons divers éléments de base d'une GUI tels que les widgets, les images, ainsi que les polices de caractères.

1.3.1 Widgets

Les widgets [1] se trouvent aussi sous le nom de *control* (appellation propre à Microsoft), ou encore de *composant d'interface graphique* en français. La notion de widget est très abstraite, puisqu'elle définit l'élément de base de l'interface graphique d'une application. C'est en disposant différents widgets dans une ou plusieurs fenêtres qu'un développeur conçoit l'interface graphique d'un programme.

Les widgets permettent à un utilisateur et à une application de s'échanger des informations et d'interagir. Ils se présentent sous des formes très diverses, et se classent selon différentes catégories. Parmi les plus courantes, on trouve les boutons (bouton radio, bouton poussoir, case à cocher), qui permettent à un utilisateur de valider (ou invalider) une donnée, et les zones de texte (zone de saisie, d'affichage) qui sont utilisées pour transmettre une information textuelle de l'utilisateur vers l'application, ou dans le sens inverse. Il en existe bien d'autres, comme les conteneurs par exemple qui sont des widgets de haut niveau pouvant regrouper d'autres widgets.

Pour mettre en place les widgets qui constitueront l'interface graphique de son application, le développeur utilise une bibliothèque logicielle appelée boîte à outils graphique, ou widget toolkit. C'est dans cette bibliothèque que sont définies toutes les catégories de widget disponibles (boutons, zones de texte, etc.), ainsi que leur style graphique. Le développeur y trouve aussi les fonctions permettant d'associer à un widget un comportement particulier, comme par exemple la fermeture de l'application sur pression du bouton approprié. Swing, Qt (utilisé par KDE) et GTK+ (utilisé par Gnome) sont trois des widget toolkit les plus utilisés.

1.3.2 Images

Un ordinateur stocke une image sous forme de fichier. Il existe deux types d'images numériques distincts : les images vectorielles et les images matricielles.

Les images vectorielles sont des compositions de formes géométriques simples (courbes, segments, points, plans) auxquelles sont associées des caractéristiques comme la couleur ou la position. Une fois agencées les unes par rapport aux autres, ces formes géométriques constituent une image. Comme ces images sont essentiellement basées sur les équations mathématiques des formes géométiques qui les composent, elles demandent à l'ordinateur un certain calcul avant chaque affichage. Ainsi, travailler sur des images vectorielles est assez gourmand en ressources. Pour les mêmes raisons, ces images sont considérées comme étant zoomables à l'infini, puisque la représentation graphique d'une équation est continue (du moins dans le cas des figures géométriques simples).

Les images matricielles sont quant à elles représentées d'une manière totalement différente : ce sont de simples tableaux de pixels (ou bitmap) auxquels sont associées des couleurs. Elles sont caractérisées, entre autres, par leur résolution qui s'exprime en pixel par unité de surface, et qui est une mesure de la qualité visuelle de l'image : plus la résolution est bonne, plus la quantité d'information visuelle est concentrée, et donc plus l'image est détaillée.

On notera que malgré les avantages que présentent les images vectorielles, elles restent beaucoup moins utilisées que les images matricielles, car il est à l'heure actuelle difficile de vectoriser un document numérisé. De plus, même si les images vectorielles sont composées de formes géométriques et non de grilles de pixels, elles sont tout de même converties en bitmaps pour leur affichage : on dit que l'image est rasterisée.

Qu'une image soit matricielle ou vectorielle, elle est stockée dans un certain format. C'est le format de l'image qui définira comment les données de l'image (formes géométriques ou pixels) seront représentées en mémoire, puis lues par un visionneur. Selon le format, les données peuvent être compressées ou non. Dans le cas d'un format avec compression, la compression peut se faire avec ou sans perte d'information. Les pertes se font généralement sur les couleurs de la manière suivante : pour une zone de l'image où les couleurs sont sensiblement les mêmes, on affecte à la zone en question une seule

1. Il convient de ne pas confondre les widgets d'interface graphique avec les widgets dits interactifs qui sont, eux, de petits programmes ou outils intégrés dans une page web, un système d'exploitation, ou encore un logiciel.

et unique couleur. Parmi les formats d'images les plus courants, on trouve BMP (sans compression), JPEG (compression avec perte), et PNG (compression sans perte) [Jab94].

Le format PBM est un format sans compression moins utilisé, mais nous allons le détailler car il a le mérite d'illustrer clairement le principe selon lequel les images matricielles ne sont que des tableaux de pixels colorés. PBM fait partie de la famille des formats du projet Netpbm, qui visait à faciliter l'échange de fichiers d'images. C'est un format extrêmement simpliste : on peut représenter une image au format PBM grâce à du texte. Il suffit d'écrire P1 sur la première ligne du fichier (ce signe indique que le fichier est un fichier PBM), puis sur la seconde le nombre de colonnes et le nombre de lignes, séparés par des espaces. Enfin, chaque pixel sera identifié par sa ligne et sa colonne, ainsi que par sa valeur : 1 ou 0, selon que le pixel doit être noir ou blanc (le format PBM est un format d'images en noir et blanc) [Pbm06].

```
1  P1
2  8 8
3  0 0 0 0 0 0 0 0
4  0 1 1 0 0 1 1 0
5  0 1 1 0 0 1 1 0
6  0 0 0 0 0 0 0 0
7  0 0 0 1 1 0 0 0
8  0 1 0 0 0 0 1 0
9  0 0 1 1 1 1 0 0
10 0 0 0 0 0 0 0 0
```

FIGURE 1.5 – Version texte d'une image au format PBM

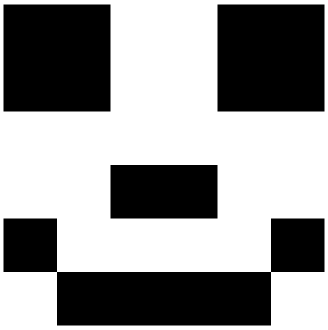

FIGURE 1.6 – Version visuelle de l'image

1.3.3 Polices de caractères

Une police de caractères, ou police d'écriture, est un ensemble logiciel de glyphes [2] qui sont utilisés pour représenter du texte dans un style graphique particulier. Les glyphes sont représentés, selon les polices, en tant qu'images matricielles ou bien vectorielles. On parle alors de police matricielle ou de police vectorielle. Les caractéristiques de ces deux types de polices ne sont pas difficiles à entrevoir : une police matricielle sera plus rapide et plus facile à afficher pour l'ordinateur, mais il faudra dessiner autant de versions d'un glyphe qu'il y aura de tailles possibles pour la police. Ce n'est pas le cas des

2. Un glyphe est un élément typographique de base comme un caractère, un accent, un symbole, ou encore un casseau.

polices vectorielles, qui peuvent être redimensionnées sans perte de qualité. Par contre, pour des soucis de lisibilité, il convient de ne pas agrandir (ou rétrécir) toutes les formes géométriques qui composent les glyphes avec le même coefficient de proportionalité. Il faut donc, pour une police vectorielle, instaurer des règles de mise à l'échelle afin de rendre les glyphes lisibles, quelque soit leur taille d'affichage [Tan04]. Avant les années 90, les polices matricielles étaient les plus couramment utilisées. Elles ont été ensuite remplacées pour la plupart par les polices TrueType, qui sont des polices vectorielles.

1.4 Interactions avec l'utilisateur

Cette partie traite des interactions entre un utilisateur et sa machine.

1.4.1 Claviers et souris

Sur un ordinateur grand public, c'est essentiellement par le biais du clavier et de la souris qu'un utilisateur transmet des informations à sa machine.

Jusqu'à la fin des années 80, les claviers utilisaient un connecteur de type DIN relié directement à la carte mère, qui fut remplacé pour des raisons marketing par un connecteur PS/2 [PS/], qui ne présentait aucun nouvel avantage, si ce n'est celui de pouvoir vendre une nouvelle catégorie de claviers. En fait, le protocole de communication était le même, et il suffifisait d'un simple connecteur PS/2 - DIN (ou l'inverse) pour passer de l'un à l'autre. Aujourd'hui, la grande majorité des claviers récents sont connectés à l'ordinateur par le biais d'un port USB, et certains disposent d'une connectique sans fil comme la technologie Bluetooth.

Les souris sont des dispositifs relativement simples. L'évolution de leur connectique est semblable à celle des claviers : autrefois branchées grâce à des connecteurs PS/2 (le protocole de communication était par contre différent de celui des claviers), elles sont maintenant majoritairement USB ou sans fil. Les deux types de souris les plus répendus sont les souris à boule, et les souris optiques. Les premières peuvent déterminer leurs déplacements grâce aux rotations de deux rouleaux entraînés par la boule lorsqu'elle tourne, l'un étant placé selon l'axe des abscisses, et l'autre selon l'axe des ordonnées. Les souris optiques, elles, sont munies de photodétecteurs, et peuvent percevoir un déplacement en comparant deux images successives du support sur lequel elles sont posées.

1.4.2 Gestion des événements utilisateurs

Dans cette partie, nous allons détailler la façon dont Linux gère les événements clavier, depuis la pression d'une touche jusqu'à sa transmission à un programme applicatif. La gestion des événements souris est assez similaire. Linux a été pris comme support d'étude car il est plus aisé de trouver de la documentation pour ce système, mais les principes sous-jacents sont généralisables aux autres systèmes d'exploitation.

Le clavier possède en interne un microprocesseur qui vérifie en permanence si une touche a été pressée ou relâchée. Lorsqu'un événement a eu lieu, le microprocesseur en informe le contrôleur de clavier (keyboard controller). Le contrôleur de clavier récupère alors un scancode : le scancode correspond à l'identifiant unique d'une touche. Il est dépendant du matériel, ainsi que du layout utilisé (aussi appelé "disposition", et qui varie selon la langue de l'utilisateur et la disposition des touches du clavier). Le scancode est ensuite fourni au pilote de clavier (keyboard driver).

Il existe trois modes de fonctionnement pour un pilote de clavier [Bro02], lorsqu'il reçoit un scancode. Le premier est le mode raw. Dans ce mode, le pilote n'effectue aucun traitement et transmet tel quel le scancode au programme applicatif. Ce mode permet à l'application d'établir un contrôle très fin des entrées de l'utilisateur. C'est à l'application de déterminer alors quelle touche a été pressée, ou quelle combinaison de touche a été effectuée. Le second mode est le mode medium raw. Ce mode est similaire au mode raw, mais plutôt que d'envoyer un scancode, le pilote de clavier enverra un keycode, qui identifie lui aussi la touche, mais de manière normalisée : le keycode est indépendant du

matériel et du layout. Enfin, il existe le mode ASCII/Unicode (qui sont en fait deux modes distincts mais très similaires puisqu'ils ne diffèrent que par l'encodage des caractères), dans lequel le pilote détermine lui même le code ASCII ou Unicode de la touche pressée, ou de la combinaison de touche : il faudra envoyer le numéro 65 qui correspond au code ASCII du caractère "A" si les touches pressées par l'utilisateur sont *MAJ* et *a*.

Le serveur X demande au pilote clavier de passer en mode raw [Wro], et le pilote de clavier envoie donc au serveur X des scancodes. C'est alors au serveur X, qui reçoit le scancode dans sa file d'événéments, de savoir quelle est la fenêtre qui a le focus (ou plutôt quel widget de quelle fenêtre a le focus). Le serveur envoie alors une information au bon client, qui est sans cesse en attente d'un événement de la part du serveur. Cet événement est ensuite traité par les couches successives de la librairie de widget (très souvent GTK+ ou Qt) pour être enfin exploitable par l'application.

Conclusion Alors qu'il était dans le passé relativement flou et incohérent, le modèle structurel d'une GUI est aujourd'hui bien défini grâce au paradigme WIMP, qui a fait ses preuves : toutes les applications de bureau sont aujourd'hui basées sur ce paradigme. Nous avons présenté de façon succinte comment étaient gérés les événements utilisateurs, ce qui nous a permis d'entrevoir la grande complexité de l'implémentation d'une interface graphique. Nous détaillerons plus précisément les composants qui entrent en jeu dans une GUI dans la partie suivante.

Chapitre 2

Architecture d'une interface graphique

Dans cette partie, nous présenterons les quatre couches qui composent une interface graphique, de la plus basse (proche du matériel) à la plus haute. Nous détaillerons leur rôle, leur fonctionnement et la manière dont elles communiquent entre elles pour former un ensemble cohérent.

2.1 Matériel

Commençons notre exploration de l'architecture d'une interface graphique par ses fondements : le matériel. Le but final étant d'afficher des informations à l'écran, il faut communiquer avec la carte graphique. Pour ce faire, trois solutions existent :

- Écrire un pilote spécifique à la carte graphique utilisée,
- Écrire un pilote VGA,
- Utiliser les modes standards VGA par l'intermédiaire des fonctions du BIOS.

L'écriture d'un pilote de carte graphique est un travail long et complexe, car il existe très peu de documentation à ce sujet. Il présente également l'inconvénient d'être spécifique à une carte, et donc d'être très peu réutilisable. Nous nous intéresserons donc ici à VGA, qui est universel. Notons tout de même que le développement d'un pilote de carte graphique est incontournable si l'on veut disposer de fonctionnalités avancées comme l'accélération 3D.

2.1.1 VGA

VGA, pour Video Graphics Array, est un standard d'affichage lancé en 1987 par IBM, qui définit un ensemble de modes d'affichage (texte ou graphique) que les cartes graphiques doivent supporter. Les modes graphiques standard sont :

- 640x480 en 16 couleurs, résolution la plus souvent associée au VGA
- 640x350 en 16 couleurs
- 320x200 en 16 couleurs
- 320x200 en 256 couleurs

Ces 16 ou 256 couleurs peuvent être choisies parmi une palette de 262144 couleurs (2^{6+6+6}, 6 bits par composante RVB). Les cartes VGA disposent d'au moins 256 Kio de mémoire vidéo [Nea98].

Bien qu'aujourd'hui, les cartes graphiques reposent sur des standards beaucoup plus récents et évolués que VGA, elles émulent toujours les composants des premières cartes VGA pour des raisons de compatibilité. L'intérêt de VGA est donc double :

– D'une part, sa relative simplicité en fait un très bon sujet d'étude. Un pilote VGA simple mais fonctionnel peut s'écrire en moins de 200 lignes de code.

– D'autre part, étant le dernier standard IBM que la majorité des fabricants de PC ont suivi, il est considéré comme le plus petit dénominateur commun que toutes les cartes graphiques sont censées supporter, sans recourir à un driver spécifique.

Présentons maintenant brièvement les composants d'une carte VGA et leurs fonctions (figure 2.1) :

– La **mémoire vidéo** : elle contient l'image affichée à l'écran. En mode texte, les codes ASCII des caractères ainsi que leurs attributs (couleur, couleur de fond...) y sont stockés ; en mode graphique, la couleur de chaque pixel (ou plutôt son index dans la palette) y est stockée.

– Le **CRTC** (Cathod Ray Tube Controller) : comme son nom l'indique, il contrôle le déplacement du spot sur l'écran [1]. C'est grâce à ses registres de configuration que l'on peut changer la résolution de l'écran par exemple. Comme il connaît en permanence la position du spot, il est également chargé de déterminer l'adresse en mémoire vidéo de l'information (caractère ou pixel) à afficher à l'écran.

– Le **GDC** (Graphics Data Controller) : c'est l'intermédiaire entre le processeur et la mémoire graphique. Il propose plusieurs modes d'accès à la mémoire vidéo en lecture et en écriture, en jouant sur les différents "plans" sur lesquels est répartie la mémoire vidéo (voir la section sur le TS).

– L'**ATC** (Attribute Controller) : c'est lui qui détermine ce qui doit être affiché à l'écran en fonction des informations présentes en mémoire vidéo (caractères ou pixels). En mode graphique par exemple, c'est lui qui détermine l'index à utiliser dans la palette du DAC.

– Le **DAC** (Digital to Analog Converter) : il est chargé de convertir les signaux numériques (index sur 8 bits dans sa palette de couleurs) en signaux analogiques permettant au moniteur de commander l'intensité des 3 faisceaux d'électrons (rouge, vert, bleu) qui constituent les couleurs affichées.

– Le **TS** (Timing Sequencer) : c'est le chef d'orchestre de tous les composants sus-cités. Il gère l'horloge et est donc chargé du rafraîchissement de la mémoire vidéo. Il gère également le découpage en "plans" de cette mémoire : nous avons vu que les cartes VGA disposent de 256 Kio de mémoire vidéo, mais cette mémoire est en réalité découpée en 4 "plans" de 64 Kio. Le séquenceur gère le plan auquel le processeur accède lorsqu'il veut lire ou écrire en mémoire vidéo.

1. Aujourd'hui, les écrans LCD procèdent à une conversion analogique/numérique du signal et l'interprètent pour afficher les pixels au bon endroit.

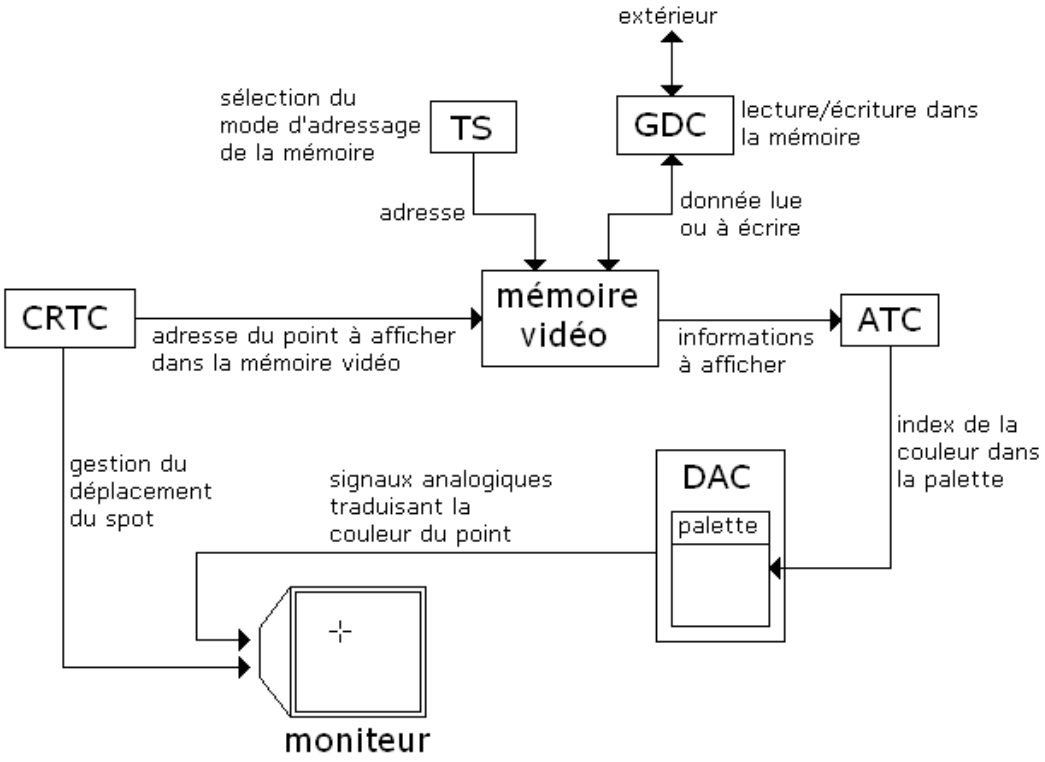

FIGURE 2.1 – Interaction entre les 6 composants principaux d'une carte VGA

Ces composants sont configurables de deux manières [OSD11] :

- La plus simple est d'utiliser les fonctions du BIOS, qui permettent de configurer un mode standard à l'aide d'une interruption. La résolution 320x200 en 256 couleurs correspond par exemple au mode 13h. La configuration de l'ensemble des registres est alors assurée par le BIOS.
- On peut également configurer tous ces registres manuellement, grâce aux ports d'entrée/sortie de la carte utilisée. Cela permet de "créer" de nouveaux modes, puisque l'on n'est alors plus limité aux modes standard. En théorie, n'importe quelle résolution jusqu'à 800×600 est possible. C'est également très utile lorsque l'on n'a pas accès aux interruptions du BIOS, par exemple en mode protégé.

Notons l'existence de **Super VGA** ou **SVGA** pour Super Video Graphics Array, une extension du standard VGA lancée la même année que VGA par IBM. À la différence de VGA, SVGA n'est pas un standard officiellement défini par IBM. La Video Electronics Standard Association (VESA), un consortium de fabriquants de matériel informatique visant à définir des standards vidéo, a cependant défini les extensions VBE (VESA Bios Extensions) qui s'approchent d'une définition officielle de SVGA [Ass94]. Elles permettent des résolutions jusqu'à 1600×1200 selon les cartes graphiques, les résolutions supérieures à 1280×1024 n'étant pas couvertes par le standard VBE. La dernière version de VBE (VBE 3.0) date de 1998 [Ass98].

2.1.2 La couche d'accès au matériel : le pilote graphique

La couche logicielle permettant le dialogue avec la carte graphique est le pilote (ou driver) graphique. Il permet à la couche supérieure, le système de fenêtrage, de dessiner à l'écran en s'abstrayant de la configuration du mode graphique.

Le fonctionnement de VGA présenté en section 3.1.1 peut laisser penser que l'écriture d'un pilote graphique est relativement simple, mais il ne faut pas oublier que les cartes graphiques actuelles mettent en œuvre des technologies beaucoup plus complexes (accélération 3D, CUDA...). Leur taille peut alors atteindre la centaine de Mio... Comme tout logiciel volumineux, ces pilotes sont alors découpés en

composants distincts s'acquittant chacun d'une tâche particulière. Dans le monde Linux, les drivers graphiques libres s'articulent autour de 3 composants [Zha09] :

- **DDX**, pour Device Dependent X : c'est le pilote d'affichage du serveur X (le système de fenêtrage utilisé sous Linux). Le système de fenêtrage l'utilise pour les manipulations matérielles de base, son rôle principal étant de configurer le mode d'affichage : résolution, profondeur des couleurs... Comme son nom l'indique, il est spécifique à chaque matériel. Il est également utilisé pour déléguer certaines opérations au processeur graphique (GPU pour Graphics Processing Unit) pour qu'elles puissent être accélérées (déplacement des fenêtres, scroll, effets visuels, rendu vidéo...). En résumé, il s'occupe de toute la partie 2D, et uniquement de la 2D.
- **DRI**, pour Direct Rendering Infrastructure, le pilote Mesa : Mesa est l'implémentation libre pour Linux d'OpenGL, un procédé d'accélération 3D. Elle se compose de deux parties : une bibliothèque compatible avec OpenGL que les applications 3D peuvent utiliser, et des pilotes DRI qui traduisent les instructions internes à Mesa en instructions que la carte graphique peut comprendre et exécuter. Après un appel à une fonction de la librairie Mesa et sa conversion en un langage spécifique au GPU, les instructions correspondantes doivent être envoyées au matériel pour être exécutées. Mesa ne peut s'acquitter de cette tâche, car elle s'exécute en espace utilisateur (seul du code noyau peut accéder directement au matériel). Elle pourrait transmettre ces instructions via X par l'intermédiaire de DDX, mais cela poserait de sérieux problèmes de performance. Elle s'appuie donc sur un dernier composant :
- **DRM**, pour Direct Rendering Manager, le pilote du noyau : c'est un module du noyau Linux. Il expose certaines fonctionnalités du matériel en fournissant une couche d'abstraction à l'espace utilisateur, permettant à des applications comme Mesa d'exploiter le GPU plus efficacement.

2.2 Système de fenêtrage

Le système de fenêtrage (windowing system) permet à l'utilisateur d'interagir avec plusieurs applications graphiques, affichées dans des zones rectangulaires appelées fenêtres, par l'intermédiaire d'un clavier et d'un dispositif de pointage (souris, trackball, touchpad...).

Le système de fenêtrage n'est pas chargé de la décoration des fenêtres (bordures, barre de titre...) ni de l'apparence ("look and feel") de leur contenu. Il propose des primitives graphiques (tracé de lignes, de rectangles, rendu de polices de caractères...) qui constituent une couche d'abstraction pour la couche supérieure, le gestionnaire de fenêtres.

2.2.1 Zoom sur X Window

Le système de fenêtrage le plus célèbre est X, ou X Window System, qui est le système de fenêtrage des systèmes UNIX. À l'origine publié par le MIT en 1984, la version actuelle de X (X11) date de 1987 [GP86] [GKM90].

X utilise un modèle client/serveur (figure 2.2) : un serveur X s'exécute sur un ordinateur qui dispose d'un écran, d'un clavier et d'une souris, et communique avec plusieurs clients. Il sert d'intermédiaire entre l'utilisateur et les programmes graphiques (clients X) : il accepte les requêtes d'affichage des clients, ce qui leur permet de s'afficher sur l'écran de l'utilisateur, et reçoit les entrées de l'utilisateur (clavier, souris) pour les transmettre aux programmes clients.

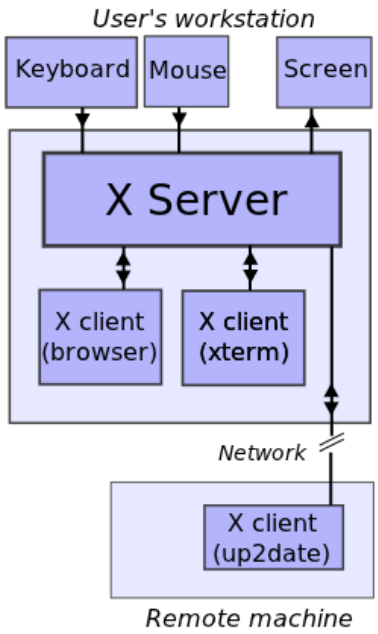

FIGURE 2.2 – Exemple d'utilisation de X Window dans un contexte client/serveur

Le protocole de communication entre clients et serveur X (X protocol) peut utiliser le réseau de manière totalement transparente : dans le cas le plus courant (poste de travail local), le client et le serveur s'exécutent tous deux sur la même machine, mais ils peuvent également s'exécuter sur deux machines différentes (avec des architectures et/ou des systèmes d'exploitation différents) et communiquer à travers le réseau. Un exemple bien connu est l'utilisation de la commande ssh avec renvoi graphique (option -X).

Observons de plus près la communication entre client et serveur X : le client initie la connexion en envoyant le premier paquet. Le serveur peut éventuellement demander une authentification puis répond en indiquant l'acceptation ou le refus de la connexion. Dans le cas d'une acceptation, le serveur fournit au client un identifiant à utiliser pour les échanges suivants. À partir de là, 4 types de paquets peuvent être échangés :

- **Requête** : le client souhaite obtenir une information du serveur ou lui demande d'effectuer une action.
- **Réponse** : le serveur répond à une requête du client.
- **Événement** : le serveur envoie un événement au client : entrée clavier/souris, déplacement ou redimensionnement de fenêtre...
- **Erreur** : le serveur envoie un paquet d'erreur au client lorsqu'il a formulé une requête invalide.

Grâce à ces 4 types de paquets, client/serveur et clients entre eux vont pouvoir collaborer pour présenter une interface graphique à l'utilisateur [Jon89] [Man93].

Fenêtres

Lorsque l'on parle de "fenêtre" dans le langage courant, on parle en réalité de "fenêtre de premier ordre" ("top-level window"). Le terme "fenêtre" est ambiguë car il est également utilisé pour désigner les fenêtres filles d'une fenêtre mère. Au sens de X, une fenêtre n'est rien d'autre qu'une zone rectangulaire : les widgets affichés à l'intérieur d'une application (menus, icônes, boutons...) sont des fenêtres filles de la fenêtre principale de l'application.

Toute fenêtre est rattachée à une fenêtre mère : les fenêtres forment un arbre, dont la racine ("root window") est créée automatiquement par le serveur X. Cette fenêtre spéciale a les mêmes dimensions

que l'écran, et située en dessous de toutes les autres. Les fenêtres de premier ordre sont les filles directes de la fenêtre racine.

Identifiants

Tous les objets créés par le serveur X possèdent un identifiant unique (un entier) que les clients peuvent utiliser lorsqu'ils dialoguent avec le serveur. Un client peut par exemple demander au serveur X de créer une fenêtre avec un identifiant donné, identifiant qui lui permettra plus tard d'afficher du texte à l'intérieur de ladite fenêtre. Ces identifiants sont bien entendus uniques pour le serveur (et pas seulement pour le client), ce qui permet à un client d'accéder à un objet grâce à son identifiant, même s'il n'a pas lui-même créé cet objet. Cela permet par exemple aux gestionnaires de fenêtres d'interagir avec les fenêtres créées par les autres applications.

Attributs et propriétés

Le serveur X stocke, pour chaque fenêtre, un ensemble d'attributs et de propriétés. Les attributs sont des données caractérisant la fenêtre (taille, position, couleur de fond...) tandis que les propriétés ne sont que des données arbitraires associées à la fenêtre, qui n'ont aucun sens pour X. Ces attributs et propriétés peuvent être lus et écrits par les clients, via les requêtes appropriées. Un exemple de propriété est `WM_NAME`, qui contient le nom de la fenêtre. X n'a que faire de cette information (ce n'est pas lui qui gère la décoration des fenêtres), mais elle permet au gestionnaire de fenêtres de faire son travail : lors de la décoration de la fenêtre, il peut lire la valeur de cette propriété et l'afficher dans la barre de titre. Les propriétés sont donc un bon moyen pour les clients de communiquer entre eux.

Événements

Les événements permettent au serveur d'indiquer à un client que quelque chose susceptible de l'intéresser a eu lieu. Ils peuvent également être utilisés pour la communication entre clients : un client peut demander au serveur d'envoyer un événement à un autre client. Si par exemple un client veut récupérer le texte actuellement sélectionné, il peut demander au serveur X d'envoyer un événement au client à qui appartient la fenêtre qui a actuellement le focus.

Les entrées clavier/souris ou la création de nouvelles fenêtres sont d'autres exemples d'événements qui sont notifiés aux clients. La création de nouvelles fenêtres intéresse tout particulièrement les gestionnaires de fenêtres.

Bien que certains types d'événements soient systématiquement envoyés aux clients, la majorité ne sont envoyés que si le client a manifesté son intérêt pour ce type d'événements. Un client peut ainsi s'"abonner" aux entrées souris mais pas clavier.

Couleurs

X représente les couleurs à l'aide d'un entier de 32 bits appelé "pixelvalue". Cet entier est en réalité un index dans une table (la "colormap") qui contient des triplets RVB. Il existe plusieurs modes de représentation des couleurs sur ce principe, selon que le client puisse ou non modifier la colormap, que celle-ci soit monochrome (niveaux de gris) ou non, et que le pixelvalue soit utilisé dans son intégralité pour indexer une unique colormap ou divisé en 3 parties pour indexer 3 colormaps distinctes (une pour le rouge, une pour le vert, une pour le bleu). Le mode de représentation des couleurs est négocié à la connexion du client.

Xlib et bibliothèques de widgets

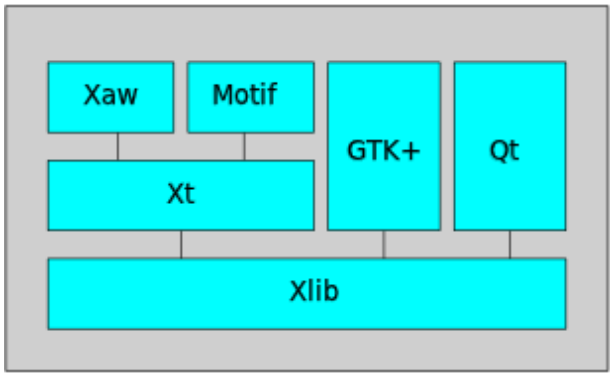

FIGURE 2.3 – Librairies utilisées pour la programmation X : Xlib et librairies de widgets

Différents choix s'offrent aux développeurs d'applications X pour dialoguer avec le serveur X (figure 2.3) [Man01] :

- Programmer directement au niveau du protocole X. Cette solution n'est évidemment jamais retenue.
- Utiliser la bibliothèque X ou Xlib, qui propose un premier niveau d'abstraction mais n'est guère très commode à utiliser car elle reste très proche du protocole X. Elle est donc en pratique surtout utilisée pour développer des surchouches, appelées "toolkits" ou "librairies de widgets".
- Utiliser le niveau d'abstraction suivant : le X Toolkit (Xt), qui est orienté objets et permet de manipuler des widgets. Il existe plusieurs jeux de widgets pour Xt, on peut en particulier citer Xaw (X Athena Widgets) et Motif. Ces librairies sont aujourd'hui de moins en moins utilisées.
- Utiliser de nouvelles surcouches de la Xlib, disjointes de Xt, plus complètes et plus simples à utiliser que les solutions vieillissantes basées sur Xt. GTK+ et Qt, utilisées respectivement par les environnements de bureau GNOME et KDE, sont les deux librairies de widgets les plus répandues aujourd'hui. On comprend alors que ce sont ces librairies qui se chargeront du dessin des widgets à l'écran en faisant des appels à la Xlib, et que ce sont donc elles qui contrôlent l'apparence (le "look and feel") de l'intérieur des fenêtres.

Gestionnaire de session et gestionnaire d'affichage

Deux logiciels s'acquittent de tâches intimement liées au système de fenêtrage, mais que celui-ci ne prend pas en charge lui-même :

- Le gestionnaire de session ou session manager permet à l'utilisateur de retrouver lors de sa reconnexion les fenêtres ouvertes avant sa déconnexion, dans l'état dans lequel il les avait laissées. Pour cela, le gestionnaire de session mémorise la liste des applications ouvertes au moment de la déconnexion de l'utilisateur, et les relance à sa reconnexion. Pour que l'état de ces applications puisse être restauré, elles doivent être capables de sauvegarder leur état d'exécution sur demande du gestionnaire de session, et de le recharger lors de leur prochain lancement. X Window inclut un gestionnaire de session par défaut nommé *xsm*, mais la plupart des environnements de bureau proposent leur propre gestionnaire de session (*ksmserver* pour KDE, *gnome-session* pour GNOME par exemple).
- Le gestionnaire d'affichage ou display manager lance un serveur X local et s'y connecte pour proposer à l'utilisateur un écran de connexion graphique. Il peut également accepter des connexions provenant de serveurs X distants et permettre ainsi à des machines distantes d'ouvrir une session et d'utiliser des applications graphiques de la machine locale : les applications sont lancées sur la même machine que le gestionnaire d'affichage, mais leurs entrées/sorties dont redirigées

depuis/vers la machine distance (sur laquelle un serveur X est lancé). X Window inclut un gestionnaire d'affichage appelé *XDM*, mais la plupart des environnements de bureau proprosent leur propre gestionnaire de d'affichage (*KDM* pour KDE, *GDM* pour GNOME par exemple).

Jusqu'à 2004, XFree86 était l'implémentation de X la plus répandue sur les systèmes Unix. Depuis 2004, X.Org, un fork de XFree86, est devenu prédominant.

Maintenant que nous avons décrit le rôle et le fonctionnement du système de fenêtrage, intéressons nous de plus près à un client particulier que nous avons évoqué plusieurs fois dans cette section : le gestionnaire de fenêtres.

2.3 Gestionnaire de fenêtres

2.3.1 Rôle

Le gestionnaire de fenêtres a pour rôle principal de permettre à l'utilisateur d'intéragir entre les différentes fenêtres des applications, notamment leur position, leur taille et la possibilité d'icônifier la fenêtre. Pour celà, il commence par "décorer" les fenêtres des applications avec un cadre qui contiendra le titre de la fenêtre ainsi que des boutons permettant de la manipuler. Ce cadre est géré par le gestionnaire et permettra à l'utilisateur de manipuler la fenêtre. Pour gérer l'icônification, le gestionnaire propose le plus souvent une barre de tâches contenant toutes les icônes des applications ouvertes.

Pour pouvoir démarrer les applications, le gestionnaire propose également des barres de menus ainsi que des zones de notification. Pour une meilleure expérience utilisateur, il peut également proposer d'autres services tels que la gestion de bureaux virtuels permettant de séparer des groupes de fenêtres ainsi que des raccourcis claviers divers.

Selon le gestionnaire, l'ensemble de ces éléments peut être plus ou moins personnalisé par l'utilisateur. Il peut par exemple modifier le thème de l'affichage (look and feel), le fond d'écran, créer/déplacer les barres de tâches et menus, créer de nouvelles zones de notifications, etc.

Il existe deux principales catégories de gestionnaire de fenêtres. Les gestionnaires de fenêtres "fottants" et "tilling" [neG]. Les flottants sont ceux pour lesquels nous sommes les plus familiers. Les fenêtres sont gérées comme une pile. En cas de superposition, c'est la fenêtre le plus en haut de la pile qui est affichée. Comme un tas de feuilles qu'on peut réorganiser. La fenêtre au premier plan est celle du haut de la pile. Les gestionnaires de fenêtres "tilling" organisent, comme leur nom l'indique, les fenêtres comme un carrelage. Les fenêtres sont donc affichées et l'écran est partagé entre toutes les fenêtres. Il est possible de personnaliser l'agencement pour arriver à avoir des fenêtres plus grandes que d'autres comme la disposition suivante par exemple :

FIGURE 2.4 – Bureau avec Awesome

2.3.2 Implémentation avec X server

Dans le cas de X, l'interfaçage du gestionnaire de fenêtre est assez déroutant. Le gestionnaire est en fait un client comme n'importe quelle application au détail prêt qu'elle s'affiche sur toute l'étendue de la fenêtre racine du serveur X. Comme nous l'avons vu précédemment, les applications peuvent modifier les paramètres des autres applications. Lors de son démarrage, le gestionnaire de fenêtres s'abonne à des évènements spéciaux tels que la création d'une nouvelle application. Lorsqu'une application (disons gedit) démarre, elle va créer une nouvelle fenêtre fille de la fenêtre racine du serveur X qu'elle a choisi. Le serveur va annoncer cette création au gestionnaire de fenêtres qui va créer une nouvelle fenêtre contenant les décorations et va rendre la fenêtre de gedit fille de celle-ci. La fenêtre de gedit sera donc gérée par cette fenêtre mère qui affichera la barre de titre et les bordures. L'utilisateur interagira sur la fenêtre mère pour la déplacer/redimensionner/icônifier. Mais quand il interagira avec le contenu de la fenêtre, les signaux seront interprétés directement par l'application soit gedit.

Remarque : Un gestionnaire de fenêtre est considéré par le serveur X comme une application comme une autre. En revanche, lorsqu'on essaye de lancer un second gestionnaire de fenêtres, le lancement échoue car il détecte qu'il y a déjà un autres gestionnaire de fenêtre lancé. Le gestionnaire de fenêtre se déclare donc pas exactement de la même façon que n'importe quelle application. Il est cependant possible de demander de remplacer le gestionnaire par un nouveau.

Les gestionnaire de fenêtres les plus répandus sont les gestionnaires de fenêtres flottants. Dans le monde d'Unix on trouve metacity maintenant remplacé par Mutter depuis la version 3 de l'environnement de bureau Gnome dont nous allons bientôt parler. Compiz est particulièrement utilisé pour remplacer metacity car il permet un affichage 3D des bureaux virtuels. Un dernier est KWin qui est le gestionnaire de fenêtre de l'environnement de bureau KDE. Ce dernier a une interface assez proche de Windows et est dit plus convivial que metacity qui a une interface très sobre.

Les gestionnaires de fenêtres "tilling" sont beaucoup moins répandus. Nous citerons Awesome qui est réputé être très configurable et xmonad qui a été écrit avec seulement 1000 lignes de code. Il est

donc très léger mais tout de même réputé pour sa stabilité et son extensibilité.

Un gestionnaire de fenêtres est indispensable mais il n'est pas suffisant pour créer une interface graphique complète. Pour cela, on utilise un environnement de bureau qui, lui, comprend un gestionnaire de fenêtres mais aussi d'autres outils indispensables.

2.4 Environnement de bureau

Lorsqu'on installe une distribution Linux, en général, on a le choix entre plusieurs environnements de bureau. Un environnement de bureau est en fait un ensemble de logiciels et de standardisations qui rend le système cohérent d'un point de vue graphique mais permet également de standardiser les communications entre les applications. Par exemple, l'environnement de bureau GNOME conseille l'utilisation de la bibliothèque graphique GTK+ pour ses applications alors que KDE préfère la bibliothèque Qt. Ces bibliothèques ont chacune leur façon d'afficher les widgets. Si toutes les applications utilisent la même bibliothèque, leurs affichages seront similaires. Les boutons par exemple seront affichés de la même façon. Quand on démarre une application faite pour KDE sous GNOME, on le remarque très rapidement : le rendu visuel est totalement différent. Le choix d'une bibliothèque graphique commune à toutes permet donc d'avoir un rendu visuel cohérent entre les différentes applications.

Le rôle de l'environnement de bureau ne s'arrête cependant pas à rendre l'aspect graphique cohérent. Il offre également une suite d'outils et de logiciels qui permet d'utiliser son ordinateur de manière graphique. Ces logiciels sont d'ailleurs entre autres choisis en fonction de la bibliothèque graphique qu'ils utilisent. Les principaux outils logiciels sont les suivants :

- Un gestionnaire de fenêtres
- Un outil graphique de configuration du système
- Un outil graphique de configuration du réseau
- Un système de connexion à sa session utilisateur
- Un éditeur de texte graphique
- Un navigateur de fichiers permettant de manipuler graphiquement le système de fichier
- Un moteur de rendu de pages web
- Un système de fichiers virtuel
- Des logiciels de communication (mails, messagerie instantanée, communication audio et vidéo...)

Les deux environnements de bureau les plus utilisés sous Linux sont GNOME et KDE [Coo]. Ces deux suivent des standards de Freedesktop.org qui proposent des règles à respecter pour faciliter l'interopérabilité entre les différents environnements de bureau sans pour autant uniformiser ceux-ci.

Il est très conseillé de respecter ces règles mais ce n'est pas pour autant obligatoire. Il est donc possible d'utiliser une application codée pour KDE sous GNOME. Seulement, d'un point de vue autre que graphique, il se peut que certaines fonctionnalités ne fonctionnent pas par défaut. Par exemple, si ce programme souhaite utiliser une application qui est installée par défaut sous KDE mais pas sous GNOME, il risque d'y avoir des erreurs lors de la tentative d'utilisation si celle-ci n'est pas installée.

Conclusion Nous avons maintenant une vue d'ensemble de l'architecture d'une interface graphique :

- le matériel (carte graphique), et la couche logicielle qui permet de dialoguer avec ce matériel (pilote),
- le système de fenêtrage, qui permet aux différentes applications de dessiner dans des zones rectangulaires qui leur sont dédiées (fenêtres) et à l'utilisateur d'interagir avec ces applications,
- le gestionnaire de fenêtres, qui décore les fenêtres et permet à l'utilisateur de les manipuler,
- l'environnement de bureau, qui définit des conventions (notamment la librairie de widgets utilisée, qui conditionne l'apparence des applications) et fournit une collection de logiciels formant un environnement graphique cohérent.

Nous avons présenté plus en détails le système X Window, dans lequel chaque couche est bien distincte. Ce choix tend aujourd'hui a être remis en cause pour des raisons de performances : ainsi,

Wayland, proposé comme successeur à X.Org, est à la fois système de fenêtrage, gestionnaire de fenêtres et compositeur (il gère les effets visuels). Il élimine ainsi des intermédiaires entre les applications et le matériel. Des distributions Linux comme Ubuntu, Fedora ou encore Mandriva envisagent de remplacer X.Org par Wayland dans leurs futures versions.

Conclusion

Cette étude bilbiographique nous a permis d'acquérir une vision globale des interfaces graphiques, tout en mettant l'accent sur les points spécifiques qui nous seront utiles pour implémenter notre solution. Nous avons, en particulier, pu discerner bien distinctement les différents acteurs d'une interface graphique, dont les frontières sont habituellement assez floues.

Nous avons été doublement séduits par le modèle X Window : tout d'abord, l'architecture client/serveur est intéressante puisqu'elle permet une utilisation très souple du gestionnaire de fenêtrage. Ensuite, la dissociation des différentes couches logicielles, chacune remplissant un rôle bien défini, permet de conserver une certaine simplicité, et confère à l'ensemble une grande modularité.

Nous allons donc nous inspirer de X Window pour notre système de fenêtrage, et implémenter une architecture client/serveur, même si nous ne disposons pas du réseau à l'heure actuelle. Nous implémenterons ensuite un gestionnaire de fenêtres flottant simple, puisque c'est cette solution qui nous semble être la plus ergonomique. Enfin, nous développerons une librairie de wigets qui permettra la création d'applications graphiques que nous utiliserons pour tester et mettre en oeuvre notre solution : une horloge, un éditeur de texte, ou encore des petits jeux de type casse brique.

Bibliographie

[Ass94] Video Electronics Standards Association. VESA BIOS EXTENSION (VBE) Core Functions Standard - Version 2.0. 18 novembre 1994.

[Ass98] Video Electronics Standards Association. VESA BIOS EXTENSION (VBE) Core Functions Standard - Version 3.0. 16 septembre 1998.

[Bro02] Andries Brouwer. The linux keyboard and console howto. `http://www.faqs.org/docs/Linux-HOWTO/Keyboard-and-Console-HOWTO.html#s2`, 2002. [En ligne ; Consulté le 29 janvier 2011].

[Coo] Alan Coopersmith. Desktops. `http://www.freedesktop.org/wiki/Desktops`. [En ligne ; Consulté le 12 janvier 2011].

[GKM90] James Gettys, Philip L. Karlton, and Scott Mcgregor. The X Window System Version 11. *ACM Transactions on Graphics*, 5 :79–109, 1990.

[GP86] James Gettys and Keith Packard. The X Window System. *ACM Transactions on Graphics, Vol*, pages 79–109, 1986.

[Jab94] Gaël Jaboulay. Les formats de fichiers d'images. `http://tecfa.unige.ch/tecfa/teaching/staf13/fiches-mm/formatfichier.htm`, 1994. [En ligne ; Consulté le 28 janvier 2011].

[Jer05] Jeremykemp. Early mouse patents. `http://en.wikipedia.org/wiki/File:Mouse-patents-englebart-rid.png`, 2005. [En ligne ; Consulté le 31 janvier 2012].

[Jon89] Oliver Jones. *Introduction to the X Window System*. 1989.

[Man93] Niall Mansfield. *The Joy of X / An Overview of the X Window System*. 1993.

[Man01] Daniel Manrique. X Window System Architecture Overview HOWTO. `http://tldp.org/HOWTO/XWindow-Overview-HOWTO/index.html`, 2001. [En ligne ; Consulté le 31 janvier 2012].

[Mus98] Digibarn Computer Museum. Xerox alto operating system and alto applications. `http://www.aresluna.org/attached/usability/articles/biurkonaekranie`, 1998. [En ligne ; Consulté le 31 janvier 2012].

[Nea98] J. D. Neal. VGA/SVGA Video Programming. `http://www.osdever.net/FreeVGA/vga/vga.htm`, 1998. [En ligne ; Consulté le 31 janvier 2012].

[neG] neowillow et Graphox. Zoom sur les gestionnaires de fenêtres sur les systèmes unix. `http://www.siteduzero.com/news-62-42074-zoom-sur-les-gestionnaires-de-fenetres-sur-les-systemes-unix.html`. [En ligne ; Consulté le 8 janvier 2011].

[OSD11] OSDev. VGA Hardware. `http://wiki.osdev.org/VGA_Hardware`, 2011. [En ligne ; Consulté le 31 janvier 2012].

[Pbm06] The pbm format. `http://netpbm.sourceforge.net/doc/pbm.html`, 2006. [En ligne ; Consulté le 14 janvier 2011].

[PS/] Ps/2 connector. `http://en.wikipedia.org/wiki/PS/2_connector`. [En ligne ; Consulté le 14 janvier 2011].

[Tan04] Andrew Tanenbaum. *Systèmes d'exploitation*. Pearson Education, 2004.

[Wic03] Marcin Wichary. Biurko na ekranie. http://www.aresluna.org/attached/usability/articles/biurkonaekranie, 2003. [En ligne ; Consulté le 31 janvier 2012].

[wik12] Environnement graphique. http://fr.wikipedia.org/wiki/Environnement_graphique, 2012. [En ligne ; Consulté le 16 janvier 2012].

[Wro] Gunnar Wrobel. Linux and the keyboard. http://gunnarwrobel.de/wiki/Linux-and-the-keyboard.html. [En ligne ; Consulté le 31 janvier 2011].

[Zha09] Yang Zhao. Linux graphics driver stack explained. http://yangman.ca/blog/2009/10/linux-graphics-driver-stack-explained/, 2009. [En ligne ; Consulté le 31 janvier 2012].

[Ét06] Étienne. Une histoire du design interactif (première partie). http://www.my-os.net/blog/index.php?2006/11/26, 2006. [En ligne ; Consulté le 31 janvier 2012].

1

1. Note : en raison de la nature du sujet de notre recherche bibliographique (un domaine assez technique de l'informatique), la majorité des sources sur lesquelles nous nous sommes basés sont des ressources en ligne.