Approche IDM pour le développement d'applications mobiles multimodales

Nadia Elouali, Jean-Claude Tarby, Xavier Le Pallec, José Rouillard

MajecSTIC 2012
Lille, France, du 29 au 31 octobre 2012

Approche IDM pour le développement d'applications mobiles multimodales

Nadia Elouali, Jean-Claude Tarby, Xavier Le Pallec et José Rouillard

Université Lille 1, Laboratoire LIFL, 59655 Villeneuve d'Ascq Cedex - France.

Contact : nadia.elouali@ed.univ-lille1.fr

Résumé
L'informatique mobile propose différentes façons pour interagir avec les utilisateurs. La variété des capteurs qui équipent les périphériques mobiles (smartphone, tablette, ...) avec leurs modalités associées représente un écosystème important pour tester et valider les propositions scientifiques concernant la spécification des interactions multimodales. Dans cet article et en s'inspirant des travaux précédents de modélisation et de développement des applications multimodales, nous présentons une approche IDM (Ingénierie Dirigée par les Modèles) pour modéliser et générer des applications mobiles multimodales.

Abstract
The mobile computing provides different ways to interact with users. The variety of sensors in mobile phones (smartphone, tablet,...) represents a perfect ecosystem to test and validate any scientific proposition about specifying multimodal interactions. Taking our inspiration from previous works, we present in this paper a MDI (Model Driven Engineering) approach to model and generate multimodal mobile application.

Mots-clés : Application mobile multimodale, Ingénierie Dirigée par les Modèles (IDM), modélisation, évènement en entrée et en sortie.

Keywords: Multimodal mobile application, Model Driven Engineering (MDE), modeling, input and output events.

---

1. Introduction

Le principal objectif de la multimodalité est de favoriser autant que possible une communication naturelle avec l'utilisateur final. Elle améliore aussi la robustesse et la fiabilité de l'interaction et facilite l'adaptation au contexte en mobilisant plusieurs modalités en entrée et/ou en sortie. Après les travaux de Bolt [4] qui sont à l'origine de la conception et le développement des applications multimodales, l'ordinateur s'est enrichi de nouvelles modalités d'interaction pour les soutenir. En outre, l'émergence de l'informatique mobile a ouvert de nombreuses perspectives pour ce type d'applications [10]. La grande variété des capteurs qui équipent les périphériques mobiles (smartphones, tablette tactile,...) a permis l'apparition de nouvelles modalités telles que l'interaction en inclinant le téléphone ou en modifiant son orientation. Ainsi l'interaction multimodale est devenue beaucoup plus riche et proche de ses utilisateurs finaux. La multimodalité à son tour atténue significativement les limites des téléphones mobiles (petits écrans, clavier inconfortable...) en prévoyant des modalités alternatives et en facilitant la correction des erreurs. Le tandem mobile & multimodalité est donc promoteur pour l'évolution de l'interaction homme-machine.
Dans le cadre de nos travaux, nous nous intéressons à la modélisation et le développement des applications mobiles multimodales. Dans cette optique, nous avons fait le choix d'aborder ce sujet selon une approche « bottom-up » en partant des considérations et des contraintes matérielles (périphériques existants et capteurs associés), leurs modes de programmation, etc. Un exemple

caractéristique de notre recherche est le projet ANR MOANO [1] qui vise à fournir à des jardiniers un équipement informatique mobile (smartphone ou tablette) dans le but de les aider dans leur travail, par exemple en leur permettant de saisir/rechercher des informations contextualisées d'une façon multimodale (quel engrais a été utilisé pour cette parcelle ? par qui ? quand ? quelle météo ce jour-là ?...).
En parallèle de notre approche « bottom-up », nous utilisons une approche IDM (Ingénierie Dirigée par les Modèles) qui vise à valider notre travail. L'IDM nous permet, notamment, de modéliser puis de générer des applications mobiles multimodales prenant en compte les considérations et les contraintes matérielles évoquées précédemment.
Cet article présente tout d'abord des travaux en relation avec les nôtres. Puis nous expliquons notre approche en utilisant un exemple de modèle d'application mobile multimodale et l'application générée grâce à l'IDM.

2. Etat de l'art et positionnement

Au cours de ces dernières années, de nombreuses approches pour la modélisation et la création des interfaces multimodales sont apparues. Ces travaux ont permis de faire face aux difficultés accrues qu'implique le développement de la multimodalité (la gestion de plusieurs modalités, les différentes coopérations en entrée et/ou en sortie, la fusion, la fission,...). Nous présentons ici cinq travaux où la modélisation a eu une place importante.
Dynamo [3] est la poursuite des travaux autour d'ICARE [5] et d'OpenInterface [13]. Ces travaux permettent de modéliser et d'implémenter des interfaces multimodales en entrée avec une approche à base des composants. Ils définissent une modalité d'interaction comme une paire de composant : périphérique et langage d'interaction. Des environnements de modélisation graphique sont proposés par ces frameworks afin de permettre d'assembler graphiquement les composants des modalités d'interaction et de leurs coopérations selon les propriétés CARE [6], puis générer l'interface multimodale en entrée. La particularité de DynaMo est de définir des interactions multimodales partielles : le moteur d'exécution « intelligent » comblera les parties manquantes selon le contexte d'exécution. **HephaisTK** [7] est une boite à outil à base d'agents qui gère les interactions multimodales en entrée des applications auxquelles elle est rattachée. La boite est configurée par des modèles décrits dans le langage à base XML SMUIML [7] qui décrit le dialogue homme machine multimodal. HephaisTK se base alors sur le modèle créé à l'aide d'un éditeur graphique dédié [8] pour gérer les données provenant des différentes modalités d'interaction ainsi que leurs coopérations suivant les propriétés CARE. **Squidy** [11] est une bibliothèque destinée à la conception des interfaces naturelles. Elle permet de tester les différents périphériques d'interaction en les reliant aux clavier et souris standard. L'environnement de conception associé permet la modélisation du flot de données et offre un nouveau concept appellé « semantic zooming ». Ce dernier permet aux concepteurs de faire des zooms sur leurs modèles et voir leurs détailles. Ainsi, ils peuvent accéder au code des composants de leurs modèles en restant sur le même environnement. **i*Chameleon** [15] est un framework d'interaction multimodale qui permet aux concepteurs et aux programmeurs de personnaliser leurs applications en changeant les modalités d'interaction en entrée. Son éditeur permet de créer des modèles simplifiés afin que les utilisateurs finaux puissent aussi d'adapter les entrées des applications selon leurs besoins. De même, **CrossWeaver** [14] est un outil déstiné aux utilisateurs finaux. Il permet la création des storyboards multimodaux. L'utilisateur peut spécifier à l'aide d'un environnement graphique les différentes scènes et les transitions entre elles en sélectionnant les modalités en entrée ainsi que les coopérations possibles entre ces modalités.

Nous pouvons classer ces travaux suivant quatre critères (cf. tableau 1) :
la puissance d'expression qui renvoie à la finesse de description du langage de modélisation concernant les différents types de combinaison de modalités (les propriétés CARE : Complémentarité (C), Assignation (A), Redondance (R), Equivalence (E)). **L'audience**, c'est-à-dire le public visé par l'éditeur de modèles. Si ce sont les utilisateurs finaux qui sont concernés, il y a de fortes chances que la puissance d'expression soit limitée. **La logique de combinaison** entre les modalités. Ce critère se réfère à la présence ou non d'une définition formelle ou rigoureuse de la logique adoptée pour chacun des opérateurs de combinaisons associés aux propriétés CARE. **Le support**

	DynaMo	**HephaisTK**	**Squidy**	**i*Chameleon**	**CrossWeaver**
Puissance d'expression (propriétés CARE/ niveau de détail)	C, A, R, E / élevé (bas niveau)	C, A, R, E /moyen	A / en fonction du zoom	A / très faible	C, A, E / très faible
Audience	Informaticien	Informaticien	Informaticien	Utilisateur final	Utilisateur final
Logique de combinaison	Moteur de fusion (algo. à base de frames [9])	Moteur de fusion (algo. à base de frames + algorithme à base de HMM (Hidden Markov models) [7])	Pas de fusion	Pas de fusion	Moteur de fusion (algo. à base de frames)
Support logiciel	Mono-plateforme	Mono-plateforme	Multi-plateforme	Multi-plateforme	Multi-plateforme

TABLE 1 – Travaux pour la modélisation et la génération des interfaces multimodales.

logiciel : le projet se destine-t-il à être multi-plateforme ? Sinon, les concepts de modélisation sont souvent influencés par ceux de la plateforme utilisée.
Le tableau montre que les projets Dynamo et HephaisTK/SMUIML ont apporté des réponses concrètes au problème de création des interfaces multimodales. Toutefois, une faiblesse de ces projets concerne l'écosystème associé à leur plateforme : les configurations matérielles sont inhabituelles et peu répandues ou dans le cas contrainte (c'est-à-dire ordinateurs de bureau classiques) le nombre de modalités est réduit.

Nous proposons de déplacer l'étude de cette problématique sur les plateformes actuelles de développement mobile et dans notre cas sur la plateforme Android (la plus répandue). Les smartphones contiennent un grand nombre de périphériques d'interactions (plus d'une dizaine), et les utilisateurs ont l'habitude d'utiliser différentes modalités d'interaction (rotation/GPS pour google map, accéléromètre pour les jeux, gyroscope pour la gestion des bureaux, capteur de proximité pour le téléphone). Enfin, le nombre d'utilisateurs et la fréquence d'utilisation sont très élevés. Cet écosystème dispose ainsi d'une richesse scénaristique d'usage des modalités qui permettra une mise à l'épreuve des concepts de modélisation bien plus efficace qu'auparavant. Nos travaux visent à fournir un environnement conceptuel et logiciel similaire aux projets présentés précédemment, mais appliqué au contexte des smartphones Android.

3. Approche IDM pour la création d'applications mobiles multimodales

Dans le contexte de multimodalité sur mobile, nous avons adopté une approche IDM afin de faciliter le développement des applications mobiles (Android) multimodales en utilisant les modèles. D'un point de vue logiciel, notre proposition est constituée d'un éditeur de modèles et d'un générateur de code associé. L'éditeur permet de spécifier des modèles d'applications Android - basées sur les widgets standards associés - avec une focalisation sur les interactions multimo-

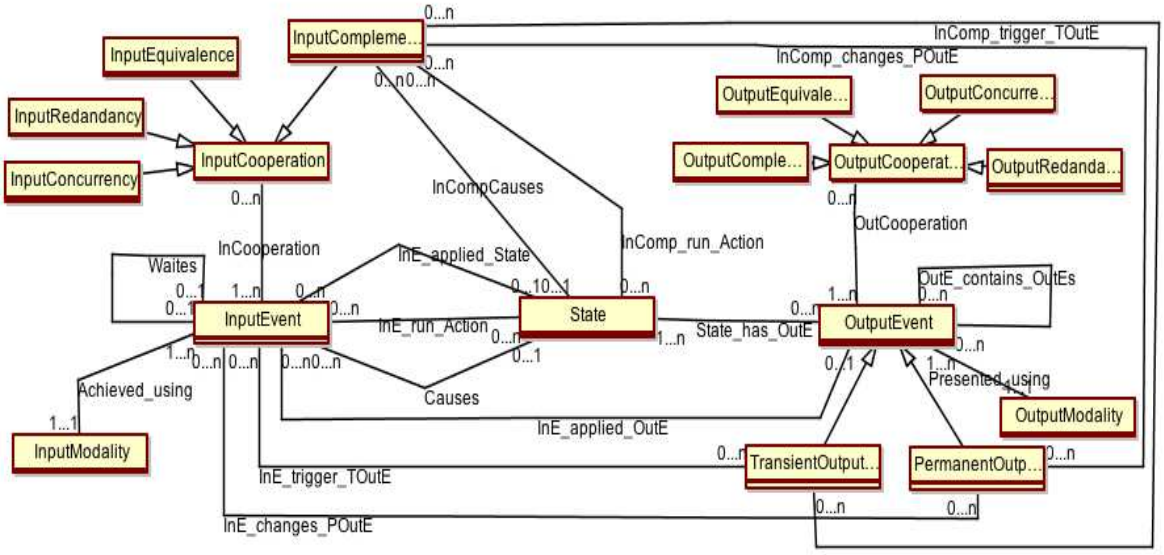

FIGURE 1 – Méta-modèle de l'interaction mobile multimodale

dales. Quand un développeur a défini son modèle d'applications, il peut le transformer en code Android correspondant grâce au générateur de code. Pour l'éditeur de modèles, nous avons utilisé le méta-éditeur ModX [2], qui, à partir d'un méta-modèle, génère un éditeur graphique de modèles correspondant. Le générateur de code est implémenté sous forme de code JavaScript au sein de ce méta-éditeur.
Nous présentons ici les concepts de modélisation (le méta-modèle), un exemple de modèle et les règles implémentées au sein du générateur.

3.1. Le méta-modèle d'interactions mobiles multimodales

Notre méta-modèle s'inspire de SMUIML [7]. Une application est représentée au travers d'états avec des événements en entrée et en sortie (non traité dans SMUIML) associés à des modalités et possédant des effets. La figure 1 présente notre méta-modèle réalisé avec ModX.
La classe « State » modélise les états qui correspondent aux états que l'application peut prendre pendant son exécution. À ces états sont associés des événements en entrée « InputEvent » provenant de l'utilisateur ou du système (clic sur un bouton, réception d'un SMS, etc.) et en sortie « OutputEvent » (vibration, affichage d'une image, synthèse vocale, etc.). Un événement en sortie peut être permanent « PermanentOutputEvent » ou transitoire « TransientOutputEvent ». Un événement permanent existe durant toute la vie de l'état associé ; il peut par exemple être un bouton qui reste affiché jusqu'au changement d'état. Un événement transitoire se produit et se termine pendant la durée de vie de son état. Une boite de dialogue ou une vibration sont des exemples d'événements transitoires en sortie.

Enfin, chaque événement en entrée ou en sortie :

- possède une modalité d'interaction (les classes « InputModality » et « OutputModality »), ces modalités étant en lien direct avec les capteurs disponibles sur les smartphones et tablettes. Une particularité de notre approche est que nous ne considérons pas que les coopérations se font entre modalités, mais plutôt entre les évènements qui utilisent ces modalités. Si nous prenons l'exemple « put that there » de Bolt où les modalités vocale et gestuelle sont utilisées pour déplacer un objet, ce sont les évènements « prononciation », « sélection de l'objet » et « sélection de la nouvelle position » qui coopèrent en complémentarité pour définir la commande finale.
- coopère ou non avec d'autres évènements du même type (en entrée « InputCooperation » ou en sortie « OutputCooperation »). En nous basant sur les espaces de conception CARE et TY-

COON [12], nous modélisons quatre natures différentes de coopérations entre évènements : Complémentarité, Redondance, Equivalence et Concurrence.
– déclenche des effets ou des traitements particuliers. Chaque événement en entrée peut avoir des effets sur l'application tels que le passage vers un autre état (l'association « Cause »), la modification d'événements permanents en sortie (« InE_changes_POutE »), le déclenchement d'événements transitoires en sortie (« InE_trigger_TOutE »), l'attente d'un autre événement en entrée (« Waites ») ou l'exécution d'une action système telle que l'accès à une base de données par exemple (« InE_run_Action »). À la différence des autres natures de coopération en entrée, la complémentarité provoque des effets (de même types que ceux des évènements en entrée) qui lui sont propres. Ainsi, la complémentarité entre un évènement de clic (c.-à-d. Un appui) sur un bouton et un évènement vocal n'aura pas un effet égal à la somme des effets des deux évènements, mais un effet personnel tel que par exemple le déclenchement d'une action système. Un évènement en entrée peut également être appliqué sur un évènement en sortie « InE_applied_OutE » (par exemple, un clic tactile s'applique sur un bouton affichage) ou sur l'état tout entier « InE_applied_State » (par exemple, l'inclinaison à droite du téléphone mesurée par accéléromètre).

Notre méta-modèle décrit donc rigoureusement les concepts fondamentaux de l'interaction mobile multimodale. Il permet non seulement de modéliser les modalités utilisées, mais aussi l'utilisation de ces modalités pour que l'utilisateur ou le système réalise sa tâche interactive.

3.2. Un modèle d'interaction mobile multimodale

Prenons l'exemple d'une application qui permet de prendre des notes. Cette application, parmi ses fonctionnalités, affiche la liste des notes qui répond à deux interactions différentes. La première interaction met en oeuvre le clic et l'accéléromètre ; si l'utilisateur clique sur une note tout en basculant rapidement le téléphone à gauche, le téléphone vibre pour signaler la bonne prise en compte de l'interaction puis affiche la page qui permet d'éditer la note. De même, la seconde interaction utilise le tactile et l'accélération ; si l'utilisateur clique sur une note tout en basculant rapidement le téléphone à droite, le téléphone vibre puis passe à une nouvelle page qui permet de partager l'image de la note par courrier électronique.

La figure 2 montre le modèle de l'application conforme au méta-modèle et construit grâce à l'éditeur ModX. Ce modèle contient trois états contenant des événements et des modalités en entrée et en sortie. Par exemple pour l'état « Liste des notes », l'évènement en entrée « Clic note » avec modalité tactile s'applique sur l'évènement en sortie « Liste » avec modalité affichage. Ainsi, si cet évènement en entrée survient dans un laps de temps imparti avec l'évènement « Aller à gauche » (modalité accélération) qui s'applique sur le même état, la première complémentarité déclenche la vibration du téléphone et un passage vers l'état d'édition. On notera de même la deuxième complémentarité entre l'évènement de clic et « Aller à droite » qui déclenche une vibration et un passage vers l'état de partage de note.

3.3. La génération de l'application mobile multimodale

Notre générateur transforme automatiquement le modèle conforme à notre méta-modèle en code Android de l'application multimodale correspondante. Le déploiement de l'application générée se fait à l'aide d'Eclipse. Le développeur commence par créer un projet Android vide sous Eclipse puis donner l'adresse de ce dernier au générateur afin qu'il puisse générer les fichiers java et xml de l'application. Suivant le modèle de la figure 2, il transforme les trois états à trois « activités » différentes de l'application Android. Pour chaque activité, il génère un « layout » pour les évènements en sortie possédant une modalité affichage (la vibration est déclarée dans l'activité associée à l'état « Liste des notes »). Il génère aussi le fichier XML « AndroidManifest » qui déclare ces trois activités ainsi que la permission de vibration. Le code nécessaire pour recevoir les évènements en entrée et effectuer les effets associés est également généré dans l'activité de l'état sur lequel ils s'appliquent. Un écouteur est ainsi associé à la liste de l'activité de liste des notes afin d'écouter les clics. De même, le capteur d'accélération est déclaré dans cette activité et la méthode permettant d'écouter ses changements est définie. Afin de gérer la complémentarité entre les deux évènements, le générateur produit aussi une classe java qui correspond au moteur de fusion des

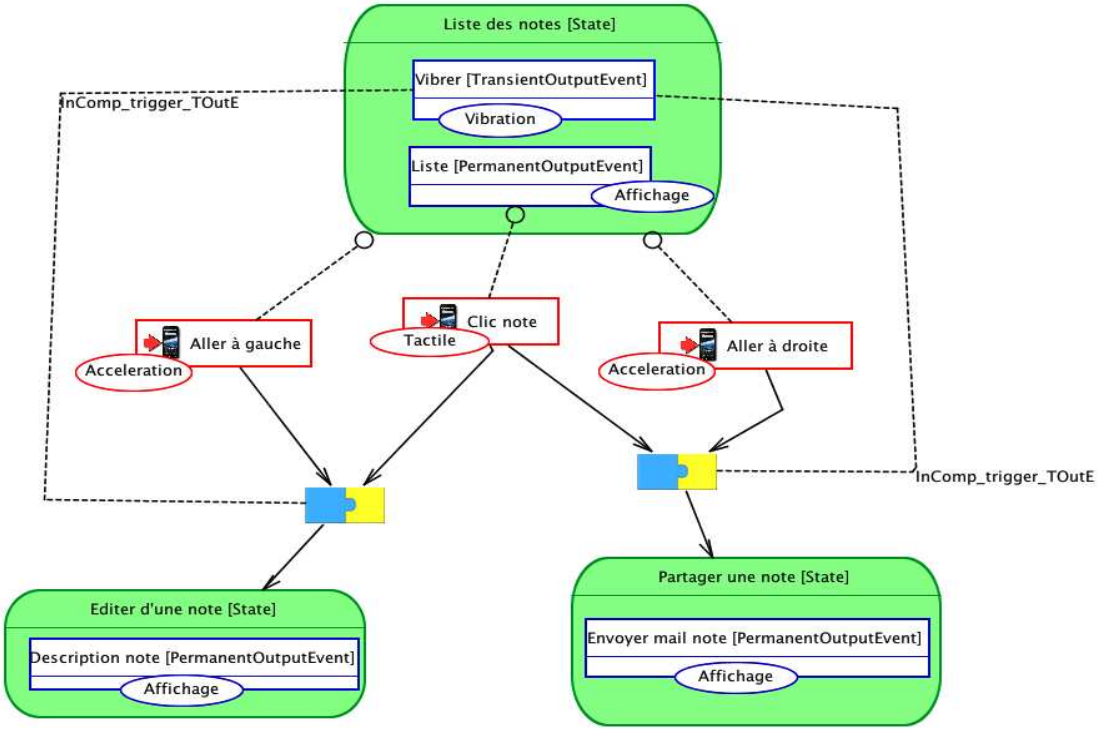

FIGURE 2 – Modèle d'interactions mobiles multimodales

données. Le moteur attend les évènements en entrée issus de l'activité courante, fusionne les données et renvoie le résultat vers l'activité cible afin qu'elle exécute les effets de la complémentarité.

Après la génération de code, le développeur n'a plus qu'à ajouter manuellement le nom des évènements envoyés vers le moteur de fusion, spécifier la commande finale à exécuter après la fusion et définir éventuellement quelques paramètres (par exemple les valeurs de l'accéléromètre pour distinguer un mouvement vers la gauche ou vers la droite). Par exemple, dans l'activité « liste des notes », pour le clic sur un item de la liste, la seule action du développeur est d'ajouter la chaine « Item » pour informer le moteur de fusion (mf) qu'il y a eu un clic sur un item. Le code en gras ci-dessous indique le code écrit par le développeur ; le reste a été généré par notre système.

```
list.setOnItemClickListener(new OnItemClickListener() {
public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
if(!mf.addElement("Item"/*le string a fusionner*/)){ }}});
```

De même, il suffit au développeur d'ajouter la condition « x>7 » et le caractère « x » pour informer le moteur de fusion qu'il y a eu une accélération vers la gauche.

```
public void onSensorChanged (SensorEvent event) {
Sensor sensor = event.sensor;
if (sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
float x = event.values[0];
float y = event.values[1];
float z = event.values[2];
if(x>7)mf.addElement("x"/*le string a fusionner*/);}
}
```

Enfin, il ajoute « xItem » ou « Itemx » comme les chaînes attendues après fusion.

```
public void update (Observable observable,Object data) {
if (data.equals("xItem"/*la commande final de la complémentarité*/)|| data.equals("Itemx")
    ){
vibrator.vibrate(300);
Intent intentStateEdite = new Intent( getApplicationContext(), StateEdite.class);
startActivity(intentStateEdite);
}}
```

De la même façon, il ajoute une condition sur les coordonnées de l'accéléromètre pour considérer l'accélération vers la droite. Finalement, le développeur ajoute le code fonctionnel de son application dans les fichiers générés et exécute le projet final.

4. Conclusion

Nous avons montré dans cet article comment, à partir d'une approche dirigée par les modèles, il nous est possible de modéliser et de générer des applications mobiles multimodales prenant en charge les capteurs disponibles dans les smartphones. Cette approche de type IDM est issue d'un travail ascendant qui nous a permis d'extraire les considérations et les contraintes matérielles associées à ce type de périphériques.

À très court terme, nous visons la complétude de notre générateur afin de pouvoir générer les différentes interactions multimodales. Ensuite, nous allons essayer, à partir de notre modèle, d'identifier des nouvelles modalités en combinant les plus basiques, d'extraire des bouts de modèles associés à ces modalités et de permettre leur réutilisation pendant la modélisation de nouvelles applications mobiles multimodales.

Bibliographie

1. Le site du projet ANR MOANO http ://moano.liuppa.univ-pau.fr/ (consulté le 30.07.2012).
2. Le site du méta-éditeur Modx http ://www.lifl.fr/modx (consulté le 30.07.2012).
3. P-A. Avouac, P. Lalanda, and L. Nigay. Service-oriented autonomic multimodal interaction in a pervasive environment. *ICMI 2011*, (8) :369–376, 2011.
4. R. Bolt. Put-that-there : Voice and gesture at the graphics interface. *Computer Graphics*, (9) :262–270, 1980.
5. J. Bouchet and L. Nigay. Icare : a component-based approach for the design and development of multimodal interfaces. *CHI Extended Abstracts*, (4) :1325–1328, 2004.
6. J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May, and R. M. Young. Four easy pieces for assessing the usability of multimodal interaction : The care properties. *Proceedings of Interact'95*, (6) :115–120, 1995.
7. B. Dumas. *Frameworks, Description Languages and Fusion Engines for Multimodal Interactive Systems*. PhD thesis, Laboratoire d'informatique de Grenoble, Université de Fribourg, December 2010.
8. B. Dumas, B. Signer, and D. Lalanne. A graphical uidl editor for multimodal interaction design based on smuiml. *Proceedings of the Workshop on Software Support for User Interface Description Language*, 2011.
9. R. A. Goubran and C. Wood. Building an application framework for speech and pen input integration in multimodal learning interfaces. *Proceedings of ICASSP'96*, (4) :3545–3548, 1996.
10. Frédéric Jourde. *Collecticiel et Multimodalité : spécification de l'interaction, la notation COMM et l'éditeur e-COMM*. PhD thesis, Laboratoire d'informatique de Grenoble, Université de Grenoble, June 2011.
11. W. A. König, R. Rädle, and H. Reiterer. Squidy : a zoomable design environment for natural user interfaces. *CHI Extended Abstracts*, (6) :4561–4566, 2009.
12. J. C. Martin. Tycoon : Six primitive types of cooperation for observing, evaluating and speci-

fying cooperations. *AAAI Fall, Symposium on Psychological Models of Communication in Collaborative Systems*, 1999.
13. M. Serrano, L. Nigay, J-Y. L. Lawson, A. Ramsay, R. Murray-Smith, and S. Denef. The openinterface framework : a tool for multimodal interaction. *CHI'08 extended abstracts on Human factors in computing systems*, (6) :3501–3506, 2008.
14. A. K. Sinha and J. A. Landay. Capturing user tests in a multimodal, multidevice informal prototyping tool. *ICMI*, (8) :117–124, 2003.
15. W. W. Tang, K. W. K. Lo, A. T. S. Chan, S. Chan, H. V. Leong, and G. Ngai. i*chameleon : a scalable and extensible framework for multimodal interaction. *CHI Extended Abstracts*, (6) :305–310, 2011.