

L'évolution de Linux Vers les Nouvelles Formes d'Ordinateurs Personnels

Stéphane CHATTY^{*,**}, Mohamed-Ikbel BOULABIAR^{***}, Benjamin TISSOIRES^{*}

**Université de Toulouse - ENAC, 7 avenue Edouard Belin, Toulouse, France*
chatty@enac.fr, benjamin.tissoires@enac.fr

***IntuiLab, Les Triades A, rue Galilée, Labège, France*
chatty@intuilab.com

****LAB-STICC, Telecom-Bretagne, Brest, France*
boulabiar@gmail.com

Abstract : Les tablettes tactiles représentent une première étape dans l'évolution de la forme des ordinateurs depuis le poste de travail graphique vers une plus grande variété. Il faut adapter les systèmes d'exploitation à une plus grande variabilité des moyens d'entrée, et à la remise en cause d'hypothèses de base telles que la présence d'un clavier ou d'un unique pointeur de désignation. Cela demande donc de multiples évolutions dans Linux, qui doit continuer à faire fonctionner les applications existantes. La gestion des entrées, dans le noyau et le serveur X, est modifiée pour tenir compte de flux d'information plus complexes. La reconnaissance de gestes est introduite, tant pour simuler le clavier ou la souris que pour proposer de nouveaux styles d'interaction. De nouvelles conventions d'interaction adaptées aux tablettes sont en cours de définition dans la distribution Ubuntu. Des modifications plus profondes de la gestion des entrées pour préparer les prochaines évolutions sont à l'étude.

Mots clés : Linux, système d'exploitation, entrées, tactile multiple, informatique ubiquitaire

INTRODUCTION

Le succès en 2010 des tablettes tactiles, après celui des téléphones tactiles, marque l'entrée dans une nouvelle ère que les chercheurs annonçaient depuis deux décennies : celle de l'informatique ubiquitaire [WEI 93]. Certains ont prédit des ordinateurs invisibles [NOR 99], qui écoutent nos pas dans le couloir pour s'assurer de notre présence ou surveillent nos gestes pour afficher les informations requises sur le bon écran. D'autres annoncent des ordinateurs qui restent dans nos poches et exploitent les moyens d'interaction disponibles là où nous sommes [OLS07]. On voit poindre ces évolutions, par exemple avec les nouvelles caméras 3D des consoles de jeu [OK10].

L'étape des tablettes, plus modeste en apparence, se prépare depuis plusieurs décennies [BUX 08]. Elle est suffisante pour remettre en cause notre usage des ordinateurs et l'architecture logicielle d'une partie de leurs systèmes d'exploitation. Depuis vingt ans les ordinateurs personnels étaient réputés posséder une souris et un clavier et la majeure partie des systèmes d'exploitation et des applications sont conçus sur cette hypothèse. Désormais, il devient normal

d'avoir un écran tactile, voire un écran tactile multiple, et des accéléromètres. Cela remet en cause de nombreux principes établis, depuis le noyau du système d'exploitation jusqu'aux logiciels applicatifs : la présence d'un périphérique connu à l'avance par le programmeur de l'application, la logique du "click, drag, double-click", l'unicité du pointeur, et jusqu'à l'utilité d'afficher un curseur sur l'écran. Et pourtant, il faut autant que possible permettre aux acquéreurs de nouveaux ordinateurs d'utiliser leurs logiciels habituels. C'est donc un défi technique qui touche toutes les couches logicielles.

Dans cet article nous recensons les évolutions requises dans les systèmes d'exploitation pour gérer le tactile multiple et le geste. Nous décrivons ensuite les travaux en cours dans Linux [BOV 02], auxquels participent les auteurs, et nous esquissons des perspectives pour le futur de la gestion des entrées dans Linux.

1. Du calculateur à l'ordinateur invisible

Les ordinateurs ont périodiquement changé de forme au cours de leur histoire. A chaque étape, les

moyens d'entrée ont changé : interrupteurs des premiers calculateurs, cartes perforées, terminaux et bandes magnétiques des ordinateurs centraux, postes de travail graphiques des ordinateurs individuels. L'étape actuelle est celle d'une multiplication des formes : tablette, console de jeu, mur interactif, etc. C'est elle qui mènera probablement à la disparition des ordinateurs dans notre environnement, quand il n'y seront plus discernables en tant qu'ordinateurs.

En laissant de côté son impact sur les usages de l'informatique, cette évolution a deux effets sur les systèmes d'exploitation et la programmation d'applications. Tout d'abord chaque forme laisse des traces profondes et durables dans la structure des systèmes d'exploitation. A chaque changement de forme, l'évolution est délicate tant cette influence touche des couches profondes dont la modification perturbe de nombreux programmes. Ainsi, la génération des ordinateurs centraux à terminaux déportés a donné naissance dans Unix au concept de terminal et de terminal virtuel (tty, vt), qui est toujours très présent dans les dérivés d'Unix et dans de nombreuses applications. Les bandes magnétiques et leur lecture séquentielle continuent d'influencer les primitives de lecture de fichiers (open, read). Le concept de focus dans les postes graphiques est probablement issu du concept de tâche active dans les terminaux. Quant au clavier, il bénéficie d'un traitement particulier dans le noyau Linux: pendant longtemps il a été le seul périphérique de saisie utilisable lorsque l'ordinateur est en mode "console", les autres périphériques étant des citoyens de seconde zone destinés à être traités en espace utilisateur. La souris avait à peine commencé à le rejoindre dans ce statut privilégié que sa présence systématique est maintenant remise en cause ! Il faut donc, lorsqu'on introduit de nouveaux périphériques, bien comprendre où les périphériques précédents ont laissé leurs traces et intervenir à tous ces endroits.

Le second effet de cette évolution est qu'on ne peut plus supposer que tous les ordinateurs sont équipés de la même manière. Pendant longtemps, tout programme pouvait supposer la présence d'exactly un clavier et une souris. Désormais, il faut soit que les applications puissent s'adapter aux circonstances, soit que le système d'exploitation les protège de cette variabilité. C'est le rôle, par exemple, des claviers virtuels ou des gestes qui remplacent le bouton droit de la souris sur certaines tablettes tactiles. A long terme, si les ordinateurs doivent s'interconnecter automatiquement avec les périphériques disponibles dans une pièce, il est probable que les programmeurs d'applications devront intervenir dans les choix effectués. A plus court terme, les écrans multi-tactiles et capteurs de mouvements sont suffisamment voisins des souris pour faire porter les efforts d'adaptation en priorité dans les systèmes d'exploitation.

2. Adapter Linux au tactile multiple

Nous décrivons dans [CHA 07] des services à rendre pour gérer l'adaptation à des périphériques multiples. La mise en oeuvre concrète de cette approche dans un système d'exploitation se heurte bien sûr à des obstacles pratiques. Ces derniers proviennent principalement du respect de l'existant du besoin d'une transition continue. Toutefois il n'est pas satisfaisant, comme c'est le cas parfois avec Microsoft Windows 7, qu'un utilisateur d'écran multi-tactile doive brancher une souris pour lancer ses applications parce que des parties entières du système d'exploitation ne savent pas traiter le multi-tactile. Il faut donc intervenir en de multiples endroits pour assurer la transition :

- dans le noyau pour intégrer la gestion de périphériques à structure plus complexe qu'une souris ;
- dans toutes les couches chargées du routage des événements d'entrée vers les applications pour leur faire propager les bonnes informations, mais surtout pour adapter les règles de routage : le principe du focus, c'est-à-dire de router vers une fenêtre tous les événements après qu'on a cliqué dans cette fenêtre, n'est plus adapté. Il faut aussi déterminer le niveau approprié pour intégrer la reconnaissance de gestes et les éventuels claviers virtuels ;
- et enfin dans l'environnement graphique, nommé Finder, Explorer ou encore Desktop selon les systèmes d'exploitation. La disparition du curseur et du pointeur unique y remet en question les principes établis de l'interaction icônique. Le tactile multiple y rend par ailleurs possible de nouveaux styles d'interaction basés sur le geste, à l'image du 'pinch-zoom' popularisé par les téléphones tactiles.

Les systèmes d'exploitation distribués par Apple et Microsoft ont intégré tout ou partie de ce support mais sans détailler les études et la démarche suivie pour le faire. Dans les parties suivantes de cet article nous détaillons les étapes en cours pour supporter les gestes multi-tactiles sur les systèmes à base de Linux. Nous nous appuyons en particulier sur les travaux effectués dans le système Ubuntu et sa pile de développement uTouch. Nous décrivons d'abord l'enrichissement de la gestion des entrées, puis l'introduction du geste multi-tactile, puis l'évolution du routage des événements (tactile ou geste) vers les applications.

3. Enrichir la gestion des entrées

Un accéléromètre comme celui qu'on trouve dans les tablettes permet des styles d'interaction nouveaux, mais ne remet pas en cause de nombreux principes de la gestion des entrées. Il faut seulement transmettre trois valeurs (X, Y, Z) au lieu de deux et s'assurer que leur signification n'est pas confondue involontairement avec les X et Y de la souris. C'est dans ce but qu'ont été introduites par le passé différentes modélisations abstraites des entrées, qui permettent de décrire un périphérique et ses caractéristiques, afin de savoir com-

ment les interpréter. Trois modèles différents sont mis en oeuvre dans les systèmes à base de Linux :

- le modèle HID, utilisé par les périphériques moderne pour se décrire eux-mêmes lors de leur branchement [USB01] ;
- le modèle evdev, utilisé par le noyau Linux pour décrire le périphérique aux programmes de l'espace utilisateur ;
- le modèle XInput, utilisé par le serveur X pour décrire le périphérique aux applications.

Ces trois modèles ont chacun leurs forces et leurs limites, mais reposent sur le même principe : la description d'un périphérique sous forme d'une liste d'attributs. Chaque attribut est décrit (valeurs min et max, précision, etc) et est associé à une entrée dans un vocabulaire standard défini par ailleurs. Le vocabulaire d'evdev consiste en une liste de définitions dans le fichier `input.h` ; on y trouve le X et le Y de la souris, celui des écrans tactiles, les différentes touches des claviers et boutons des joysticks, etc. Le vocabulaire de HID est géré sous forme d'un document géré par un comité, dont chaque page correspond à une catégorie de périphériques : environnement bureautique, joysticks, etc. L'accéléromètre a donc seulement nécessité l'ajout de nouvelles constantes dans `input.h`, charge aux applications de savoir les interpréter.

En revanche, les écrans multitactiles atteignent les limites des modèles que nous venons de décrire. Tout d'abord, lorsque trois doigts sont en contact avec l'écran, le périphérique va émettre trois n-uplets d'attributs (par exemple X, Y, et pression). On s'éloigne donc de la structure de liste décrite plus haut pour une structure d'arbre, dans lequel un même type d'attribut peut apparaître plusieurs fois. De plus, le nombre de doigts sur l'écran varie au cours du temps en fonction des actions de l'utilisateur. La quantité d'informations transmises varie donc, ce qui introduit un besoin de dynamique dans le modèle. Nous allons voir comment ces besoins ont été pris en compte dans le noyau Linux, tant dans la gestion du HID que dans l'évolution d'evdev. Les évolutions de XInput seront décrites plus loin dans l'article, car dans le serveur X le modèle d'entrée est mêlé au routage des événements qui est un autre sujet important.

3.1. HID et le tactile multiple

Le protocole HID n'est pas limité à la structure de liste évoquée plus haut. Il connaît la notion de "Collection", qui permet de regrouper des attributs. Dans leur description, les périphériques multi-tactiles annoncent donc une série de collections qu'ils nomment chacun "Finger". Il a suffi aux gestionnaires de HID d'ajouter à leur vocabulaire quelques entrées telles que "Contact Count" qui décrit le nombre de doigts présents à un moment donné, pour que les écrans multi-tactiles soient correctement gérés. La seule limitation résiduelle est l'absence de dynamique. Le protocole HID considère qu'un périphérique émet toujours la même quan-

tité de valeurs, et cette quantité est limitée par la taille de l'auto-description qu'un périphérique a le droit de fournir. Cela force donc les fabricants d'écrans multi-tactiles à décrire un certain nombre de doigts (2, 6, ou 10 selon les fabricants et le nombre d'attributs par doigt), à fournir des valeurs invalides lorsqu'un nombre inférieur de doigts est présent, et à envoyer plusieurs messages lorsque ce nombre est supérieur.

Le modèle evdev de description des entrées étant purement à base de liste, la gestion du HID dans Linux ignorait jusqu'à présent les collections. Un simple mécanisme de traduction d'attributs était mis en oeuvre, chaque attribut de HID étant associé à une entrée dans `input.h`. Outre que cette approche a contribué à l'inflation incontrôlée du nombre des entrées, elle a posé des problèmes pour la prise en charge des écrans multi-tactiles. Il fallait en effet retrouver la structure en collections que le gestionnaire de HID faisait disparaître. Dans un premier temps, chaque périphérique ayant sa propre structure, un traitement à base de machine à état a été mis en place dans un pilote spécifique pour chaque type de périphérique.

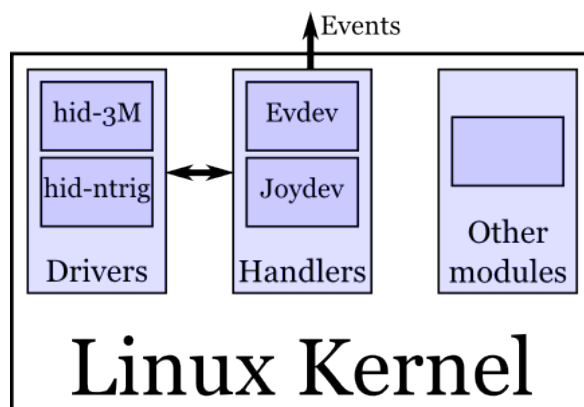


Figure 1. Gestion des entrées multi-tactiles dans le noyau Linux

Dans un second temps, nous faisons évoluer le gestionnaire de HID pour exploiter les collections et les événements répartis sur plusieurs messages, ce qui permet d'évoluer vers un pilote unique nommé "hid-multitouch". Ce travail est significatif de l'évolution nécessaire de la gestion des entrées, qui avec l'enrichissement des moyens d'entrée va devoir passer d'un mécanisme basique de gestion d'événements à un système plus riche de traitement de données. D'un simple modèle de liste d'attributs, le modèle d'entrée devra donc devenir à terme une description générale de données et de comportements.

3.2. evdev et le tactile multiple

Le travail décrit ci-dessus s'est effectué en parallèle avec une évolution nécessaire du protocole evdev utilisé par le noyau Linux pour transmettre les informations aux programmes. Cette évolution, discutée de

manière collective après des initiatives individuelles, s'est faite en plusieurs tentatives et n'a pas encore tout à fait convergé. Ces tâtonnements sont en partie une manifestation de la dynamique sociale propre aux projets de logiciel libre. Mais ils indiquent aussi que la complexité de la situation n'est pas encore bien évaluée, et que les modèles théoriques utilisés sont trop limités.

3.2.1. Protocole multi-tactile A

La première étape a consisté à introduire dans le protocole evdev la possibilité d'émettre plusieurs valeurs d'attributs similaires dans le même événement. En effet, le système de gestion des entrées de Linux contient un mécanisme de compression qui s'assure que la valeur d'un attribut n'est émise qu'une fois par événement, et seulement si cette valeur a changé. Ce mécanisme n'est pertinent que si un attribut ne peut apparaître qu'une fois par événement. Il a donc fallu introduire de nouveaux attributs insensibles à ce mécanisme de compression, et un nouveau message de synchronisation pour marquer la frontière entre les doigts.

Les nouveaux attributs ont en partie fait l'objet d'un débat pour tenter de capturer des caractéristiques communes à tous les périphériques multi-tactiles. Le résultat est un mélange du modèle proposé dans MPX [HUT 08] et de caractéristiques spécifiques à certains pavés tactiles :

- identification du contact : `BLOB_ID`, `TRACKING_ID`
- position : `POSITION_X`, `POSITION_Y`
- caractérisation du contact et de l'objet en approche sous forme d'ellipses : `TOOL_TYPE`, `PRESSURE`, `ORIENTATION`, `TOUCH_MAJOR`, `TOUCH_MINOR`, `WIDTH_MAJOR`, `WIDTH_MINOR`

Il ne s'agit que d'une évolution incomplète du modèle d'entrée, car la manière dont les périphériques sont déclarés reste une liste. Cela renforce le caractère partiel de la déclaration des périphériques dans evdev, qui décrit seulement les attributs qu'ils sont *susceptibles* d'émettre au lieu de préciser la forme des événements ou la structure des périphériques. Notons aussi que les nouveaux attributs sont parfois des doublons d'attributs existants (position, pression). Cela souligne la difficulté à gérer dans une unique liste (celle de `input.h`) un vocabulaire d'attributs qui, selon les périphériques et les situations d'usage, seront utilisés de manière différente. Les niveaux lexical, syntaxique et sémantique du protocole sont gérés dans la plus grande confusion.

3.2.2. Protocole multi-tactile Android

Le protocole A décrit ci-dessus comporte une erreur de conception : il ne permet pas d'indiquer quand un doigt quitte l'écran. La disparition du doigt dans les messages suivants est suffisante, mais encore faut-il qu'il y ait un message suivant : si le doigt relevé est la seule activité pendant plusieurs secondes, l'information restera indétectable pendant toutes ces secondes.

Android est un système d'exploitation pour téléphones tactiles, reposant sur le noyau Linux. Les écrans tactiles fournis avec ces téléphones sont en général multi-tactiles, et les fabricants fournissent des pilotes pour ces écrans. Dans la version 2 d'Android, qui intègre la gestion du tactile multiple, une convention a été employée pour contourner l'erreur décrite ci-dessus : l'attribut `WIDTH_MAJOR`, qui est le plus souvent inutilisé, est détourné pour indiquer sous forme d'un booléen si un contact a lieu ou non.

3.2.3. Protocole multi-tactile B

Le protocole B a été introduit dans des conditions similaires au protocole A, avec pour objectifs de corriger l'erreur et de réduire le nombre d'attributs émis. En effet, avec des périphériques qui détectent des dizaines de contacts, ceci génère une communication qui peut consommer beaucoup de bande passante, et de traitements pour les applications.

Un mécanisme de compression a donc été réintroduit, en utilisant le fait que la plupart des périphériques intègrent un algorithme de poursuite des contacts. Un contact est identifié de manière unique au fil des événements, et la valeur d'un attribut est donc émise seulement si elle a changé pour ce contact depuis l'événement précédent. Pour cela, en remplacement du message de synchronisation du protocole A, a été introduit la notion de créneau (*slot*). Un créneau est un contact unique, suivi au fil du temps par son identifiant. Ce dernier, un attribut nommé `SLOT_ID`, est réutilisé lors que le contact est terminé.

Contrairement au protocole A, le protocole B contient probablement toutes les informations fournies par les périphériques. Cela s'est fait au prix de la création d'un nouvel attribut, accroissant la confusion sur la sémantique des attributs. Par ailleurs, le numéro de créneau est une information supplémentaire qu'il faut calculer à partir des informations fournies par les périphériques. Cela introduit donc un calcul algorithmique dans le noyau, là où son rôle devrait être une simple traduction. De plus, une bibliothèque nommée `mtdev` est proposée pour les programmeurs d'applications afin de masquer la différence entre le protocole A et le protocole B. Pour les périphériques qui n'offrent pas la poursuite des contacts, cette bibliothèque extrait l'information manquante en utilisant appliquant l'algorithme d'affectation Munkres [KUH 55].

Tout ceci souligne tout d'abord la complexité introduite par la présence de plusieurs modèles d'entrée, tous aux limites de leur pouvoir d'expression : cela produit des "cicatrices" où des algorithmes sont nécessaires pour reproduire l'information manquante. Cela souligne aussi l'évolution du traitement des entrées depuis un simple relais vers une chaîne d'interprétation et de calcul. Cette évolution est particulièrement visible avec l'introduction de la reconnaissance de gestes, décrite dans la section suivante. Elle s'accroîtra probablement avec l'introduction des caméras et

l'introduction de l'informatique ubiquitaire, où saisie par l'utilisateur et interprétation par l'ordinateur viendront à se confondre.

4. Intégrer le geste multi-doigts

Les tablettes tactiles n'ont en général pas de clavier. Cela signifie qu'il faut proposer un moyen alternatif de saisie de texte, par exemple des claviers virtuels. Notons au passage que les claviers virtuels, tout comme certains gestes que nous évoquerons ci-dessous, soulèvent la question des *entrées simulées* : des couches logicielles de niveau élevé doivent pouvoir émettre des événements que l'on rencontre en général à bas niveau. Cela souligne l'importance d'avoir un modèle d'entrée suffisamment riche et explicite, mais aussi celle d'avoir un système de routage d'événements suffisamment flexible.

Mais l'absence de clavier implique aussi l'absence de touches spécialisées et de raccourcis au clavier. Combinée à l'absence de boutons de souris, cela restreint la capacité de l'utilisateur à indiquer ses intentions aux applications et aux couches supérieures du système d'exploitation : plus de menu du bouton droit, double clic moins naturel, etc. Parmi les alternatives possibles, les écrans tactiles multiples en apportent une nouvelle : les gestes multi-doigts. Nous décrivons ici la manière donc le composant GRAIL de la pile uTouch interprète les contacts pour produire des événements gestuels, nous concentrant sur la signification des gestes et leur reconnaissance. Nous reviendrons par la suite sur l'usage de ces événements dans la section sur le routage des événements.

4.1. Conventions et significations des gestes

Les pavés tactiles ont permis d'explorer l'utilisation de gestes à plusieurs doigts au cours des dernières années. Les premiers développeurs ont choisi des significations pour certains gestes, par exemple pour simuler l'existence d'un bouton droit de souris ou de touches de défilement de texte. Toutefois, il existe des différences entre pavés tactiles et écrans tactiles. Ainsi, un "drag" sur un pavé tactile permet de déplacer le document dans le sens contraire au mouvement des doigts et qui peut être imaginé comme le déplacement d'une fenêtre sur le document dans le même sens. Ce geste a le comportement inverse de celui utilisé pour les écrans tactiles qui sont des périphériques d'entrée directe. Cette signification opposée d'un même geste dans deux contextes différents force la réflexion. Est-ce que le modèle mental humain de la signification des gestes change du fait que c'est un périphérique d'entrée direct ou non ?

Pour un écran tactile, on veut déplacer l'élément qu'on est en train de toucher et voir en même temps. Mais tant qu'on ne voit pas l'élément en déplaçant les doigts sur le touchpad, le modèle mental s'abstrait au geste en lui-même tout en pensant qu'on veut dévoiler

la partie cachée du document, donc déplacer le voile qui le cache. Le signification opposée de ce geste se déduit de l'élément sur lequel il est effectué.

4.2. Gestes atomiques

Il a fallu définir un ensemble de gestes atomiques que le moteur de reconnaissance doit intercepter et que les couches supérieures peuvent utiliser pour reconnaître d'autres séquences plus complexes. Les quatre gestes atomiques choisis sont Contact, Déplacement (Drag), Pincement (Pinch) et Rotation (Rotate).

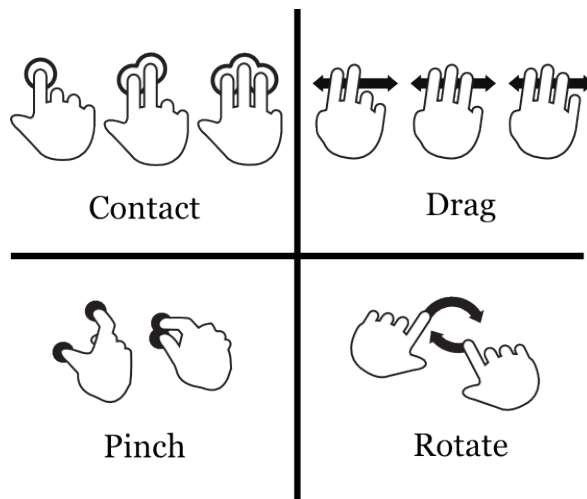


Figure 2. Les gestes atomiques détectés par le moteur de reconnaissance

Cet ensemble a inclus dans une première version le geste "Tap" au lieu de "Contact", mais il a été modifié puisque le Tap est un geste composé d'un contact puis d'un retrait des doigts et réalisé dans un délai d'attente inférieur à 300ms. La première version ne permet pas de construire des gestes comme "Tap and Hold" qui détecte la séquence Appui-Levée-Appui réalisé dans les mêmes contraintes de temps.

Chacun de ces gestes a un ensemble de paramètres physiques indiquant son centroïde, le delta des mouvements et la vélocité. Il est possible d'utiliser cet ensemble dans les couches supérieures pour composer des gestes plus complexes ou spéciales comme le "Flick". Un "Flick" peut être vu comme un déplacement avec une accélération supérieure à un seuil.

4.3. Moteur de reconnaissance

Le moteur de reconnaissance a des contraintes assez difficiles comme le temps de réponse pour retourner le geste reconnu et l'adaptabilité à des périphériques supportant un nombre variable de doigts. De ce fait, les méthodes utilisant une grande charge processeur sont écartés. Le moteur reconnaît le geste en appliquant des opérations simples de calcul géométrique.

La reconnaissance est indépendante du nombre de doigts impliqués, elle nécessite seulement la reconnaissance de la position des contacts dans le court de temps précédant.

Les gestes de rotation sont les plus complexes. Ils sont reconnus avec le calcul de la somme des différences en delta angle de la rotation des n doigts par rapport au barycentre.

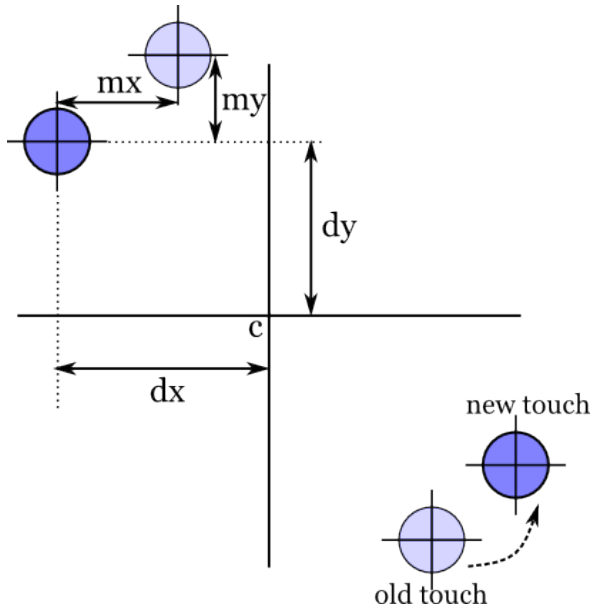


Figure 3. Les mesures pour dans le calcul de la Rotation dans le cas de 2 doigts.

$$rotation = \frac{1}{r^2} * \frac{1}{n} \sum_n (dx_i * my_i - dy_i * mx_i)$$

$$r = \sqrt{\frac{1}{n} \sum_n (dx_i^2 - dy_i^2)}$$

La notion simplifiée de la signification centroïd est tirée des travaux de Görg [GÖR 10] mais elle peut évoluer vers la notion de pivot qui est le point fixe lors d'une rotation en améliorant l'algorithme. L'utilisation du point pivot comme référence améliore les mesures et réduit les approximations, dans le cas où on a plus de 2 doigts impliqués, si une partie font une rotation combinée à un mouvement partiel.

Les gestes de type "Drag" sont reconnus avec le calcul de la moyenne des mouvements de l'ensemble des doigts, on y ajoute ensuite un filtre pour réduire le bruit. Le filtre utilisé crée une moyenne pondérée exponentielle augmenté par la réponse [THA 96].

Les gestes de pincement sont reconnus avec le rectangle englobant les contacts et en calculant la variation du changement de taille de ce rectangle.

Les gestes de rotation, de pincement et de déplacement sont émis de façon concurrente avec les

paramètres respectives. C'est la mission des applications de filtrer l'utilisation selon les seuil déclarés.

4.4. États de reconnaissance

Les 3 états principaux dans la reconnaissance des gestes sont :

- Gesture Start : Se déclenche lorsqu'on détecte un mouvement après la mise en place des doigts et le dépassement d'un seuil.
- Gesture Update : On continue à suivre l'évolution de la reconnaissance d'un geste et mettre à jour son état tant qu'on enlève pas les mains et qu'il y a un mouvement.
- Gesture Finish : On détecte la fin d'un geste par l'enlèvement des mains.

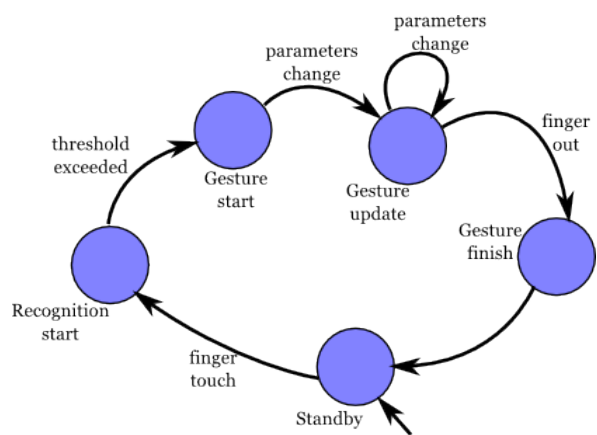


Figure 4. La FSM de la reconnaissance de gestes.

4.5. Instanciation des événements

Après la phase de reconnaissance, il faut aussi savoir émettre ces gestes reconnus, ceci nécessite savoir un minimum le contexte de l'entrée. Le contexte inclus la fenêtre où le geste est effectué ainsi que les gestes que le client veut recevoir. La fenêtre peut être détecté en regardant celle qui a le focus et qui contient le barycentre des doigts.

On cite aussi la notion importante des gestes tentatives, c'est la mise en mémoire tampon des événements de plusieurs alternatives de gestes jusqu'à ce qu'une correspondance soit confirmée. Le fait d'être nommée tentative vient de la possibilité d'annuler des gestes déjà reconnus si on n'a pas le contexte adéquat.

5. Adapter le routage des événements

L'un des rôles essentiels et méconnus des systèmes d'exploitation est d'adresser les entrées des utilisateurs aux bons programmes. Sur les terminaux textuels, ce travail reposait sur le concept de tâche courante (*foreground, background*) : le texte tapé au clavier était

adressé au programme actif. Sur les postes de travail graphiques, le routage repose sur le fenêtrage, le curseur de la souris, et le concept de focus : les événements sont adressés soit à l'application propriétaire de la fenêtre où réside le curseur de la souris, soit à l'application dont la fenêtre possède le focus. Le focus est modifié par des opérations explicites, comme le clic dans une fenêtre, soit par des actes implicites : lors d'un "drag", la fenêtre où l'opération a débuté possède le focus jusqu'à la fin de l'action.

Avec l'arrivée des tablettes multi-tactiles, le routage est appelé à changer de plusieurs manières : il n'y a plus de curseur, un geste peut se produire sur plusieurs fenêtres à la fois, et à terme il se peut même que le concept de fenêtre soit remis en cause. Or dans la plupart des systèmes à base de Linux, c'est le serveur X qui effectue le routage. La gestion des entrées y est intimement mêlée au système de fenêtres pour réaliser ce routage.

5.1. Le serveur X et le tactile multiple

XInput est le composant de gestion des périphériques d'entrée dans le serveur d'affichage X.org. Ce système est resté, jusque l'introduction de sa version 2, dans le paradigme d'une souris unique et d'un seul clavier. Les autres périphériques venaient se greffer de manière annexe autour de ces deux périphériques de référence. XInput 2.0 a permis au serveur X d'évoluer et de permettre d'utiliser plus d'une souris et d'un clavier sur un même serveur d'affichage.

5.1.1. La configuration des périphériques d'entrée

XInput 2.0 a repris les principes de MPX et introduit la notion de périphériques *maîtres* et *esclaves*.

Les *périphériques esclaves* correspondent aux périphériques physiques. Si l'on connecte plusieurs souris et claviers à un serveur X, on aura autant de périphériques esclaves que l'on a de souris et de claviers réels.

Les *périphériques maîtres* sont des périphériques "virtuels" qui sont la composition des périphériques esclaves. À chaque périphérique maître correspond un curseur graphique présenté à l'écran ainsi qu'un clavier associé à ce curseur. Ces périphériques virtuels permettent le routage des événements émis par les périphériques physiques vers les applications pointées par le curseur. Ils permettent aussi de maintenir la cohérence des couples claviers/souris grâce à l'association d'un clavier virtuel aux curseurs.

Cette combinaison des périphériques maîtres et esclaves permet aux utilisateurs de configurer leur environnement en fonction des périphériques d'entrée à leur disposition.

5.1.2. Transmission des événements

Les applications adaptées pour utiliser XInput 2.0 peuvent s'abonner aux événements émis soit par

les tous périphériques maîtres, soit par tous les périphériques esclaves, soit par un périphériques (maître ou esclave) spécifique. L'émission des événements se fait en plusieurs temps :

- le noyau émet les événements physiques pour chacun des périphériques,
- le serveur X digère ces événements et les traduit éventuellement (par exemple les événements "moulette" de la souris sont des événements composites de plusieurs modifications de l'axe Z) avant de les réémettre vers les applications,
- les périphériques esclaves transmettent ces événements vers les applications abonnées au périphérique esclave concerné,
- puis les événements sont à nouveau réémis vers les applications abonnées à leur périphériques maîtres.

Dans le cas d'un abonnement à un ou plusieurs périphériques maîtres, le protocole d'événements émet à chaque modification de la source des événements un événement notifiant ce changement et qui permet d'annoncer la source et la configuration (les différents axes pour un dispositif d'entrée, ou le nombre de touches pour un clavier par exemple).

5.1.3. Le multi-tactile dans X.org

Nous avons essayé plusieurs approches avant d'arriver à la solution actuelle, qui est encore en cours de développement. Tout d'abord, nous avons essayé une solution reposant sur une approche multipointeurs. Ensuite, nous avons envisagé une solution plus proche de ce qui est réalisé sous des systèmes d'exploitation comme Microsoft Windows ou Apple MacOS. Ces deux essais ont conduit la communauté à étendre le protocole XInput 2.0 pour gérer le multi-tactile.

La première idée, a été de considérer chacun des doigts du périphérique d'entrée comme un périphérique esclave (au sens d'XInput 2.0). Cette approche a soulevé des problèmes. Tout d'abord, du point de vue applicatif, cette approche est trop différente de ce qui est fait actuellement sous les autres systèmes. Les applications doivent gérer un nombre de périphériques différents potentiellement grand (autant qu'il y a de contacts). Cette approche supprime l'information que les différents contacts sont émis d'un même périphérique physique. D'un point de vue utilisateur, assimiler les contacts à des curseurs pose problème puisque l'on perd la notion de contact tactile. Un contact est une zone par nature assez vague dans ses dimensions, et donc il est frustrant pour l'utilisateur de travailler avec un curseur qui a comme propriété principale la précision. Les opérations classiques de pointage et de redimensionnement de fenêtres par exemple sont fastidieuses, voire impossible. Enfin, une limite technique est apparue dans cette approche : il n'est pas possible d'attacher tous ces périphériques esclaves à un même périphérique maître. Chacun des contacts envoie une information et le périphérique maître en

fait une synthèse qui ne représente pas les volontés de l'utilisateur. Il a donc fallu créer pour chacun des périphériques esclaves un périphérique maître. Cependant, pour gérer l'apparition/disparition de ces contacts, il est nécessaire de créer un logiciel tournant en espace utilisateur, avec les problèmes de synchronisme que cela comporte.

La deuxième approche, a consisté à utiliser les différents axes disponibles sur les périphériques esclaves. Cette approche est celle utilisée par les systèmes Windows et MacOS. Ici, on rencontre un premier problème technique : le nombre d'axes maximal admissible par un périphérique est fixé, arbitrairement, à 40. Une allocation statique de ces axes est faite. Changer une telle constante imposerait de nombreux changements internes (alors que l'objectif de l'approche est d'être le plus conservatif possible). Au lieu de réaliser ces changements internes non satisfaisant d'un point de vue programmeur et utilisateur, il a été choisi d'introduire une modification au protocole XInput 2.0.

XInput 2.1 introduit de nouveaux événements `XI_Touch{Begin|End|Motion}`. Il a été décidé que seules les applications supportant XInput 2.1 pourraient utiliser les fonctionnalités multi-tactiles. Ceci permet d'éviter de tromper les applications n'utilisant pas ce protocole évitant ainsi de frustrer les utilisateurs.

5.2. Les difficultés du tactile multiple pour X.org

Le travail d'X.org, le serveur d'affichage, concernant la gestion des entrées est de préparer les événements du noyau puis de les retransmettre aux fenêtres à sa connaissance. XInput 2.0 introduit la gestion de plusieurs couples claviers/souris, mais ne gère pas encore le multi-tactile (qui sera introduit dans XInput 2.1). Dans le cas d'interfaces tactiles, la notion de survol n'existe pas. En effet, un contact ne possède que deux états : *présent* ou *absent*. Les événements classiques (Enter, Leave) ne se gèrent pas de la même manière pour les applications. L'événement `TouchBegin` correspond au couple Enter-Press et l'événement `TouchEnd` au couple Release-Leave. Les applications doivent donc principalement s'adapter au tactile pour cette raison.

Cependant, à la différence de la souris, une information importante à prendre en compte est le *chemin* parcouru par le contact. Dès lors que l'on introduit de la reconnaissance de gestes, ce ne sont plus les états initiaux, courants, et finaux qui importent seulement. L'historique est une donnée autant, voire plus importante.

X.org possède une hiérarchie de fenêtres de profondeur au moins égale à deux. Il est en effet nécessaire d'avoir un gestionnaire de fenêtrage qui encapsule au niveau de X.org les applications sous-jacentes. X.org choisi à quelle application transférer les événements dans cette hiérarchie. Dans le cas d'événements souris, un délai est introduit puisque le serveur demande au

gestionnaire de fenêtre si ce clic s'adresse à lui ou à l'application en dessous. Si le clic ne s'adresse pas au gestionnaire de fenêtre, XInput "rejoue" l'événement clic auprès du client suivant. Ce délai n'est en général pas senti par les utilisateurs finaux.

Cette gestion pose par contre un problème pour la reconnaissance de gestes. Si le gestionnaire de fenêtre doit attendre d'avoir un certain nombre d'événements (de l'ordre de quelques dizaines de millisecondes) avant de déterminer si oui ou non ce geste s'adresse à lui, XInput doit enregistrer ces événements tant que le gestionnaire de fenêtre ne lui a pas répondu. Si les événements ne s'adressent pas au gestionnaire de fenêtre, il faudrait alors les rejouer auprès du client suivant. Les délais introduits risquent de gêner de manière perceptible les utilisateurs finaux.

La solution retenue et en cours de réalisation consiste à transmettre à tous les clients concernés les événements, mais en ajoutant un drapeau leur disant si oui ou non ils sont actuellement "propriétaires" des événements en question. Si le gestionnaire de fenêtres *relâche* les événements, le client suivant peut alors utiliser tous les pré-traitements qu'il a pu réaliser au cours de la phase dans laquelle il ne savait pas si les événements lui étaient adressés ou pas.

5.3. Transformation et injection des événements

Une méthode a été définie pour supporter les gestes multi-tactiles dans les anciennes applications. Il a fallu construire un consommateur de gestes multi-tactiles qui s'abonne aux sources de ces gestes, puis injecte des événements clavier et souris dans les applications choisies.

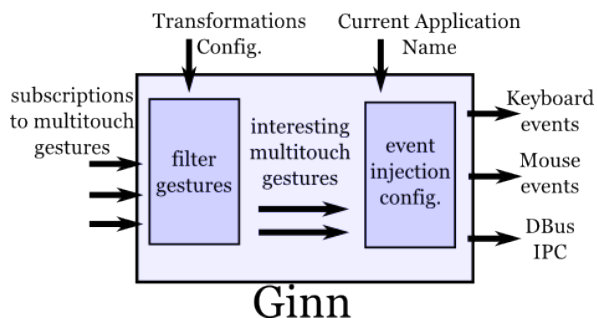


Figure 5. Schma du transformateur de gestes Ginn dans uTouch

L'application elle-même ne reconnaît pas qu'elle est dans un environnement multi-tactile, mais répond inconsciemment à ces gestes qui les reçoit sous forme d'appuis clavier, de boutons souris ou via la méthode de communication inter-processus DBus.

Avec ce genre d'utilisation on est limité aux raccourcis fournis par l'application, donc il n'y a pas une évolution dans le mode d'interaction globale de l'application, mais c'est un autre routage des informations.

Cette méthode a été utilisée d'une façon similaire dans des anciens travaux [WAN 03] et elle est approuvée par l'utilisation courante de son efficacité pour un support rapide et de prototype.

6. Conclusion

Nous avons décrit dans cet article les évolutions en cours pour gérer les tablettes tactiles dans Linux, en analysant ces évolutions comme les premières étapes vers l'adaptation de Linux à l'informatique ubiquitaire. Le travail décrit, qui mélange analyse théorique et réalisations concrètes dans un cadre communautaire, est nécessairement incomplet car les contraintes techniques et sociales viennent s'imposer à la théorie. Toutefois, l'analyse effectuée dans le présent article permet d'orienter les travaux à venir. Le tactile multiple, en remettant en cause de nombreuses hypothèses utilisées jusqu'à présent dans les systèmes d'exploitation, met en lumière ce qui devra évoluer à l'avenir. En particulier, l'absence d'un modèle théorique suffisamment riche des entrées conduit à la prolifération de modèles partiels et de traductions entre ces modèles, et l'absence d'un système de routage des entrées suffisamment flexible et indépendant du graphisme complique l'introduction de la reconnaissance de gestes, des claviers virtuels et de tous les systèmes d'adaptation que l'on peut imaginer. L'avenir est donc probablement à la mise en place d'un nouveau système de gestion et de routage d'événements. Plus de dix ans après le refus d'un tel système dans Linux, le moment est peut-être venu pour ce changement.

REMERCIEMENTS

Ce travail a été en partie financé par l'Agence Nationale de la Recherche française (projet Istar) et le Fonds Unique Interministériel (projet ShareIT). La gestion du tactile multiple dans Linux est une activité collective, il faut en particulier signaler les contributions d'Henrik Rydberg, Peter Hutterer, Rafi Rubin, Thomas Hansen, ainsi que les développeurs d'Ubuntu.

REFERENCES

[BOV 02] BOVET D. et CESATI M. *Understanding the Linux Kernel, Second Edition*. O'Reilly & Associates, Inc., 2^e édition, 2002.

[BUX 08] BUXTON B. Multi-touch systems that I have known and loved (overview). Available from <http://www.billbuxton.com/multitouchOverview.html>, 2008.

[CHA 07] CHATTY S., LEMORT A., et VALÈS S. Multiple input support in a model-based interaction framework. *Proceedings of the 2nd IEEE workshop on horizontal interactive human-computer systems (Tabletop 2007)*. IEEE computer society, Octobre 2007.

[GÖR 10] GÖRG M.T., CEBULLA M., et GARZON S.R. A framework for abstract representation and recognition of gestures in multi-touch applications. *Proceedings of the 2010 Third International Conference on Advances in Computer-Human Interactions, ACHI '10*, pages 143–147. IEEE Computer Society, 2010.

[HUT 08] HUTTERER P. *Groupware Support in the Windowing System*. Thèse de Doctorat, School of Computer and Information Science, University of South Australia, 2008.

[KUH 55] KUHN H.W. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.

[NOR 99] NORMAN D.A. *The invisible computer : why good products can fail, the personal computer is so complex, and information appliances are the solution*. MIT press édition, 1999.

[OK10] OpenKinect Project Gallery. Available from <http://openkinect.org/wiki/Gallery>, 2010.

[OLS07] UIST 2.0 interviews – Dan Olsen. Available from <http://www.acm.org/uist/archive/uist2.0/DanOlsen.html>, 2007.

[THA 96] THAM M. Dealing with measurement noise. <http://lorien.ncl.ac.uk/ming/filter/filewma.htm>, 1996.

[USB01] USB implementers' forum. Universal Serial Bus – device class definition for human interface devices. 2001.

[WAN 03] WANG J. et MANKOFF J. Theoretical and architectural support for input device adaptation. *Proceedings of CUU*, pages 85–92, 2003.

[WEI 93] WEISER M. Some computer science issues in ubiquitous computing. *Communications of the ACM*, Juillet 1993.