Notes en vrac

- Swing / GTK / QT reposent sur un modèle de “widgets” - des boites dans des boites qui forment des boites qu’on peut remettre dans des boites - avec une expressivité “bas niveau”. On définit “à la main” l’emplacement des différents éléments -> c’est naze.
- Une interface créée à l’aide de Swing / GTK / QT offre la possibilité d’adapter son aspect visuel à l’OS hôte (sans avoir à changer le code) -> intéressant.
- Swing / GTK / QT n’offrent pas une “portabilité” pour les interfaces créées : il n’est pas aisé de transporter une interface d’une appli à une autre sans avoir à beaucoup toucher au code -> peut-être intéressant ?
- Toujours sur la “portabilité” : Swing n’offre pas un langage intermédiaire indépendant des bindings pour différents langages hôtes. GTK offre une description XML des interfaces (à vérifier) -> concept intéressant. Quid de QT ?
- “Portabilité / adptativité” : les interfaces créées via Swing / GTK / QT ne s’adaptent pas automatiquement au support matériel : clavier / souris, écran tactile, quelle taille d’écran ... c’est naze !
- HTML / CSS offre justement cette approche, cependant des problèmes se posent dans le cas HTML / CSS : les différents navigateurs interprêtent de manières assez différentes les styles, est-ce un problème ?
- Bien que n’étant pas fait pour la création d’interfaces graphiques dans des applications autre que web, le couple HTML / CSS a une approche intéressante : séparer le style de la description des éléments.
- Swing / GTK n’offrent à priori aucune analyse statique de “débilité” d’une interface : des éléments toujours invisibles / non atteignables etc etc ...
- Swing est une horreur à débugger, quid de GTK / QT ?
- Swing est très verbeux, quid de GTK / QT ?
- Quid des performances de Swing / GTK / QT ?
- La gestion des actions est nulle dans Swing : verbeuse, assez peu intuitive. Quid de GTK / QT ?
- A compléter ... les idées sont à fouttre en vrac, il faudra trier ensuite.
- Interface résultante modelable par l’utilisateur.
- Debugging aisé pour le programmeur.
- etc etc ...

1 Qu’est ce qu’une interface utilisateur et comment en programmer

1.1 Définition des concepts liés à la notion d’interface utilisateur

Qui sont les utilisateurs ?

- Deux catégories : nous utiliserons suivante afin de distinguer les deux catégories d’utilisateurs : utilisateur application, et programmeur.
- Dans les deux cas, il faut avoir une approche orientée utilisateur, les besoins seront différents, mais l’approche à adopter la même.
- Utilisateur application : il faut prendre en compte à la fois l’aspect sensoriel (visuel si écran il y a) et l’aspect “contrôle” (gestion des périphériques d’entrée).
- Intéraction utilisateur application - programme : plusieurs “niveaux” de compétences parmi les utilisateurs avec des besoins différents.

Conception - génie logiciel

Utilisation de la conception centrée utilisateur qui est un modèle itératif.

- **Analyse :** analyse des besoins, un panel représentatif d'utilisateurs des deux catégories concernées doit être constitué afin d'établir les dits besoins.
- **Conception :** un prototype doit être conçu en fonction des besoins établis à l'étape précédente.
- **Evaluation :** sur la base du prototype réalisé, une évaluation est faite. Le procédé étant itératif, cette évaluation servira de base à la modification des besoins de la première étape.
- Il faut établir des "topic" d'évaluation - besoins, les deux sont liés :
  - Pour l'utilisateur application :
    - Vitesse d'apprentissage - aide nécessaire / intégrée - nombre d'erreurs commises lors d'un test.
    - Correction des erreurs.
    - Temps de réponse.
    - Efficacité - navigation rapide ?
  - Pour le programmeur :
    - Programmation intuitive.
    - Bonne expressivité - ne pas avoir un code verbeux à produire (capacité d'adaptation au support etc etc ?).
    - Typage fort - sûreté.
    - Debugging aisé - analyse de pertinence (détecter le maximum d'absurdités)
    - Intégration aux IDE populaires ? Création d'un IDE ?
- **Les outils :** des outils du génie logiciel peuvent nous aider à l'intérieur des phases décrites précédement :
  - Scénarios / diagrammes de cas d'utilisation : à la fois pour établir les besoins et faire l'évaluation au niveau de l'utilisateur application. Peut aussi être utilisé pour le côté programmeur bien que le point suivant me semble être plus approprié.
  - Diagrammes relationnels - diagrammes de classes : relations entre les différents objets de l'interface, est utile dans la phase de conception pour "se mettre à la place" du programmeur.
  - Design pattern : beaucoup d'interfaces utilisateurs utilisent des principes similaires d'intéraction avec l'utilisateur, l'implémentation "native" de comportements génériques peut être pratique.

1.2 Division du travail

La conception d'une interface graphique par dessus un moteur d'application peut se diviser en quatre parties qui devraient demeurer indépendantes le plus possible.

- Communication entre le moteur et l'interface :
  - Affichage : ensemble des données du moteur affichées (sous diverse forme) par l'interface ;
  - Actions : ensemble des actions de l'utilisateur modifiant l'état du moteur. Tout élément de l'interface permettant d'agir sur le moteur devrait être lié à une "Action". Une même Action doit pouvoir être effectuée par différents éléments d'une interface, et par différentes interfaces.

  Cette section doit être indépendante de l'interface finale (éléments, aspect, etc) et de la plateforme.
- Éléments de l'interface ; dépendant de la plateforme, lié à la partie précédente.
- Personnalisation des éléments : positionnement, taille, aspect...

  Une telle configuration est évidemment dépendante de la partie précédente.

  Certains paramètres de cette configuration doivent pouvoir être modifiés par l'utilisateur final.

  Ces configurations doivent pouvoir être enregistrées et s'échanger facilement.
- Aspect général.

2 Outils à développer

2.1 Bindings

Liaison d'une propriété d'un élément de l'interface graphique à une valeur du moteur de l'application : par exemple, la "clickabilité" (oui) d'un bouton à un booléen, le champs d'un label à une chaîne de caractères, la valeur d'un curseur à une valeur du moteur logique, le contenu d'un panel à une image...

La liaison doit être effective dans les deux sens : le changement du champs du label par l'utilisateur doit modifier la valeur de la chaîne de caractère, tout changement tiers de la valeur doit être répercuté dans le label de l'interface.

Solution : la librairie fournit à l'intention du moteur de l'application des classes pour des types simples comprenant un pattern observer à destination des éléments de l'interface.