Chapitre 19

Interfaces Graphiques de Base

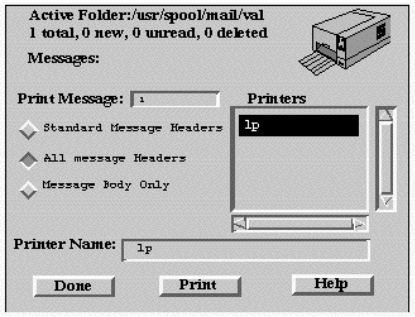
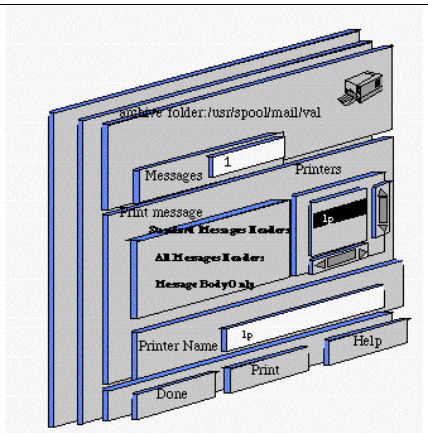
1. Interfaces Graphiques

- Une interface graphique c'est un ensemble de programmes permettant d'utiliser un système donné de manière plus souple et intuitive que les lignes de commande qui sont parfois difficiles à comprendre.

- Avec la rapidité croissante de l'internet, de plus en plus de personnes ou de sociétés aimeraient offrir des services ou contrôler divers équipements à partir du réseau.

- L'interface graphique coté utilisateur devra donc tourner indépendamment du matériel et va agir comme un simple client, recevant et envoyant des données critiques (nécessitant des mesures de sécurité appropriées) ou non critiques.

2. La face cachée d'une interface graphique

Quand vous voyez ceci	En réalité vous avez codé ceci
	

- Il existe donc des composants graphiques qui en contiennent d'autres (et gèrent leur apparition, leur positionnement, ...) et d'autres qui n'en contiennent pas comme les boutons poussoirs, les fenêtres textes de saisie, ... Ces derniers sont parfois appelés les contrôles.

3. Environnement fenêtré

- C'est une surcouche d'un système d'exploitation.
- Il présente les bases fonctionnelles d'une interface graphique qui sont:
    - Le système de fenêtres (Windows),
    - Les icônes (Icons),
    - Les menus (Menus),
    - et un dispositif de pointage (Pointing device).
- Il porte le nom de WIMP (Windows Icons Menus Pointing).
- Il a été inventé par XEROX et sa sortie grand public a été faite grâce à Apple dans le milieu des années 80 à l'aide entre autres du Macintosh.
- Il est constitué le plus souvent de:
    - Espace de travail (bureau).
    - Ensembles de fenêtres (contrôler des programmes, lister le contenu de répertoires etc.).
    - Icônes (représentations graphiques d'une commande ou d'une application).

4. Gestion d'un environnement fenêtré

- C'est le module "Windows Manager" qui se charge suivant certaines règles établies de l'affichage des fenêtres, icônes, menus, barre des tâches, bureau virtuel etc.

5. Développement indépendant de la plate-forme

- Avant de se lancer dans la programmation proprement dite d'une interface graphique, il est essentiel de répondre d'abord à la question : « **Pour quel système d'exploitation doit-on développer cette application** ? »

- Est-ce uniquement pour Linux ? Windows ? Peut-être les deux ? Pourquoi pas Solaris ? Pourquoi pas les trois ? Peut-être d'autres etc., pour la faire fonctionner sur quel matériel : un PC, un SUN ou un Mac etc.

- Si l'indépendance est un facteur clé, il est important de regarder s'il existe des bibliothèques indépendantes des plateformes permettant de réaliser ces interfaces graphiques.

- A noter, qu'il faut tenir compte des coûts associés pour effectuer le portage de l'application vers d'autres systèmes.

6. Bibliothèques indépendantes

On distingue deux catégories :

- Bibliothèques natives : une bibliothèque spécialement écrite pour une architecture matérielle donnée (processeur, système d'exploitation etc.).

- On maximise l'utilisation des différentes routines et fonctionnalités associées au système d'exploitation.

- Le but de cette opération est d'avoir des interfaces graphiques et les événements associés (cliques, déplacements etc.) qui ressemblent « parfaitement » aux autres applications graphiques du système considéré.

- Un comportement et un look : l'idée est de programmer un comportement (bouton, menu, boite etc.) et un look (couleur, forme, mise en page etc.) particulier (comportement et un look : look and feel) qui s'affiche de la même manière peu importe le système d'exploitation utilisé.

7. Boite à outils (« toolkits »)

- Un ensemble d'outils conçus pour fonctionner les uns avec les autres de façon complémentaire.
- Ils sont regroupés dans une bibliothèque.
- Parmi ces outils, on peut citer : bouton, menu, boite à cocher etc. ainsi que des fonctions de haut niveau pour la gestion de divers événements (cliquer sur une souris, appuyer sur une touche d'un clavier etc.)
- Les deux points suivants sont des critères essentiels dans le choix d'une boite à outils :
    - Documents et support disponibles : pour qu'une boite à outils soit efficace, il faudra au moins qu'elle soit bien documentée et qu'elle ait une porte où frapper en cas de pépins !
    - Les coûts : la plupart des bibliothèques sont gratuites pour un usage personnel et parfois, c'est le cas aussi pour un usage académique. Mais dès qu'il est question de commercialiser un produit fabriqué à base d'une de ces boites, c'est toute une autre histoire. Il faudra bien lire la licence d'utilisation avant de se lancer dans la tâche.
    - Contraintes de distribution : là aussi il faut bien lire la licence d'utilisation de ces boites à outils avant de distribuer une interface graphique à base d'une de ces boites à outils !

8. Environnement de développement

- Environnement de développement intégré, réunissant tous les outils nécessaires à la création d'applications aussi complexes qu'elles soient.
- Il permet de simplifier (dans un sens mais pas dans tous les cas) le travail du développeur.

9. Langages de programmation

- Pour programmer votre interface graphique, vous aurez besoin d'une boite à outils.
- La boite à outils a été programmée en utilisant un langage de programmation donné (C, C++, Java etc.).
- Ainsi donc le choix du langage de programmation à utiliser pour coder votre interface graphique peut être du même type que celui de la boite à outil. Comme il peut être différent, si la boite à outils le permet.

10. Boites à outils et Langages de programmation

- **C/C++ :**

  Qt : développée par « Trolltech ».
  Langage : C++.
  Disponible sous licence GPL ou commerciale.
  Disponible sous GPL depuis peu à la communauté Windows.
  Exemple : KDE, gestionnaire de fenêtre sous Linux. Navigateur Opera.

  wxWidgets : développée par la communauté « Open Source ».
  Langage : C++.
  L'ancien nom « wxWindows ».
  Disponible sous licence LGPL.
  Disponible depuis plus d'une dizaine d'années.
  Exemple : AOL Communicator.

  GTK+/GTKMM : développée par la communauté « Open Source ».
  Langage : C (gtkmm C++)
  Elle a vu le jour à cause des contraintes de licences associées à l'utilisation de Qt.
  Version Windows chaotique ! Il faut être sacrément très patient !
  Exemple : Gnome, gestionnaire de fenêtre sous Linux. Gimp (images).

- **Java :**

  - Développer par la compagnie SUN en 1995.
  - Indépendant de la plateforme.
  - AWT est le paquetage de base pour construire et manipuler des interfaces utilisateurs graphiques. Il est parmi les paquetages originaux de Java.
  - Swing un nouveau paquetage dont les composantes sont écrites, manipulées et affichées complètement en Java ("pur" java).

- **Et Microsoft ? :**

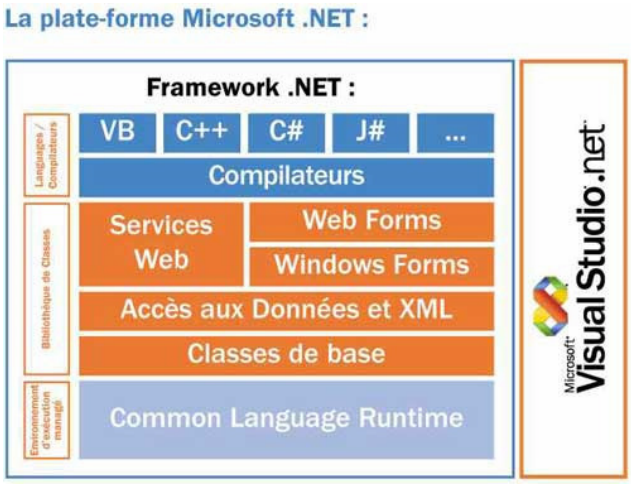
  - Avant 2001, deux possibilités ont été offertes (version 6 et moins): Visual Basic ou bien Visual C/C++ (MFC).
    - Visual Basic : pour des applications simples pas complexes de tous les jours.
    - Visual C/C++ (MFC : Microsoft Foundation Classes) : pour les interfaces plus complexes, s'il y a une forte interaction avec le gestionnaire de fenêtre à la Windows etc.

- .NET (depuis 2000)
    - Sa naissance Juillet 2000 et, le premier « framework » la version 1.0 a été rendue publique le 15 janvier 2002.
    - L'aspect le plus intéressant de « .NET » se situe au niveau de la plateforme de développement et des langages qu'il met en avant.
    - Il a permis d'unifier l'environnement de développement.
    - Avant son arrivé, le choix d'un outil ou d'une technologie de développement de Microsoft était une tâche assez ardue! Pour la simple raison que les solutions étaient vastes et pouvaient impliquées une des gadgets Visual Basic 6.0, que C++, VBScript, MFC, DCOM, ATL etc.
    - Par ailleurs, Microsoft perdait du terrain devant la concurrence de SUN et son langage Java.
    - Il fallait une solution intégrée, ouverte vers le web.
    - L'objectif est donc le développement de manière simple d'applications web interportables d'où l'arrivée de « .NET ».

Pour la suite de ce cours, nous allons décrire brièvement l'architecture de « .NET » afin de présenter quelques applications graphiques simples en utilisant « .NET ».

11. « .NET »

11.1 Architecture de la plateforme « .NET »



- Le Framework .NET contient deux composants principaux : la Common Language Runtime (CLR) et la bibliothèque de classes du Framework .NET[1].

  - Le CLR peut être considéré comme un agent qui manage le code au moment de l'exécution, fournit des services essentiels comme la gestion de la mémoire, la gestion des threads et l'accès distant.
  - Il applique également une stricte sécurité des types et d'autres formes d'exactitude du code qui garantissent un code sécurisé et robuste.
  - Le code qui cible le CLR porte le nom de code managé (managed) par opposition au code non managé (unmanaged).
  - Il supporte du code écrit dans plusieurs langages différents (C++, VB, C#, Pascal, Cobol ...) et simplifie le développement, la gestion et le déploiement d'applications. On peut la comparer à la Java Virtual Machine (JVM).

  - La bibliothèque de classes est une collection complète orientée objet de types réutilisables que vous pouvez utiliser pour développer des applications allant des traditionnelles applications à ligne de commande ou à interface graphique utilisateur (GUI, Graphical User Interface) jusqu'à des applications qui exploitent les dernières innovations fournies par ASP.NET.

[1] La description de cette architecture est extraite de la « FAQ » se trouvant à cette adresse. Une autre description intéressante se trouve à cette adresse.

11.2 Compilateur

- Les programmes (peu importe le langage utilisé) sont compilés sous la forme d'un code binaire intermédiaire, indépendant du matériel et du système d'exploitation.
- Ce langage est MSIL (Microsoft Intermediate Language).
- Le MSIL est traduit par la suite en code machine par un compilateur "Juste à temps" (JiT compiler) pour être exécuté par le CLR.

- Ce qui signifie que quelque soit le langage utilisé au départ (C++, C#, VB.NET etc.) dans votre programme, vos exécutables et DLL seront toujours déployés sous la forme de code MSIL.
- Il n'y a donc aucune différence entre un composant écrit en C++, C# ou VB .NET.

- Ainsi donc en réalité « .NET » « ne supporte » qu'un seul langage : MSIL! Pour votre confort, Microsoft vous propose d'écrire ce code MSIL avec une syntaxe Visual Basic, ou C++, ou C# etc.

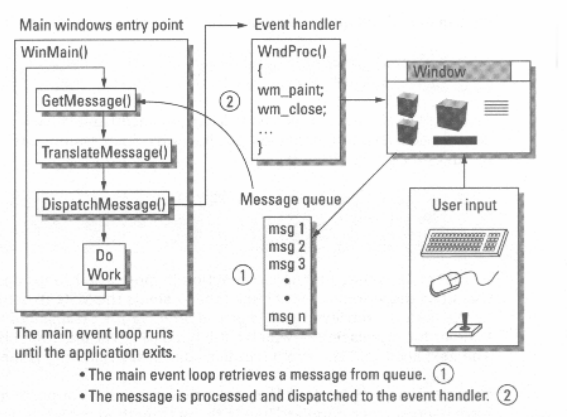
- Une autre conséquence directe, du fait que tous les langages de « .NET » sont compilés sous la forme d'un même code intermédiaire, est une classe écrite dans un langage et peut être dérivée dans un autre langage. On peut instancier ainsi dans un langage un objet d'une classe écrite dans un autre langage.

- Pour qu'un langage soit éligible au rang de langage supporté par la plateforme « .NET », il faut qu'il fournisse un ensemble de possibilités, de constructions qui sont recensées dans un contrat de service dénommé CLS (Common Language Specification).

- Pour ajouter un langage à « .NET », il suffit donc a priori qu'il satisfasse aux exigences de la CLS et que quelqu'un développe un compilateur depuis ce langage vers MSIL.

11.3 Gestion des événements dans Windows

API SDK,

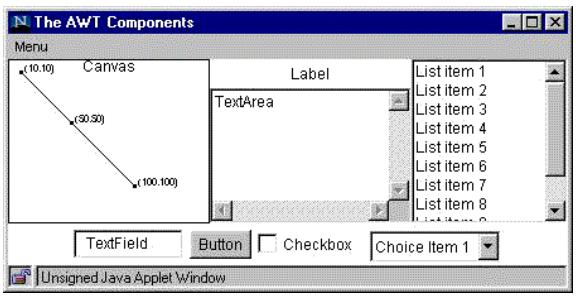


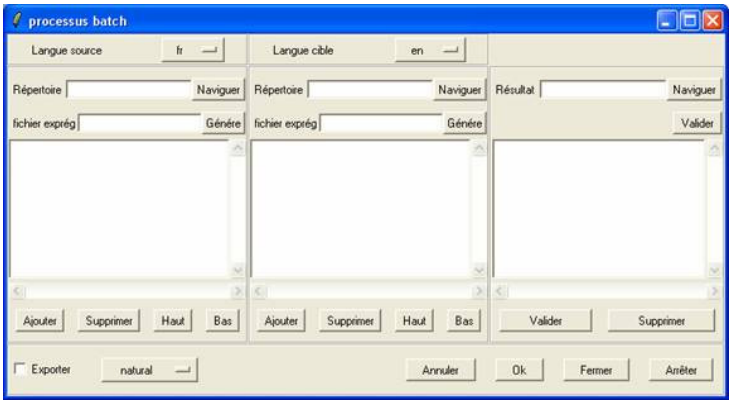
- Pour communiquer avec une application ou ses diverses fenêtres, Windows leurs envoie des messages.
- Windows va créer un message approprié qu'il va envoyer dans la file d'attente de l'application.
- La file d'attente est un tampon où sont stockés les messages en attente de traitement.
- À vous de coder, l'extraction des messages de la file d'attente (GetMessage), traduire le contenu (TranslateMessage) et les renvoyer (DispatchMessage) au bon gestionnaire d'événements associé à la fenêtre en question.

MFC/.NET

- Dans la plupart des cas, vous pouvez éviter de rentrer dans les détails et gérer par vous-même la file d'attente des messages. Ceci peut être simplifié par l'introduction de l'appel suivant « Application::Run(……); » dans le programme principal (WinMain).

11.3 Anatomie d'une fenêtre





11.4 Développement d'interfaces graphiques avec « .NET »

Quelques exemples à développer en cours ….