T A S C P A D

TOUCH **A**ND **S**PEECH
COMMANDED
PHONE **A**ND **D**IRECTORY

Joel ALLRED & Julien THOMET

27 mai 2009

Interfaces Multimodales / Semestre de printemps 2009

Université de Fribourg

Table des matières

1 INTRODUCTION 3

2 MODALITÉS 3

3 OUTILS 3

3.1 *DiamondTouch* 3
3.2 *DiamondSpin* 4
3.3 Windows SAPI 4
3.4 *FreeTTS* 4
3.5 *Skype* 4
3.6 *Phidgets* 5
3.7 Autres 5

4 FONCTIONNALITÉS 5

4.1 Affichage des contacts 5
4.2 Marquage 5
4.3 Filtrage 5
4.4 Suppression 5
4.5 Copie 5
4.6 Affichage des détails 5
4.7 Appel 6

5 IMPLÉMENTATION 6

5.1 Processus 6
5.2 Gestes 6
5.3 Reconnaissance vocale 7
5.4 Synthèse vocale 7
5.5 *Skype* 7
5.6 *Phidget* 7

6 PROBLÈMES ET LIMITATIONS 7

7 AMÉLIORATIONS POSSIBLES 8

8 CONCLUSION 8

1 Introduction

Dans le cadre du cours *Interfaces Multimodales*, un projet a été réalisé afin de découvrir le monde de la multimodalité à travers des périphériques et des librairies existants.

Tous les téléphones portables d'aujourd'hui permettent d'utiliser des commandes vocales pour appeler les contacts du carnet d'adresses en prononçant le nom de l'appelant ou un mot-clé préenregistré ; certains, comme *l'iPhone*, permettent également d'interagir avec la plupart des fonctionnalités du téléphone par commandes vocales.

Dans le contexte d'un salon intelligent, l'idée du projet est de concevoir une interface multimodale permettant d'interagir avec un téléphone (fixe) et un carnet d'adresses à la fois grâce à la voix et à une surface interactive.

2 Modalités

L'un des objectifs principaux du projet est de coupler différente modalités dans une même application. Dans *TASCPAD*, quatre modalités sont utilisées, deux en entrée et deux en sortie :

- tactile et visuelle via la *DiamondTouch*,
- vocale via des commandes vocales,
- auditive via un système de synthèse vocale.

Le concept de multimodalité devient intéressant lorsque les modalités sont utilisées en parallèle de manière combinée (synergie). Ainsi dans *TASCPAD*, les commandes vocales ne sont activées que si une destination (contact ou *tag*) est sélectionnée sur la *DiamondTouch*. Le geste de marquage est également synergique puisqu'il nécessite le pointage de deux objets différents (un *tag* et un contact) en même temps.

3 Outils

3.1 *DiamondTouch*

L'élément central du projet *TASCPAD* est la table interactive *DiamonTouch* distribuée par *Circle Twelve* Inc.. *DiamondTouch* est la première table tactile multiutilisateur, avant tout dédiée à l'aide à la collaboration pour de petits groupes. La plupart des tables tactiles se basent sur la pression exercée sur la surface pour déterminer les points de touche. La *DiamondTouch* quant à elle intègre sous sa surface un quadrillage d'antennes transmettant des signaux radios à très faible fréquence. Lorsque l'utilisateur touche la surface, l'utilisateur couple une partie des signaux provenant de quelques antennes à un récepteur connecté à sa chaise. Puisque chaque récepteur est unique, on peut aisément savoir qui fait quoi.

Circle Twelve Inc. fournit un SDK assez complet permettant de développer des applications pour la *DiamondTouch*. Cependant, son utilisation s'avère très complexe lorsque l'on désire rapidement développer des interfaces graphiques comportant des fenêtres, des claviers virtuels, … À cet effet, nous avons utilisé une extension de ce kit de développement : *DiamondSpin*.

3.2 *DiamondSpin*

DiamondSpin est l'un des principaux *Tabletop Toolkit Project* étendant les fonctionnalités de la *DiamondTouch*. L'une des fonctionnalités principales de ce *toolkit*, est l'adaptation automatique de l'orientation des objets affichés sur la table, sachant que les utilisateurs ont tous un point de vue différent. Mis à part cela, *DiamondSpin* offre un kit de développement graphique complet. Parmi les fonctionnalités principales l'on peut citer les opérations de zoom, de rotation et de redimensionnement sur les objets `DSFrame` (similaires aux `JFrame` de Swing), la possibilité d'intégrer un clavier virtuel (`DSKeyboardPanel`) ou encore d'annoter directement des éléments (`DSStroke`).

DiamondSpin ne fut toutefois pas aisé à prendre en main à cause de son architecture en couche et particulièrement les classes `DSTabletop` et `DSView` qui complexifient le rafraîchissement complet de l'affichage.

3.3 Windows SAPI

Concernant la reconnaissance vocale, l'idée initiale était d'utiliser *Sphinx-4*, un système de reconnaissance vocale entièrement en *Java*. Malheureusement, ce système n'est pas utilisable pour la reconnaissance de commandes en temps réel.

À la place, nous avons utilisé la reconnaissance vocale intégrée à *Windows Vista* (et configurable sous *Windows XP*), *SAPI* (*Speech Application Programming Interface*) version 5.1 qui est très efficace et prompt à recevoir des commandes composées généralement d'un ou deux mots.

Pour pouvoir accéder à cet *engine* de *Microsoft* via un programme *Java*, nous avons utilisé un « pont », *Quadmore Java To Microsoft SAPI bridge for Windows* (version 2.5). Ce pont utilise l'interfaçage *JNI* (*Java Native Interface*) afin d'intégrer du code « natif » *C* et *C++* dans un projet *Java*. Sa configuration est aisée et bien documentée ; l'appel au système se fait simplement à l'aide d'une méthode récupérant le dernier mot dicté.

3.4 *FreeTTS*

Bien que *Microsoft SAPI* fournisse également des fonctionnalités de synthèse vocale, sa configuration via *Quadmore Java To SAPI* n'a pas abouti. Nous nous sommes donc tournés vers une implémentation gratuite de *Java Speech*, à savoir *FreeTTS*. Ce dernier est un système de synthèse vocale programmé entièrement en *Java* et basé sur *Flite*, un système de synthèse en temps réel développé par à l'Université Carnegie Mellon (*CMU*).

3.5 *Skype*

Skype est le logiciel de téléphonie *VoIP* par excellence. Il fournit un service gratuit de voix par *IP* entre utilisateurs possédant un compte *Skype*, mais offre également la possibilité d'appeler des téléphones fixes ou mobiles sur n'importe quel réseau.

En plus de fournir un logiciel robuste, *Skype* propose un *API* décliné pour plusieurs langage de programmation.

Dans le contexte du projet *TASCPAD*, seules deux fonctionnalités ont été utilisées : appeler un contact et terminer un appel.

3.6 *Phidgets*

Les *Phidgets* sont des « blocs » *plug and play* (senseurs et contrôleur) possédant une interface *USB*. Un *API* complet permet d'utiliser facilement ces périphériques à l'aide de plus d'une dizaine de langage différents. Dans le cadre de notre projet, nous avons utilisé deux *Phidgets RFID*.

3.7 Autres

TASCPAD a été développé en *Java* à l'aide de l'environnement de programmation *Eclipse*. La *DiamondTouch* était connectée à un *PC* tournant sous *Windows XP*, aucun pilote pour *Windows Vista* n'étant disponible lors du développement.

4 Fonctionnalités

TASCPAD est un carnet d'adresses intuitif couplé à un système de téléphonie (*Skype*), sous la forme d'un écran tactile. Un registre général de contacts est stocké sur un serveur. Chaque utilisateur possède un carnet d'adresses personnalisable, lui aussi stocké sur le serveur. Tout ce système est entièrement transparent à l'utilisateur, qui peut accéder à son répertoire simplement en approchant sa carte *RFID* d'un récepteur du *TASCPAD*.

Les sections suivantes décrivent les fonctionnalités disponibles dans *TASCPAD*.

4.1 Affichage des contacts

Approcher sa carte *RFID* du lecteur permet d'afficher sur la table l'image des contacts les plus souvent appelés. Lorsque le tag est retiré, les images sont effacées de la table et les modifications apportées sont enregistrées sur le serveur.

4.2 Marquage

La commande vocale « *tags* » permet d'afficher une liste de *tags* personnalisés sur la table. Ces derniers peuvent ensuite être assignés a des contacts en sélectionnant le *tag* d'un doigt, et l'image du contact d'un autre.

4.3 Filtrage

Il est possible de filtrer les contacts qui s'affichent sur la table, soit par nom, soit par *tag*. La commande « *filter tag* », suivie d'un nom de *tag*, filtre le carnet d'adresses par rapport à ce *tag*. La commande « *filter name* » fait la même chose avec le nom des contacts. La table affiche toujours d'abord les contacts les plus fréquents.

4.4 Suppression

La commande vocale « *remove* », en maintenant le doigt sur l'image d'un contact, efface ce dernier du répertoire.

4.5 Copie

Pour partager un contact avec un autre utilisateur de la table, il suffit de glisser l'image du contact vers le carnet d'adresses de l'autre personne.

4.6 Affichage des détails

En pointant sur l'image d'un contact et en prononçant « *details* », la table affiche toutes les informations sur ce contact dans une boîte, ainsi que les tags qui lui sont assignés. Il est également

possible grâce à cette vue de retirer un *tag* d'un contact en sélectionnant un *tag* et en utilisant la commande vocal « *remove* ».

4.7 Appel

Les commandes vocales « *phone* » ou « *call* » établissent une communication téléphonique via *Skype*, avec le contact pointé du doigt.

5 Implémentation

5.1 Processus

L'implémentation s'articule autour de quelques processus légers principaux ; la première classe importante est `MainThread` qui représente le processus récupérant les événements de gain et de perte de *tag RFID* sur les détecteurs. Lorsqu'un *tag* est détecté par un *Phidget*, un `User` est créé en récupérant l'identifiant du *tag*. Finalement un objet `UserThread` est instancié et lancé.

Un tel objet a accès à toutes les informations courantes concernant un utilisateur. En intégrant un objet `Workspace`, le processus peut aisément récupérer un `AddressBook` ou un contact. En plus de gérer les données utilisateurs, le `UserThread` gère l'affichage pour un côté de la *DiamondTouch*, ajoute un *listener* pour les commandes vocales auprès d'un `SpeechModifier` disponible et un autre *listener* pour les gestes de marquage et de sélection.

Les `UserThread` ne communiquent pas directement entre eux. Une référence vers chacun des *threads* est détenue par `MainThread` afin de synchroniser les comportements lors de gain ou de perte de *tags RFID*.

Le processus léger suivant concerne la récupération des points de touche ; un objet `BBThread` (*BB* pour *Bounding Box*[1]) est instancié pour chaque `UserThread` auquel est attaché un *listener* qui permet d'être averti d'un ou de deux nouveaux points de touche. Sachant que la *DiamondTouch* ne renvoie qu'une *bounding box*, nous avons défini une surface maximale pour laquelle la *bounding box* désigne un seul point de touche. Lorsqu'un point est détecté, il est sauvegardé dans l'état du `BBThread` et est utilisé lorsqu'une *bounding box* plus grande est détectée. Connaissant le premier point de touche, on peut aisément récupérer le second.

Finalement `SpeechModifier` vérifie la capture de nouvelles commandes vocales.

5.2 Gestes

Les gestes décrits plus haut ont été implémenté comme `BBListener`. Ainsi par exemple, la classe `TagGestureListener` implémente le geste permettant à l'utilisateur de marqué un contact avec l'un des *tags* disponibles défini dans son `AddressBook` ; lorsque deux point de touche sont détectés, on récupère les éléments de type `DSFrame` qui contiennent les points. Si l'une des fenêtre est un contact et l'autre un *tag*, alors on les associe. Sinon, rien ne se passe.

Le geste de copie fonctionne différement. Sachant que seul `MainThread` connait les deux *threads* des utilisateurs, c'est dans cette classe qu'il faut implémenter la copie de contact. L'événement de copie est quant à lui directement implémenté dans le comportement des `ContactImage` (objet de

[1] Une *bounding box* représente un rectangle contenant les points de touche d'un utilisateur.

type `DSImage` spécifique à un objet `Contact`) à l'aide de la méthode `elementDropped` du *listener* `DSElementListener`.

5.3 Reconnaissance vocale

La reconnaissance vocale de *Microsoft SAPI*, via *Quadmore* est simple. Comme cité précédemment, c'est un processus léger de type `SpeechModifier` qui récupère les nouvelles commandes vocales et les envoie aux éléments concernés, à savoir les *threads* utilisateurs.

5.4 Synthèse vocale

La synthèse vocale est utilisée afin de donner un retour après une opération gestuelle ou une commande vocale. L'utilisation de *FreeTTS* se fait via un objet `Voice` sur lequel est appelée la méthode `speak` avec une chaîne de caractère en argument. Un retour vocal est émis à chaque dictée de commande (pour autant qu'aucun appel téléphonique ne soit en cours) et lorsque l'on filtre, copie, affiche les détails d'un contact, supprime un contact ou un tag, marque un contact avec un tag,...

5.5 *Skype*

Skype fournit un *SDK* très complet qui permet facilement d'utiliser les fonctionnalités de *Skype* à l'intérieur d'un programme *Java* (ou pour de nombreux autres langages). Les seules fonctionnalités utilisées dans ce projet furent l'appel d'un contact et l'annulation d'un appel. La première consiste en une méthode `call` appelée statiquement sur la classe `Skype`. Cette méthode renvoie un objet de type `Call` sur lequel la méthode `finish` peut être appelée pour mettre fin à l'appel.

5.6 *Phidget*

Un *SDK* est également fournit pour l'utilisation des *Phidgets*. L'architecture est événementielle est permet de récupérer, dans le cas de *Phidgets RFID*, deux types d'événement importants : `TagLossEvent` et `TagGainEvent` en implémentant un `TagLossListener` respectivement `TagGainListener`.

6 Problèmes et limitations

À l'origine, il était question d'implémenter un geste intéressant qui consistait en un « lancé » d'éléments d'un côté de la table vers l'autre côté afin de copier l'élément dans le contexte du second utilisateur. Pour cela, il fallait détecter la vitesse avec laquelle l'élément de la table était lancé. Un geste rapide aurait dû fournir une vitesse élevée et ainsi copier l'élément, alors qu'un geste « normal » aurait simplement déplacé l'élément. Nous avions commencé à étudier le code source du projet de master *Gesture Engine*, mais sans succès. La tâche était d'autant plus ardue puisque l'intégration de *Gesture Engine* avec *DiamondSpin* et le *SDK* original de la *DiamondTouch* devint vite un casse-tête, car certaines choses étaient présentes dans chacun des *kits*, mais programmé différemment.

L'autre problème récurrent auquel nous avons été confrontés fut la concurrence entre les différents processus. Nous nous sommes rendu compte de ces problèmes de synchronisation assez tardivement, lorsque l'application était utilisable par deux utilisateurs, ce qui implique qu'une partie n'a pas été résolue. Il en résulte un comportement relativement aléatoire principalement pour ce qui est de la copie de contacts, le marquage et la sauvegarde du carnet d'adresse.

Les limitations du projet s'articulent autour d'une remarque importante ; certes la *DiamondTouch* est multiutilisateurs, mais les périphériques que nous avons utilisés ne le sont pas. En particulier le microphone et les haut-parleurs. Ainsi, il n'est « pas » possible de gérer sont carnet d'adresse via des commandes vocales lorsqu'un second utilisateur est en train de téléphoner à l'aide du système. Le retour auditif pourrait également être divisé et rendu dans un casque pour chaque utilisateur de façon indépendante.

7 Améliorations possibles

Les améliorations possibles sont nombreuses, en témoigne les problèmes et limitations évoqués dans la section précédente. La première amélioration importante serait la séparation de tous les périphériques (microphone, haut-parleurs) et logiciels ad-hoc utilisés (*Skype*) afin de rendre le système 100% multiutilisateurs (de manière non restrictive).

L'intégration complète des fonctionnalités de *Skype* pourrait également être étudiée, notamment la connexion aux profils *Skype* lorsqu'un *tag RFID* est détecté ou la synchronisation des contacts de *Skype* avec ceux de la table. Outre la téléphonie, l'intégration d'un client e-mail élargirait l'éventail de fonctionnalités.

Bien que notre implémentation permette de copier ou supprimer des contacts, elle ne permet pas d'ajouter des contacts à un carnet d'adresse privé à partir d'un annuaire public. Une fonction de recherche serait nécessaire. Des cartes de visite *RFID* pourrait également être couplées à la table afin d'afficher les détails du contact, de l'appeler directement,…

Enfin, toutes les fonctionnalités des téléphones portables tactiles actuels pourraient être intégrées.

8 Conclusion

La conception de *TASCPAD* nous a permis d'approcher l'univers de la multimodalité de manière appliquée. Même si l'ensemble de fonctionnalités de départ paraissaient « maigre », les divers problèmes rencontrés, principalement pour la détection de la vitesse d'un lancé, ont complexifié considérablement l'implémentation. De plus, l'implémentation d'un carnet d'adresse même simple nécessita de nombreuses fonctionnalités de base.

Bien que la prise en main ait été laborieuse, la combinaison de différentes modalités fut motivante, en particulier la possibilité d'utiliser un périphérique tel que la *DiamondTouch* ou des *Phidgets RFID*.