# PROJET BAD TOUCH

Guillaume Gnaegi Tony Svenson Raphael Bösch

## Introduction

Le projet Bad touch a été implémenté dans le cadre du cours Interfaces multimodales. Il était demandé pour ce projet de mettre au point un éditeur de texte collaboratif utilisant plusieurs modalités. Dans le cas présent, l'éditeur fonctionne sur une diamond touch, une table avec un écran tactile ainsi que reconnaissance de l'utilisateur mais utilise aussi la reconnaissance vocale, car il est possible de donner des ordres au système directement à la voix.

## Architecture

### Diamondspin

**http://diamondspin.free.fr/**

Afin de donner un aspect agréable et nous éviter de devoir programmer l'ensemble des fonctionnalités pour la diamond touch, nous avons pu utiliser le toolkit diamondspin mis au point par Fred Vernier. Le gros avantage de diamondspin c'est qu'il se base directement sur la librairie swing de java. En fait les classes les plus fréquentes en swing sont simplement complétées afin de pouvoir correspondre aux standards de la diamond touch. Ainsi lorsque que l'on veut créer une nouvelle JFrame que l'on trouve dans la librairie swing de java, il nous suffit d'utiliser un objet DSFrame de diamondspin. Cette JFrame étendue supporte totalement les différentes manipulations possibles sur une diamond touch, mais surtout « reconnaît » l'utilisateur qui l'emploie – exemple concret, la fenêtre change de couleur en fonction de l'utilisateur -.

Ce toolkit nous a permis de gagner du temps, car ainsi nous n'avions plus à nous consacrer aux problèmes d'implémentation liés à la diamondtouch (gestion des utilisateurs, gestion du pointeur, gestion des vues etc…). Dès lors nous avons pu tout de suite imaginer la conception graphique de notre éditeur et réfléchir aux possibles implémentations de la voix dans notre projet.

### Sphinx

Sphinx est un utilitaire de reconnaissance vocale open source et surtout, dans sa version 4, implémenté en java. De ce fait il a été très facile d'intégrer sphinx à l'ensemble du projet. Néanmoins, nous avons remarqué les faiblesses de sphinx comme par exemple que celui-ci était limité au niveau de la longueur des ordres que l'on pouvait lui soumettre sans qu'il y ait confusion. En fait le risque de fausse redirection avec sphinx est assez élevé, du moins si les différents ordres n'ont pas été testés au préalable et le cas échéant modifiés afin que tous ait une probabilité de reconnaissance à peu près équivalente.

Comme vous pourrez le constater par la suite, afin que les ordres soit bien reconnus, certains ont été allongés (ceux qui risquaient d'être le plus souvent reconnus) ou écourtés (ceux qui par contre avaient à chaque fois une faible chance de faire un score suffisant lors de l'évaluation par sphinx). A l'heure actuelle le système reconnaît une vingtaine d'ordres soit :

***Copie du texte:*** copy

***Collage du texte:*** paste

***Couper le texte:*** cut selection

***Effacer du texte:*** ((delete selection)|(remove selection)| (erase selection))

***Déplacer du texte:*** ((move selection)|(move)|(drag selection)|(displace))

***Choix de la partie que l'on veut éditer:*** ((take part one)|(take part two)|(take part three) | (take part four))

***Couleur du texte:*** ((text color blue)|(text color red)|(text color green)|(text color orange)|

(text color black)|(text color pink) | (text color magenta) | (text color yellow))

***Mise en evidence :*** ((highlight green) | (highlight pink) | (highlight orange) | (highlight yellow))

***Alignement du texte:*** ((align left) |(align center) |(align right) | (align justify))

***Taille de l'éditeur:*** ((bigger window)|(smaller window))

***Rotation:*** rotate window

Afin de pouvoir utiliser sphinx dans les meilleures conditions, nous avons volontairement restreint les possibles interactions entre le système de reconnaissance vocale et l'utilisateur. Ainsi un utilisateur ne pourra donner qu'un seul ordre à sphinx à chaque fois qu'il pressera sur le bouton speak (nous reviendrons en détail sur ce point dans GUI Editeur). Ceci a deux gros avantages. D'une part on évite d'avoir un Thread sphinx toujours en fonctionnement et d'autre part on peut assurer une reconnaissance vocale de meilleure qualité.

## JMF

Dans la version de base de notre projet, il était question de faire de la dictée directe au système de reconnaissance vocale et que celui-ci la transcrive directement en texte. Néanmoins nous avons eu le problème que sphinx n'est pas assez efficace pour des ordres très long et qu'il nous aurait fallu refaire un modèle de parole si nous voulions l'utiliser dans notre projet (ce qui prend un temps considérable, ce que nous n'avions pas à disposition).De ce fait, nous avons utilisé la possibilité de faire l'annotation de texte en enregistrant directement un fichier de type wav -à l'heure actuelle l'enregistreur est terminé, mais il n'a pas encore été intégré au projet-. Pour implémenter ce projet nous avons utilisé directement la librairie JMF qui est en fait un plus multimedia des librairies java standard.

## Gui Editeur

Comme nous l'avons déjà précisé plus haut notre projet a pu être implémenté plus facilement grâce à diamondspin. Après avoir un peu fait connaissance avec les différents objets de diamondspin, nous avons pu nous atteler à la conception de notre éditeur. Mais avant de parler des fonctionnalités de l'éditeur, faisons un petit état des lieux. L'éditeur est composé d'une barre de menu que l'on trouve au bas de la surface de travail de chaque utilisateur, d'un clavier « virtuel » (l'utilisateur tapant directement les lettres sur la diamond touch) ainsi que d'une surface d'édition (L'utilisateur choisit au début de la réunion quelle partie du texte il veut travailler).

### *Barre de menu*

La barre de menu contient quelques fonctionnalités intéressantes dans l'édition de texte, comme par exemple le choix des fontes, des tailles de fontes, des styles, de l'alignement ou encore de la couleur du texte. Il est bon de noter que toutes ces fonctionnalités ont été développées pour le projet et ne font en aucun cas partie du toolkit diamondspin. Ainsi nous avons par exemple implémenté un système de JList qui répertorie l'ensemble des polices présentes sur le système et qui permet le cas échéant à l'utilisateur de choisir la police qui lui convient. Ceci en va de même pour les tailles de police ou encore les différentes couleurs applicables au texte (dans ce cas-là nous utilisons directement un utilitaire de java que nous avons modifié pour correspondre à nos attentes)

### *Clavier virtuel*

Afin de pouvoir éditer du texte, l'utilisateur a besoin d'une interface idoine. Ainsi nous mettons à sa disposition un clavier virtuel - qui est en fait un assemblage de JButton – qui lui permet d'éditer le texte, via la reconnaissance tactile de la diamond touch. Pour finir, sur chaque clavier figure un bouton speak. Ce bouton ne peut être activé que par un utilisateur à la fois et permet l'utilisation de la reconnaissance vocale via sphinx.

***Surface d'édition***

Pour que l'utilisateur puisse modifier le texte sans encombre, il faut qu'il ait une surface d'édition à disposition qui lui permette de visualiser les changements opérés ainsi que son clavier virtuel soit rattaché à cette surface d'édition. C'est en fait le cas dans notre projet, pour chaque clavier, il existe sur la surface de travail une surface d'édition qui lui correspond. De même si l'utilisateur quitte sa surface d'édition et qu'il désire reprendre une autre partie du texte principal alors une nouvelle surface d'édition lui sera assignée et le clavier virtuel y sera rattaché. Notre éditeur de texte utilise un type de Document RTF particulier qui génère à chaque création de document tous les styles qui peuvent être créés sur le système hôte (seule restriction le système doit fonctionner sur windows, car le chemin d'accès est donné pour un système windows). Dès lors il est possible d'éditer un texte avec toutes les polices du système

## Fonctionnalités

Nous allons maintenant explorer les différentes fonctionnalités offertes par le système. Nous allons par contre analyser seulement les fonctionnalités qui ne sont pas standard à un éditeur de texte normal.

***Ouverture, fermeture et sauvegarde de fichier***

Lorsque l'on veut démarrer une édition de texte, l'utilisateur responsable (un seul utilisateur a la possibilité d'ouvrir et de fermer des fichiers) doit ouvrir un fichier RTF correspondant aux standards pour notre éditeur. En fait celui-ci doit contenir les balises nécessaires pour que l'utilisateur puisse ensuite obtenir sa partie de texte. Ces balises sont les suivantes : ←PART1→ ←PART2→ ←PART3→ ←PART4→ et ←END→. Si ces balises ne sont pas présentes alors il est impossible d'obtenir une partie de texte pour chaque utilisateur.

Lorsque l'on veut terminer l'édition de texte et finalement quitter l'application il faut tout d'abord que les utilisateurs ferment leur éditeur de texte, lors de la fermeture de celui-ci le texte modifié sera ajouté à la partie qui leur correspond sur le texte principal. Ensuite, il suffit à l'utilisateur responsable d'appeler dans le menu File de la barre de menu l'onglet save (qui ouvre un outil de java pour le choix de fichier et la sauvegarde de ceux-ci).

Petit bémol : les onglets du menu File utilise un JFileChooser pour permettre de naviguer, et sélectionner un fichier. Ce JFileChooser n'étant pas de type DiamondSpin il n'est pas possible de l'utilisé en restant assis à la table, mais il est nécessaire d'utiliser la souris de l'ordinateur.

***Choix de la partie à éditer***

Une fois que le fichier texte principal est chargé, il suffit à l'utilisateur de donner un ordre vocal au système pour obtenir sa partie de texte. Ainsi il lui suffit de presser le bouton speak sur le clavier et de dire « take part one », « take part two » « take part three » ou « take part four ». A ce moment là, la partie du texte concernée sur l'éditeur principal est alors déplacée sur l'éditeur de l'utilisateur.

***Ordres vocaux***

Tout au long de l'édition de texte il est possible de donner un certain nombre d'ordres vocaux au système. Ceux-ci ont déjà été répertoriés plus haut. Néanmoins, nous allons nous attarder sur les différents ordres que sont copy, cut selection, paste et move.

Pour la plupart des ordres qui concerne le texte (couleur de texte, mise en évidence etc..) il s'agit d'abord de sélectionner la partie de texte que l'on veut éditer et d'ensuite donner un ordre vocal au système en pressant le bouton speak.

Lorsque l'utilisateur effectue un copy ou un cut selection sur son éditeur, alors le texte est automatiquement enregistré sur un buffer ad-hoc utilisé pour notre projet. Ainsi lorsqu'un autre utilisateur effectue la commande paste sur son système, alors il obtiendra le dernier bout de texte enregistré sur le buffer.

En ce qui concerne la fonctionnalité move, celle-ci permet d'afficher dans une Frame supplémentaire le contenu de texte sélectionné. De ce fait il est possible de garder un extrait de texte dont on ne sait pour l'instant que faire, ou que l'on pense que celui-ci doit être retravaillé mais que l'on ne veut pas s'y atteler de suite. Lorsque l'on veut effectivement garder ce texte et le placer dans notre éditeur, il suffit alors de faire un drag n'drop de la Frame contenant le texte déplacé sur la surface d'édition. Le texte sera alors ajouté après la position actuelle du pointeur.

## Conclusions

Le projet a été passionnant à implementé, et nous avons pu de ce fait nous familiariser avec le concept de multimodalité. Néanmoins nous pensons que quelques points négatifs viennent un peu ternir ce projet. Tout d'abord, nous sommes en fin de master et nous n'avons pas encore intégré l'utilisation d'un svn comme un impératif pour le bon déroulement d'un projet. Sans cela nous sommes un peu déçus de n'avoir pas pu implémenter complètement notre projet.