Université Lille1
Sciences et Technologies

Laboratoire d'Informatique Fondamentale de Lille

THÈSE

présentée par

Nadia ELOUALI

en vue d'obtenir le grade de

Docteur de l'Université des Sciences et Technologies de Lille
(Discipline : Informatique)

Approche à base de modèles pour la construction d'applications mobiles multimodales

---

Date de la soutenance : 27 octobre 2014

Composition du jury :

Mme. Laurence NIGAY	Université Joseph Fourier Grenoble 1	Rapporteur
M. Yacine BELLIK	Université Paris Sud	Rapporteur
Mme. Marianne HUCHARD	Université Montpellier 2	Examinateur
M. Lionel SEINTURIER	Université Lille 1	Examinateur
M. José ROUILLARD	Université Lille 1	Directeur
M. Jean-Claude TARBY	Université Lille 1	Co-encadrant

UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE
Laboratoire d'Informatique Fondamentale de Lille 8022
U.F.R. d'I.E.E.A. - Bât. M3 - 59655 VILLENEUVE D'ASCQ CEDEX

Tél. : +33 (0)3 28 77 85 41 – Télécopie : +33 (0)3 28 77 85 37 - email : direction@lifl.fr

Résumé

Cette thèse s'inscrit dans le domaine de l'Interaction Homme-Machine (IHM) et plus spécifiquement dans celui de l'ingénierie des interfaces mobiles multimodales. Les travaux de recherche présentés décrivent les problèmes posés par le développement de telles interfaces pour les nouveaux dispositifs mobiles riches de capteurs innovants, et proposent des solutions qui sont testées a travers des réalisations et des expérimentations. Le travail est divisé en trois étapes. La première, à caractère exploratoire, décrit une enquête auprès des développeurs mobiles concernant le développement des applications multimodales. Cette enquête a permis de révéler les problèmes de développement de ce type d'applications. Elle a souligné aussi la nécessité de proposer des solutions pour simplifier et accélérer le développement de leurs interfaces. La seconde étape décrit notre approche à base de modèles pour faciliter et accélérer le développement des interfaces mobiles multimodales riches de modalités à base de capteurs. L'approche est composée d'un langage de modélisation, d'une bibliothèque d'évènements d'interaction à base de capteurs, et d'un framework de génération de code. Le langage, baptisé M4L (Mobile MultiModality Modeling Language), est basé sur un ensemble de concepts pertinents et concis, ainsi qu'une notation graphique qui présente une efficacité cognitive. La modélisation d'un ensemble d'interfaces mobiles multimodales a permis d'évaluer le langage ; et d'en dégager des enseignements intéressants sur ses forces et faiblesses. La bibliothèque fournit les différents types d'interaction à base de capteurs, en entrée et en sortie, pouvant être utilisés lors de la modélisation. Elle permet non seulement aux concepteurs d'identifier les types d'évènements possibles, mais elle leur fournit également des modèles réutilisables modélisant ces évènements. Le framework de génération de code, nommé MIMIC (MobIle MultImodality Creator), se base sur notre langage de modélisation (M4L) ainsi que la bibliothèque. Il permet aux développeurs de modéliser graphiquement puis générer automatiquement les parties correspondantes aux interactions (en entrée et en sortie) de leurs applications mobiles multimodales. Dans la dernière étape, notre approche est évaluée avec des développeurs d'applications mobiles. L'objectif visé est double : d'une part, valider nos hypothèses de départ concernant la facilité et l'accélération de développement des interfaces mobiles multimodales grâce à notre approche, et d'autre part, évaluer les différentes caractéristiques du framework (notation graphique, vérification de modèle, guidage, etc.). Des résultats préliminaires encourageants sont obtenus et des perspectives prometteuses sont discutées en conclusion.

Mots clés : Interaction Homme-Machine (IHM), Interfaces Multimodales, Applications Mobiles, Ingénierie Dirigée par les Modèles (IDM).

Abstract

Today, smartphones present a great variety of features and embed an important number of sensors (accelerometer, touchscreen, light sensor, orientation sensor...). Created for giving context-aware abilities, these sensors also allow new types of interaction like shaking the phone or changing its orientation. Such type of interaction reduces limitation of mobile phones and paves the way to a great expansion of multimodal mobile interactions. Unfortunately, the current context of mobile software development makes difficult the development of multimodal applications. In this thesis, we intend to help developers to deal with this difficulty by providing a model- based solution that aims to facilitate the development of multimodal mobile interfaces. We adopt the principles of the Model-Driven Engineering (MDE), which is particularly fitted for such context. Our proposition includes M4L (Mobile MultiModality Modeling Language) modeling language to model the (input/output) multimodal interactions and the MIMIC (MobIle MultImodality Creator) framework that allows the graphical modeling and automatic generation of multimodal mobile interfaces. Our approach respects the main criteria of the MDE in order to define an efficient model-based approach. The results of our evaluations suggest that using our approach facilitates the development of sensor-based multimodal mobile interfaces. With our contributions, we aim to facilitate the penetration of multimodal applications and benefit from their advantages.

Keywords : Human-Computer Interaction (HCI), Multimodal Interfaces, Mobile Applications, Model Driven Engineering (MDE).

Remerciements

Je tiens à exprimer ma profonde gratitude et mes remerciements à mes encadrants de thèse : José Rouillard et Jean-Claude Tarby pour la confiance qu'ils ont placée en moi, et qui a été une motivation pour avancer dans les moments les plus délicats. Ils m'ont guidé par leurs nombreux conseils tout au long de ma thèse.

Immense merci à Xavier Le Pallec pour m'avoir donné envie de réaliser cette thèse au travers de son encadrement de mon stage en Master II. Je le remercie également pour sa disponibilité à chaque fois que j'ai sollicité son aide, ainsi que pour ses multiples encouragements et sa bonne humeur.

Mes vifs remerciements vont aussi à Madame Laurence Nigay et Monsieur Yacine Bellik pour avoir accepté d'examiner ce travail en qualité de rapporteurs. Je tiens également à exprimer ma gratitude à Madame Marianne Huchard et Monsieur Lionel Seinturier pour avoir accepté de faire partie du jury en qualité d'examinateurs.

Je remercie évidemment mes amis et mes collèges de bureau : Hajer Sassi, Rabab Bouziane, Antoine Bertout, Michel Dirix et en particulier Daniel Liabeuf pour les discussions et les pauses café partagées ensemble ainsi que pour leur soutien et leurs encouragements.

Enfin le plus grand des mercis à ma très chère famille, et tout particulièrement Abdelkader, mon père, et Fatiha, ma mère pour leur amour et leur confiance qui me donne toujours le courage pour avancer. Merci à mes chères soeurs Amel, Fatima Zohra, Malika, Fadela et Aya, dont le soutien et les encouragements n'ont jamais connu de faille. Merci d'avoir été présents chaque fois que j'ai eu besoin de vous. Je vous dois beaucoup et vous aurez toujours une place importante dans mon cœur.

Je dédie cette thèse à Amr Khaled qui m'a aidée à définir mon but dans la vie.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Sommaire

1.1 Contexte

L'émergence des smartphones a apporté des restrictions aux interfaces graphiques et à l'interaction en général. Cela est dû à leurs petits écrans, à l'interaction avec le doigt qui n'est pas assez précise, au clavier virtuel inconfortable, à l'obligation d'utiliser les yeux et les deux mains pour certaines interactions, etc [93]. Cependant, les smartphones intègrent une grande variété de capteurs qui ont permis l'apparition de nouvelles modalités d'interaction telle que l'interaction en inclinant le téléphone ou en modifiant son orientation. Ces modalités peuvent être utilisées pour repousser ou atténuer les limites de la technologie mobile et de confort. Par exemple, l'utilisation d'une modalité gestuelle telle que le secouage ou l'orientation du téléphone peut aider l'utilisateur à faire des interactions « yeux libres », la synthèse vocale lui facilite la lecture à partir d'un petit écran et la vibration lui notifie à distance. Dans ce contexte, même si l'interaction avec plusieurs modalités n'atteint pas encore l'objectif principal de la multimodalité qui consiste à assurer une communication naturelle avec l'utilisateur, elle semble permettre au moins de faciliter son interaction avec les appareils mobiles. La fiabilité et la robustesse de l'interaction qu'elle peut améliorer, permet la prévention des erreurs et l'augmentation des chances de succès de l'utilisateur.
Cependant, malgré le nombre important de modalités d'interaction issues des capteurs qui équipent nos téléphones mobiles, la multimodalité utilisée actuellement est encore limitée au tactile et/ou vocal en entrée et à l'affichage et/ou son en sortie. L'analyse des 100 applications les plus téléchargeables sous Android [1] , par exemple, montre que les pourcentages de développement/d'utilisation des nouveaux capteurs en entrée/en sortie et leurs modalités associées sont très réduits (tableaux 1.1 et 1.2).

Capteurs en entrée	Pourcentage d'utilisation
GPS	26%
Caméra	13%
Accéléromètre	10%
NFC	5%
Gyroscope	2%
Boussole	1%
Capteur de proximité	0%
Capteur de lumière	0%

TABLE 1.1 – Utilisation des capteurs en entrée pour les 100 applications Android les plus téléchargeables (Juillet 2013 France)

1. Nous avons effectué cette analyse en juin 2013

Capteurs en sortie	Pourcentage d'utilisation
Haut-parleur	70%
Vibrateur	43%
La synthèse vocale	3%

TABLE 1.2 – Utilisation des capteurs en sortie pour les 100 applications Android les plus téléchargeables (Juillet 2013 France)

Les résultats d'une enquête que nous avons réalisée suggèrent trois raisons possibles pour expliquer ce manque. Premièrement, les clients qui commandent des applications mobiles ne demandent pas l'intégration des différentes modalités issues des capteurs. Ils sont habitués aux modalités d'interaction classiques et ne veulent pas prendre de risque. Par conséquent, les utilisateurs finaux d'applications mobiles ne se voient pas proposer des nouvelles modalités. Deuxièmement, les nouveaux capteurs mobiles présentent souvent des données brutes (le x, y et z de l'accéléromètre par exemple) dénuées de sens. Il est donc très compliqué de définir à partir de ces données des modalités d'interaction ergonomiques (c'est-à-dire des modalités qui peuvent être utilisées avec le maximum de confort). Beaucoup de tests avec les utilisateurs et des algorithmes performants sont nécessaires pour les concevoir et les mettre au point. Enfin, le développement de la multimodalité sur mobile nécessite plus de temps et d'effort que les applications monomodales (programmation bas-niveau, synchronisation de modalités, etc.). De plus, avec la fragmentation des matériels (variété des capteurs mobiles sous les différentes plateformes) et le manque d'outillage standard pour les configurer, le développement multiplateformes devient plus coûteux.

Pour ces raisons, les développeurs évitent de s'engager dans la création des applications multimodales.

1.2 Objectifs et contributions de la thèse

Ces travaux de thèse s'adressent aux concepteurs et développeurs d'applications mobiles. Ils visent à faciliter le développement des interfaces mobiles multimodales riches de nouvelles modalités à base de capteurs et de combinaisons de modalités en entrée et/ou en sortie. Il est clair que ce type d'applications n'est pas prioritaire actuellement. Toutefois, plusieurs indicateurs montrent que la prochaine génération d'applications mobiles utilisera plus les interactions à base de capteurs. Les nouvelles applications de Samsung et Google, par exemple, proposent de plus en plus des interactions originales à base de capteurs pour facili-

ter l'interaction telles que la rotation du téléphone pour écouter les traductions sous « Google Translate ». Ces interactions reçoivent de plus en plus d'acclamations de la part des utilisateurs. Les deux dernières éditions de Consumer Electronic Show (CES 2013 et 2014, Las Vegas) ont montré aussi l'intérêt croissant des capteurs mobiles et des périphériques connectés aux smartphones (comme « les smartwatchs » par exemple [2]). Ils ont ouvert ainsi des nouvelles perspectives de développement pour faciliter la communication avec l'utilisateur et surtout son adaptation au contexte.

Les utilisateurs d'applications mobiles devraient se voir proposer prochainement des nouvelles modalités d'interaction et des interfaces mobiles multimodales riches. Ils auront la possibilité de voir l'intérêt des différentes interactions, d'évaluer leurs risques d'utilisation et de choisir la multimodalité qu'ils préfèrent [110] [111]. Mais pour cela, les développeurs mobiles doivent aussi se voir proposer des solutions pour exploiter pleinement les capteurs des supports mobiles. Ces solutions doivent fournir des modalités d'interaction (cas d'utilisation des capteurs) facilement utilisables et permettre un développement facile des interfaces mobiles multimodales.
Cette thèse propose une telle solution en considérant à la fois la multimodalité en entrée (de l'utilisateur vers l'application) et en sortie (de l'application vers l'utilisateur). Nos résultats élèvent le niveau d'abstraction lors de la création de ce type d'interfaces, permettent aux concepteurs/développeurs d'avoir une vue générale des différentes modalités utilisées et automatisent autant que possible leurs développements pour les différentes plateformes mobiles.

Nous avons adopté une approche dirigée par les modèles pour atteindre nos buts. Le principe de l'Ingénierie Dirigée par les Modèles (IDM) est d'aider les concepteurs/développeurs en fournissant des niveaux d'abstraction élevés pour gérer les entités logicielles complexes et automatiser leurs implémentations. Ainsi, en vertu de l'IDM, nous avons augmenté le niveau d'abstraction de développement d'applications mobiles multimodales, tout en permettant une réduction du temps de développement grâce à la génération de code pour différentes plateformes. En utilisant une modélisation graphique, l'approche permet aussi de fournir une vue générale des modalités d'interaction et donc de vérifier leurs problèmes, incohérences, contradictions, etc.
Le cadre de ce travail intègre deux domaines : l'interaction homme-machine et l'ingénierie

2. `http://lc.cx/dKD` (dernière consultation le 24/06/2014)

dirigée par les modèles. Vis-à-vis de nos objectifs qui regroupent les deux domaines, nos principales contributions sont à la fois conceptuelles et pratiques. La contribution conceptuelle prend la forme d'un langage de modélisation nommé M4L (Mobile MultiModality Modeling Language) pour la spécification des interactions mobiles multimodales en entrée et en sortie. M4L est basé sur le paradigme « état-transition » et un ensemble de concepts de modélisation dont la plupart ont été inspirés à partir des langages existants. Il permet la modélisation des interactions mobiles multimodales en entrée et en sortie, ainsi que leurs combinaisons à travers un ensemble d'opérateurs basées sur les propriétés Tycoon et CARE. La proposition conceptuelle est complétée par une mise en œuvre concrète. Ainsi, la deuxième contribution de nos travaux, qui prend la forme d'une réalisation logicielle, est un environnement de modélisation des interfaces mobiles multimodales nommé MIMIC (MobIle MultImodality Creator). Cet environnement facilite la modélisation suivant le langage M4L, propose une bibliothèque d'évènements d'interaction en entrée et en sortie et permet, à l'heure actuelle, la génération de code pour trois plateformes mobiles (Android, iPhone et HTML5/CSS3). MIMIC accorde aussi un grand intérêt aux critères IDM tels que la vérification de modèle et le guidage.

Par ces contributions, ce travail doctoral définit une approche particulière pour faciliter et accélérer le développement des interfaces multimodales ; cette multimodalité étant basée sur les capteurs mobiles en entrée et en sortie.

1.3 Organisation du mémoire

Le document est organisé en deux parties.
La première partie définit le cadre général de notre étude ainsi qu'un état de l'art sur les approches similaires existantes.

- Le chapitre 2 fait le point sur l'interaction multimodale, l'informatique mobile et présente l'Ingénierie Dirigée par les Modèles pour la multimodalité mobile.
- Le chapitre 3 analyse des approches orientées modèles pour l'ingénierie des interfaces mobiles et multimodales.

La deuxième partie du manuscrit est consacrée à nos contributions conceptuelles et logicielles.

- Le chapitre 4 présente notre langage de modélisation pour les interfaces mobiles multimodales.

- Le chapitre 5 présente l'environnement de modélisation et de génération des interfaces mobiles multimodales, appelé MIMIC.
- Le chapitre 6 montre nos évaluations de l'approche proposée ainsi qu'une synthèse des contributions de cette partie.

En conclusion générale, les points contributifs sont soulignés. Puis, les avantages et limites de notre approche sont présentés. Enfin, de nombreuses perspectives à ce travail sont exposées.

Chapitre 2

Interaction sur supports mobiles : multimodalité et IDM

Sommaire

Les interactions et interfaces multimodales font l'objet d'études depuis l'introduction du célèbre « Put that there » par R. Bolt [13] en 1980. Pour soutenir ces études, l'ordinateur a été enrichi de nouvelles modalités d'interaction en entrée (de l'utilisateur vers le système) et/ou en sortie (du système vers l'utilisateur). Toutefois, les configurations matérielles de ces modalités sont inhabituelles et peu répandues. Au contraire, les smartphones contiennent un grand nombre de périphériques d'interaction natifs (plus d'une dizaine), et les utilisateurs peuvent les utiliser sans aucune configuration. Le nombre d'utilisateurs et la fréquence d'utilisation sont ainsi plus élevés. L'informatique mobile dispose donc d'une richesse scénaristique d'usage des modalités qui permet une mise à l'épreuve des concepts de modélisation bien plus efficace qu'auparavant.

Dans la première partie de ce chapitre, après avoir donné la terminologie de la multimodalité, nous décrivons les types d'interaction et de combinaisons entre modalités. Nous présentons les différents plateformes mobiles ainsi que les nouveaux capteurs et types d'interaction. Puis, nous détaillons l'intérêt de la multimodalité pour mobile et l'état actuel du marché des applications mobiles multimodales.
Dans la deuxième partie, nous présentons l'Ingénierie Dirigée par les Modèles (IDM) qui représente une ingénierie intéressante pour faciliter le développement des applications mobiles multimodales. Nous décrivons les différents principes et concepts de cette ingénierie et les critères qui permettent de l'exploiter efficacement. Nous concluons par l'intérêt d'une approche IDM pour les applications mobiles multimodales.

2.1 L'informatique mobile et la multimodalité

2.1.1 Interfaces multimodales (en entrée/en sortie)

2.1.1.1 Terminologie

La littérature identifie plusieurs termes dans le domaine de l'interaction multimodale (schéma de la figure 2.1) dont trois sont les plus importants : Modalité, Mode et Multimodalité.

Le terme « modalité d'interaction » est ambigu [82]. Bellik la définit comme « une forme concrète d'un mode de communication particulier » où le mode « fait référence aux cinq sens humains : la vue, le toucher, l'ouïe, l'odorat et le goût (réception d'informations) et aux différentes formes d'expression humaine : le geste, la parole (production de l'information).

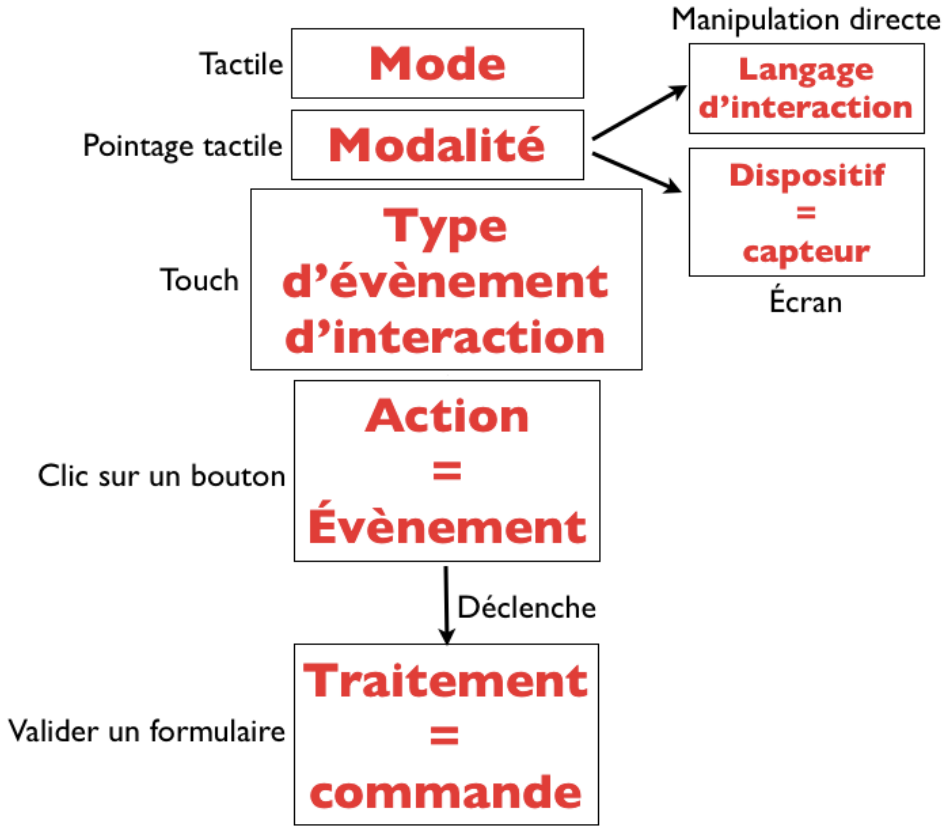


FIGURE 2.1 – Les différents termes utilisés dans cette thèse : exemple d'une interaction avec un écran tactile.

Il définit la nature des informations utilisées pour la communication (le mode visuel, sonore, gestuel, etc.) ». Par exemple, selon cette définition, le bruit, la musique et la parole sont les modalités du mode sonore [11].

Nigay et Coutaz [81] définissent la modalité par <d,r> où « d » représente le périphérique physique d'entrée/sortie, et « r » le système représentationnel ou le langage d'interaction. Par exemple, la modalité parole (vocale) peut être définie par le « Microphone » comme dispositif physique et le « langage pseudo naturel » comme langage d'interaction.

Dans [49], le terme modalité « se réfère à une forme de perception sensorielle : audition, vision, toucher, goût et odeur » (qui désigne le mode pour Bellik) alors que dans [14], c'est un « mécanisme de codage d'information pour les présenter aux humains ou machines ». Parmi ces définitions controversées, nous réutilisons celles de Bellik et Nigay pour définir la modalité comme **une forme d'un mode particulier référant un des sens humains ou des façons d'expression, et utilisant un langage d'interaction et un périphérique (physique ou virtuel)**. Avec cette définition, nous avons l'intention de souligner le fait qu'une

modalité d'interaction change selon les quatre points suivants : un sens humain ou un moyen d'expression référant un mode, une forme particulière de mode, un langage d'interaction et un dispositif d'entrée ou de sortie (physique ou virtuel). Les tableaux 2.1 et 2.2 donnent quelques exemples de modalités d'interaction en entrée et en sortie.

Modalité	**Mode**	**Langage d'interaction**	**Périphérique**
Accélération	Gestuel	Manipulation directe	Accéléromètre
Localisation	Gestuel	Positionnement GPS	GPS
Parole	Vocal (la voix)	Langage (pseudo) naturel	Microphone
pointage tactile	Tactile	Manipulation directe	Écran tactile
Orientation	Gestuel	Manipulation directe	Boussole

TABLE 2.1 – Quelques exemples de modalités d'interaction en entrée

Modalité	**Mode**	**Langage d'interaction**	**Périphérique**
Synthèse vocale	Audio	Langage (pseudo) naturel	Haut-parleurs
Affichage de widgets	Visuel	Widgets	Écran

TABLE 2.2 – Quelques exemples de modalités d'interaction en sortie

Les modalités d'interaction peuvent être actives ou passives. Une modalité peut être considérée comme active lorsqu'elle est utilisée consciemment par l'utilisateur, et passive lorsqu'elle est utilisée inconsciemment. Par exemple, la localisation de l'utilisateur avec un GPS est le plus souvent considérée comme passive, car elle n'a pas besoin de l'attention de l'utilisateur. Mais si l'utilisateur se déplace consciemment sur une carte (pour un jeu par exemple), le positionnement par GPS est considéré comme une modalité active.

Une application est multimodale si son interface intègre deux ou plusieurs modalités d'interaction en entrée et/ou en sortie en référant un ou plusieurs modes de communication. Ainsi, ce type d'interfaces offre en théorie un moyen d'interaction plus adapté aux besoins des utilisateurs qui peuvent choisir une ou plusieurs modalités combinées selon leurs préférences et en fonction de la tâche effectuée. Ceci réduit le nombre d'erreurs et rend plus robuste l'interprétation des informations (l'utilisation des modalités en parallèle peut augmenter le taux de reconnaissance correcte d'interaction) [85]. Aussi, Sharon Oviatt a montré dans [84] que non seulement les utilisateurs interagissent avec moins d'erreurs face à des interfaces multimodales, mais aussi qu'ils les préfèrent fortement (même si ce n'est pas évident de généraliser ces résultats à tout type d'interfaces multimodales).

2.1.1.2 Types d'interaction et de combinaison entre les modalités

Une interface multimodale intègre plusieurs modalités d'interaction différentes en entrée et/ou en sortie. On peut se demander quels rapports ces modalités entretiennent les unes aux autres et la façon dont leur combinaison peut être mise à profit par le système.
En France, les types de combinaison entre modalités ont été introduites par J. C. Martin durant les cinquièmes journées sur l'ingénierie des Interfaces Homme-Machine [10]. Par la suite, elles ont été présentées plus formellement par Coutaz et Nigay [26] sous le nom de propriétés CARE (Complémentarité, Assignation, Redondance, Equivalence) et étendu dans [68] sous le nom de TYCOON (TYpes and goals of COOperatioN).

CARE Les propriétés CARE (Complémentarité, Assignation, Redondance, Equivalence) décrivent les façons dont les modalités d'interaction peuvent être utilisées dans un système multimodal. Une modalité d'interaction dans CARE est définie par le couple <d,r> (voir la section 2.1.1.1).

Complémentarité : Des modalités sont dites complémentaires si elles doivent être utilisées ensemble dans une fenêtre temporelle pour la réalisation d'un traitement. Plus formellement, la complémentarité entre les dispositifs ou les langages d'interaction exprime qu'il faille utiliser tous les dispositifs ou tous les langages d'interaction pour obtenir une commande complète. Cela signifie qu'aucun des dispositifs ou des langages d'interaction ne se suffit à lui seul. Par exemple, en entrée, l'utilisateur peut prononcer la phrase « afficher les plantes de ce jardin » (l'interface est une liste des jardins d'un parc) tout en pointant du doigt l'icône du jardin en question. L'interprétation de la phrase « afficher les plantes de ce jardin » indique la commande à exécuter, mais ne permet pas de connaître l'objet sur lequel elle doit s'appliquer. De même, le pointage par le doigt permet de connaître l'objet cible, mais pas la commande qui doit être appliquée dessus. La compréhension complète de l'énoncé nécessite donc la fusion des deux messages.

Assignation : On dit qu'une modalité est assignée à une tâche donnée si cette dernière ne peut être effectuée qu'à l'aide de cette modalité. Cette propriété exprime donc l'absence de choix. Par exemple, on peut affecter le clavier pour entrer les mots de passe et la modalité vocale pour présenter les alertes.

Équivalence : Deux modalités sont dites équivalentes s'il est possible d'effectuer une

tâche donnée indifféremment au moyen de l'une quelconque de ces deux modalités. Formellement, l'équivalence d'un ensemble de modalités d'interaction est vérifiée si chaque périphérique ou langage d'interaction permet d'atteindre le même but en produisant les mêmes données. Par exemple, en entrée, l'utilisateur peut avoir le choix entre prononcer le nom de la commande qu'il désire exécuter ou la désigner sur une barre d'outils en utilisant la souris. En sortie, par exemple, le système peut afficher une notification ou la signaler par un message sonore.

Redondance : La redondance est un sous-cas de l'équivalence. Deux modalités sont dites redondantes si elles sont équivalentes, et si elles peuvent être utilisées simultanément ou successivement pour l'expression d'une tâche donnée (transmettre un même énoncé sur les deux modalités) [48]. En entrée, la redondance d'information en provenance de l'utilisateur implique la prise en compte d'une seule des modalités d'interaction par le système, l'autre pouvant éventuellement contribuer à fiabiliser l'expression obtenue. Par exemple, le montage redondant d'un microphone et d'une caméra qui observe le mouvement des lèvres d'un utilisateur permet d'augmenter la robustesse d'un système de reconnaissance vocale [15].

TYCOON En plus des quatre propriétés définies dans CARE, Martin définit deux autres combinaisons possibles entre les modalités d'interaction : le transfert et la concurrence.

Transfert : Deux modalités d'interaction coopèrent par transfert quand une information produite par une modalité peut être utilisée par la deuxième modalité. Par exemple, une information donnée par l'utilisateur à travers la modalité vocale, peut être transférée et affichée sur l'écran par la modalité affichage.

Concurrence : Les modalités sont concurrentes si elles sont utilisées en parallèle sans être fusionnées : plusieurs tâches peuvent être exécutées en parallèle. Cela permet une interaction homme-machine rapide [68].

Les propriétés CARE et TYCOON approfondissent l'usage des modalités d'interaction au sein d'une application multimodale. L'usage combiné est défini par les propriétés de complémentarité et de redondance. L'existence de choix est décrite par la propriété d'équivalence, alors que l'absence de choix est définie par l'assignation (CARE) ou la spécialisation (TYCOON). TYCOON ajoute en plus les propriétés de transfert et de concurrence.

2.1.1.3 Fusion et fission de données

Dans la littérature, on trouve plusieurs définitions pour les processus de fusion et fission : « La fusion fait référence à la combinaison de plusieurs blocs d'information pour former de nouveaux blocs alors que la fission fait référence au phénomène de décomposition. Les deux processus font partie des phénomènes d'abstraction et de matérialisation » [27].

« La fusion de données multimodales est liée à l'intégration des informations dans des systèmes interactifs homme-machine où plusieurs modalités de communication sont proposées à l'utilisateur ». « La fission de données multimodales consiste à diviser l'information en plusieurs parties compte tenu de l'objectif et du contexte » [59].

« La fusion est la combinaison de plusieurs unités d'information pour former de nouvelles unités. La fission correspond au processus inverse. L'une et l'autre traduisent deux activités importantes des processus d'interprétation et de restitution » [81].

Toutes ces définitions sont unanimes sur le fait que la fusion permet la combinaison et l'intégration des informations provenant des modalités d'interaction en entrée, alors que la fission permet la décomposition des informations fournies aux modalités d'interaction en sortie.

Ces deux mécanismes sont des composantes fondamentales pour les applications interactives multimodales. Ils sont aussi les défis technologique clés pour la création de ces applications [107]. Les informations en entrée et en sortie peuvent varier selon le contexte, la tâche, l'utilisateur et le temps. Les modalités sont aussi très différentes les unes des autres. Par exemple, la parole et l'orientation, la reconnaissance faciale et le tactile, l'affichage et la synthèse vocale... n'ont pas forcément des similitudes ou des moyens simples pour être combinés ensemble. Mais l'aspect le plus difficile à gérer pendant la fusion/fission est la synchronisation (le temps) [107].

Méthodes de fusion. Différents critères peuvent être considérés pour la fusion des informations tels que la proximité temporelle, la complémentarité logique des informations, la complétude des structures de fusion, la compatibilité des types d'informations à fusionner, etc. Basée sur ces critères, la fusion des informations peut être réalisée à trois niveaux différents : niveau « données » (directement sur le flux d'entrée), niveau « fonctionnalités »

(les caractéristiques extraites des données) et niveau « décision » (fusion sémantique). La fusion au niveau décision est la plus utilisée, car elle s'effectue après la récupération des informations à partir des entrées, ce qui lui permet d'être plus sécurisée par rapport aux deux autres, qui ont souvent des problèmes de gestion du bruit (même si les deux premières n'ont pas le problème de perte de données comme le niveau décision) [10]. Il existe plusieurs algorithmes de fusion sémantique dans la littérature, tels que « frame-based » fusion, « unification-based » fusion et « hybrid symbolic/statistical » fusion [32].

Méthodes de fission. Comme la fusion, la fission de données se fait sur différents niveaux dont le plus utilisé est le niveau sémantique [95]. Selon Foster [41], la fission sémantique est constituée de trois étapes :

1. Sélection du contenu et de la structuration des données : dans cette étape le système identifie les données en sortie et les structures selon la présentation.
2. Sélection des modalités d'interaction : après la structuration des données, le système choisit les modalités d'interaction adéquates en se basant sur des bases de connaissances ou des règles prédéfinies.
3. La coordination des sorties : finalement, le système coordonne les sorties afin de donner des messages cohérents à l'utilisateur.

Au contraire de la multimodalité en entrée, en sortie c'est le système qui prend en charge les choix des modalités d'interaction et la coordination des données, ce qui rend la tâche beaucoup plus difficile pour les développeurs, surtout en cas d'adaptation au contexte (fission multimodale intelligente).

2.1.2 Informatique mobile

La mobilité est devenue au fil du temps une véritable philosophie de vie [1] . Elle caractérise ce qui peut changer de place et de position et façonne graduellement la relation à l'information, aux transports, au temps... En situation de nomadisme, l'informatique mobile permet à un utilisateur de conserver une partie de sa relation avec des solutions numériques. Elle traite ces solutions sur des plateformes comme les téléphones portables, les smartphones, les tablettes ... qui favorisent l'informatique mobile grâce à leurs miniaturisations.

1. `http://www.gfi.fr/gfilabs/common/docs/Offre-Technologique-Informatique-Mobile.pdf` (dernière consultation le 24/06/2014)

Dans cette section, nous allons présenter les différentes plateformes mobiles ainsi que les nouveaux périphériques et capteurs. Nous présentons aussi les limites de l'informatique mobile, du côté du matériel et du côté utilisateur.

2.1.2.1 Les plateformes mobiles

Plusieurs plateformes mobiles existent actuellement sur le marché : Symbian OS, Windows Phone, Palm OS, Android, iOS, Linux (OpenMoko, ...), Blackberry OS...
En 2005, Palm, Symbian et BlackBerry ont dominé le marché. Un an plus tard, Microsoft les avait dépassés tous les trois. Sur la période 2008-2010, BlackBerry a réussi encore une fois à se hisser en première position, avant de céder du terrain devant Android en 2011. En revanche, bien qu'il représente un acteur majeur du marché, iPhone n'a jamais été leader [2].

A ce jour, Android et iPhone sont les deux plateformes principales du marché des smartphones.

Système d'exploitation Android. Android est le leader selon les résultats Comscore [3] aux État- Unis en 2013. Il dispose de plus de 50% de parts de marché et d'un succès croissant auprès des développeurs d'applications embarquées de type smartphone, ordiphones, tablettes, téléviseurs connectés, etc. C'est le premier système d'exploitation mobile open source qui a permis aux développeurs de bénéficier de tout ce que peut offrir un appareil mobile. Il est devenu totalement open-source (sous licence Apache) en 2008 après sa sortie officielle en 2007 par Google. L'ouverture du code a permis de réduire les coûts de production, de laisser les développeurs externes améliorer ou corriger le code et de permettre à chacun de modifier le système à sa guise [76].
Google a créé en 2012 « Google Play », une boutique en ligne pour les applications Android. Le Google Play regroupe toutes les applications gratuites ou payantes pour le système Android et permet leurs téléchargements et installations sur les appareils mobiles compatibles. Il permet aussi la location des films et séries télévisées, l'achat de musiques, de livres, de magazines ainsi que des smartphones et tablettes.
En 2014, Google Play a atteint le chiffre de 1,5 millions d'applications à télécharger [4], ce qui

2. `http://www.journaldunet.com/ebusiness/internet-mobile/comscore-mobile-2013/evolution-des-os.shtml` (dernière consultation le 24/06/2014)
3. `http://www.comscore.com/` (dernière consultation le 24/06/2014)
4. `http://www.frandroid.com/applications/google-play/225043_google-play-store-15-millions-dapps-revenus-doubles` (dernière consultation le 24/06/2014)

le place comme étant la plus grosse boutique d'applications au monde.

Système d'exploitation iOS. iOS (anciennement iPhone OS) est le système d'exploitation mobile développé par Apple pour l'iPhone, l'iPod touch et l'iPad. La première version du système est sortie en 2007 (rendue disponible en Europe en 2008) avec le premier iPhone vendu par Apple qui a rencontré un succès important.
Depuis 2008, « App Store » est devenu une plateforme de téléchargement d'applications, distribuée par Apple sur les appareils mobiles fonctionnant sous iOS (iPod Touch, iPhone et iPad). L'App Store est accessible de deux façons : avec le logiciel iTunes ou avec une application dédiée qui permet d'accéder à l'App Store et d'y télécharger les applications directement sur les appareils iPhone, iPod Touch et iPad. Les applications peuvent être gratuites ou payantes (comme pour Android).
En septembre 2013, App Store a atteint le chiffre de 950 000 applications à télécharger [5].

Bien qu'Android domine le marché des smartphones, la concurrence reste rude entre les différents plateformes. L'iOS, par exemple, dispose de plusieurs bons points comme son interface graphique beaucoup plus simple, Android possède un grand avantage concernant l'innovation (capteurs mobiles, nouveaux moyens d'interaction,...), et Windows Phone présente un multitâche plus ergonomique et convivial.

Les développeurs de chaque plateforme travaillent sans relâche afin de fournir aux utilisateurs une expérience beaucoup plus intéressante que celle trouvée sur les autres plateformes. Cependant, cela augmente significativement la différence entre ces plateformes et réduit la possibilité d'avoir des développeurs qui les maîtrisent toutes.

2.1.2.2 Nouveaux capteurs et nouveaux types d'interaction

Les nouveaux smartphones et tablettes sont équipés aujourd'hui par un ensemble de capteurs innovants destinés à détecter les données relatives aux appareils, aux états des utilisateurs (mouvements et actions [70]) [60] et à leur environnement. Certains capteurs (caméra, microphone et GPS) peuvent être utilisés pour donner des informations concernant l'état de l'utilisateur alors que d'autres (gyroscope, accéléromètre et capteur de proximité) peuvent être utilisés collectivement pour estimer des informations sur le contexte. De plus, des capteurs supplémentaires peuvent être facilement interfacés avec le téléphone via Bluetooth ou

5. `http://9to5mac.com/2013/09/24/app-store-stops-by-950k-apps-on-its-way-to-1-million/` (dernière consultation le 24/06/2014)

des connexions filaires (par exemple, capteur pour la pollution de l'air ou des capteurs biométriques) [22].

Un capteur (appelé aussi « détecteur ») est défini comme un dispositif qui mesure une grandeur physique et la transforme en un signal qui peut être lu par un observateur ou par un instrument [6]. Il peut être réel (physique) ou virtuel. Par exemple, l'accéléromètre est un capteur qui existe réellement pour mesurer l'accélération alors que le capteur de gravité n'est qu'un capteur virtuel basé sur l'accéléromètre. La qualité des capteurs virtuels dépend non seulement de la qualité des données physiques, mais aussi de la sophistication des algorithmes utilisés pour le calcul [7]. Un capteur peut aussi être actif ou passif (comme sa modalité d'interaction associée) ; actif lorsqu'il est utilisé consciemment par l'utilisateur et passif lorsqu'il est utilisé inconsciemment.

Les premiers capteurs intégrés aux périphériques mobiles donnaient, parfois, des informations erronées. Par exemple, les boussoles de plusieurs smartphones ne donnaient pas la bonne valeur. Toutefois, les fabricants ont rapidement résolu ce genre de problème en utilisant des algorithmes plus précis et en introduisant des bibliothèques de capteurs [8]. Avec ces bibliothèques, les développeurs arrivent à capter les mouvements du téléphone et des utilisateurs avec une grande précision.

Les capteurs peuvent être utilisés maintenant dans différents domaines tels que la surveillance de la santé personnelle, la surveillance du bruit et de l'ambiance sonore, les systèmes d'information géographiques (SIG), les jeux, etc.

Dans les tableaux 2.3 et 2.4, nous listons l'ensemble des capteurs qui existent actuellement en entrée et en sortie. Nous donnons aussi quelques types d'interaction (des types d'évènements d'interaction) qui peuvent être créés à partir de ces capteurs ainsi que quelques cas de leurs utilisations.

Capteur	**Type**	**États d'utilisation**	**Cas d'utilisation**	**Types d'interaction**
Accéléromètre (détermine l'accélération appliquée sur le périphérique)	Réel	Actif et passif [89]	Utilisé pour déterminer l'orientation du téléphone (affichage portrait/paysage), déplacer des widgets [33], estimer des informations sur le contexte [22]	Secouage, chute libre, accélération (droite, gauche, haut, bas)

6. `http://fr.slideshare.net/cvs26/sensors-on-android-10220894` (dernière consultation le 24/06/2014)
7. `http://lc.cx/d9R` (dernière consultation le 24/06/2014)
8. `http://lc.cx/d9a` (dernière consultation le 24/06/2014)

Capteur de champ magnétique (boussole ou magnétomètre) (mesure le champ magnétique ambiant dans les axes x, y et z)	Réel	Actif (il nécessite la localisation [60])	Utilisé pour détecter les rotations (carte, graphe, interfaces...), détection d'orientation, détection des champs magnétiques [9]	Orientation (nord, sud, est et ouest), orientation vers la Mecque
Gyroscope (mesurer la vitesse de rotation autour des axes [10])	Réel	Actif (il nécessite la localisation)	Utilisé pour la détection des mouvements précis : beaucoup dans les jeux [11] et en réalité augmentée (Sky Map (Android))	Équilibrage
Capteur de luminosité (RGB)	Réel	Actif et passif (ex : l'application Night Mode [12])	Utilisé pour la détection de lumière, adapter le rétro-éclairage de l'écran en fonction de l'environnement, mesurer la lumière réfléchie et ambiante (en photographie) (ex : l'application Light Meter Tools [13])	Blackout (un masquage total de la luminosité du smartphone), éclairage
Capteur de température	Réel	Actif et passif (Deprecated [14] pour Android à partir de la version 3.2)	Utiliser pour mesurer la température externe [15]	

9. `http://www.uni-weimar.de/medien/wiki/images/Zeitmaschinen-smartphonesensors.pdf` (dernière consultation le 24/06/2014)

10. `http://fr.slideshare.net/datta_jini/android-sensors-18449038?ref=http://www.techjini.com/blog/all-about-mobile-sensors/`(dernière consultation le 24/06/2014)

11. `http://www.mobile88.com/news/read.asp?file=/2012/4/21/20120421165938` (dernière consultation le 24/06/2014)

12. `https://play.google.com/store/apps/details?id=pt.bbarao.nightmode&feature=search_result&hl=fr` (dernière consultation le 24/06/2014)

13. `http://lc.cx/dtK` (dernière consultation le 24/06/2014)

14. `http://fr.slideshare.net/cvs26/sensors-on-android-10220894` (dernière consultation le 24/06/2014)

15. `http://lc.cx/dtr` (dernière consultation le 24/06/2014)

Capteur de pression ou Baromètre (mesure la pression de l'air ambiant)	Réel	Actif	Utilisé pour aider le GPS à définir la position précise (la pression est proportionnelle à l'altitude)	
Capteur d'humidite	Réel	Actif	Utilisé pour détecter l'humidité de l'environnement	
GPS	Réel	Actif et passif	Utilisé pour détecter l'emplacement de l'utilisateur	
Écran tactile	Réel	Actif	Utilisé pour détecter l'interaction tactile	« Touch » (un appui sur l'écran du smartphone), « Long Touch » (un appui prolongé), « Multitouch » (un appui avec plusieurs doigts) - « Gestural touch » (des gestes tactiles), « Gestural Multi-touch » (des gestes tactiles avec plusieurs doigts)
Caméra	Réel	Active	Utilisé pour prendre des photos ou vidéos et en réalité augmentée	Prendre des photos/vidéos
Capteur de gestes (Air gesture) [16]	Réel	Actif (seulement sur des appli particulières pour le moment)	Utilisé pour détecter quelques gestes effectuées avec les mains : parcourir les emails, images, musiques, afficher les notifications, répondre/ refuser à un appel	Des gestes particuliers avec la main
Capteur de Hall	Réel	Actif	Utilisé pour détecter si l'écran est couvert ou pas	Disponible sur le Samsung Galaxy S4

16. `http://allaboutgalaxys4.com/galaxy-s4-features-explained/air-gesture/` (dernière consultation le 24/06/2014)

Bluetooth/wifi (Pas vraiment des capteurs dans le sens traditionnel) [17]	Réel	Actif	Utilisé dans les jeux pour jouer à plusieurs (plusieurs téléphones), utilisé pour télécharger ou se connecter à un autre dispositif	
Capteur d'accélération linéaire (mesure l'accélération pure du téléphone (Accélération - Gravité)	Virtuel	Actif	Utilisé dans les applications pour le calcul de la vitesse de voiture, distance parcourue [4], etc., détecter et corriger les mouvements/activités des utilisateurs (en médecine) [53] [103]	Presque les mêmes moyens d'interaction que l'accéléromètre
Capteur de gravité (mesure la force de gravité appliquée sur les axes x,y,z = Accélération - Accélération linéaire)	Virtuel	Actif	Utilisé pour détecter l'orientation du téléphone [18], déplacer l'avatar dans les jeux, détecter des mouvements [19]	Peut être utilisé pour définir des secousses particulières
Capteur d'orientation (mesure le degré de rotation que le périphérique fait autour de ses 3 axes)	Virtuel	Actif (Deprecated [20] pour Android à partir de la version 2.2)	Utilisé pour détecter les orientations	Orientation (haut, bas, droite et gauche)
Capteur de proximité (mesure la distance des objets par rapport au téléphone)	Virtuel	Actif (basé sur le capteur de lumière)	Utilisé pour détecter la proximité de l'oreille afin d'éteindre l'écran et arrêter la détection du tactile - utilisé aussi en sport [21])	S'approcher/ s'éloigner du téléphone

17. `http://web.stanford.edu/class/cs75n/Sensors.pdf` (dernière consultation le 24/06/2014)

18. `https://play.google.com/store/apps/details?id=com.plexnor.gravityscreenofffree` (dernière consultation le 24/06/2014)

19. `http://developer.android.com/guide/topics/sensors/sensors_motion.html` (dernière consultation le 24/06/2014)

20. `http://fr.slideshare.net/cvs26/sensors-on-android-10220894`(dernière consultation le 24/06/2014)

21. `https://play.google.com/store/apps/details?id=kr.co.ldroid.pushups.toe&feature=search_result` (dernière consultation le 24/06/2014)

Vecteur de rotation	Virtuel	Actif (une combinaison de l'accéléromètre, le magnétomètre, et parfois le gyroscope)	Utilisé pour détecter la rotation	Rotation de périphérique
Microphone	Réel	Actif et passif	Utilisé pour détecter le son (texte, commande vocale, musique, bruit,..) et détecter les battements cardiaques (en médecine)	Dicter et donner une commande vocale

TABLE 2.3 – Description des capteurs en entrée sur mobile

Capteur	**Type**	**États d'utilisation**	**Cas d'utilisation**	**Moyens d'interaction**
Haut parleur	Réel	Actif et passif	Utilisé pour diffuser du son	Synthèse vocale et diffusion de musique
Écran	Réel	Actif	Utilisé pour afficher l'interface graphique	Affichage des widgets
Vibreur	Réel	Actif	Utilisé pour vibrer	Vibration courte, vibration longue, distinguer un appel téléphonique d'un autre

TABLE 2.4 – Description des capteurs en sortie sur mobile

Ces capteurs fournissent des mesures très précises, mais très sensibles, bruitées et sans aucune interprétation [22]. Ainsi, pour les utiliser en interaction, beaucoup de programmation de bas niveau est nécessaire. De plus, beaucoup de tests doivent être réalisés avec les utilisateurs finaux afin de détecter les interactions ergonomiques.

2.1.2.3 Les limites des dispositifs mobiles

Les appareils mobiles disposent d'un ensemble de limitations en ce qui concerne l'interaction avec l'utilisateur.
Une première limite concerne l'utilisation des deux mains pour la plupart des interactions. Bien que des recherches montrent que la majorité des utilisateurs des dispositifs mobiles préfèrent utiliser une seule main pour l'interaction [50], avec les grands appareils, les utilisateurs ne peuvent pas accéder à toutes les options avec une seule main et plusieurs mouvements successifs de pouce sont difficiles à effectuer et peu ergonomiques [93][42].
En revanche, les utilisateurs préfèrent de plus en plus de grands écrans pour bénéficier d'un meilleur affichage/lisibilité et pour faciliter l'usage du tactile pour les grandes mains, par des personnes âgées, avec des gants, etc [79].

Une deuxième limite concerne l'utilisation des doigts en interaction tactile. Avec une seule main, l'utilisateur du périphérique mobile n'utilise que le pouce pour interagir avec l'écran. Cependant, le pouce cache généralement la cible et ne permet pas de la sélectionner correctement, ce qui augmente le nombre des erreurs. Certains périphériques proposent l'utilisation d'un stylet au lieu du pouce, mais cette solution n'est pas appropriée pour l'interaction à une seule main [93] et elle n'est pas toujours fournie. Aussi, lors de l'utilisation du clavier virtuel (même avec deux mains), le taux de saisie de texte est faible [83]. Cela est due à l'utilisation des doigts (les deux pouces généralement) ainsi qu'à la position du clavier qui peut gêner l'utilisateur [83].

Une autre limitation concerne le retour d'information sous mobile. Puisque les interactions en entrée se font en tactile, les informations en sortie sont souvent présentées visuellement (affichage sur l'écran). Toutefois, l'attention visuelle des utilisateurs lors de l'interaction avec les dispositifs mobiles est limitée, car elle dépend de plusieurs facteurs (l'environnement, la position de l'utilisateur et/ou du téléphone, etc.) [3]. Il est ainsi nécessaire de prévoir d'autres modalités d'interaction en sortie qui ne nécessitent pas une attention visuelle (vibration, son, etc.).

En plus de limites d'interaction, les dispositifs mobiles présentent aussi le problème de

22. `http://lc.cx/d9a` (dernière consultation le 24/06/2014)

l'autonomie réduite (connexion WiFi, 3G/4G, Bluetooth, applications gourmandes,...) et la faible capacité en terme de puissance de traitement et de stockage. Leur utilisation est souvent limitée même si les fabricants produisent des batteries de plus en plus performantes et des capacités de stockage de plus en plus grandes.

2.1.3 L'intérêt de la multimodalité pour mobile

La grande variété de capteurs embarqués sur les dispositifs mobiles et le nombre important de nouvelles modalités d'interaction issues de ces capteurs soutiennent vivement la multimodalité. Ce type d'interaction, qui existe depuis les années 80, a trouvé finalement un écosystème riche qui va le rendre beaucoup plus intéressant et proche des utilisateurs finaux. La multimodalité, à son tour, offre un moyen puissant pour surmonter les limites des systèmes et dispositifs mobiles.
Les principaux intérêts de la multimodalité que nous avons recensés pour l'informatique mobile sont les suivants :

Prévoir/proposer des modalités équivalentes. la multimodalité permet de prévoir différentes modalités d'interaction afin d'avoir une interaction plus souple. Les utilisateurs qui ne peuvent pas utiliser certaines modalités d'interaction peuvent basculer sur d'autres modalités équivalentes. Par exemple, les personnes âgées qui ne peuvent pas utiliser les mouvements, peuvent utiliser le tactile. Sinon, s'ils trouvent des difficultés avec le tactile aussi (clavier inconfortable,...), ils peuvent utiliser les commandes vocales. Cela joue un rôle important lors de l'adaptation au contexte également. Par exemple, s'il y a beaucoup de soleil ou si l'affichage d'un texte n'est pas clair sur un petit écran de téléphone, l'utilisateur peut écouter son texte avec une synthèse vocale.

Faciliter la correction des erreurs. l'intégration de plusieurs modalités d'interaction permet d'éviter les erreurs d'interprétation notamment si deux ou plusieurs modalités sont utilisées en redondance. Par exemple, l'utilisation redondante du microphone et du tactile permet aux utilisateurs de corriger les erreurs de reconnaissance vocale ce qui augmente la fiabilité des expressions obtenues.

Bénéficier des capteurs mobiles. l'implémentation des différentes modalités d'interaction issues des capteurs permet de bien exploiter les capacités de nos appareils mobiles qui les véhiculent en nombre croissant. Qu'il s'agisse de luminosité pour adapter le rétro-éclairage, de l'accéléromètre pour exploiter l'accélération de l'appareil, du GPS pour détecter la position de l'utilisateur, les capteurs sont de plus en plus intéressants et utiles pour la vie quotidienne.

Impliquer des nouveaux utilisateurs. Les utilisateurs ordinaires préfèrent les interfaces multimodales comme indiquées dans [110] [111]. De plus, la spécification de plusieurs modalités d'interaction va permettre d'impliquer de nouveaux utilisateurs en interaction avec les mobiles notamment les personnes malades et/ou handicapées.

Le tandem mobile & multimodalité est donc prometteur pour l'évolution de l'interaction homme-machine (ou plutôt homme-mobile).

2.1.4 L'état actuel du marché : faible exploitation de capteurs et fonctionnalités des nouveaux dispositifs mobiles

En analysant les applications mobiles les plus téléchargeables sous Android, nous avons constaté [23] que les applications mobiles qui existent actuellement sur le marché n'utilisent que peu les capteurs mobiles et les combinent très rarement. Ainsi, et dans le but de découvrir les causes de ce manque, nous avons réalisé une enquête auprès des développeurs d'applications mobiles. Nous avons envoyé un questionnaire par mail aux développeurs des 60 applications les plus téléchargeables sous Android et iPhone. Nous avons par ailleurs réalisé des interviews avec des développeurs mobiles dans des entreprises d'informatique sur Lille.
Les résultats suggèrent trois causes de ce manque :

1. Il n'y a pas de demande : les développeurs disent que les clients et les utilisateurs ne demandent pas des applications avec plusieurs modalités d'interaction. Ils sont familiers avec les interactions classiques (tactile/vocale en entrée et affichage/son en sortie) et ne veulent pas prendre de risques.
2. Manque de connaissance chez les développeurs : ils connaissent les capteurs, mais trouvent qu'ils donnent des mesures de l'environnement sans perspective d'interprétation (des mesures sans sens) ce qui ne permet pas de trouver facilement des cas d'usage intéressants.
3. L'intégration de plusieurs modalités d'interaction est coûteuse : les développeurs trouvent que l'intégration de plusieurs modalités d'interaction nécessite forcément plus de temps et d'efforts (programmation de bas niveau) par rapport aux applications ordinaires [24]. De plus, ils trouvent que le développement des applications multimodales pour les différentes plateformes mobiles peut être très coûteux.

23. Comme ici : `http://lc.cx/d9a` (dernière consultation le 24/06/2014)
24. Applications monomodales ou avec une multimodalité classique

L'enquête nous a permis de fixer notre objectif avec plus de détails : pour faciliter la pénétration des applications mobiles multimodales à base de capteurs, nous devons proposer une solution permettant aux développeurs d'exploiter avec moins d'effort et plus rapidement les différents capteurs et fonctionnalités de nouveaux smartphones/tablettes. Avec cette approche, les développeurs pourront proposer aux clients et utilisateurs finaux des exemples d'interaction multimodale au lieu d'interaction classique. De ce fait, les utilisateurs pourront voir les intérêts des nouvelles modalités d'interaction et les demander par la suite.

Notre solution est basée sur une approche IDM (Ingénierie Dirigé par les Modèles). Dans le chapitre suivant, nous présentons l'IDM, les critères de cette ingénierie ainsi que les raisons qui nous ont poussés à choisir une telle approche.

2.2 L'Ingénierie Dirigée par les Modèles (IDM) pour la multimodalité mobile

L'Ingénierie Dirigée par les Modèles (IDM), ou Model Driven Engineering (MDE) en anglais est une ingénierie qui a amélioré le développement des systèmes informatiques en se concentrant sur une préoccupation plus abstraite que la programmation classique [24]. Actuellement, le recours à l'IDM prend de l'ampleur afin de faciliter et d'accélérer le développement des applications. L'IDM se base sur les modèles pour générer tout ou partie des applications.

2.2.1 L'IDM : Métamodèles, Langages, Modèles

2.2.1.1 Les principes généraux de l'IDM

L'IDM est un domaine du génie logiciel où « tout est modèle ». Pour faire face à la complexité technique croissante, les chercheurs ont décidé, il y a une dizaine d'années, d'explorer une voie de l'ingénierie, où les modèles deviennent les objets centraux à manipuler [47]. Le principal avantage de l'IDM est lié à la génération automatique de code. Comme les modèles sont considérés indépendants de la plateforme, il est plus facile de construire des générateurs de code pour plusieurs plateformes. Par exemple, les codes pour Android, iOS, Windows Phone ou BlackBerry OS peuvent être générés à partir d'un modèle d'une application mobile indépendant de la plateforme. Bien que le code généré ne constitue pas

l'ensemble du code de l'application envisagée, la génération de code multiple réduit considérablement le coût de développement de chaque plateforme. Aussi, l'IDM raccourcit le cycle de réalisation [72] en réduisant le temps de développement (jusqu'à trois fois [25]), améliore la qualité du code et augmente l'efficacité des logiciels ainsi générés.

Les concepts centraux de l'IDM n'ont généralement pas de définition universelle. Néanmoins, de nombreux travaux dans la littérature s'accordent à un relatif consensus. Dans la suite de cette sous-section, nous définissons et présentons les trois principaux concepts de l'IDM : métamodèle, langage et modèle.

Métamodèle et langage de modélisation Un métamodèle signifie littéralement le modèle du modèle (le préfixe méta veut dire que l'on applique un concept à lui même ; par exemple, une méta-discussion est une discussion sur les discussions). L'OMG (Object Management Group) le définit par « un modèle qui définit le langage d'expression d'un modèle » [24]. Il permet d'identifier les concepts utilisables dans les modèles qui lui sont conformes (identifier le vocabulaire ou la grammaire), indique comment ces concepts sont organisés et quelles sont les informations contenues. Ainsi, le métamodèle doit définir rigoureusement le langage de modélisation, car s'il n'est pas défini précisément, les modèles ne le seront pas non plus et ne peuvent pas être opérationnels.
La métamodélisation est la technique permettant d'identifier les concepts à utiliser pour modéliser les systèmes. On ne peut pas métamodéliser un métamodèle universel pour décrire tous les systèmes informatiques. Un métamodèle doit être défini pour un objectif particulier. Ainsi, il existe une multitude de métamodèles pour créer de nouveaux langages répondant aux problèmes spécifiques à un domaine particulier [69]. On nomme ces langages : « langages de modélisation spécifique au domaine » (Domain Specific Modeling Language (DSML)), contrairement aux langages généraux, tels qu'UML (Unified Modeling Language), qui peuvent être utilisables pour plusieurs domaines d'applications. Un DSML est défini par une syntaxe abstraite qui décrit les concepts du langage et leurs relations (décrite dans son métamodèle), une sémantique qui décrit de manière précise et non ambiguë la signification des concepts et une syntaxe concrète qui décrit le formalisme (la notation) permettant à l'utilisateur de manipuler ces concepts. Les syntaxes concrètes des DSMLs peuvent être graphiques ou textuelles. La séparation entre syntaxe abstraite et concrète est une technique pour gérer la complexité de la définition d'un langage de modélisation, car elle

25. `http://ercim-news.ercim.eu/en91/ri/advances-in-model-driven-software-engineering` (dernière consultation le 24/06/2014)

permet de définir les éléments d'un langage indépendamment de leur représentation [34].

Modèle Dans [55], « un modèle est une description (d'une partie) d'un système écrit dans un langage bien formé ». Minsky le définit par « pour un observateur B, M est un modèle de l'objet O, si M aide B à répondre aux questions qu'il se pose sur O » [73]. À base des différentes définitions qui existent dans la littérature, nous avons défini la nôtre qui nous semble la plus adéquate : un modèle est une représentation abstraite d'un système, problème ou situation pour un but donné et suivant un point de vue particulier.
La modélisation est la tâche de conception d'un modèle conformément à un langage. Le recours à la modélisation pour le développement logiciel est devenu indispensable depuis longtemps [92], car les systèmes deviennent de plus en plus complexes et leur maîtrise nécessite de l'abstraction. En IHM, l'activité de modélisation a donné lieu à une riche littérature. Elle a pour but de produire une description du système interactif en rapport avec les besoins de l'utilisateur [5]. Initialement les modèles ont été utilisés pour faciliter l'implémentation de l'IHM et sa relation avec le système, ainsi que la communication entre les concepteurs, développeurs et clients. Avec l'évolution de l'IDM, les modèles sont de plus en plus utilisés pour la génération de code [106] [34]. Les différents modèles en IHM peuvent être classés en deux catégories : les « modèles d'architecture » qui décrivent l'organisation du logiciel du système interactif et les « modèles d'interaction » qui décrivent les différents mécanismes utilisés pour assurer un dialogue entre l'utilisateur et le système.

Modèles d'architecture. Les modèles d'architecture définissent l'organisation logicielle du système interactif. Ils séparent le noyau fonctionnel (qui met en œuvre les concepts spécifiques au domaine d'application) de l'interface utilisateur afin d'avoir une meilleure modularité. Cette séparation doit théoriquement permettre de modifier l'interface sans toucher le noyau fonctionnel et vice-versa, mais, en réalité, cela est difficile à réaliser pour toute l'interface. En pratique, seule une partie de l'interface est réellement séparée du noyau fonctionnel [11]. Les modèles d'architecture les plus connus sont : Seeheim [88], Arch [100], MVC [57] et PAC [25]. Seeheim est le premier modèle qui sépare les interfaces du noyau fonctionnel de l'application. Il se compose d'un composant de présentation, un composant de contrôle de dialogue et un composant d'interface d'application (figure 2.2). L'utilisateur interagit directement avec le composant de présentation. Ce composant passe les événements d'interaction au contrôleur de dialogue qui détermine les services de l'application qui doivent être interrogés. Le modèle Arch détaille le modèle Seeheim en ajoutant d'autres composants intermédiaires afin de s'approcher encore plus de la réalité des systèmes interactifs.

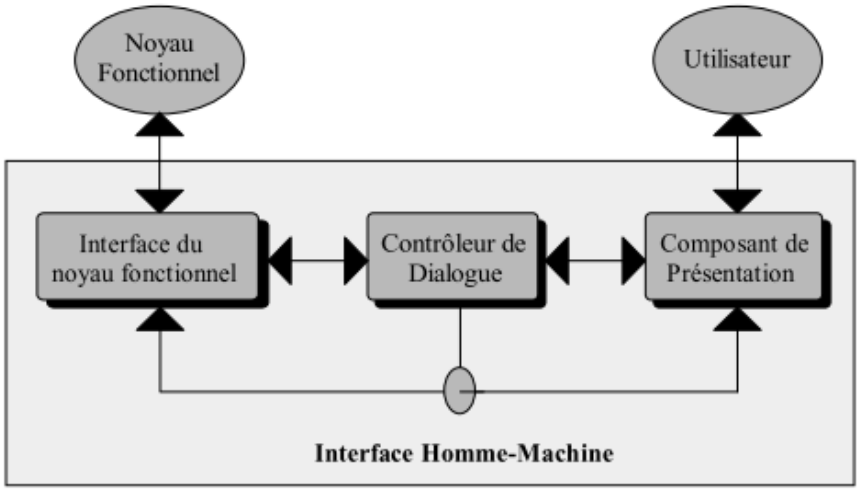


FIGURE 2.2 – Le modèle Seeheim [88]

Les modèles MVC (Model, View, Controller) et PAC (Presentation, Abstraction, Controller) sont basés sur les agents. Ils structurent les systèmes interactifs comme une collection d'agents (suivant le même principe que Seeheim et Arch, mais à un grain plus fin). Dans le modèle MVC (figure 2.3), chaque agent comporte trois facettes : le modèle (qui définit les fonctionnalités du domaine), la vue (qui définit la présentation avec laquelle l'utilisateur interagit) et le contrôleur (qui traite les entrées de l'utilisateur).

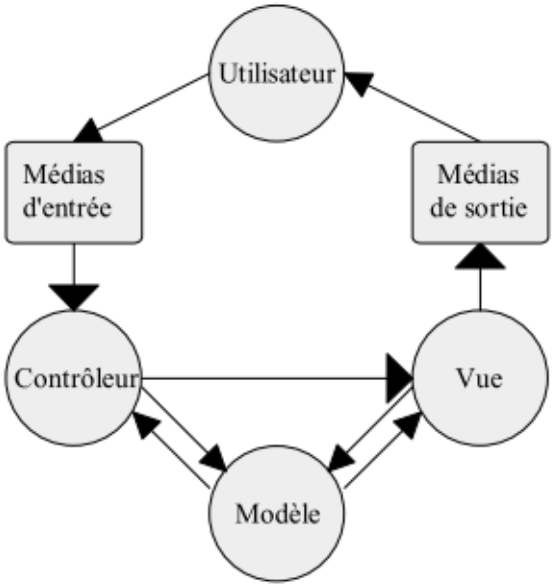


FIGURE 2.3 – Le modèle MVC [57]

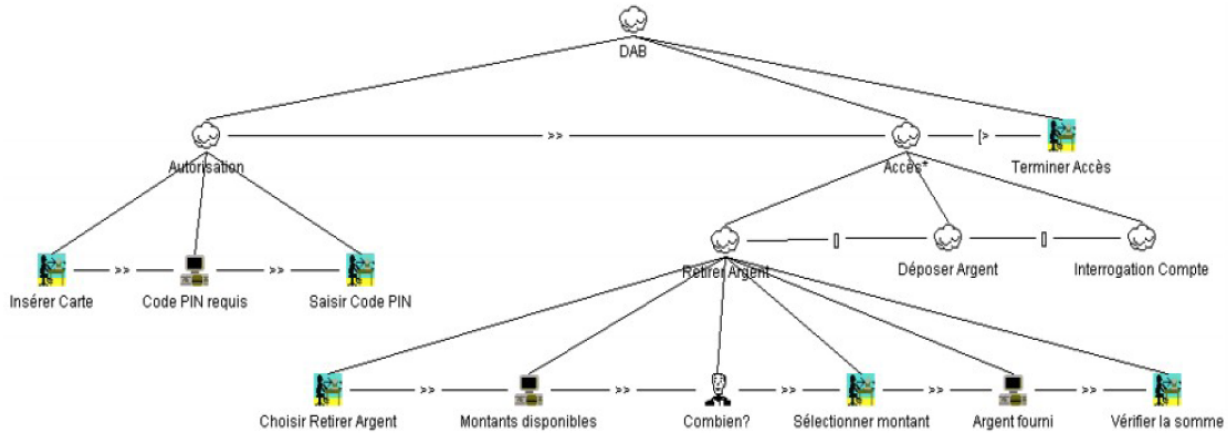


FIGURE 2.4 – Exemple de modèle CTT (distributeur automatique de billets)

Les agents dans le modèle PAC comportent aussi trois composants : la présentation (définit la partie en contact direct avec l'utilisateur), l'abstraction (définit les fonctions propres au domaine de l'application) et le contrôle (maintient la cohérence entre le composant d'abstraction et le composant présentation). La principale différence entre le modèle MVC et PAC est que le modèle MVC sépare les entrées (contrôleur) des sorties (vue), alors que PAC les rassemble au sein d'un même composant (présentation) [11].

Modèles de tâches. Contrairement aux modèles d'architecture, les modèles de tâches sont loin des implémentations des systèmes. Comme leur nom l'indique, ils définissent les tâches d'interaction entre l'utilisateur et l'application. L'arbre de tâches représente une décomposition des activités de l'utilisateur et du système [34]. Le modèle de tâches ConcurTaskTrees (CTT) [87] est le modèle le plus utilisé comme modèle d'interaction. Il définit les activités de l'utilisateur et le feedback système et propose de nombreux opérateurs temporels permettant de décrire différents comportements associés au système interactif (interruption, concurrence, désactivation, ...). Il possède quatre types différents de tâches : tâche abstraite (représente une tâche qui nécessite une interaction complexe), tâche interactive (représente une interaction de l'utilisateur vers l'application), tâche système (représente une tâche entièrement réalisée par l'application) et tâche utilisateur (représente une tâche entièrement réalisée par l'utilisateur).

Le modèle CTT (figure 2.4) a été utilisé pour la génération de code dans certains projets/approches visant à générer des interfaces homme machine (UsiXML [26]) par exemple).

26. `http://www.usixml.org/` (dernière consultation le 24/06/2014)

2.2.1.2 Transformation des modèles

Dans le contexte de l'IDM, la notion de transformation de modèle joue un rôle fondamental. Elle est définie dans [55] par « une génération automatique d'un modèle cible à partir d'un modèle source selon une définition de transformation ». Une définition de transformation regroupe un ensemble de règles pour transformer un modèle en un autre. Une règle de transformation est une description de la façon dont les concepts du langage source peuvent être transformés en d'autres concepts du langage cible. Ainsi, un moteur de transformation prend en entrée un modèle source, exécute les règles de transformation, et produit en sortie un modèle cible.
Il existe deux types de transformations de modèles : transformation Model-to-Model et transformation Model-to-Text. Le premier type de transformation permet le passage d'un modèle à un autre. Le deuxième type prend en entrée un modèle et le transforme vers du texte (souvent du code) telle que la génération du code Java par exemple. La transformation du modèle vers le code est généralement considérée comme un cas particulier de la transformation Model-to-Model (il suffit de définir le métamodèle du langage de programmation cible).

Dans [12], l'auteur décrit trois approches de transformations de modèles : l'approche par programmation, l'approche par template et l'approche par modélisation :

1. **L'approche par programmation**. Cette approche utilise les langages de programmation pour réaliser la transformation d'un modèle à un autre. Elle décrit les transformations sous forme d'un programme écrit généralement par un langage orienté objet [31].
2. **L'approche par template**. Les templates sont très utilisés dans le développement web. Par exemple, avec le langage PHP, le développeur peut créer des pages dynamiques à partir des canevas de pages ou des templates contenant des paramètres qui font référence aux données. De même, l'approche de transformation par template consiste à prendre un modèle template et à remplacer ses paramètres par les informations du modèle source pour définir un modèle cible. Ce type de transformation peut être dans un format graphique (template UML par exemple) ou textuel (template basé sur XMI [27], Acceleo [28], etc.).
3. **L'approche par modélisation**. La transformation par modèle permet de modéliser les transformations de modèles. Elle applique le principe de l'IDM sur les transforma-

27. `http://www.omg.org/spec/XMI/` (dernière consultation le 24/06/2014)
28. `http://www.obeo.fr/pages/acceleo/fr` (dernière consultation le 24/06/2014)

tions et définit des modèles de transformation (décrivant la correspondance entre les concepts du modèle source et celles du modèle cible) indépendants des plateformes. MOF 2.0 QVT (Query, Views, Transformation) est un exemple de standard qui a pour but de définir un métamodèle permettant l'élaboration des modèles de transformation de modèles.

Le plus important dans la transformation de modèles est la spécification des règles de passage d'un modèle source à un modèle cible. Le choix de la façon dont les règles sont implémentées vient en deuxième position. Les trois sortes de transformation sont très utilisées dans l'ingénierie dirigée par les modèles. Pour choisir une approche, les développeurs/concepteurs se basent généralement sur l'outillage disponible (comme dans notre cas, voir chapitre 6).

2.2.2 Critères IDM

L'existence de nombreux langages de modélisation indique l'utilité des modèles abstraits pour faciliter et accélérer le développement des applications informatiques. En plus de l'abstraction, l'IDM a d'autres caractéristiques importantes. Leur présence est essentielle pour permettre une modélisation efficace et une génération de code complète [38]. Les cinq principales caractéristiques sont décrites dans la suite.

2.2.2.1 Pertinence et concision des concepts

Cette caractéristique concerne les DSMLs. La pertinence consiste à identifier les concepts les plus pertinents et les plus utiles lors de la modélisation pour les définir comme une partie du DSML. Dans [44], Ulrich Frank explique la pertinence par « Si un concept est prévu pour être utilisé régulièrement, le définir comme une partie du DSML augmente la valeur du langage pour la plupart des utilisateurs. Si un concept est prévu pour être utilisé rarement, le définir comme une partie du langage augmente sa taille, le rend plus difficile à apprendre et la plupart des utilisateurs n'en bénéficient pas ».
La concision fait référence aux modèles produits. Un modèle est concis s'il est suffisamment clair et pas inutilement vaste [61]. Cela permet de se débarrasser des parties inutiles de modèle telles que les perspectives de génération, par exemple.

2.2.2.2 Guidage de modélisation

Le guidage lors de la modélisation est une caractéristique très importante et nécessaire pour réussir le processus de création des modèles de qualité. Elle doit fournir aux concepteurs les lignes directrices de modélisation afin de réduire l'effort et faciliter la difficulté liée aux activités de modélisation [43].
En génie logiciel, un grand nombre d'approches existent pour aider à résoudre les problèmes macroscopiques tels que les méthodes Agiles, les processus unifiés (Rational Unified Process), etc. On trouve aussi quelques solutions pour les problèmes microscopiques comme dans ArgoUML, où des questions de conception sont proposées aux praticiens UML pour les aider à évaluer la qualité de leur modèle.

2.2.2.3 Vérification des modèles

Malgré les lignes directrices qui doivent être fournies par les systèmes de guidage, le concepteur peut encore faire des erreurs de modélisation. Pour éviter la génération d'applications incorrectes, des mécanismes de vérification des modèles (model-cheking) sont nécessaires. Ces mécanismes sont alimentés par des règles et des contraintes que tous les modèles doivent satisfaire.
Selon l'environnement de modélisation, le moteur de vérification de modèle peut produire des rapports contenant les erreurs de modélisation et éviter la génération de code ou bien juste détecter ces erreurs et les montrer aux concepteurs. En outre, l'environnement peut aussi permettre aux concepteurs d'ajouter facilement de nouvelles règles de vérification.

2.2.2.4 Réutilisation des modèles

Par défaut, la stratégie de résolution des problèmes dans les activités humaines est la réutilisation [65]. La réutilisation de modèles est alors une bonne stratégie pour aider le concepteur et faire évoluer les modèles [46]. Les mécanismes de réutilisation des modèles sont principalement nécessaires pour éviter la re-modélisation des parties de modèles déjà modélisées. Avec de tels mécanismes, le concepteur gagnera du temps en modélisant une seule fois les éléments de modèle qui peuvent se répéter.
La réutilisation peut se faire par copie, par valeur ou par référence (copier et coller la partie du modèle), composants (partie de modèle encapsulée comme une boîte noire), des modèles de templates, etc., mais peut aussi être plus profonde à travers le mécanisme d'héritage par exemple.

2.2.2.5 Génération de code

La génération automatique de code est le type de transformation de modèle (modèle vers texte) qui oriente les modèles vers une utilisation plus productive. Grâce à la génération, les erreurs de programmation sont réduites et le code devient plus facile à lire et à maintenir [90].

La présence de ces différentes caractéristiques dans les approches basées sur des modèles est essentielle pour permettre une modélisation efficace (facile et sans erreur) et une génération de code complète et correcte. Les créateurs des approches à base des modèles doivent les prendre en compte dès la définition du langage de modélisation (pertinence et concision des concepts), puis lors de la création de l'environnement de modélisation associé (guidage, vérification et réutilisation de modèle), et finalement lors de l'implémentation des générateurs de code. Ainsi, ces approches tireront plus d'avantages de l'IDM et gagneront plus d'utilisateurs en facilitant et accélérant la création d'applications informatiques.

2.2.3 La modélisation graphique (notation visuelle)

En IDM, la modélisation graphique est favorisée [35]. Les diagrammes sont souvent utilisés pour définir la syntaxe concrète des langages de modélisation. Un diagramme est défini par Larkin [62] comme « une représentation dans laquelle l'information est indexée par un positionnement 2D » (« is a representation in which information is indexed by 2D location »). Il est défini par les humains pour les humains et il ne possède qu'une petite (voire nulle) valeur ajoutée dans la communication homme-machine [75]. Sa plus grande valeur est qu'il permet au développeur de monter en abstraction et d'avoir une vue complète de son application (en plus de la communication entre développeurs, concepteurs, etc.).
On appelle « Notation visuelle » l'ensemble des symboles graphiques que le créateur d'un langage de modélisation affecte aux concepts. Le développeur/concepteur utilisera ainsi ces symboles pour créer le diagramme qui correspondra à une vue graphique sur le modèle de son application (conformément au langage de modélisation). Il a été démontré qu'une notation visuelle doit être de qualité pour qu'elle soit intelligible. Daniel Moody [74] a défini la « physique des notations » qui donne neuf critères pour concevoir des notations visuelles de qualité (cognitivement efficaces), [64], à savoir :

1. **Clarté sémiotique** : elle traite la relation bijective entre l'ensemble des concepts abstraits du langage de modélisation et l'ensemble des éléments de la syntaxe concrète. Pour éviter la redondance, la surcharge, l'excès ou le déficit de symboles, il est conseillé

d'avoir un et un seul élément concret pour chaque élément abstrait. Cet élément concret ne doit pas être lié à plusieurs éléments abstraits.

2. **Transparence sémantique** : à la lecture d'un diagramme, l'intelligence perceptive du lecteur va attacher du sens à chaque élément visuel. Il est donc important que le sens de la représentation visuelle de chaque concept ne soit pas trop éloigné de son sens (la forme implique le contenu).
3. **Discriminabilité perceptive** : elle définit le niveau de discrimination visuelle entre deux représentations visuelles différentes. Plus le niveau est élevé, plus la perception des diagrammes est rapide et plus le traitement cognitif qui suivra en sera donc facilité. Il est donc primordial de pouvoir discerner facilement chaque notation graphique par rapport aux autres.
4. **Gestion de la complexité** : la représentation d'un système complexe est une thématique de recherche transversale à de nombreuses disciplines. C'est aussi une problématique lorsqu'il s'agit de représenter des modèles complexes, c'est-à-dire constitués de nombreux éléments et/ou nombreuses connexions. L'encapsulation graphique, la fragmentation en différents diagrammes sont des exemples de mécanismes possibles pour gérer la complexité.
5. **Intégration cognitive** : lorsque le lecteur navigue dans les différents diagrammes via les mécanismes de gestion de complexité associes, il lui faut des éléments visuels lui permettant de se rappeler dans quel contexte se situe le diagramme qu'il/elle a sous les yeux. En d'autres termes, il faut pouvoir intégrer la partie du modèle étudié dans la représentation mentale du modèle global. Par exemple, pour une page web, il est classique d'afficher le chemin de la page actuelle au sein de la hiérarchie du site.
6. **Expressivité visuelle** : une syntaxe concrète est expressive visuellement si elle exploite un grand nombre de variables visuelles et utilise un grand nombre de valeurs pour chacune d'elles. Plus l'expressivité visuelle est grande, plus une syntaxe est efficace cognitivement. Ce critère est proche de la discrimination perceptive.
7. **Double codage** : même s'il n'est pas conseillé d'utiliser seulement du texte pour représenter un élément de modèle, l'annotation textuelle est par contre très conseillée pour renforcer une forme géométrique ou une icône.
8. **Economie graphique** : il convient de ne pas avoir un vocabulaire visuel trop important, c'est-à-dire d'utiliser un trop grand nombre de formes/liens différents. Pour une notation trop riche, l'activité mémorielle dédiée à l'association représentation-sens de-

vient, chez les lecteurs (non-experts), trop importante et gênera la lecture des diagrammes. Ce critère implique par exemple de partitionner les modèles en diagrammes de types différents. La clarté sémiotique est ici en défaut, car l'économie graphique peut impliquer, dans le cas de langages sémantiquement riches, un déficit de symboles dû au partitionnement.

9. **Adaptation cognitive** : les capacités de dessin sont en rapport avec le support utilisé pour le dessin des diagrammes. Par exemple, l'utilisation des images complexes est peu viable lorsque les diagrammes se feront au stylo sur une feuille de papier. Parallèlement, le niveau d'expérience du lecteur concernant l'écriture de diagrammes est aussi à prendre en compte. En effet, les différents mécanismes comme la fragmentation en plusieurs diagrammes, par exemple, demandent moins d'effort dans leur utilisation chez une personne expérimentée que chez un débutant.

La considération de ces critères permet de mettre en œuvre une notation visuelle de qualité. Cependant, il n'est pas facile de les prendre tous en compte. Peu de travaux et encore moins d'outils abordent l'évaluation de la qualité des notations. Ainsi, lors de la création des langages de modélisation, la qualité de la syntaxe concrète est souvent négligée [64].

2.3 Conclusion : pourquoi l'IDM pour la multimodalité mobile ?

Nous concluons ce chapitre en présentant l'intérêt d'une approche IDM pour la création des applications mobiles multimodales. Nous discutons aussi les avantages de la modélisation graphique de ce type d'applications.

2.3.1 L'intérêt d'une approche IDM pour la multimodalité mobile

En 1992, une étude sur la programmation des interfaces homme-machine [78] a montré que 48% du code des 74 logiciels informatiques étudiés, 45% de leur temps de conception, 50% de leur temps de développement et 37% de leur temps de maintenance sont consacrés aux interfaces homme-machine. Une autre étude (1999) montre que la programmation des IHMs, même en utilisant des boîtes à outils, n'est pas aisée [77]. Cela reste toujours valable aujourd'hui [72], car les besoins des systèmes interactifs ont considérablement augmenté pendant ces dernières années. De plus, l'utilisation des différentes modalités d'interaction en entrée et en sortie et leurs combinaisons dans des applications multimodales augmentent

encore l'effort et le temps de développement des interfaces homme-machine.

Les modèles sont depuis longtemps utilisés pour faciliter et réduire le temps de développement des interfaces. Au début, ils ont été utilisés pour faciliter l'implémentation avec les systèmes de gestion d'interfaces (User Interface Management Systems (UIMS)). Le but était de séparer le noyau fonctionnel de l'interface graphique dans les applications (modèle Seeheim, Arch, etc.). Ensuite, avec l'apparition de l'IDM, on est passé de la programmation à l'ingénierie des IHMs. Les modèles (modélisation des tâches, des évènements d'interaction, etc.) sont devenus plus productifs, réduisant ainsi le temps de développement. Beaucoup des premières approches de génération automatique (et semi-automatique) du code à partir des modèles ont échoué [77]. Cependant des progrès significatifs ont été réalisés au fil des années pour la génération totale ou partielle des interfaces.
L'utilisation d'une approche à base de l'IDM pour la modélisation et la génération des interfaces mobiles multimodales a de nombreux intérêts :

- Elle réduit significativement le temps de développement (jusqu'au tiers du temps de développement manuel). Cela a été prouvé pour les applications en médecine et pour les systèmes de contrôle nucléaire par exemple [29]. La génération automatique du code joue un rôle très important dans la réduction du temps de développement.
- Elle permet aussi d'élever le niveau d'abstraction, et réduire ou éviter partiellement la programmation à bas niveau (moins d'effort de programmation, mais on ajoute toujours l'effort de modélisation).
- Les modèles issus d'une approche IDM peuvent être utilisés par des concepteurs d'interface qui ne sont pas forcément des programmeurs.
- L'utilisation des modèles accélére le développement d'applications dans le contexte des plateformes hétérogènes, comme le contexte actuel des plateformes mobiles (Android, Windows Mobile, iOS, Blackberry, Badda, Symbian, etc.).
- La modélisation graphique des applications mobiles multimodales donne une vue d'ensemble des interactions en entrée et en sortie. Cette vue aidera à la conception ergonomique. Elle facilitera la détection des conflits entre modalités d'interaction ainsi qu'entre combinaisons de modalités qui peuvent être facilement perdues dans le code.

On pourrait se demander « pourquoi ne pas utiliser des APIs de haut niveau pour implémenter les interfaces au lieu de suivre une approche IDM ? ». Et aussi, « pourquoi ne pas juste copier/coller les parties de code qui peuvent se répéter (lors de l'utilisation des capteurs

29. `http://ercim-news.ercim.eu/en91/ri/advances-in-model-driven-software-engineering` (dernière consultation le 24/06/2014)

par exemple) au lieu de les générer automatiquement ? ».
Notre réponse est que, premièrement, il y a une grande différence entre générer et copier/coller du code. Copier/coller est une activité manuelle et elle peut donc provoquer des erreurs alors que la génération est automatisée et sans erreur. De plus, le gain n'est pas toujours assuré avec le copier/coller. Il faut copier des grandes parties du code puis les adapter une seule fois aux besoins des nouvelles applications afin de gagner en temps de développement. Deuxièmement, bien que les APIs puissent être une bonne solution pour une manipulation de plus haut niveau des capteurs (implémentation des modalités d'interaction) et pour l'implémentation des mécanismes de synchronisation entre modalités, elles sont néanmoins coûteuses à développer et utiliser. Il faut développer des API pour les différentes plateformes mobiles (Android, Windows Mobile, iOS, Blackberry, Badda, Symbian, Meego, etc.) et les développeurs doivent être familiers avec elles ce qui nécessite beaucoup d'efforts et de compétences en programmation. Alors que dans une approche à base de modèles, il suffit de créer un seul modèle pour générer du code sous toutes les plateformes (les modèles étant indépendants des plateformes) sans aucun effort de développement (ou beaucoup moins). En outre, l'utilisation des modèles résiste à l'évolution technologique (puisqu'ils sont abstraits) et simplifie la compréhension de l'application, même après un certain temps. La modélisation graphique joue aussi un rôle important dans la facilité de lecture de l'application surtout en ce qui concerne l'interaction homme-machine.

2.3.2 Modélisation graphique pour la multimodalité sous mobile

La modélisation graphique est préférable pour la création des interfaces mobiles multimodales, car elle permet de donner une vue sur l'ensemble des interactions et de cacher les détails de l'implémentation. La vue d'ensemble facilitera la détection des conflits. Par exemple, un conflit peut se révéler entre des interactions traitées en même temps, surtout lorsque leurs capteurs sont basés sur le même périphérique physique (le capteur de gravité est un périphérique virtuel basé sur l'accéléromètre, le capteur de proximité est basé sur le capteur de lumière, etc.). Du point de vue multimodalité, la représentation graphique facilite la détection des modalités d'interaction non compatibles (un secouage du téléphone combiné en complémentarité avec un geste tactile) ainsi que les problèmes de synchronisation entre interactions. Les modèles graphiques peuvent être aussi un bon moyen de communication entre les participants (développeurs, concepteurs, ergonomes, etc.), comme ils peuvent faire partie d'une documentation facile à lire et à maintenir.
Cependant, il est difficile de définir une notation graphique claire et facile à comprendre/

apprendre. Il faut non seulement prendre en compte les critères qui permettent de concevoir une notation de qualité (section 2.2.3), mais des tests et des évaluations doivent aussi être faits avec les utilisateurs finaux afin d'évaluer les réponses à leurs besoins.

Chapitre 3

État de l'art sur les approches orientées modèles pour l'Ingénierie des interfaces mobiles multimodales

Sommaire

Dans les deux chapitres précédents, nous nous sommes concentrés sur l'informatique mobile, la multimodalité et l'IDM. Nous avons montré l'intérêt de la multimodalité pour mobile ainsi que l'intérêt d'une approche IDM pour la multimodalité mobile. Dans ce chapitre, nous présentons des approches que nous avons choisies pour illustrer les travaux qui utilisent les modèles pour le développement des applications multimodales et des applications mobiles. Nous étudions d'abord les approches pour créer des interfaces multimodales, notamment celles qui s'intéressent au mobile. Nous décomposons ces approches en quatre catégories selon les quatre paradigmes de modélisation utilisée : état-transition, réseaux de Petri, hiérarchique et à base de composants [38]. Puis nous étudions les plateformes dédiées à la création d'applications mobiles, notamment celles qui permettent l'intégration des interactions multimodales à base de capteurs. Nous détaillons puis analysons ces approches en fonction des quatre points ci-dessous qui représentent, pour nous, les spécificités de la multimodalité mobile que chaque langage/approche doit respecter :

Support de la multimodalite : les approches doivent prendre en compte les interactions mobiles multimodales en entrée et en sortie. Le retour d'information (feedback) est très important pour les applications mobiles [36]. Les interactions en entrée ont des effets directs sur les sorties de l'application comme le déclenchement de vibrations, le changement d'affichage, etc. Lorsque l'utilisateur emploie une modalité inhabituelle, il est essentiel de l'informer qu'il a (bien) effectué cette commande. Pour les interactions non tactiles, on ne peut pas être sûr que l'utilisateur sera en mesure de voir l'écran. Ainsi, le créateur d'application mobile doit avoir la possibilité de choisir les modalités en entrée ainsi que les modalités adéquates en sortie pour fournir les bons feedbacks. La modélisation de ces interactions avec le même langage permet d'avoir une vue générale de l'interaction afin de faciliter la conception ergonomique.
L'approche doit intégrer aussi les combinaisons de modalités en entrée et/ou en sortie qui permettent de rendre l'interaction plus sécurisée (secouer le téléphone en cliquant sur le bouton physique du volume pour effectuer une action est plus sécurisée par rapport au secouage seul, cette combinaison permet d'éviter le déclenchement des traitements en cas des secouages involontaires) ;

Support des capteurs mobiles : il est essentiel que les approches permettent d'exploiter les capteurs mobiles en interaction. Pour cela, le niveau d'abstraction doit être suffisamment élevé afin de cacher la complexité d'implémentation (les données de bas niveau, le bruit, etc.). En même temps, ce niveau doit permettre de définir les interactions en détails (pour la modalité « Accélération » par exemple, il faut savoir si c'est

« secouage », « chute libre », etc.). Cela permet de donner plus de lisibilité lors de la modélisation ;

Productivité des modèles : les approches qui permettent la génération de code sous les différentes plateformes mobiles permettent d'améliorer la productivité, réduisent l'effort et les erreurs de programmation et rendent le code plus facile à lire et à maintenir [90] ;

Traitement de l'hétérogénéité sur mobile : la prise en compte de l'hétérogénéité sur mobile est devenu nécessaire surtout avec la fragmentation de la façon dont les capteurs mobiles sont accessibles et configurés, l'évolution des différentes plateformes, l'apparition de nouveaux appareils mobiles, etc.

Nous concluons ce chapitre par une synthèse de l'état de l'art qui montre les limites des approches existantes et les besoins nécessaires pour faciliter la création des applications mobiles multimodales riches de nouvelles modalités à base de capteurs et de combinaisons de modalités en entrée et/ou en sortie.

3.1 Approches pour la création des interfaces multimodales

Nous avons décomposé les approches à base de modèles pour l'ingénierie des interfaces multimodales en quatre catégories selon les quatre paradigmes de modélisation de l'interaction multimodale :

- Paradigme état-transition
- Paradigme réseaux de Petri
- Paradigme hiérarchique
- Paradigme des composants

3.1.1 Le paradigme état-transition

En utilisant le paradigme état-transition, la modélisation de l'interaction multimodale est composée d'un nombre fini d'états, de transitions et d'actions. Les états sont reliés par les transitions afin de permettre le passage d'un état à un autre et/ou provoquer une action. Chaque transition est étiquetée par un événement en entrée et la modalité d'interaction associée, tandis que les états sont généralement étiquetés par les événements de sortie et leurs modalités en sortie.

3.1.1.1 SMUIML / HephaisTK

SMUIML (Synchronized Multimodal User Interaction Modeling Language) [32] est un langage basé sur XML pour modéliser les interfaces multimodales en entrée. Il a été créé afin de configurer la plateforme HephaisTK [32] pour la création rapide d'interfaces multimodales. Cette plateforme basée sur les agents s'intègre dans l'application multimodale cible. Selon le document de configuration SMUIML, qui décrit le dialogue homme-application, la plateforme gère les « Recognizers » de modalités, reçoit et encapsule les données à partir de ces « Recognizers », gère la fusion des interactions combinées et notifie l'application multimodale.

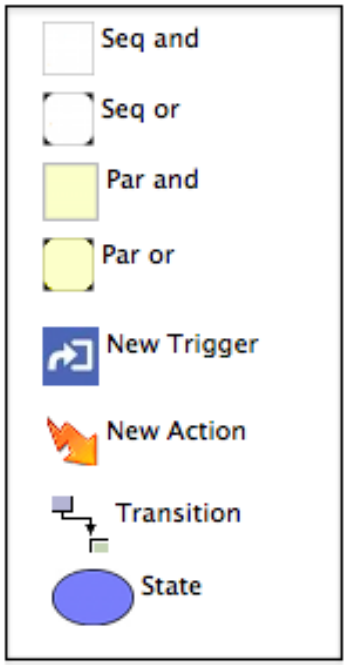


FIGURE 3.1 – Notation visuelle des concepts SMUIML

SMUIML est basé sur le paradigme état-transition. Il modélise les interactions multimodales de la manière suivante :

- Le dialogue d'interaction est modélisé par une machine à états où les états représentent les éléments du contexte connectés par les transitions.
- Une transition définit l'évènement en entrée avec la modalité d'interaction associée.
- Chaque transition permet soit le passage vers un autre état, soit de rester dans le même état en déclenchant une action.
- Les combinaisons entre modalités d'interaction en entrée sont modélisées par les opérateurs « seq and », « seq or », « par and » et « par or » (« par » pour « parallel » et

« seq » pour « sequential ») (propriétés CARE).

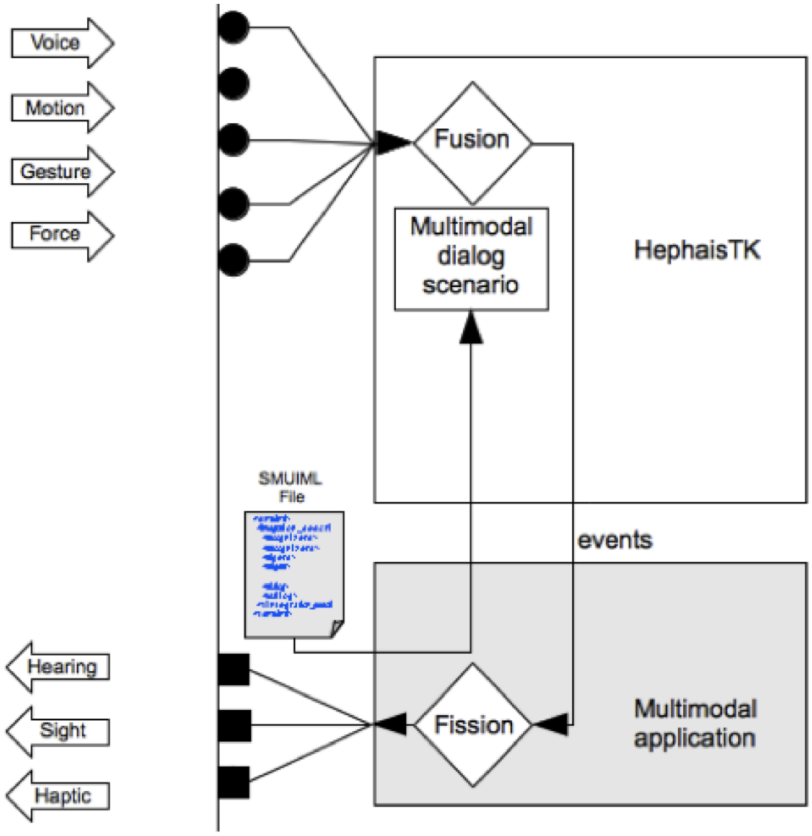


FIGURE 3.2 – Le fonctionnement de SMUIML/HephaisTK [71]

SMUIML permet une modélisation structurée et claire [38]. Il possède un éditeur graphique pour modéliser graphiquement les interfaces multimodales en entrée [71]. La modélisation graphique a un sens de lecture bien précis. La majorité des symboles ont une transparence sémantique. Cependant, la distinction entre les symboles des combinaisons entre modalités (« seq and », « seq or », « par and » et « par or ») n'est pas évidente (figure 3.1). La discrimination visuelle (le troisième critère de la physique des notations) est très réduite ce qui ne permet pas de discerner facilement chaque symbole.

Après la modélisation graphique, l'éditeur SMUIML génère un script de configuration (« la productivité des modèles est limitée à la configuration de HephaisTK ») utilisé avec un autre fichier Java pour configurer HephaisTK rattaché à l'application cible. HephaisTK reçoit et fusionne les entrées de l'application selon les combinaisons des modalités spécifiées et informe l'application. Ainsi, l"interaction multimodale d'entrée est gérée automatiquement sans la participation des développeurs (même si certaines configurations manuelles sont nécessaires). Cependant, « la multimodalité en sortie n'est pas traitée dans

SMUIML/HephaisTK », c'est l'application multimodale qui s'en charge (figure 3.2).

À la base, SMUIML/HephaisTK n'a pas été créé pour permettre la modélisation des interactions avec des périphériques mobiles. Cependant, son adaptation pour concevoir des applications mobiles multimodales en entrée a été réalisée. Dans [28], par exemple, le langage de modélisation proposé est basé sur SMUIML pour modéliser des applications mobiles multimodales (applications Android uniquement -pas de traitement d'hétérogénéité). Ensuite, le framework (pas encore disponible publiquement) gère automatiquement le processus de fusion comme le fait HephaisTK. Le problème de cette approche est qu'elle se limite au traitement des modalités vocale et tactile uniquement. Elle ne traite pas les interactions à base de nouveaux capteurs mobiles bien que SMUIML propose un niveau d'abstraction adapté pour les modéliser. Il définit l'interaction à deux niveaux : 1) le niveau réception (concepts Recognizer dans le méta-modèle (figure 3.3)) qui permet de définir le capteur responsable ainsi que la modalité de l'interaction ; 2) le niveau de détection d'évènement issu de cette interaction (concepts « Trigger »). L'utilisateur du langage peut ainsi modéliser les interactions en détail (« Recognizer »= Accéléromètre, « Modality »= Accélération et « Trigger »= Secouage, par exemple).

3.1.1.2 MMIR

Le framework MMIR (Multimodal Mobile Interaction and Rendering) [96] fonctionne presque de la même façon que HephaisTK, mais pour les applications web sur mobile. Son but est de faciliter la construction des systèmes légers de dialogue (sous le navigateur). MMIR **traite les interactions en entrée et en sortie pour des modalités particulières** (tactile, reconnaissance vocale et gestuelle en entrée, affichage, audio et synthèse vocale en sortie). Cependant, **il ne traite pas les combinaisons entre interactions** (ni en entrée ni en sortie).

En entrée, il se base sur le langage SCXML (State Chart XML) pour décrire les interactions utilisateurs qui seront traitées et envoyées vers l'application (un exemple est montré à la figure 3.4). En sortie, les interactions sont modélisées **séparément** avec des templates HTML. Le développement de MMIR est en cours. Cependant, il ne fonctionne que sous Android et le navigateur Chrome (il utilise des APIs qui ne fonctionnent que sous Android pour la gestion de la reconnaissance et de la synthèse vocale -**pas d'hétérogénéité**), alors que les applications web sont censées être multi-plateforme.

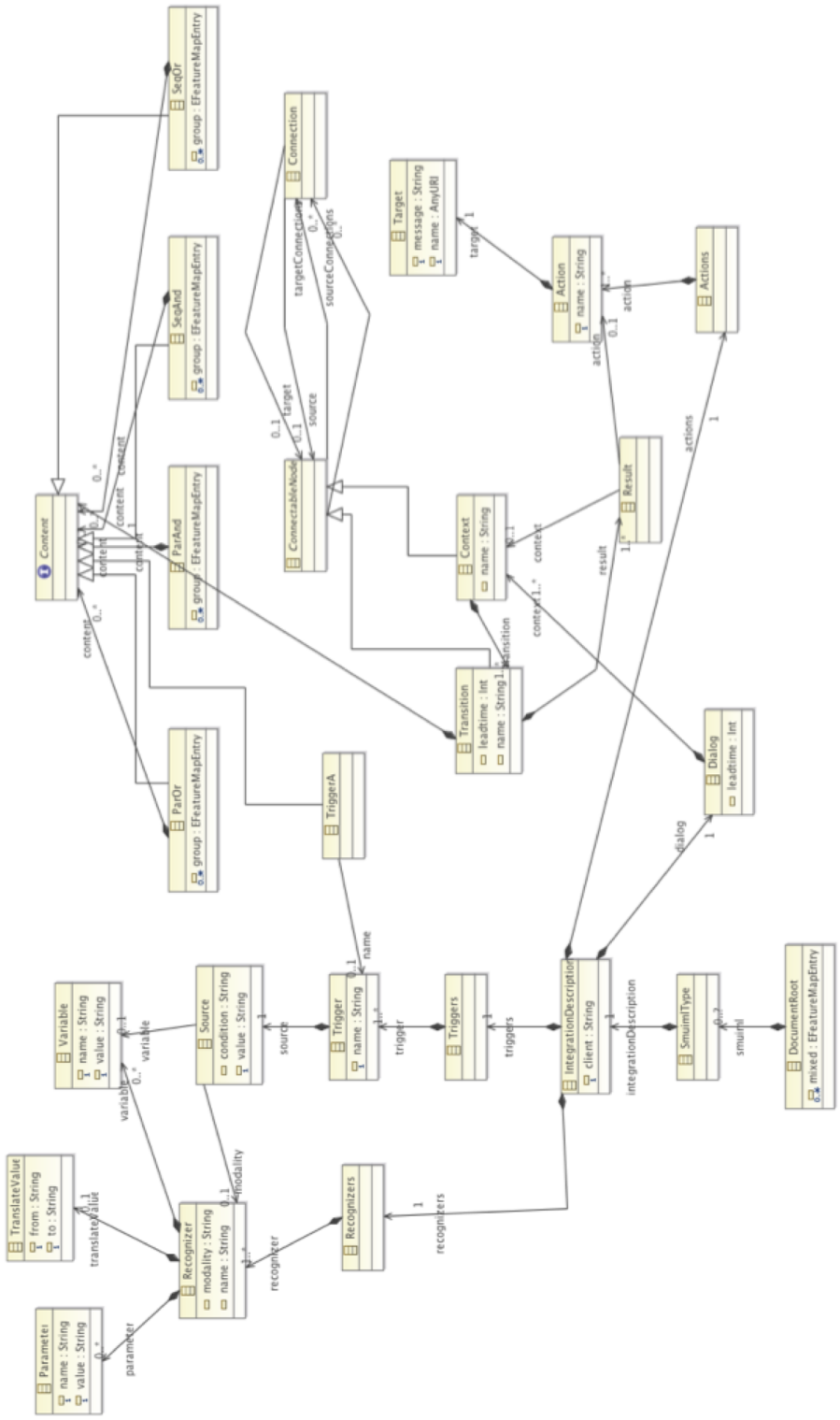


FIGURE 3.3 – Le méta-modèle de SMUIML [71]

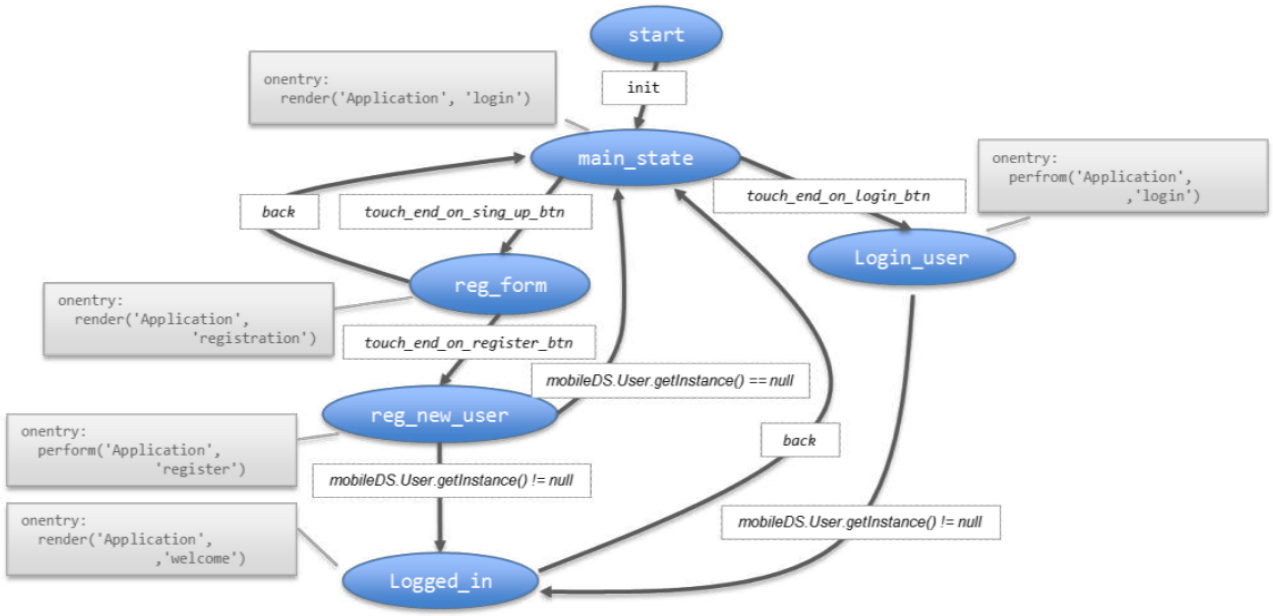


FIGURE 3.4 – Un exemple d'une modélisation graphique de la procédure de connexion d'une application sous SCXML[96]

3.1.1.3 IMBuilder / MEngine

IMBuilder/MEngine [16] utilise les machines à états finis (Finite State Machine ou FSM) pour modéliser puis exécuter rapidement des interfaces multimodales (mobile ou classique). Il est composé de deux modules : IMBuilder et MEngine. IMBuilder est l'environnement graphique de modélisation. Il permet de spécifier les interactions multimodales sous forme d'automates finis. En utilisant les fichiers XML générés par IMBuilder, MEngine exécute directement les interactions des utilisateurs combinant parole et geste. MEngine lance la reconnaissance vocale ainsi que la reconnaissance des gestes (mouvement de souris uniquement) et traite les actions utilisateurs selon les machines à états modélisées.

Cette approche ne génère pas le code des interfaces, mais permet de réaliser des prototypes rapides des interfaces afin de les tester avant l'implémentation (**productivité limitée**). C'est un framework destiné aux concepteurs d'interfaces. Le langage de modélisation est générique (**permet de modéliser tous types d'interaction**) alors que le moteur d'exécution ne traite que les interactions **vocales et gestuelles** (gestes 2D effectués à la souris). **Il traite aussi les combinaisons entre modalités, mais en entrée uniquement** (combinaisons séquentielles, complémentaires (figure 3.5), équivalentes ou redondantes). Les concepteurs qui utilisent IMBuilder pour modéliser les combinaisons doivent faire attention à l'ordre des

modalités d'interaction. Si l'ordre n'est pas important dans une commande multimodale, le concepteur doit modéliser tous les ordres possibles d'utilisation des modalités.

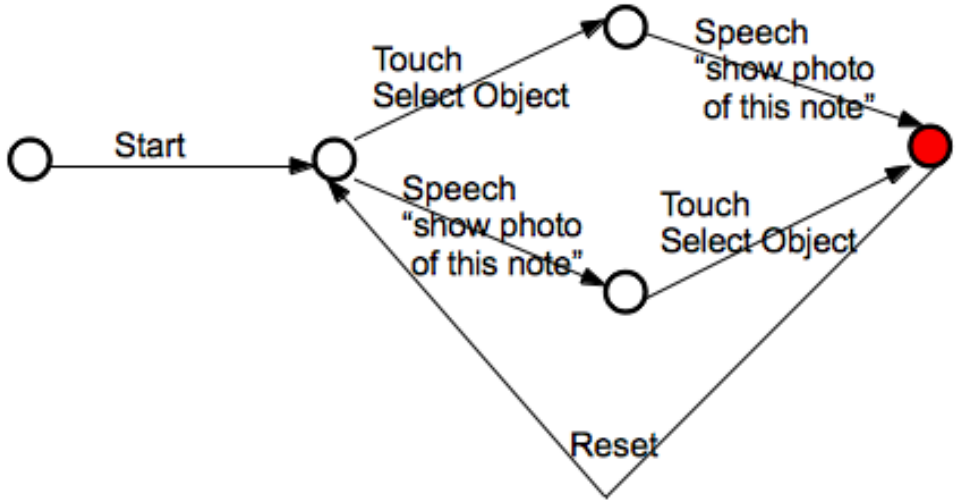


FIGURE 3.5 – La consultation de la photo d'une note dans l'application « Prise de notes » exprimées à l'aide de IMBuilder

Par exemple, dans la figure 3.5, nous modélisons une complémentarité pour sélectionner une photo d'une note à partir d'une liste des notes. Nous avons modélisé cette complémentarité entre deux interactions (vocale et tactile) en deux séquences différentes, car l'ordre n'est pas important : sélectionner la note puis dire qu'on veut voir la photo de la note ou bien parler avant de la sélectionner.
Le concepteur doit alors assimiler les règles de modélisation avec les FSMs afin de modéliser correctement et tester différentes versions d'interaction multimodale sous IMBuilder / MEngine.

3.1.1.4 UMAR / SIHMM

L'approche UMAR (User Modalities Artefact Representation)/SIHMM (Simulateur de l'Interaction Homme-Machine Multimodale) [63] permet la description des interactions multimodales (avec UMAR) puis leur simulation (**pas de génération de code**) en utilisant la plateforme SIHMM (d'un point de vue utilisateur).

UMAR permet de modéliser à la fois **les interactions multimodales en entrée et en sortie**. Cependant, **il ne modélise pas les interactions en détail (on ne voit pas le type de l'évènement)**. Les modalités sont organisées en trois catégories :

- Les modalités d'action : elles sont utilisées en entrée pour interagir avec le système comme le tactile. Elles sont notées par le 7-uplet <Cat, P, R, Co, Tr, Ti, Te> où Cat : la Catégorie de la modalité (tactile, vocale...), P : le dispositif Physique utilisé (écran, microphone...), R : le système Représentationnel (langage gestuel, langage naturel, etc.), Co : la modalité de Contrôle attachée à la modalité d'action, Tr : le Temps de réalisation, Ti : le Temps d'interprétation et Te : le Temps d'exécution.
- Les modalités de sortie (ou feedback) : elles sont utilisées pour exprimer les retours des systèmes tels que l'affichage. Elles sont modélisées par le triplet <P,R,D> où D définit le temps nécessaire pour exprimer la modalité (la durée de l'interaction en sortie).
- Les modalités de contrôle : elles sont utilisées pour contrôler d'autres modalités, par exemple, la modalité de contrôle du vocal est l'audition.

Les propriétés CARE de combinaison entre les modalités sont aussi traitées (« ° » pour complémentarité, « & » pour redondance et « || » pour l'équivalence).
La modélisation UMAR se base sur le paradigme d'état. Il utilise une notation de graphes à base d'états simultanés (parallèles) et hiérarchiques (figure 3.6). Le parallélisme est utilisé, car le fil du dialogue homme-machine est souvent parallèle (les modalités sont souvent indépendantes les unes des autres). Les modèles sont ainsi modulaires et peuvent être structurés sur des sous-diagrammes simultanés. La description modulaire et le parallélisme entre les sous-diagrammes fournissent plus de lisibilité. Cependant, le formalisme ne possède pas un sens de lecture précis.

UMAR/ SIHMM permet aussi la modélisation et la simulation des systèmes mobiles. La simulation avec SIHMM montre aux concepteurs et développeurs la réalité de l'interaction prévue au sein des systèmes interactifs multimodaux mobiles.

La synthèse des approches basées sur le paradigme état-transition est résumée dans le tableau 3.1.

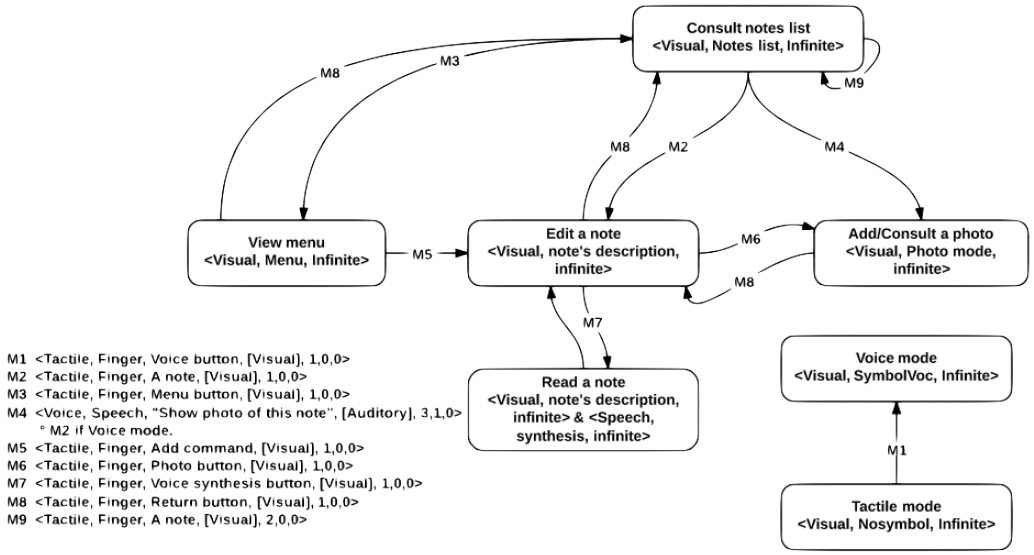


FIGURE 3.6 – L'application « Prise de notes » modélisée avec UMAR

	Interaction multimodale	**Combinaisons**	**Interaction à base de capteurs mobiles**	**Productivité des modèles**
SMUIML / HephaisTK	En entrée uniquement	CARE	Oui (deux niveaux de modélisation)	Interprétation (pas d'hétérogénéite)
MMIR	En entrée et en sortie (deux langages différents)	/	Non	Interprétation (pas d'hétérogénéite)
IMBuilder / MEngine	En entrée uniquement	CARE	Non	Prototypage uniquement
UMAR / SIHMM	En entrée et en sortie	CARE	Non	Simulation

TABLE 3.1 – Synthèse des approches basées sur le paradigme état-transition

3.1.2 Paradigme réseau de Petri

Un réseau de Petri est constitué de quatre concepts :

– Les états (ou places) qui modélisent généralement les interactions en entrée.
– Les noeuds (ou transitions) qui modélisent les évolutions de l'application entre deux

places.
- Les arcs entre les places et les transitions.
- Les jetons distribués sur les places qui modélisent l'état actuel de l'application en fonction de leur répartition dans les places.

Plusieurs extensions du paradigme réseau de Petri ont été créées pour modéliser les interactions. Cependant, très peu d'entre elles s'intéressent aux interactions multimodales. Nous notons ICO/PetShop comme approche basée sur des réseaux de Petri pour modéliser la multimodalité.

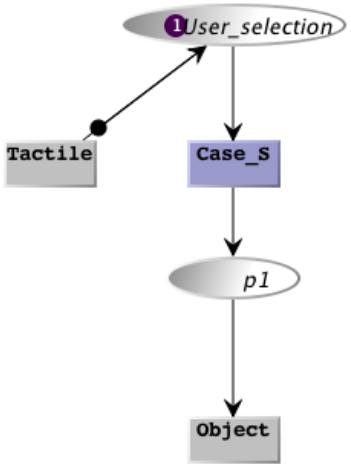


FIGURE 3.7 – La sélection en tactile modélisée avec PetShop

3.1.2.1 ICO / PetShop

Le réseau de Petri est utilisé dans le formalisme ICO (Interactive Cooperative Object) pour modéliser les aspects dynamiques et comportementaux des applications. Les aspects structurels de ces applications sont modélisés par des concepts orientés objet.

Les modèles ICO sont exécutables afin de tester les interactions multimodales **en entrée** avant leurs implémentations (**productivité limitée**). PetShop [86] est un outil graphique pour modéliser les applications multimodales selon le formalisme ICO. Les interactions multimodales sont donc modélisées graphiquement avec la spécification des places et des transitions des réseaux de Petri (voir l'exemple de la figure 3.7) puis exécutées directement à partir de l'éditeur (déplacement des jetons). L'exécution permet au développeur de vérifier le comportement prévisible de l'application.

Les modalités d'interaction sont bien gérées dans cette approche. Les états (ou places) peuvent définir les modalités souhaitées ainsi que leurs évènements détaillés (figure 3.7). Cependant, **les combinaisons entre ces modalités ne sont pas explicitement définies dans les modèles**. Il faut créer des classes ICO pour définir les algorithmes de combinaisons. L'intégration des objets permet d'augmenter le pouvoir d'expression des réseaux de Pétri. Ainsi, le pouvoir d'expression des modèles créés avec PetShop est très élevé et n'importe quelle application multimodale peut être modélisée selon [15].

ICO/PetShop fournit une solution pour faciliter la phase de conception des applications multimodales. Les problèmes et conflits d'interaction peuvent être détectés tôt et résolus avant de passer à l'implémentation.

La synthèse de l'approche est résumée dans le tableau 3.2.

	Interaction multimodale	**Combinaisons**	**Interaction à base de capteurs mobiles**	**Productivité des modèles**
ICO/ PetShop	En entrée uniquement	CARE	Oui	Exécution des modèles

TABLE 3.2 – Synthèse de l'approche basée sur le paradigme réseau de Petri

3.1.3 Paradigme hiérarchique

En interaction homme-machine, le paradigme hiérarchique est généralement utilisé pour modéliser les arborescences des tâches d'interaction. CTT (section 2.2.1.1) est une des notations les plus populaires pour la modélisation des tâches. Elle est très utilisée dans des environnements de modélisation et réalisation d'applications multimodales tels que les environnements décrits dans cette section.

3.1.3.1 CTT / DynaMo-AID

DynaMo-AID (Dynamic Model-Based User Interface Development) [23] est un environnement de modélisation et de développement des interfaces sensibles au contexte. Il utilise un modèle de tâches et d'autres types de modèles (modèle de contexte, de dialogue et de présentation) pour spécifier les interfaces. Le modèle de tâches est réalisé par le développeur en

utilisant une extension du modèle CTT pour des tâches adaptables au contexte avec une modélisation explicite des modalités d'interaction. Les modèles de contexte et de présentation sont aussi définis par le développeur alors que le modèle de dialogue est généré automatiquement à partir des deux modèles (de tâches et de contexte). L'outil offre quatre vues pour visualiser les quatre modèles et propose des mises en correspondance entre modèles afin de conserver les choix de conception [102]. **DynaMo-AID permet la modélisation des interfaces multimodales en entrée et en sortie**. Il permet aussi l'exécution des modèles de tâches multimodales et facilite la simulation de contexte. Les modèles sont sérialisés dans un langage à base d'XML appelé « DynaMOL » et l'architecture à l'exécution les utilise pour générer le code de l'interface (**en J2ME pour les interfaces mobiles, Java Swing pour les applications classiques ou HTML pour les applications web**).
Le problème de cette approche est qu'elle se base sur plusieurs modèles différents qui peuvent rendre la modélisation difficile à réaliser.

3.1.3.2 UsiXML

Les modèles de tâches sont aussi utilisés (avec d'autres modèles) pour la modélisation des applications **web mobiles** multimodales **en entrée et en sortie** sous UsiXML (USer Interface eXtensible Markup Language) [66].
UsiXML utilise le modèle de contexte, de tâches, de dialogue et de domaine pour pouvoir **générer le code de l'application (modèles productifs)**. Il est structuré en fonction des quatre niveaux du canevas Cameleon [19] : T&C (Task & Concepts), AUI (Abstract User Interface), CUI (Concrete User Interface) et FUI (Final User Interface).
La modélisation des interactions multimodales s'effectue au niveau CUI. Le méta-modèle concret (figure 3.8) intègre des modalités d'interaction particulières (tactile, vocal et graphique). Cependant, ce problème peut être résolu par l'ajout d'autres modalités ou même l'ajout des types d'interactions de ces modalités (comme des concepts qui héritent leurs modalités). Ainsi, l'utilisateur aura la possibilité de modéliser les différentes interactions à base de capteurs.
Des outils comme IdealXML proposent plusieurs éditeurs pour la création des modèles de tâches, modèles abstraits et concrets. Ils permettent aussi la mise en correspondance entre les modèles, la simulation des arbres des tâches et la génération des modèles de dialogue [102]. Basé sur ces modèles et après de nombreuses transformations, le code final de l'interface est généré. Comme DynaMo-AID, le principal problème d'UsiXML réside dans la multitude des modèles, de transformation de modèle et d'outils existants [97].

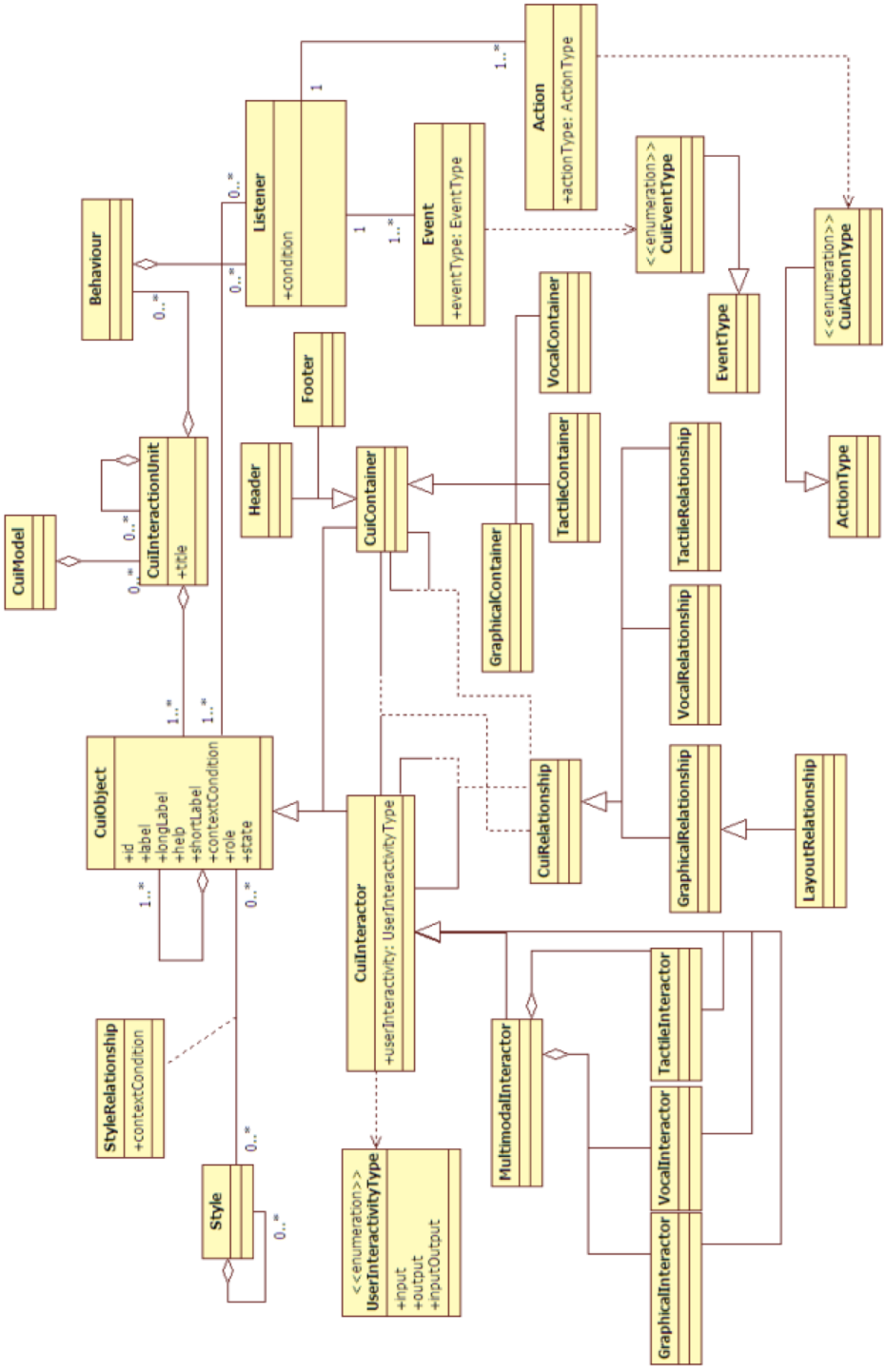


FIGURE 3.8 – Le méta-modèle du « Concrete User Interface » avec UsiXML (schéma issu de [1])

Cette variété de modèles rend l'approche complexe et augmente le temps et l'effort nécessaire pour la création des applications sous UsiXML.
La synthèse des approches basées sur le paradigme hiérarchique est résumée dans le tableau 3.3.

	Interaction multimodale	**Combinaisons**	**Interaction à base de capteurs mobiles**	**Productivité des modèles**
CTT / DynaMo-AID	En entrée et en sortie	CARE	Non	Génération de code pour trois plateformes différentes (présente une certaine hétérogénéité)
UsiXML	En entrée et en sortie	CARE	Non	Génération de code (web uniquement)

TABLE 3.3 – Synthèse des approches basées sur le paradigme hiérarchique

3.1.4 Paradigme de composants

Le quatrième paradigme de modélisation est le paradigme de composants. Un composant est « une unité réutilisable de composition et de déploiement accessible à travers des interfaces » [104]. Les approches à base de composants fournissent des environnements pour l'assemblage et le déploiement des applications construites avec ces composants [15]. Les modèles d'assemblage modélisent les composants comme des boîtes noires connectées entre elles. Pour la modélisation des interactions multimodales, les approches à base de composants modélisent généralement les modalités d'interaction avec les composants : « dispositif » (le périphérique d'interaction) et « langage d'interaction ». Elles modélisent aussi les opérateurs de combinaison entre modalités avec des composants adaptés (complémentarité, redondance...).
Les modèles d'assemblage des composants de telles approches incluent généralement beaucoup de détails d'implémentation. De plus, puisque les composants logiciels sont généralement implémentés par et pour une technologie particulière, ils ne peuvent pas être utilisés pour des perspectives multiplateformes.

3.1.4.1 ICARE et ACICARE

La plateforme ICARE (Interaction Complementarity Assignation Redondance Equivalence) [15] permet aux concepteurs de manipuler graphiquement et d'assembler des composants logiciels afin de spécifier les interactions multimodales en entrée des systèmes interactifs. Les composants de base fournis par ICARE sont les « Dispositifs », les « Langages d'interaction » et les diverses formes de « Combinaisons de modalités ». Le composant « Dispositif » peut être par exemple le pilote qui contrôle le clavier, la souris ou le GPS. Il possède plusieurs attributs tels que le nom du périphérique physique, le domaine des valeurs en sortie, le temps de traitement moyen, etc. Le composant « Langage d'interaction » transforme les données en sortie du composant « Dispositif » en une forme compréhensible par le reste du système. Il possède aussi des attributs tels que le domaine des valeurs en entrée et en sortie, la fréquence des données, etc. Ainsi, une modalité d'interaction peut être modélisée par l'assemblage d'un composant « Dispositif » avec un composant « Langage d'interaction ». Les composants de combinaison entre modalités sont de type complémentarité, redondance, équivalence et redondance/équivalence. Ils possèdent aussi des attributs relatifs à la fenêtre temporelle (l'intervalle de temps pendant lequel les informations des modalités d'interaction peuvent être combinées), à l'ordre des modalités, etc.
La figure 3.9 montre un exemple d'utilisation des composants ICARE. Une fois la modélisation graphique terminée, la génération automatique du code de l'assemblage est déclenchée. Le code nécessaire comme interface avec le reste de l'application est aussi généré automatiquement. Ainsi, l'assemblage s'intègre facilement, distinguant la partie interactive des autres parties de l'application [15].

Tandis que ICARE ne permet pas la génération du code pour les téléphones mobiles, une autre implémentation a été faite afin d'explorer l'interaction multimodale sur mobiles. Cette implémentation est appelée ACICARE. C'est un outil qui couple ICARE pour mobile avec ACIDU [30] qui permet de capturer les données d'usage sur téléphones.
Le problème de ACICARE est qu'il se base sur le même modèle conceptuel que ICARE. Il fournit les mêmes types de composants « Dispositifs » / « Langages d'interaction » pour modéliser les modalités d'interaction sur mobile. Ainsi, il ne permet pas de modéliser les différents types d'interaction à base de capteurs (**les composants sont souvent basés sur des données de bas niveau** (yaw, pitch et roll pour l'orientation avec magnétomètre dans la figure 3.9) ce qui ne permet pas de définir les types d'interaction de haut niveau). Il ne propose donc qu'un nombre très limité de modalités d'interaction (vocal et clavier téléphonique).

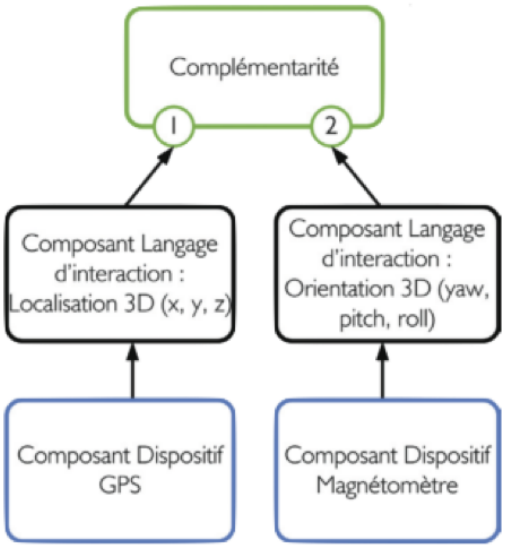


FIGURE 3.9 – Exemple d'utilisation des composants connectés dans ICARE : une complémentarité entre l'orientation et la localisation (schéma issu de [15])

De plus, ACICARE ne possède pas d'éditeur graphique. L'assemblage des composants et la génération de code (testé sur **SPV c500** [1]) doivent être faits à la main.

3.1.4.2 OpenInterface

OpenInterface [2] est une plateforme à base de composants pour la création **d'interfaces multimodales en entrée**. Il utilise des composants génériques (un composant est générique s'il ne s'applique pas à un dispositif ou une application particulière) avec un haut niveau de réutilisabilité (basés sur les composants d'ICARE) et des composants adaptés (des composants créés de manière ad-hoc pour des applications particulières). Selon le même modèle conceptuel que ICARE, OpenInterface **modélise les modalités d'interaction** avec les couples <Dispositif, Langage d'interaction> (même problème de niveau d'abstraction que ACICARE). Il propose aussi **les opérateurs de combinaisons selon les propriétés CARE**. Pour la modélisation et l'assemblage des composants, le framework offre deux environnements graphiques : OIDE et SKEMMI. Ils facilitent l'assemblage des composants basés sur des **technologies différentes** (Java, C++, .Net, Python, Matlab...). Ces composants permettent l'interaction avec plusieurs techniques telles que la reconnaissance vocale et l'ana-

1. `http://en.wikipedia.org/wiki/HTC_Typhoon` (dernière consultation le 24/06/2014)
2. Projet européen : `http://www.oi-project.org/` (dernière consultation le 24/06/2014)

lyse des images (ARToolkit, TrackIR) et avec plusieurs périphériques tels que Wiimote, iPhone, GPS, souris et clavier [99].
Après la modélisation des interactions, **le modèle est transformé en code**. Les composants de combinaison entre modalités sont transformés en **mécanismes de fusion** de données tandis que les autres composants permettent la génération du code des modalités d'interaction.

3.1.4.3 WWHT / ELOQUENCE

Contrairement aux approches précédentes dédiées à l'interaction en entrée, Rousseau et al. [95] proposent une approche pour la simulation et l'exécution (**productivité limitée**) d'interfaces intelligentes **multimodales en sortie**. WWHT est le modèle conceptuel de l'approche. Il se base sur quatre concepts fondamentaux : « What » (Quelle information présenter ?), « Which » (Quelle présentation multimodale choisir ?), « How » (Comment instancier cette présentation ?), et « Then » (Comment faire évoluer cette présentation ?). Il traite les problèmes de conception et d'évolution de la présentation multimodale en adéquation avec le contexte d'interaction [95]. Sur la base de ce modèle, la plateforme logicielle ELOQUENCE permet la conception des systèmes multimodaux en sortie. Cette plateforme propose les composants de modélisation des interactions multimodales en sortie uniquement. Cependant, ces composants ne permettent pas la génération du code associé. Ils permettent juste la simulation de l'interface en sortie.

Pour modéliser, le concepteur décompose manuellement les informations à présenter aux utilisateurs. Puis, il choisit les modalités d'interaction à utiliser en fonction des informations à présenter et au contexte de la présentation. Chaque modalité est attachée à un mode utilisateur et un dispositif physique (médium) (figure 3.10). Les liaisons entre composants peuvent être primaires ou secondaires.
Un ensemble de règles d'adaptation est également fourni afin de bien modéliser les interactions en sortie ainsi que les **combinaisons entre elles (redondance ou complémentarité)** en fonction du contexte.

L'outil de simulation [95] contient quatre services. Le premier service permet la configuration des paramètres de simulation à travers un fichier XML. Le second simule le contrôleur de dialogue qui déclenche la présentation intelligente en sortie. Le troisième service simule le serveur de contexte qui permet d'éditer le contexte de la présentation. Finalement, les résultats de la simulation sont présentés au concepteur sous forme textuelle [95].

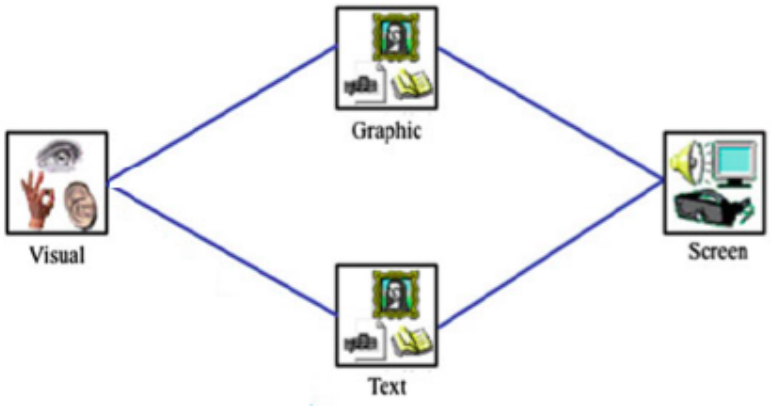


FIGURE 3.10 – Exemple de modèles de composants d'interaction dans ELOQUENCE

3.1.4.4 DynaMo et i*Chameleon

DynaMo (Dynamic Multimodality) [8] est défini dans le cadre du projet « MEDICAL » [3]. Ce projet cherche à réaliser un intergiciel pour aider les personnes âgées dans des maisons intelligentes à changer le mode d'interaction selon le contexte (avec la télévision, les jeux...). DynaMo fait partie des approches qui permettent la multimodalité pervasive. C'est une plateforme pour l'adaptation au moment de l'exécution (run-time) des interfaces **multimodales en entrée**. Il joue le rôle d'un médiateur entre l'application multimodale et ses périphériques d'interaction.
Comme SMUIML/HephaisTK, DynaMo propose **un niveau d'abstraction adapté** pour modéliser les interactions à base de capteurs. Il définit l'interaction à deux niveaux : 1) le niveau réception (concepts « Dispositif » dans le méta-modèle (figure 3.11)) qui permet de définir le capteur responsable ainsi que la modalité de l'interaction ; 2) le niveau de détection du type d'interaction issu de cette modalité (concepts « Port »).

DynaMo se base sur deux modèles pour faire l'adaptation. Les deux modèles sont définis pendant la conception : un modèle d'interaction qui décrit les interactions avec l'application (spécifié par le concepteur) et un modèle de proxy qui décrit les proxys entre l'application et la plateforme (spécifiés par le développeur). À partir de ces deux modèles, DynaMo génère la « Mediation chain » qui correspond au processus des interactions multimodales.
La plateforme est composée de trois services :

– Plateforme OSGI (contenant les modules iPOJO et ROSE) : cette plateforme permet de surveiller l'environnement d'interaction.

3. `http://medical.imag.fr/` (dernière consultation le 24/06/2014)

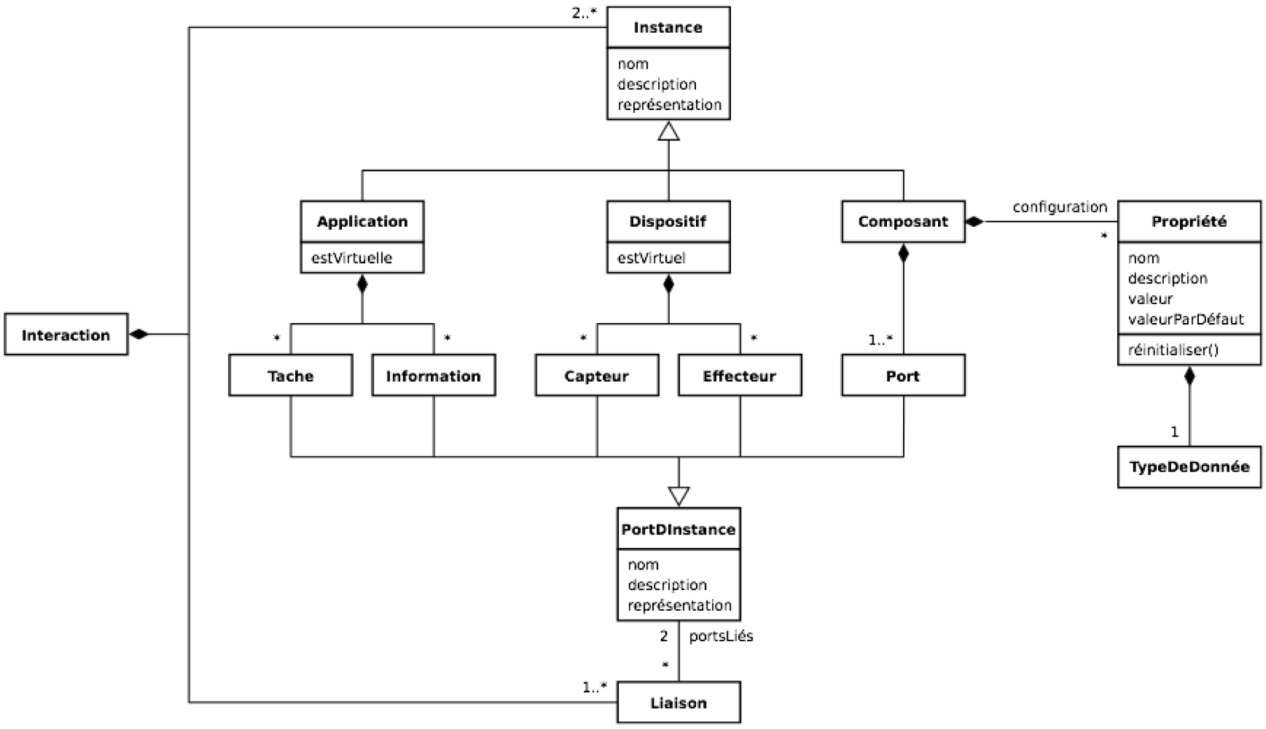


FIGURE 3.11 – Le méta-modèle de DynaMo(schéma issu de [7])

– Médiateur (Service mediation - CILIA) : ce niveau permet l'exécution du processus d'interaction et l'échange de données entre l'application et les périphériques.
– Gestionnaire autonomique : ce gestionnaire permet d'adapter les interactions multimodales selon l'environnement.

Le gestionnaire d'interaction utilise deux types d'information pour pouvoir prendre la décision d'adaptation. Le premier type concerne les informations sémantiques (spécifiées par un expert d'interaction) et le deuxième concerne les informations contextuelles. L'utilisateur final peut aussi participer à la décision prise en spécifiant, par exemple, les périphériques qui peuvent être utilisés.

Ainsi, pour interagir d'une façon multimodale avec une application comme VLC [4], par exemple, il suffit de définir son modèle de proxy et le modèle d'interaction qui décrit les interactions possibles. L'utilisateur peut aussi ajouter des nouvelles fonctionnalités telles que la fonctionnalité « mute » pour VLC. DynaMo surveille l'environnement d'interaction, exécute le processus d'interaction généré à partir des deux modèles et en cas de changement de contexte il décide quel périphérique utiliser.

4. `http://www.videolan.org/vlc/` (dernière consultation le 24/06/2014)

L'originalité de DynaMo est qu'il permet d'utiliser des modèles partiels. Le concepteur peut en effet fournir des modèles partiels pendant la phase de conception qui seront complétés par le gestionnaire autonomique pendant la phase d'exécution. Pour créer le modèle d'interaction, DynaMo fournit un éditeur graphique afin de simplifier l'assemblage des composants d'interaction. Cependant, pour définir le modèle proxy, le concepteur doit avoir des connaissances en programmation, car le modèle ne peut être réalisé qu'en programmation (Java et XML).
Concernant les combinaisons de modalités, **DynaMo assure la complémentarité et l'équivalence entre les interactions en entrée**.

De même, i*Chameleon [105] [67] permet l'adaptation au contexte en changeant les périphériques d'interaction en entrée. Il se focalise surtout sur les jeux pour ordinateur de bureau et permet aux utilisateurs finaux de les personnaliser selon leurs besoins. Il se base sur le principe de « Binding », c'est-à-dire qu'au lieu d'utiliser la souris, l'utilisateur peut choisir un autre périphérique comme la Wii par exemple pour interagir avec l'application. Cependant, les interactions avec la Wii ne feront que simuler des interactions à base de la souris (aller à gauche avec la Wii simule un clic sur le bouton gauche de la souris et ainsi de suite).
Comme DynaMo, i*Chameleon permet la modification des périphériques **d'interaction en entrée** au moment de l'exécution (clavier, souris, Wiimote, Smartphone, microphone, etc.). Les relations entre modalités **ne sont pas décrites**.
i*Chameleon (figure 3.12) facilite la tâche de modélisation pour qu'elle soit accessible par les utilisateurs finaux. La notation graphique utilisée est très simplifiée pour cette perspective (des icônes claires et liaisons automatiquement créées). Comme i*Chameleon, on trouve plusieurs outils qui permettent aux utilisateurs finaux d'adapter eux-mêmes les modalités d'interaction selon le contexte, mais sous PC généralement (Crossweaver [101], SILK [58], DENIM [5], etc.). Cependant, la définition des langages de modélisation et des notations visuelles pour les utilisateurs finaux reste toujours un grand défi [6].

5. http://repository.cmu.edu/cgi/viewcontent.cgi?article=1066&context=hcii (dernière consultation le 24/06/2014)

6. http://hal.univ-lille3.fr/docs/00/50/46/68/PDF/combemale10a.pdf (dernière consultation le 24/06/2014)

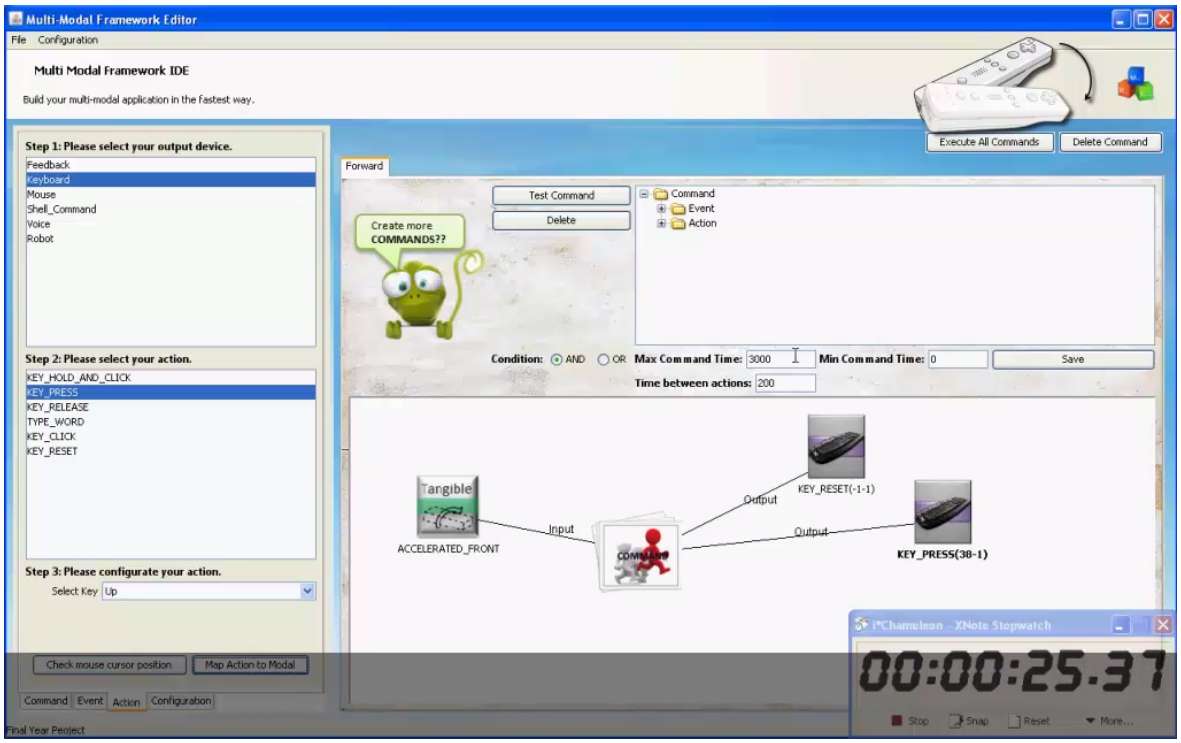


FIGURE 3.12 – L'éditeur graphique de i*Chameleon (figure issue de [105])

3.1.4.5 Squidy

Squidy [56] est une bibliothèque destinée à la conception, le prototypage rapide et la gestion des interfaces multimodales post-WIMP (**productivité limitée**). Elle se rajoute dans les applications existantes pour gérer les différents périphériques d'interaction hétérogènes en les reliant aux clavier et souris standard (principe de « Binding » comme pour i*Chameleon). Pour modéliser les interactions multimodales, Squidy présente un langage visuel simple avec une collection de composants représentant des **périphériques physiques** prêts à utiliser. Il se base sur le concept de modélisation « Pipe-and-filter ». Les tubes et les filtres servent d'intermédiaires entre les sources et les puits. Les sources sont les éléments qui fournissent des données alors que les puits sont les éléments qui les consomment. Les tubes définissent les chemins des données entre les noeuds, et les filtres les traitent en fonction des périphériques utilisés.

L'utilisateur sélectionne alors les périphériques d'entrées comme des sources (pointeur laser, microphone, Wiimote...), il les connecte à des filtres afin de traiter les données en entrée (reconnaissance des gestes, reconnaissance des paroles...), puis il définit le chemin des données vers l'application (clavier ou souris).

Le principe de « Semantic zooming ». Squidy permet de faire des zooms sur les noeuds afin de voir leur code. Le but de cette fonctionnalité est d'utiliser un seul environnement de travail pour modéliser et programmer. L'utilisateur peut zoomer pour voir le code et le modifier ; en dé- zoomant le code sera exécuté à la volée.
Squidy est caractérisé par le « Multithreading ». Chaque noeud génère son thread et gère ses propres données. Cela permet de réduire le temps de traitement qui aura un effet positif sur les performances d'interaction. La réutilisation des composants des périphériques réduit aussi le temps de modélisation. De plus, la visualisation des flots des données permet de détecter les erreurs et de les corriger facilement.
Squidy ne traite pas les combinaisons entre modalités. Cependant, il est facilement extensible (ajout de nouveaux périphériques). Il est aussi **limité aux interactions en entrée et ne considère pas les interactions en sortie de l'application**.

3.1.4.6 ICOM / ICON

ICON (Input CONfigurator) [80] est un environnement de prototypage rapide des applications Post-WIMP multimodales (**productivité limitée**). Son but est de **supporter l'adaptation en entrée** en facilitant l'ajout, la suppression et la modification des périphériques d'interaction d'un système interactif **sous PC (les interactions à base de capteurs mobiles ne sont pas supportées)**. L'outil propose une bibliothèque de dispositifs d'interaction en entrée (dispositifs systèmes, dispositifs utilitaires et dispositifs d'application) et un moteur d'exécution. Les dispositifs sont représentés sous forme de composants avec des ports en entrée et en sortie. Ainsi, pour réaliser un prototype rapide d'une interaction multimodale, le concepteur peut connecter les ports des composants de cette interaction à partir de l'éditeur graphique de ICON, c'est-à-dire créer un modèle ICOM (Input Configuration Model), puis exécuter cette interaction rapidement avec le moteur d'exécution.
Comme ICO / PetShop, « les combinaisons entre modalités d'interaction en entrée ne sont pas explicitement modélisées ». Des dispositifs « utilitaires » (non générique [15]) sont utilisés pour gérer les différents types de combinaison dans le code.
La notation graphique de ICOM est basée sur la description de flux de données avec des modules (dispositifs) et leurs relations.

Le tableau 3.4 résume la synthèse des approches basées sur le paradigme de composants.

	Interaction multimodale	**Combinaisons**	**Interaction à base de capteurs mobiles**	**Productivité des modèles**
ICARE et ACICARE	En entrée uniquement	CARE	Non	Génération de code (pas d'hétérogénéité)
OpenInterface	En entrée uniquement	CARE	Non	Génération de code (pas d'hétérogénéité)
WWHT / ELOQUENCE	En sortie uniquement	CARE (mais pas d'équivalence)	Non	Simulation
DynaMo	En entrée uniquement	CARE (mais pas redondance)	Oui (deux niveaux de modélisation)	Interprétation (une plateforme particulière)
i*Chameleon	En entrée uniquement	/	Oui (téléphone mobile en tant que dispositif d'entrée)	Interprétation (binding)
Squidy	En entrée uniquement	/	Oui (téléphone mobile en tant que dispositif d'entrée)	Interprétation (binding)
ICOM /ICON	En entrée uniquement	CARE	Non	Exécution et simulation

TABLE 3.4 – Synthèse des approches basées sur le paradigme de composants

3.2 Outils logiciels pour la création des applications mobiles

Récemment, plusieurs travaux traitent du développement des applications mobiles à base de modèles. Ils facilitent la création des applications natives, web ou hybrides pour les différentes plateformes mobiles (Android, iOS, Windows Phone, etc.) en utilisant des environnements graphiques intuitifs.

3.2.1 XDK de Intel

XDK de Intel [7] est un outil de création d'applications **web et hybrides mobiles** (comme PhoneGap [8]). Il a été créé par Intel en 2013. Il permet de modéliser les interfaces graphiquement avec un environnement intuitif [9] et de **les générer automatiquement (HTML5, CSS3 et JavaScript)**. En sortie, il permet la modélisation et la génération des éléments d'affichage, de décoration, des animations, etc. En entrée, il permet la modélisation et génération des interactions tactiles, Qrcode et GPS. Il utilise aussi l'accéléromètre et le capteur d'orientation. Cependant, pour créer des interactions exploitant ces capteurs, l'outil ne donne pas la possibilité de les modéliser et générer comme le reste des interactions. Il propose une API [10] pour faciliter leurs implémentations (coder les interactions en JavaScript). Le but de l'API appelée « Bridge API » est de fournir les fonctionnalités natives à une application hybride. Elle simplifie l'accès aux couches matérielles basses des dispositifs mobiles afin de définir des interactions exploitant les capteurs (accéléromètre, capteur d'orientation, caméra, écran tactile (« multi-touch »)...). L'API permet aussi la programmation de plusieurs fonctionnalités qui ne concernent pas l'interaction homme-machine. Elle facilite par exemple la programmation des fonctionnalités comme l'envoi de mail et de SMS, l'authentification, etc.

Le problème de cette solution (utiliser une API pour le traitement des interactions à base des nouveaux capteurs) est **qu'elle est spécifique pour une plateforme particulière** (ici le web - html5/css3). Ainsi, si on souhaite utiliser l'outil pour d'autres plateformes mobiles, il faut créer d'autres APIs (selon ces plateformes, leurs versions différentes...), ce qui augmente la quantité de code à gérer, la documentation à fournir, etc.

XDK de Intel propose aussi un émulateur pour pouvoir tester les applications générées avant de les essayer sur un dispositif mobile. Cela permet d'avoir un aperçu de l'interface graphique de l'application et de tester les interactions en entrée telles que l'interaction avec l'écran tactile, l'accéléromètre, le GPS et le capteur d'orientation [11]. Il simule (figure 3.13) la webview des différents périphériques mobiles (iPad, iPhone 4S, iPhone 5, Nexus 4, Nexus 7, Galaxy S, HTC Droid Incredible, Motorola Droid 2, Nokia Lumina 920...) (une bonne gestion de l'hétérogénéité des dispositifs mobiles).

7. `http://www.i-programmer.info/news/87/6756.html` (dernière consultation le 24/06/2014)
8. `http://phonegap.com/` (dernière consultation le 24/06/2014)
9. `http://www.sharewareconnection.com/intel-xdk.htm` (dernière consultation le 24/06/2014)
10. `https://software.intel.com/en-us/node/492826` (dernière consultation le 24/06/2014)
11. `http://lc.cx/PQd` (dernière consultation le 24/06/2014)

Après le test des applications, le XDK Intel permet leur déploiement directement à partir de l'outil. Il commence par synchroniser les applications générées (ou codées) avec les serveurs cloud d'Intel. Puis à partir de l'application native App Preview [12] l'utilisateur peut lancer la synchronisation entre les serveurs cloud et les fichiers locaux de l'application. Ainsi, il peut accéder aux différentes applications (pages web) depuis son smartphone.

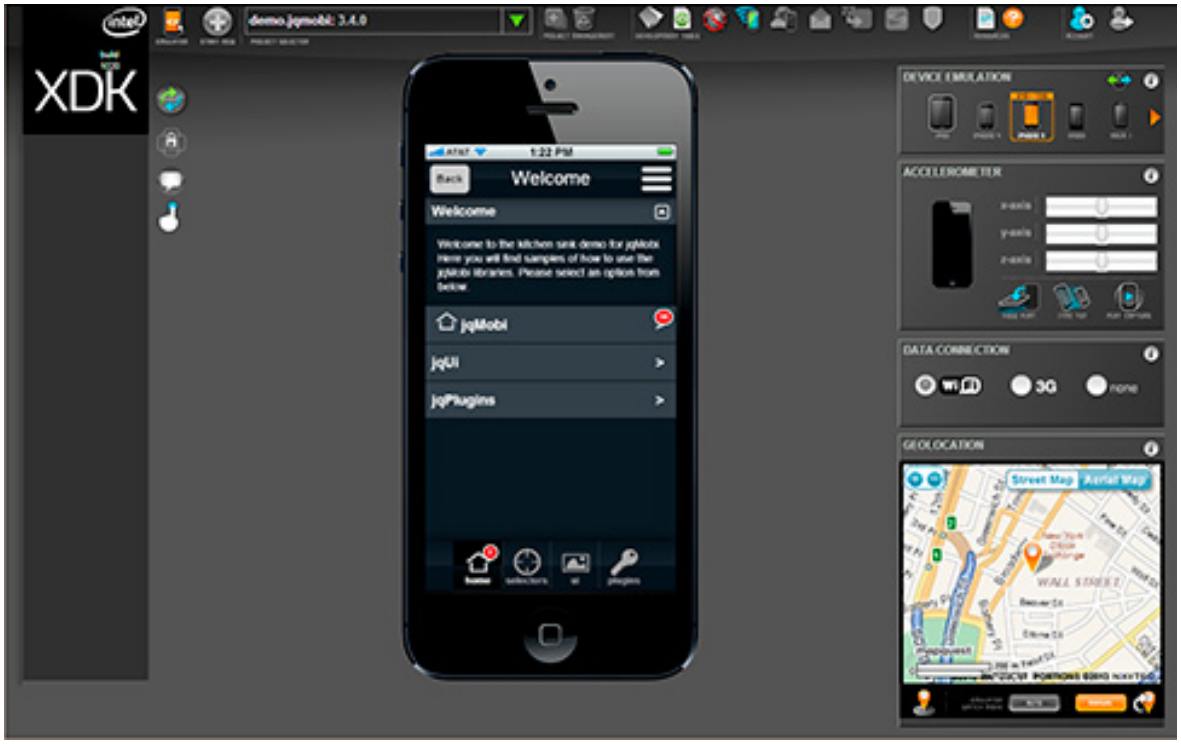


FIGURE 3.13 – L'environnement de modélisation/simulation du XDK de Intel (figure issue de [2])

La modélisation des relations entre modalités n'est pas possible avec le XDK Intel. Cependant, leur programmation manuelle est plus simple en utilisant « Bridge API » qui facilite l'implémentation des interactions à base de capteurs.

3.2.2 App Inventor

App Inventor [13] est un environnement basé sur le web pour le développement facile et rapide d'applications mobiles. Il permet de modéliser graphiquement (programmation visuelle) puis générer, tester et déployer des applications pour **Android uniquement**. Le but

12. `https://play.google.com/store/apps/details?id=com.intel.html5tools.apppreview` (dernière consultation le 24/06/2014)
13. `http://appinventor.mit.edu/explore/` (dernière consultation le 24/06/2014)

de App Inventor est de faciliter la programmation des applications mobiles surtout pour les débutants. Il a été créé par Google en 2010 puis mis à jour par le MIT (Massachusetts Institute of Technology).
La modélisation sous App Inventor ressemble à celle sous Scratch [14]. Il est basé sur OpenBlocks [91]. Les séquences d'interaction sont représentées par des blocs graphiques (vue « Blocks ») ce qui donne des modèles similaires aux diagrammes de Nassi-Shneiderman (figure 3.14). Ce type de modèle réduit le risque d'erreur (les blocs s'encastrent de façon non ambigüe) et guide bien les utilisateurs novices lors de la modélisation.
En plus de séquences d'interaction, les écrans des applications sont aussi modélisés graphiquement (vue « Designer ») avec un système de « drag and drop » des widgets proposés dans la palette (figure 3.15). Dans la vue « Designer », l'utilisateur peut aussi utiliser des capteurs tels que l'accéléromètre, le capteur d'orientation, GPS, NFC et le scanner de code-barres. Cependant, certains capteurs tels que le capteur de lumière et de proximité ne sont pas encore pris en compte. De plus, pour utiliser des interactions à base de capteurs comme l'accéléromètre par exemple, **l'utilisateur doit lui-même préciser les valeurs de x, y et z** de ces interactions (sauf pour quelques interactions prédéfinies comme le secouage « shake »).

FIGURE 3.14 – Une copie d'écran de la vue « Blocks » de App Inventor

14. http://scratch.mit.edu/ (dernière consultation le 24/06/2014)

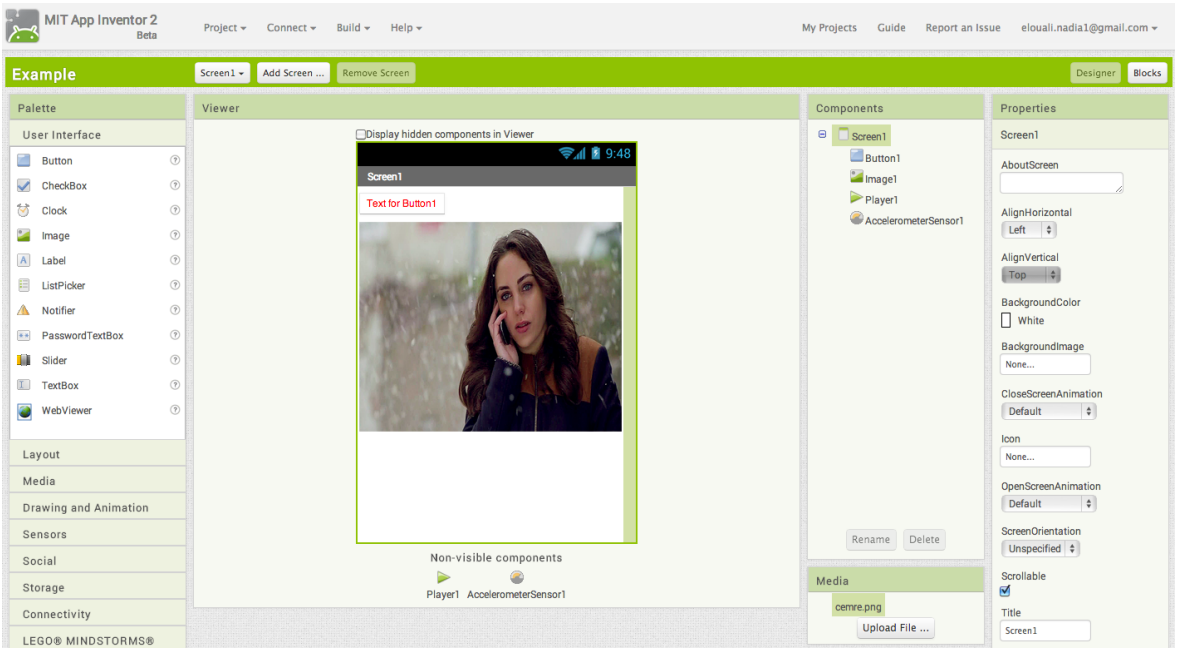


FIGURE 3.15 – Une copie d'écran de la vue « Designer » de App Inventor

L'éditeur graphique de App Inventor est très intuitif [15]**. Beaucoup d'efforts ont été faits sur son ergonomie**. Cependant, il ne permet pas de visualiser toute l'application modélisée si elle est volumineuse, ce qui ne permet pas d'avoir une vue générale. La séparation entre la séquence d'interaction et les écrans de l'application peut aussi être une difficulté pour le concepteur lors de la modélisation (le passage d'un onglet à un autre). Une autre limite de App Inventor est qu'il ne permet pas d'avoir le code source de l'application généré. Il génère directement l'APK (Application PacKage file) Android ce qui ne permet pas au créateur de l'application de visualiser le code (Java/XML) ou de le modifier.

La modélisation des relations entre modalités n'est pas prévue dans App Inventor. Les applications générées peuvent avoir une ou plusieurs modalités d'interaction en entrée et en sortie, mais sans combinaison entre elles.

3.2.3 iStuff Mobile

iStuff Mobile [16] est un outil open-source de prototypage rapide et fonctionnel d'interfaces mobiles. C'est le premier outil qui s'intéresse au prototypage des interactions à base de capteurs mobiles en informatique ubiquitaire [9]. Son but est de donner la possibilité aux

15. `http://lc.cx/WKH` (dernière consultation le 24/06/2014)
16. `http://hci.rwth-aachen.de/istuff/` (dernière consultation le 24/06/2014)

utilisateurs de tester les interfaces et les interactions à base de capteurs avant de passer à l'implémentation (**productivité limitée**).

Le projet iStuff a été réalisé en 2007. Comme il n'y avait pas encore de capteurs intégrés dans les téléphones mobiles, les participants au projet ont décidé d'ajouter des capteurs externes aux téléphones afin de détecter l'accélération, la température, la lumière, etc.

iStuff Mobile décompose l'interface à prototyper en deux parties : « Premier-plan » (Foreground) et « Arrière-plan » (Background). La partie « Foreground » est celle qui sera affichée sur le téléphone pour que l'utilisateur interagisse avec, alors que la partie « Background » est celle qui traite les interactions de l'utilisateur. La partie « Background » est distribuée sur un ou plusieurs ordinateurs (où iStuff est installé). Selon l'interaction, elle permet de déclencher les fonctionnalités suivantes : déclenchement/lecture de son, contrôle de vibration, lancement d'autres applications du téléphone, contrôle de rétro-éclairage, exécution d'applications en arrière-plan et contrôle de caméra. La communication avec le téléphone se fait en Bluetooth.

Pour prototyper une interface, le concepteur doit d'abord ajouter les capteurs externes au téléphone. Dans les tests du projet, les auteurs ont utilisé le capteur Smart-Its [45] qui contient un accéléromètre, un microphone, un capteur de lumière, de pression, de température et de tension (avec le système d'exploitation mobile « Symbian Series 60 », **une plateforme particulière**). Puis, le concepteur doit installer iStuff sur son ordinateur et connecter les deux en Bluetooth. Pour modéliser les interfaces (les parties « Foreground »), iStuff propose un environnement de programmation visuelle. Cet environnement est basé sur Quartz Composer [17] introduit dans le logiciel Xcode (MacOSX 10.4). C'est un langage de programmation visuelle qui consiste à connecter graphiquement un ensemble de composants pour réaliser un traitement particulier. iStuff étend l'environnement proposé par Apple en ajoutant un ensemble de composants pour permettre la programmation visuelle des interfaces mobiles (des composants matériels pour la gestion des capteurs par exemple et des composants logiciels de contrôle, de condition, etc.).

Beaucoup de prototypes d'applications ont été réalisés avec iStuff Mobile [9]. L'évaluation avec des concepteurs/développeurs montre que l'utilisation de la programmation visuelle réduit significativement le temps de prototypage et encourage les développeurs à faire plusieurs itérations (essayer plusieurs interactions différentes). Toutefois, pour la détection des interactions à base de capteurs, l'utilisateur doit quand même programmer **à bas niveau** (JavaScript).

17. `http://en.wikipedia.org/wiki/Quartz_Composer` (dernière consultation le 24/06/2014)

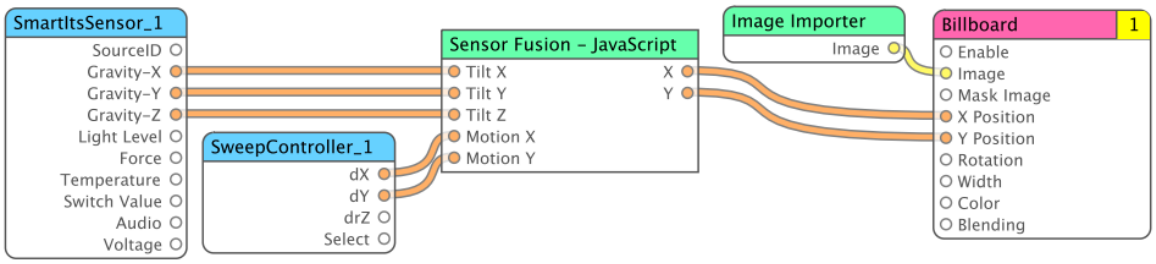


FIGURE 3.16 – Un exemple de la programmation visuelle proposé dans iStuff (figure issue de [9])

Il doit d'abord identifier les valeurs qui correspondent à l'interaction (les valeurs de x, y et z pour l'accéléromètre par exemple), puis écrire un script qui déclenche un traitement dès que les valeurs sont détectées.

iStuff Mobile ne traite pas les interactions multimodales à base de capteurs. Cependant, il permet de faire quelques fusions entre les données issues des capteurs en entrée afin d'améliorer la précision des interactions détectées (la redondance). La figure 3.16 montre un exemple de fusion entre l'accéléromètre et la caméra pour améliorer la détection des mouvements.

3.2.4 Amarino

Amarino [18] [51] est une boîte à outils qui facilite le développement d'applications utilisant les smartphones comme moyen d'interaction dans des environnements intelligents. En particulier, Amarino permet une communication transparente entre les smartphones Android et les microcontrôleurs basés sur Arduino (**mono-plateforme**). Il facilite la gestion des interactions à base de capteurs entre les différentes applications mobiles Android et les outils intelligents contenant des microcontrôleurs basés sur Arduino (une lampe, ventilateur, etc.). Le choix d'Android et d'Arduino est basé sur le fait qu'ils sont tous les deux open-source et représentent des plateformes suffisamment matures [51].

La boîte à outils Amarino contient deux composants : une application Android « Amarino » [19] et une bibliothèque Arduino « MeetAndroid » (figure 3.17). Ces deux composants permettent (1) l'accès aux capteurs Android (capteur d'orientation, accéléromètre, messages, boussole) et l'envoi d'évènements de ces capteurs vers l'Arduino connecté en Bluetooth (puisque Bluetooth est un protocole de communication peer-to-peer, Amarino communique

18. `http://www.amarino-toolkit.net/` (dernière consultation le 24/06/2014)
19. `https://play.google.com/store/apps/details?id=at.abraxas.amarino` (dernière consultation le 24/06/2014)

avec un Arduino à la fois) ; (2) le développement rapide d'applications Android qui envoient et reçoivent des données aux Arduino.

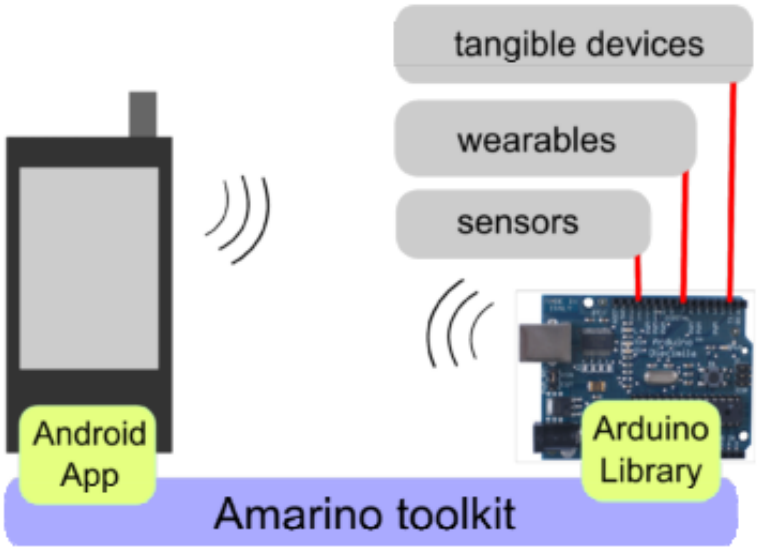


FIGURE 3.17 – Les composants d'Amarino (schéma issu de [51])

L'application Amarino possède une interface graphique qui permet à l'utilisateur de choisir l'Arduino à utiliser, d'envoyer des évènements vers cet outil intelligent, de surveiller le Bluetooth et le transfert des données, de sélectionner les données à envoyer et de personnaliser les évènements. L'application permet aussi de recevoir des données puis de les transmettre vers une autre application Android. La bibliothèque MeetAndroid permet au développeur Arduino de développer des fonctionnalités à base d'évènements Android. Si la communication est bidirectionnelle, la bibliothèque permet aussi de développer des fonctions pour envoyer des évènements vers le téléphone. Les fonctions de la bibliothèque facilitent la tâche de développement d'Arduino (utiliser des fonctions de haut-niveau et ne pas rentrer dans les détails d'implémentation) ainsi que le transfert des données depuis ou vers des smartphones Android. Dans les autres applications Android, Amarino peut être utilisé en backgroud pour transférer les données des capteurs vers l'Arduino connecté.
Comme Amarino, il existe d'autres plateformes pour simplifier le développement des interactions entre smartphones et environnement intelligents. Open Data Kit Sensors (ODK) [18], IOIO [20], WeWrite [109] sont des exemples de plateformes qui visent à faciliter le développement d'interfaces entre smartphones et capteurs externes. Toutefois, ces outils sont souvent définis pour une plateforme cible (Android généralement). De plus, leurs utilisateurs doivent

20. `https://www.sparkfun.com/products/retired/10748` (dernière consultation le 24/06/2014)

avoir des connaissances importantes en programmation.

3.2.5 Les kits de développement (SDKs) mobiles

On ne peut pas parler des outils logiciels de développement des applications mobiles sans parler des SDKs (Software Development Kits). Ce sont des outils qui permettent et simplifient le développement des applications mobiles pour les différents systèmes (Android, iOS, Symbian, Bada, Windows Phone 7, Windows Phone 8...). Ils utilisent parfois des modèles pour spécifier les interfaces graphiques des applications ou même la navigation d'un écran à un autre. Ci-dessous nous présentons les SDKs des deux plateformes mobiles les plus utilisées et qui ont des pourcentages importants de parts du marché ces deux dernières années : Android et iPhone.

3.2.5.1 SDK Android

Google fournit gratuitement un SDK Android pour les trois systèmes d'exploitation majeurs (Windows, Mac OS X et Linux). Ce kit de développement permet l'accès aux exemples fournis, à la documentation, à un émulateur pour tester les applications et surtout à l'API (Application Programming Interface).

Le développement sous Android s'effectue majoritairement en Java, ce qui augmente la popularité de la plateforme (en plus du fait qu'il est open-source). Pour pouvoir développer, le plugin Android Development Tool (ADT) peut être installé sous Eclipse pour intégrer les différentes fonctionnalités du SDK. Android Studio peut aussi être utilisé indépendamment d'Eclipse ; c'est un environnement de développement d'applications Android qui est livré avec sa propre version SDK.
Après la compilation, le SDK convertit le bytecode Java en bytecode Dalvik. L'application installable sur le téléphone est empaquetée dans un fichier APK, qui inclut le bytecode de l'application, son fichier de description « AndroidManifest », ses ressources et les signatures numériques [21].
Le plugin ADT et Android Studio donnent la possibilité de définir les interfaces graphiques des applications avec des modèles. Ils offrent deux vues différentes : une vue qui montre le code XML de l'interface et une autre vue qui donne son modèle graphique. Le code XML des interfaces modélisées est généré automatiquement.

21. `https://www.sstic.org/media/SSTIC2011/SSTIC-actes/Securite_Android/SSTIC2011-Article-Securite_Android-ruff.pdf` (dernière consultation le 24/06/2014)

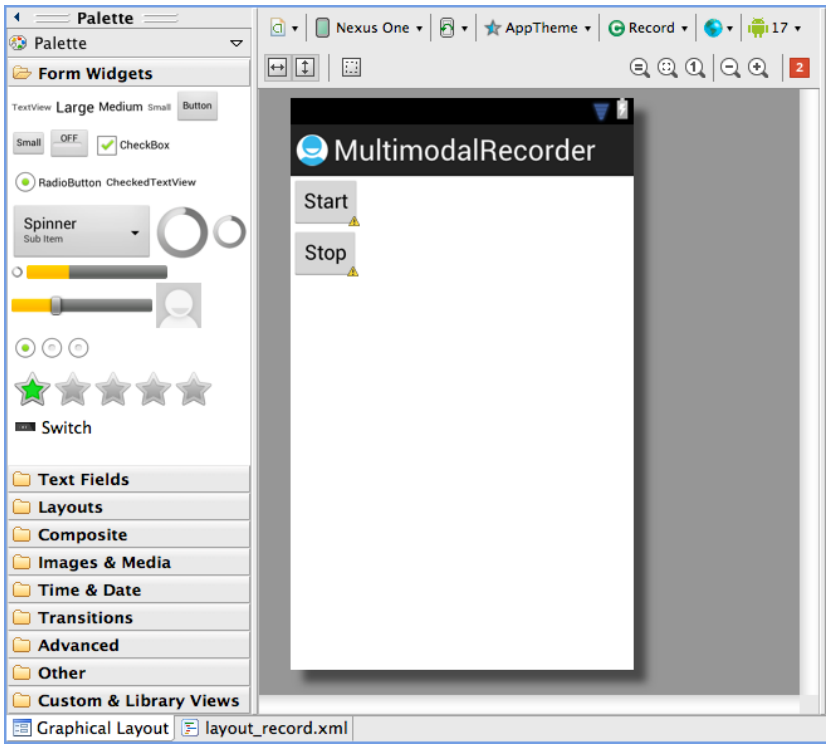


FIGURE 3.18 – Une copie d'écran de la vue graphique fournie par l'ADT Android

Aussi, en cas de modification du code, le modèle est modifié automatiquement. Pour modéliser, l'outil propose un système WYSIWYG (What You See Is What You Get). Les éléments graphiques (widgets, médias, images, layouts, etc.) peuvent être choisis à partir d'une palette, puis glissés/déposés sur un modèle représentant l'écran du smartphone (figure 3.18). L'ADT et Android Studio donnent aussi la possibilité de visualiser des interfaces sur des écrans de différentes résolutions. Les éléments mis sur le modèle peuvent être déplacés et redimensionnés. De plus, un clic droit sur un élément du modèle invoque un menu par lequel une grande variété de propriétés de l'élément sélectionné peut être modifiées.

Pour l'implémentation des interactions à base de capteurs, le SDK Android fournit des fonctions qui écoutent les capteurs et récupèrent leurs valeurs (x, y, z pour l'accéléromètre, distance pour capteur de proximité, etc.). À partir de ces valeurs, le développeur peut définir des interactions en entrée.

Cependant, **la programmation de ces interactions avec le SDK est souvent considérée comme difficile** (des données de bas niveau, différents capteurs, différentes façons d'utilisation et de configuration, etc.) [22].

22. `http://www.fiercedeveloper.com/story/sensor-based-apps-offer-\lots-potential-are-hindered-fragmentation/2012-04-20` (dernière consultation le 24/06/2014)

3.2.5.2 SDK iOS

Apple fournit un SDK complet pour le développement des applications mobiles sous iPhone. Ce SDK contient un ensemble de logiciels et d'APIs nécessaires pour la création d'applications, la gestion de leurs interfaces, la gestion des capteurs, la lecture des fichiers multimédias, etc. Le développement s'effectue avec le langage Objective C, l'IDE (Integrated Development Environment) XCode et l'interface Builder :

FIGURE 3.19 – Une copie d'écran de la vue graphique fournie par Interface Builder et l'émulateur iPhone

XCode. C'est l'IDE officiel et unique de développement sur iPhone. Il propose un éditeur de code, un compilateur GCC (GNU Compiler Collection) ainsi que d'autres fonctionnalités.

Interface Builder. C'est un constructeur visuel d'interfaces graphiques pour les applications sous iPhone. Comme l'ADT d'Android, Interface Builder donne la possibilité de modéliser graphiquement les interfaces (figure 3.19). Le code correspondant à ces interfaces est généré automatiquement. L'utilisateur peut sélectionner les éléments graphiques à partir d'une bibliothèque et les placer dans la partie « View » qui correspond à la vue affichée sur l'écran. Il peut aussi modifier leurs propriétés (la taille, la couleur, l'image de fond, etc.) à partir d'un inspecteur. Les autres fonctionnalités des applications (utilisation des capteurs, gestion des données, etc.) **ne peuvent pas être modélisées graphiquement**.

À partir de Mac OS X 10.4, Quartz Composer a été fourni avec XCode comme langage de programmation visuel. Il est spécialisé dans le traitement des éléments graphiques de l'interface, notamment les animations, les images et les vidéos. Il consiste à connecter des blocs de traitement entre eux afin de créer des économiseurs d'écran, créer des animations graphiques puissantes, réaliser des effets vidéo, etc. Cependant, même si Quartz Composer est fourni avec le SDK de l'iPhone, il n'existe actuellement aucun moyen de l'exécuter sur les appareils iOS.
Tous les éléments graphiques ainsi que les interactions (tactiles uniquement) peuvent être testés sur l'émulateur fourni avec le SDK iOS.

Le tableau 3.5 résume la synthèse des outils logiciels pour la création des applications mobiles.

3.3 Synthèse

Nous avons décomposé la revue des travaux existants en deux parties : les approches qui utilisent les modèles pour le développement d'applications multimodales et les outils logiciels qui les utilisent pour le développement d'applications mobiles. Puis nous les avons analysées selon les critères suivants : le support de la multimodalité, le support des capteurs mobiles, la productivité des modèles et le traitement de l'hétérogénéité sur mobile.

Nous constatons les points suivants :

- Concernant les approches à base de modèles pour faciliter le développement d'interfaces multimodales :
  - Elles abordent rarement simultanément la multimodalité en entrée et en sortie. Elles traitent surtout les interfaces multimodales en entrée (10 sur 14 des approches étudiées).
  - Les quelques approches qui abordent la multimodalité en entrée et en sortie les modélisent séparément (avec des langages différents), alors qu'on cherche à donner aux concepteurs une vue générale de l'interaction en entrée et en sortie dans un même modèle (pour éviter la variété de modèles qui complexifie l'approche). De plus, ces approches ne permettent pas toujours la génération de code de ces interfaces à partir des modèles. Leurs buts sont limités à l'exécution ou la simulation des interfaces afin de s'assurer de leur bon fonctionnement avant l'implémentation.

	Applications mobiles	**Interactions multimodales**	**Combinaisons**	**Interactions à base de capteurs mobiles**	**Productivité des modèles**
XDk de Intel	Web et hybride	En entrée et en sortie	/	API pour les implémenter	Modélisation et génération des interfaces graphiques
App inventor	Native (Android uniquement)	En entrée et en sortie	/	Modélisation graphique (mais les valeurs des capteurs doivent être précisées)	Modélisation et génération des applications complètes
iStuff Mobile	Native	En entrée et en sortie	Redondance	Programmation visuelle (les valeurs des capteurs doivent être précisées)	Modélisation et prototypage des applications
Amarino	Native (Android uniquement)	En entrée	/	Programmation classique (téléphone mobile en tant que dispositif d'entrée)	Pas de modèles
SDKs	Native	En entrée et en sortie	CARE	Programmation classique	Modélisation et génération des interfaces graphiques uniquement

TABLE 3.5 – Synthèse des approches pour la création des applications mobiles

  - L'hétérogénéité mobile n'est pas prise en compte. La génération de code se fait pour une seule plateforme et seulement pour les anciennes plateformes (Dynamo-AID est la seule approche qui génère pour J2ME).
  - SMUIML et DynaMo proposent une solution intéressante pour modéliser les interactions à base de capteurs avec un niveau d'abstraction adéquat. Ils définissent l'interaction sur deux niveaux : le niveau réception (concepts « Recognizer » et « Dispositif ») qui permet de définir le capteur responsable ainsi que la modalité de l'interaction. Puis, le niveau de détection d'évènement issu de cette interaction (concepts « Trigger » et « Port »).
  - Ces approches proposent rarement un support pour faciliter la modélisation tels que le guidage de modélisation (model guidance), la vérification des modèles (model checking) qui représentent des points importants dans les approches à base de modèles (voir le chapitre 2 partie 2.2.2) [38].
- Concernant les outils logiciels à base de modèles pour faciliter le développement d'applications mobiles :
  - La modélisation graphique avec la plupart de ces outils se fait pour les interfaces graphiques uniquement. Il n'est donc pas possible de modéliser des interactions à base de capteurs. Il faut passer par la programmation à bas niveau (SDK) ou utiliser des APIs (XDK Intel).
  - Les outils qui permettent la modélisation des capteurs (AppInventor) ne fournissent pas l'interactions pré-testées ni optimisées. L'utilisateur doit lui-même fournir les coordonnées des capteurs pour détecter l'interaction.
  - La modélisation se fait écran par écran. Cela joue un rôle positif pour réduire la complexité des modèles. Cependant, cela ne donne pas une vue générale des interactions possibles.
  - Les modèles sont parfois utilisés pour le prototypage rapide des applications, mais, généralement, ils permettent la génération du code de l'application ou de l'interface. Quelquefois le code source généré n'est pas disponible (avec App Inventor par exemple, le concepteur n'obtient que l'APK de l'application générée).
  - Plusieurs outils ne permettent pas la modélisation et la génération des combinaisons de modalités d'interaction en entrée et/ou en sortie.
  - Ces outils proposent la génération pour une plateforme particulière ou bien pour le web (qui fonctionne pour plusieurs plateformes). Ainsi, il n'existe pas d'outil permettant la génération d'applications web, hybride et native pour les différentes plateformes.

Ces limites rendent plus visible le fossé qui existe actuellement entre l'évolution des dispositifs mobiles avec leurs capteurs intégrés et leurs utilisations dans des interfaces multimodales.

3.4 Conclusion

Nous concluons ce chapitre en présentant les besoins nécessaires pour faciliter la création d'applications mobiles multimodales riches de nouvelles modalités à base de capteurs et de combinaisons de modalités en entrée et/ou en sortie :

- Le premier besoin concerne la définition d'un langage qui permettrait la modélisation des interactions mobiles multimodales en entrée et en sortie, qui fournit le bon niveau d'abstraction pour la modélisation des interactions à base de capteurs et qui suit le paradigme de modélisation le plus adéquat pour la multimodalité sur mobile.
- Le besoin de modélisation consiste à permettre aux concepteurs d'avoir une vue globale des interactions en entrée et en sortie de leurs applications. La vue globale avec une notation visuelle claire facilitera la détection des conflits entre modalités et combinaisons de modalités. Elle permettra aussi de comprendre rapidement tout le scénario d'interaction sans être obligé de rentrer dans le code (même après un certain temps).
- Finalement, les approches de développement des applications mobiles multimodales doivent respecter les différents critères IDM afin de définir des ingénieries à base de modèles efficaces. Le langage de modélisation doit respecter la pertinence et la concision des concepts. L'environnement de modélisation/génération doit respecter les critères de guidage, de vérification et de réutilisation de modèles et surtout la génération de code. Cette dernière doit être multi-plateformes puisque les modèles sont indépendants des plateformes, et doit respecter aussi l'hétérogénéité mobile (au moins à gros grain).

Dans les chapitres suivants, nous présentons les solutions que nous proposons pour les différents besoins décrits ci-dessus. Nous commençons par la définition de notre proposition conceptuelle consistant en un langage de modélisation des interactions mobiles multimodales en entrée et en sortie. Puis, nous définissons notre environnement de modélisation et de génération de la multimodalité sur mobile. Au fil des définitions, nous montrerons comment nous avons amené notre chaîne de création et de transformation de modèles à respecter les différents critères IDM.

Chapitre 4

Proposition conceptuelle : langage de modélisation pour les interfaces mobiles multimodales

Sommaire

Le chapitre précédent a souligné le manque de prise en compte des interactions multimodales dans les outils logiciels de création d'applications mobiles. Il a souligné également le manque de considération des spécificités des environnements mobiles (nouveaux smartphones et capteurs) dans les approches de conception et développement des interfaces multimodales. Ainsi, nous avons mis en évidence les besoins nécessaires pour faciliter la création des applications mobiles multimodales riches de nouvelles modalités à base de capteurs et de combinaisons de modalités en entrée et/ou en sortie. Ce chapitre présente notre proposition conceptuelle pour répondre principalement au premier besoin identifié qui consiste à définir un langage pour modéliser les interactions mobiles multimodales en entrée et en sortie. Nous présentons notre langage de modélisation des interactions multimodales sur mobile : Mobile MultiModality Modeling Language (M4L). M4L possède deux principales caractéristiques : 1) il permet la modélisation de la multimodalité sur mobile à la fois en entrée et en sortie suivant le paradigme « état-transition » ; 2) il permet de modéliser les différentes interactions avec un niveau d'abstraction qui cache la complexité de leur implémentation et qui les présente avec précision afin d'augmenter la lisibilité des modèles. De plus, il fournit des concepts pertinents et concis afin de réduire la taille du langage, le rendre plus facile à apprendre et permettre la création de modèles pas inutilement vastes.

Pour définir la première version de M4L, nous avons été inspirés par les langages de modélisation de la multimodalité en entrée et/ou en sortie proposés dans la littérature. Puis, nous avons suivi une approche ascendante [52] qui part des applications existantes et des contraintes d'implémentation afin de définir le langage final.
Après avoir présenté le paradigme de modélisation choisi pour notre langage, nous présentons la structure de M4L et son métamodèle en détails. Nous présentons aussi l'évolution de notre langage ainsi que son positionnement par rapport aux autres langages. Nous concluons par les intérêts et les limites de notre proposition.

4.1 Paradigme de modélisation : état transition

Avant d'identifier les concepts, le choix du paradigme de modélisation était nécessaire. Nous avons ainsi modélisé des applications mobiles multimodales existantes selon les différents paradigmes de modélisation (voir chapitre 3 section 3.1) afin de choisir le plus compatible à la modélisation de la multimodalité en entrée et en sortie sur mobile.

Les applications que nous avons modélisées ont été généralement implémentées de manière ad-hoc dans le cadre du projet ANR MOANO [1]. C'est un projet qui vise à fournir à des jardiniers ainsi qu'à des visiteurs de jardins/parcs des équipements informatiques mobiles dans le but de les aider dans leurs travaux ou leurs balades en leur permettant de saisir/rechercher d'une façon multimodale des informations contextualisées (quel est le nom de ce jardin ? quel engrais a été utilisé pour cette parcelle ? par qui ? quand ? quelle météo ce jour-là ?) [37].

eMosaic [2] est une des applications consacrées aux visiteurs du parc MOSAIC [3]. Elle indique les différents endroits du parc ainsi que des informations pratiques pour le déplacement. Elle envoie des notifications dès que l'utilisateur atteint les différents jardins (coordonnées GPS) et affiche des descriptions pour les plantes et les animaux existants. Les utilisateurs peuvent aussi utiliser l'interaction vocale et le pointage tactile en complémentarité pour voir la liste des animaux d'un jardin particulier.
Un autre exemple d'une application dans le projet MOANO est l'application « Prise de note ». C'est une application consacrée aux jardiniers. Elle permet de prendre des notes, soit en les tapant à la main (interaction tactile), soit de manière vocale. Elle permet aussi de prendre des photos associées aux notes et de scanner des QRcodes. La consultation des notes peut se faire par l'affichage ou en utilisant la synthèse vocale de la description de la note.
Nous avons modélisé ces applications avec des modèles sous les trois paradigmes : paradigme d'état transition, paradigme hiérarchique et paradigme de composants. La modélisation des applications (avec le modèle de tâches CTT pour le paradigme hiérarchique, « statechart » pour le paradigme états-transitions et l'approche ACICARE pour le paradigme de composants) a montré que le paradigme d'état-transition est le plus adapté pour modéliser les interfaces multimodales. Il permet de bien décrire le dynamisme des interfaces et la structure des échanges avec l'utilisateur. Comme le dit Cutugno et al. [28], il est aussi très utile pour la modélisation des commandes multimodales (des relations CARE [17]) et des dialogues complexes [32]. De plus, les modèles peuvent être imbriqués, ce qui réduit la densité informationnelle. À l'opposé, nous constatons que le modèle de tâches est rapidement très grand (figure 4.1) et ne permet pas de modéliser tous les aspects de l'interaction.

1. `http://moano.liuppa.univ-pau.fr/` (dernière consultation le 24/06/2014)
2. `http://emosaic.free.fr/emosaic/site/` (dernière consultation le 24/06/2014)
3. `http://www.enlm.fr/cms/home/mosaic.html` (dernière consultation le 24/06/2014)

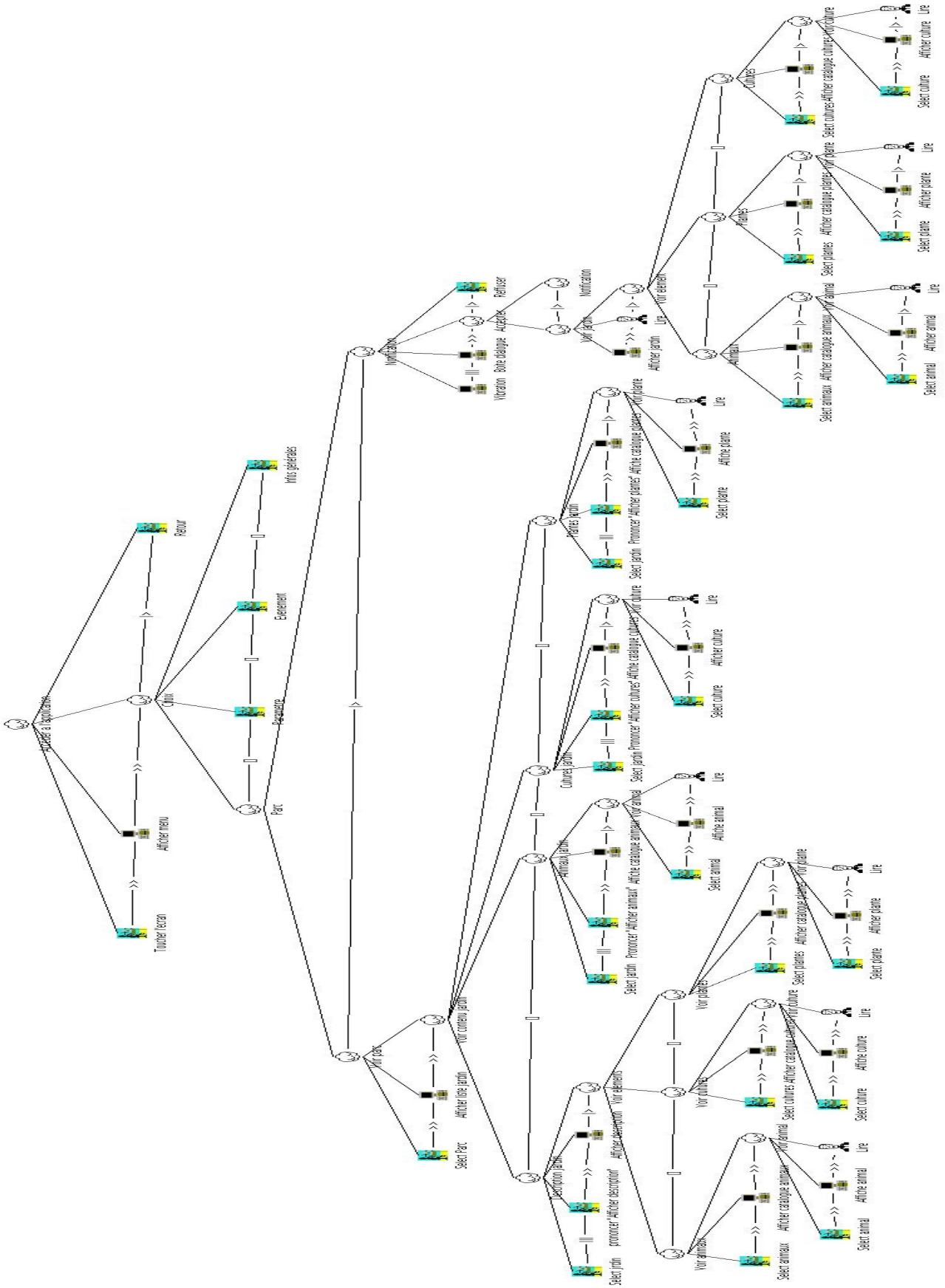


FIGURE 4.1 – La modélisation de l'application eMosaic avec le modèle de tâches CTT

Par exemple, nous n'avons pas pu modéliser les notifications qui arrivent soudainement pendant que l'utilisateur de l'application eMosaic se promène dans le parc avec le modèle CTT, car le paradigme ne permet pas de modéliser les interactions qui peuvent arriver à n'importe quel moment. Par contre, le diagramme état-transition permet de le faire facilement en utilisant un état général des interactions et en l'interrompant par un état « Notification » (figure 4.2).

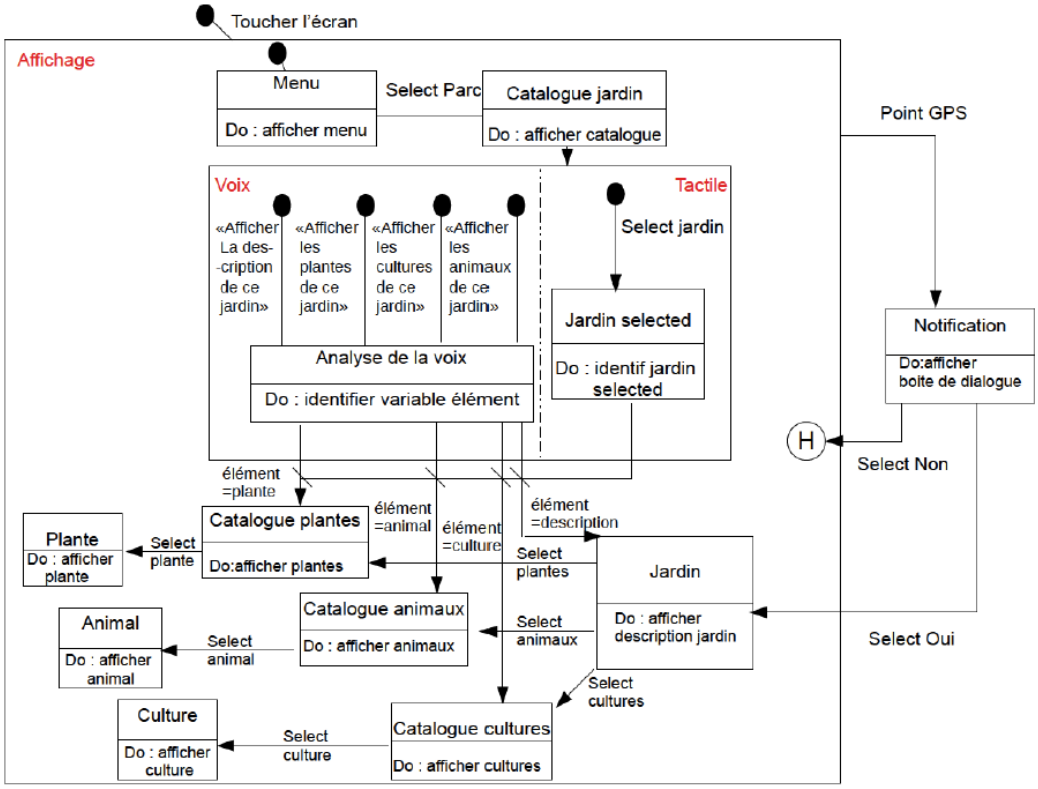


FIGURE 4.2 – La modélisation de l'application eMosaic avec le modèle état-transition (symbolique de statechart)

Contrairement au modèle CTT, la modélisation par assemblage de composants ACICARE n'a pas rapidement grandi. Elle a été faite tâche par tâche (figure 4.3) ce qui donne une bonne lisibilité. Cependant, le problème général des approches par composants est qu'il arrive fréquemment qu'un utilisateur ne puisse pas réutiliser un composant alors qu'il correspond en majeure partie à ses besoins. Par exemple, un développeur peut ne pas utiliser un composant juste parce que ses ports ne correspondent pas à ceux voulus. Ainsi, la réutilisation se réduit à faire des copies et des modifications à la main du composant (niveau code) ce que provoque une charge de travail supplémentaire avec un grand risque d'erreur [29].

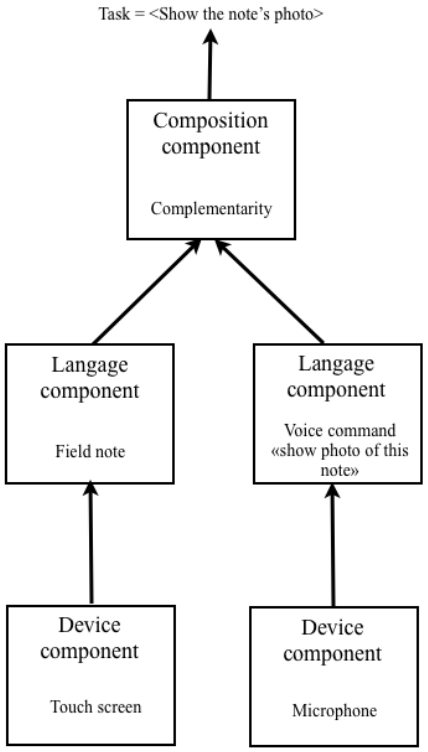


FIGURE 4.3 – La modélisation de l'application de l'interaction multimodale de l'application prise de note avec l'assemblage des composants ACICARE

Par conséquent, nous avons choisi le paradigme d'état-transition comme un paradigme de modélisation de notre langage avec une génération de code automatique (au lieu des composants réutilisables encapsulant du code). Dans la section suivante, nous présentons la structure de notre langage de modélisation. Nous présentons le méta-modèle initial et les itérations pour l'enrichir.

4.2 Structure et évolution du méta-modèle de M4L

Après l'identification du paradigme de modélisation, nous avons commencé l'identification des concepts de notre langage. Nous avons pu partir d'un langage existant tel que SMUIML et l'enrichir pour arriver à nos objectifs de modélisation. Cependant, notre but était plutôt de créer un nouveau langage pour exploiter les points positifs des différents langages de modélisation existants. Ainsi, dans une première étape, nous avons identifié les points importants de la modélisation des interactions multimodales dans les différents langages :

- L'interaction est généralement modélisée comme un évènement émis par l'utilisateur (ou son environnement) ou par l'application afin de déclencher un traitement (« une tâche » dans ICARE/OpenInterface/ACICARE ou « une action » dans SMUIML).
- Il existe deux types d'évènements : évènement en entrée et en sortie.
  - En entrée : les évènements en entrée sont les évènements les plus modélisés. Chacun possède une modalité d'interaction en entrée (définie dans UMAR par la modalité d'action). Chaque modalité peut être mise en relation avec plusieurs autres modalités en entrée. La modélisation des relations est possible dans les différents langages avec des concepts nommés différemment : « Combinaison », « Content », « Liaison », « Grouping » et « AURelationship ». Certains langages définissent les types de relation entre modalités dans le méta-modèle (généralement : complémentarité, redondance, équivalence ou concurrence), tandis que d'autres les définissent dans le modèle (instancier le concept de relation pour définir les types).
  - En sortie : comme les évènements en entrée, en sortie chaque évènement possède une modalité d'interaction et les modalités peuvent être mises en relation avec plusieurs modalités en sortie (en complémentarité, redondance, équivalence ou concurrence).
- Les traitements (tâches/actions) déclenchés par les évènements en entrée peuvent avoir des effets sur les interactions en sortie, sur le système (noyau fonctionnel) ou sur l'état de l'interaction. Les états d'interaction modélisent généralement les écrans des applications mobiles (les « States » sous MMIR et SMUIML).

Ces points nous ont permis d'identifier les concepts initiaux de notre langage ainsi que leurs propriétés nécessaires. Cependant, contrairement aux langages existants, dans M4L nous allons pas considérer que les combinaisons (relations) se font entre modalités, mais plutôt entre les évènements d'interaction [39]. Si nous prenons l'exemple « put that there » de Bolt [13], 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 seront modélisés complémentaires pour définir la commande finale de l'exemple. La considération de cette particularité revient au fait que notre langage place les évènements au centre de l'interaction plutôt que leurs modalités.

Les concepts initiaux de notre langage sont les suivants :

1. Le concept « InputEvent » : représente un **évènement en entrée** de l'application.

2. Le concept « OutputEvent » : représente un **évènement en sortie** de l'application. Cependant, en plus de ce concept nous ajoutons deux autres sous-concepts. L'ajout de ces deux concepts permet de différencier les retours d'information (feedbacks) (évènements transitoires) des interactions en sortie (évènements permanents) :
   (a) « OutputEvent Permanent » : les évènements en sortie qui existent pendant toute la durée de vie d'un état d'interaction comme l'affichage d'un bouton par exemple.
   (b) « OutputEvent Transitoire » : les évènements en sortie qui commencent et se terminent durant la durée de vie d'un état d'interaction comme une petite vibration.
3. Le concept « InputModality » : représente une **modalité d'interaction en entrée**.
4. Le concept « OutputModality » : représente une **modalité d'interaction en sortie**.
5. Le concept « Activity » : représente **l'état** actuel de l'interaction. Ce concept regroupe les différentes interactions en entrée et en sortie. Ainsi, chaque écran de l'application peut être représenté comme une « Activity ». Cependant, il est nécessaire de définir le premier état pour que l'application sache par où commencer (propriété « First »).
6. Le concept « Action » : représente le type de traitement, déclenché par les interactions en entrée, qui a un effet sur le système (pas visible pour l'utilisateur). Les autres types de traitements seront modélisés à travers des associations, car ils représentent des relations particulières entre concepts.
7. Le concept « Item » : représente un concept qu'on ajouté pour modéliser les éléments de menu de l'application. Sous Android comme sous les autres systèmes mobiles, les applications proposent des menus et chaque menu possède un ou plusieurs items.

Entre ces concepts, nous avons identifié les associations suivantes :

1. Une activité peut appeler une autre activité automatiquement (« Activity_LeadTo_Activity »). Cependant, elle ne peut appeler qu'une et une seule activité à la fois, car sur mobile un seul écran est affiché à un moment donné. Par exemple, le premier écran qui affiche le logo de l'application appelle généralement automatiquement le deuxième écran.
2. Une activité peut être déclenchée/causée (si elle n'est pas la première dans l'application) par l'arrivée d'un ou plusieurs évènements en entrée (« Causes ») comme un clic sur un bouton, une orientation du téléphone, etc.
3. Une activité peut avoir des « Items » si elle propose un menu (« Has »).
4. Une activité possède un ou plusieurs évènements permanents en sortie (« Activity_Output_Permanent »). Elle peut avoir aussi des évènements transitoires en sortie (« Activity_OutputEvent_Transitoire »).

5. L'arrivée d'un évènement en entrée peut causer l'attente d'un autre évènement en entrée (« InE_LeadTo_InE »).
6. Un évènement en entrée peut être combiné avec d'autres « InputEvents » suivant les propriétés de CARE et TYCOON suivantes : Complémentarité (« InC »), Equivalence (« InE »), Redondance (« InR »), ConCurrence (« InCC »).
7. Les évènements en sortie peuvent se combiner suivant les propriétés : Complémentarité (« OutC »), Equivalence (« OutE »), Redondance (« OutR ») et ConCurrence (« OutCC »).
8. Un évènement en entrée possède une et une seule modalité d'interaction en entrée (« Achieved_using »).
9. Un évènement en sortie possède une et une seule modalité d'interaction en sortie (« Presented_Using »). L'évènement en sortie de type « bouton » possède la modalité « affichage » par exemple.
10. Un évènement en entrée peut être reçu par un et un seul Item (« InE_appleid_Item »).
11. Un évènement en entrée peut être reçu par un et un seul évènement en sortie et un évènement en sortie peut recevoir 0 ou plusieurs évènements en entrée (« InE_applied_OutE »).
12. Un évènement en entrée peut changer un ou plusieurs évènements en sortie (type de traitement qui a un effet sur les évènements en sortie) (« InE_LeadsToChange_OutE »).
13. Un évènement en entrée peut lancer 0 ou plusieurs évènements en sortie (« Launch_new_OutputEvent »). L'application d'un long clic sur un item d'un menu, par exemple, peut provoquer l'affichage d'une alerte (un « OutputEvent Transitoire »).
14. Un évènement en entrée provoque l'application de 0 ou plusieurs actions et une action peut être causée par un ou plusieurs évènements en entrée (« Event_LeadTo_Action »).
15. Une action peut être appliquée sur une activité (« Action_AppliedTo_Activity »).

Avec les concepts et les associations précédentes, nous avons défini un méta-modèle initial pour la modélisation des interfaces mobiles multimodales (figure 4.4). Comme éditeur de modèles, nous avons utilisé le méta-éditeur ModX [4] créé dans notre équipe, qui, à partir d'un méta- modèle, génère un éditeur graphique de modèles correspondant.

4. `http://www.lifl.fr/modx/` (dernière consultation le 24/06/2014)

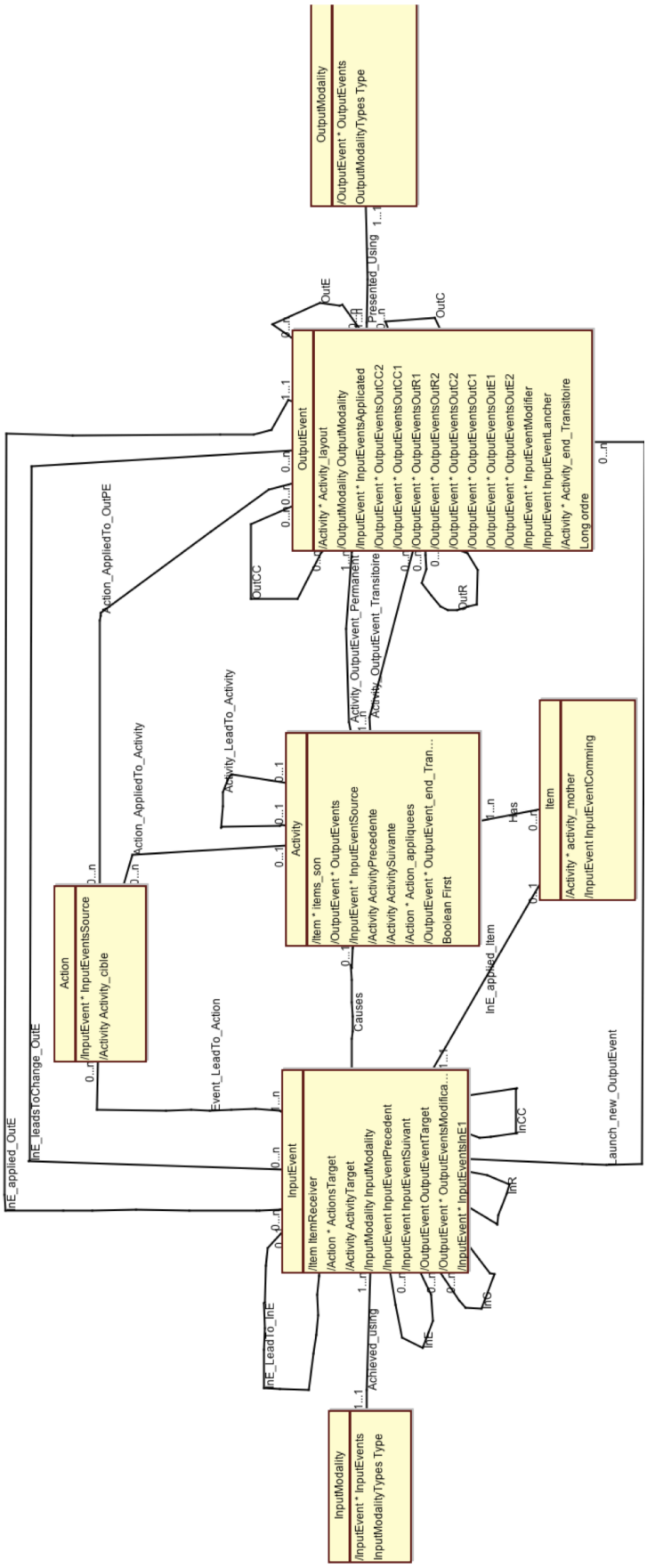


FIGURE 4.4 – La première version du méta-modèle du langage M4L (les quelques problèmes de syntaxe sont dus à l'outil de modélisation utilisé)

Nous avons enrichi ce méta-modèle initial selon une approche ascendante (« bottom-up »), c'est-à-dire à partir des applications existantes et des contraintes d'implémentation. Le schéma itératif appliqué est présenté à la figure 4.5 : Analyse d'applications mobiles multimodales - Modélisation - Métamodélisation. Dans la suite nous allons présenter les trois principaux cycles qu'on a effectué.

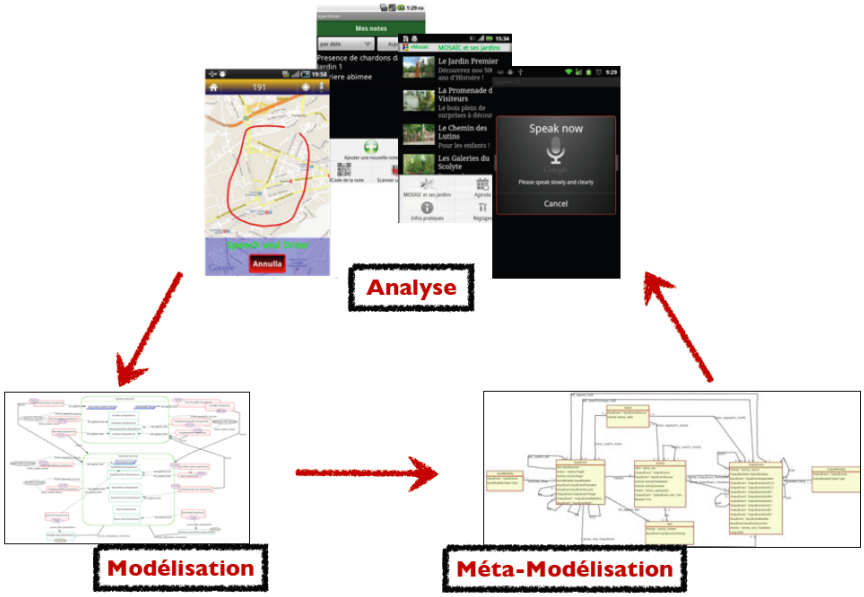


FIGURE 4.5 – L'approche ascendante pour enrichir le langage de modélisation

Basé sur la première version du méta-modèle, nous avons modélisé un ensemble d'applications mobiles multimodales (les applications « eMosaic » et « Prise de note » décrites dans la section 4.1 ainsi qu'une dizaine d'autres petits exemples comme ceux présentés dans l'annexe B). La figure 4.6 montre le modèle de l'application « Prise de note » par exemple.

La modélisation de ces applications nous a permis d'identifier certains problèmes :

– Le premier problème concerne les relations entre évènements d'interaction en entrée et en sortie : la considération de ces relations comme des associations ne permet pas de définir leurs propriétés telles que la fenêtre temporelle (le temps nécessaire en complémentarité) ou l'ordre entre interactions. Par conséquent, et puisque ces relations sont limitées, nous considérons que leurs définitions dans le méta-modèle facilitera la tâche de modélisation. Ainsi, nous définissons les concepts suivants dans le méta-modèle : « Complementarity », « Redondancy », « Equivalence » et « Concurrence » (en entrée et en sortie).

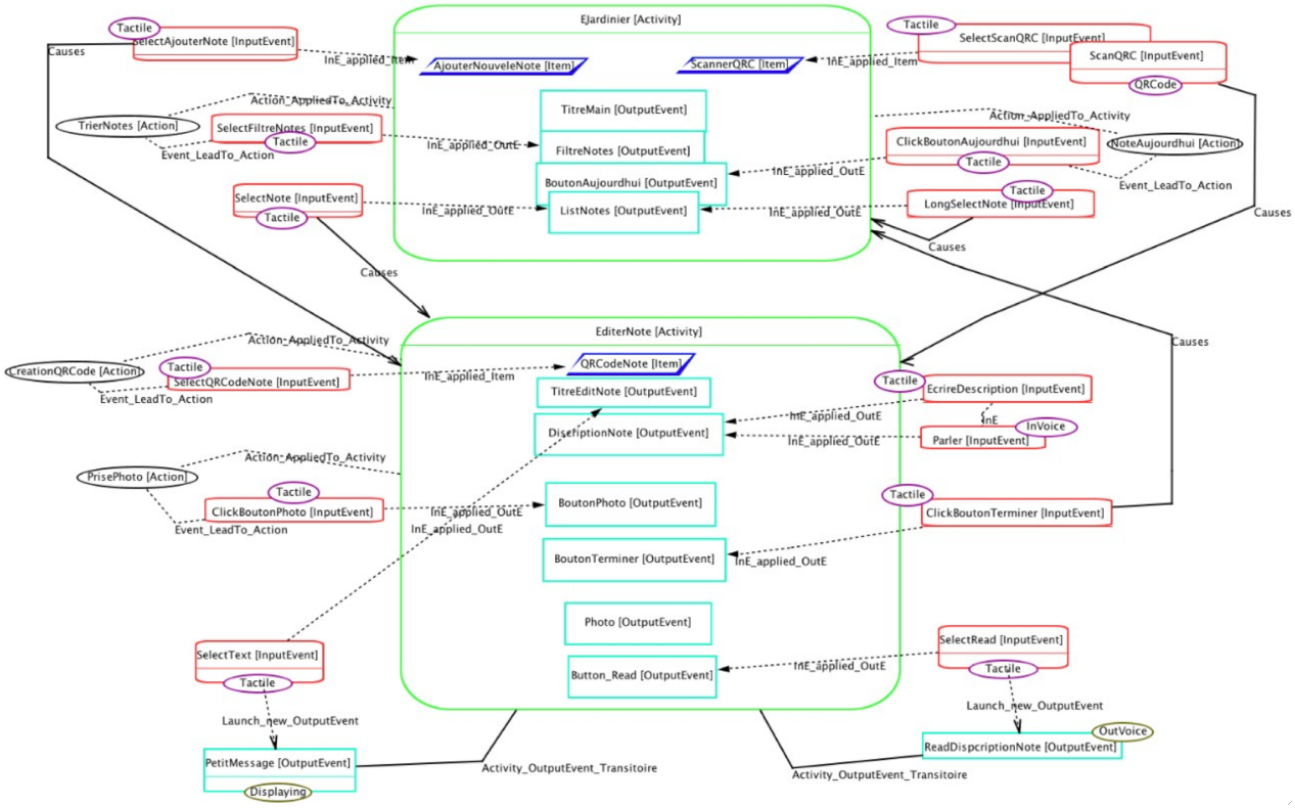


FIGURE 4.6 – Modèle de l'application « Prise de note » avec le méta-modèle initial de M4L

– Le deuxième problème concerne la modélisation des items : les menus et les items sont des types d'évènements en sortie (des évènements transitoires). Ainsi, leur modélisation à part ne fait qu'encombrer le méta-modèle.
– Le dernier problème concerne les interactions à base de capteurs : par exemple, le secouage du téléphone à base de l'accéléromètre est un évènement en entrée, mais il ne s'applique pas sur un évènement particulier en sortie. Il s'applique sur tout l'état de l'interaction et déclenche un traitement (provoque un effet). Ainsi, une nouvelle association est nécessaire entre les évènements en entrée et les états afin de permettre la modélisation de ces interactions.

Dans la deuxième version du méta-modèle [39] nous avons résolu ces problèmes (figure 4.7). Nous avons aussi modifié les noms de certains concepts et associations (« Activity » est devenu « State » pour ne pas confondre avec les activités Android).

La figure 4.8 montre une partie d'une application de description des jardins modélisée avec cette nouvelle version du méta-modèle (figure 4.7).

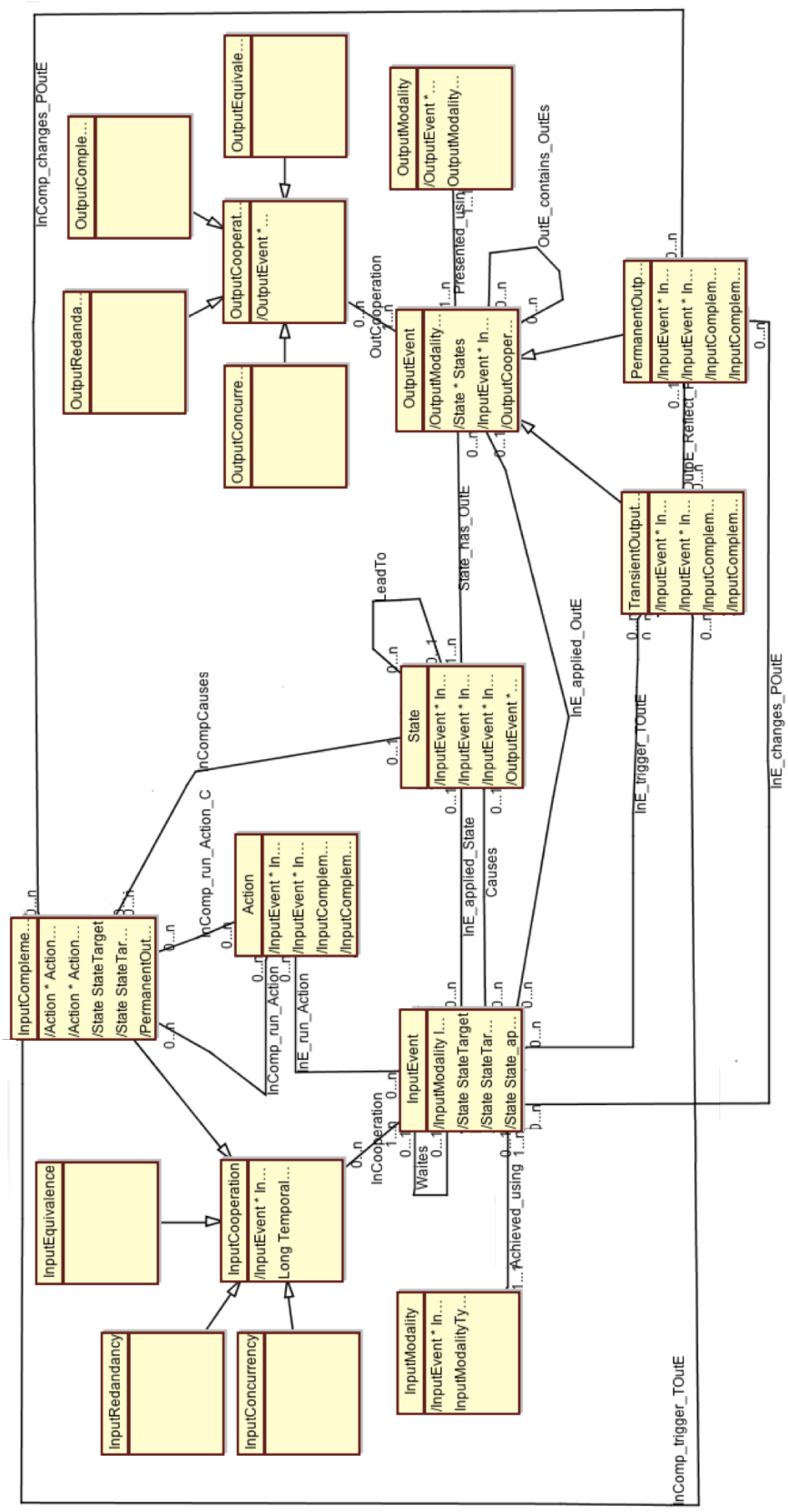


FIGURE 4.7 – La deuxième version du méta-modèle du langage M4L (les quelques problèmes de syntaxe sont dus à l'outil de modélisation utilisé)

On trouve des relations entre évènements d'interaction (« Input complementarity ») ainsi que des interactions à base de capteurs qui peuvent s'appliquer sur les états (« Aller à gauche »). Toutefois, ces interactions ne sont pas suffisamment précises. Quelques précisions peuvent être données par celui qui modélise l'application en donnant des noms significatifs pour les évènements tels que le nom « Aller à gauche » (modalité accélération), mais elles ne sont pas suffisantes. Par exemple, un évènement nommé « clic » avec la modalité « pointage tactile » est ambigu. Il peut être un « Touch », « Long Touch », avec un doigt, deux doigts, etc. Il y en va de même pour le reste des interactions. Le tableau 4.1 montre des exemples d'interaction ou de ce que nous appelons « des types d'évènements » (pour chaque modalité il existe plusieurs types d'évènements). La précision de ces types d'évènements dans les modèles n'est pas possible avec le méta-modèle actuel.

Pour résoudre ce problème, le langage SMUIML a proposé une solution (chapitre 3 section 3.1.1.1) qui consiste à définir l'interaction avec deux concepts : le concept de réception (« Recognizer ») qui permet de définir le capteur responsable ainsi que la modalité de l'interaction, et le concept de détection d'évènement issu de cette interaction (« Trigger »). Nous n'avons pas considéré cette solution dans le méta-modèle initial, car elle se base sur le concept « Recognizer » (capteur/dispositif) qui n'a pas le même niveau d'abstraction que les concepts de notre langage (son niveau est un peu bas puisqu'il est plus proche de la programmation). Toutefois, nous pouvons la considérer pour résoudre le problème de typage d'évènement, mais en élevant le niveau d'abstraction des concepts utilisés.

Nous avons déjà les concepts « InputEvent » et « OutputEvent » qui permettent de modéliser les évènements. Puis, au lieu des deux concepts « InputModality » et « OutputModality » nous pouvons utiliser deux nouveaux concepts qui permettent de décrire d'une façon abstraite les types d'évènements : « TypeInputEvent » et « TypeOutputEvent ». Ces concepts permettent aussi la définition de la modalité d'interaction associée aux types avec l'attribut « modality ». Lors de la modélisation, il est plus intéressant de mettre en avant les types d'évènements, car ils sont plus impliqués dans l'interaction que les modalités. De plus, un type d'évènement permet de référencer rapidement la modalité d'interaction ce qui n'est pas le cas pour une modalité (« Touch » référencie rapidement la modalité « pointage tactile » tandis que « pointage tactile » ne référencie pas forcément « Touch »). Ainsi, la modélisation d'une interaction en entrée ou en sortie sera définie avec l'instanciation de deux concepts qui précisent : l'interaction effectuée (l'évènement), la façon dont elle est effectuée (le type d'évènement) et la forme du mode d'interaction utilisé pour sa réalisation (la modalité d'in-

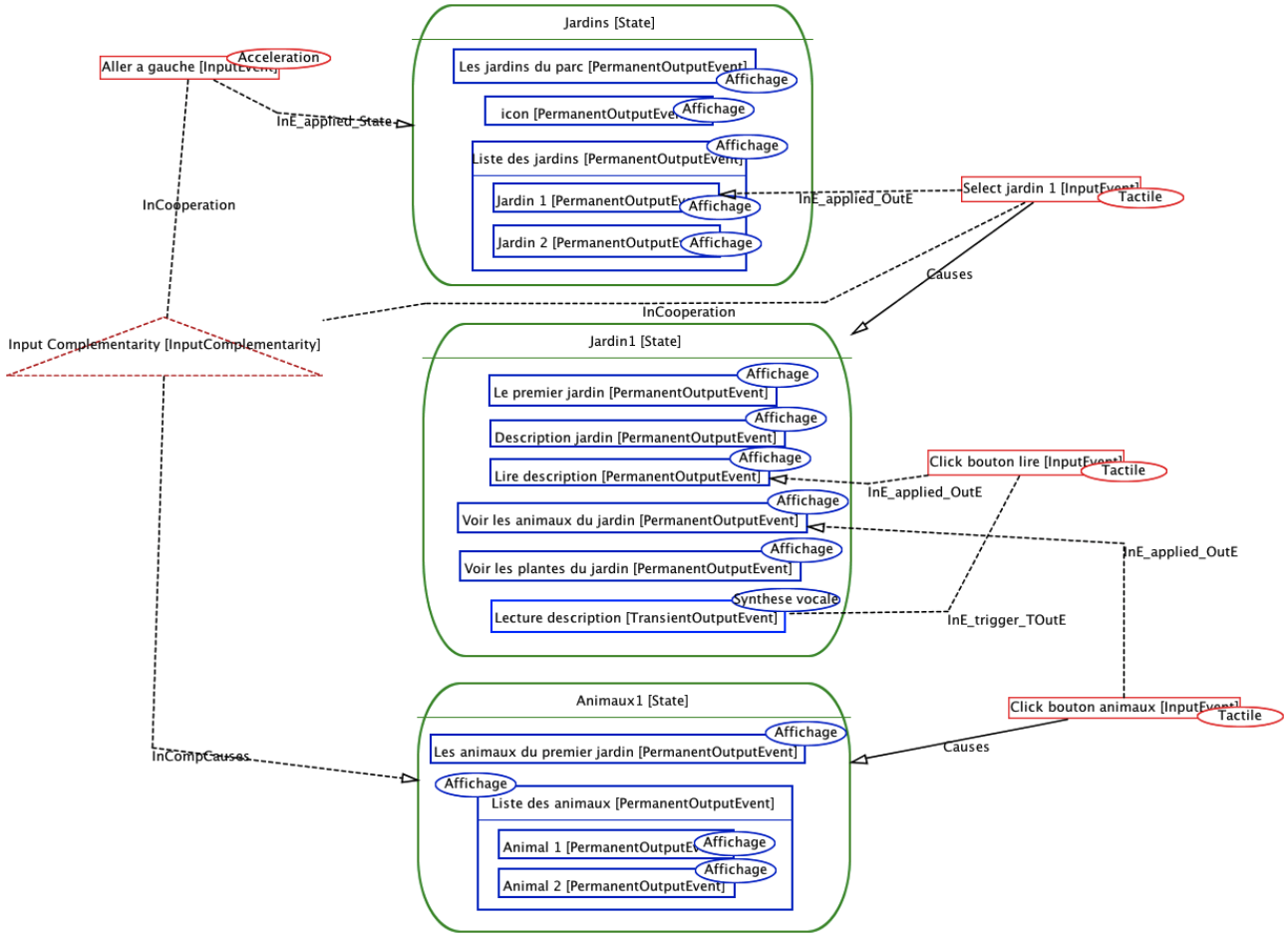


FIGURE 4.8 – Modèle de l'application « description des jardins » modélisée avec la deuxième version du méta-modèle.

teraction).

Cela permet non seulement d'augmenter la lisibilité des modèles, mais aussi de faciliter la détection des conflits entre types, modalités et modes d'interaction (voir chapitre 5 section 5.3.2). La troisième version du méta-modèle est présentée dans la figure 4.9.

Pour faciliter la tâche de modélisation et permettre aux concepteurs/développeurs de connaître les différents types d'interaction possibles en entrée et en sortie, nous avons défini une bibliothèque de modélisation. Elle fournit des modèles réutilisables modélisant les différents types d'évènements d'interaction proposés. Cette bibliothèque est détaillée dans le chapitre 5, section 5.1.3.

Après la résolution du problème de spécification des types d'évènements d'interaction, nous avons encore résolu deux problèmes dans le méta-modèle pour obtenir une version finale :

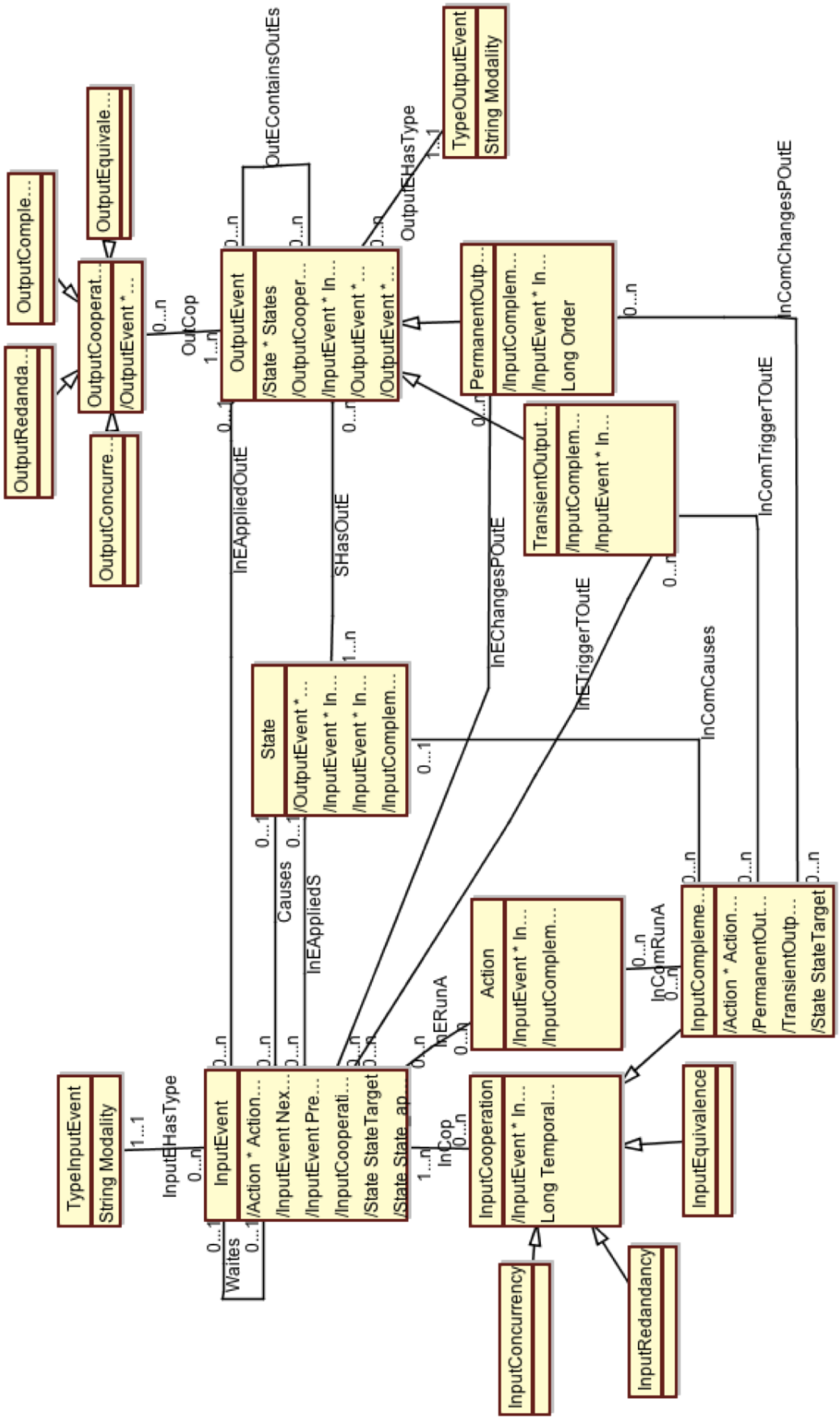


FIGURE 4.9 – La troisième version du méta-modèle du langage M4L (les quelques problèmes de syntaxe sont dus à l'outil de modélisation utilisé)

Modalités	Types d'évènements
Accélération	Secouage, Chute libre, Accélération vers le haut, Accélération vers le bas, Accélération vers la droite, Accélération vers la gauche
Orientation	Orientation vers le haut, Orientation vers le bas, Orientation vers la droite, Orientation vers la gauche, Orientation vers le nord, Orientation vers le sud, Orientation vers l'est, Orientation vers l'ouest
Luminosite	Éclairage, « Blackout »
Proximite	Loin, Proche
Localisation	Positionnement
Caméra	Scan QRcode
Pointage tactile	« Touch », « Long Touch », « Multi-touch », « Gestural touch », « Gestural multi-touch »
Vocale	Reconnaissance vocale, Synthèse vocale, Musique
Vibration	Vibration courte, Vibration longue

TABLE 4.1 – Des exemples de type d'évènement en entrée et en sortie

– Préciser les effets des changements appliqués sur les évènements permanents en sortie : le méta-modèle définit les changements appliqués sur les évènements en sortie (les widgets par exemple) comme des associations entre les évènements et les relations en entrée et les évènements permanents en sortie. Toutefois, cette modélisation ne permet pas de détecter le genre de changement appliqué (changement de couleur, édition, changement de musique, etc.). Pour résoudre cela, nous avons défini le concept « ChangeType » qui permet d'instancier les changements possibles..
– La modélisation des évènements internes (c'est-à-dire les évènements qui ne sont pas déclenchés par une action utilisateur, mais par une action interne au système comme la fin d'un timer par exemple) : pour pouvoir les modéliser, nous avons ajouté une nouvelle classe « InternalEvent » héritant d'une autre nouvelle classe « ActionEvent » qui représente les évènements ayant des effets sur l'application (évènements internes, évènements en entrée et relations).

Nous avons appliqué ces solutions pour obtenir la dernière version du méta-modèle de M4L [36] (figure 4.10). Cette version a été réalisée avec Obeo Designer [5] (voir chapitre suivant, section 5.1.1). Elle permet au concepteur de produire des modèles décrivant les interactions multimodales en entrée et en sortie avec les applications mobiles. Elle supporte les évènements d'interaction à base de capteurs, les relations entre évènements, les évènements internes des applications et les effets de chaque interaction en entrée. Avec cette version, nous avons modélisé plusieurs exemples d'applications (chapitre 6 section 6.1) ce qui nous a permis de vérifier l'habileté de M4L à modéliser les différents types d'interaction mobile multimodale. Cela nous a permis aussi de détecter les points forts et faibles du langage (voir chapitre 6 section 6.1).

La section suivante présente le positionnement de notre langage par rapport aux langages de la littérature.

4.3 Positionnement du langage M4L

M4L permet la modélisation des interactions en entrée et en sortie. Le concepteur peut modéliser les interactions en entrée et en sortie ainsi que les retours d'information approprié. De plus, il peut utiliser la relation entre évènements d'interaction telles que la complémentarité, l'équivalence, la redondance et la concurrence. M4L fournit aussi un niveau d'abstraction élevé pour la modélisation des interactions à base de capteurs. Cette modélisation ainsi que le paradigme « état-transition » augmentent la lisibilité des modèles et facilitent la détection des conflits.
Par rapport aux langages de modélisation des interfaces mobiles multimodales existants, M4L ajoute la possibilité de modéliser à la fois (avec le même langage et suivant le même paradigme de modélisation) les interactions entrantes et sortantes de l'application sans pour autant être limité à certaines modalités d'interaction (comme UsiXML, MMIR, etc.). Il ajoute la possibilité de modéliser les interactions à base de capteurs mobiles et à l'aide d'une bibliothèque des modèles réutilisables. Il facilite aussi la modélisation de ces interactions (voir le chapitre suivant, section 5.1.3). Finalement, il peut être utilisé pour modéliser n'importe quel type d'application (web, natif ou hybride).
Avec ces propriétés, nous considérons que M4L est le langage le plus adapté à la modélisation des applications mobiles multimodales.

5. `http://www.obeo.fr/pages/obeo-designer/fr` (dernière consultation le 24/06/2014)

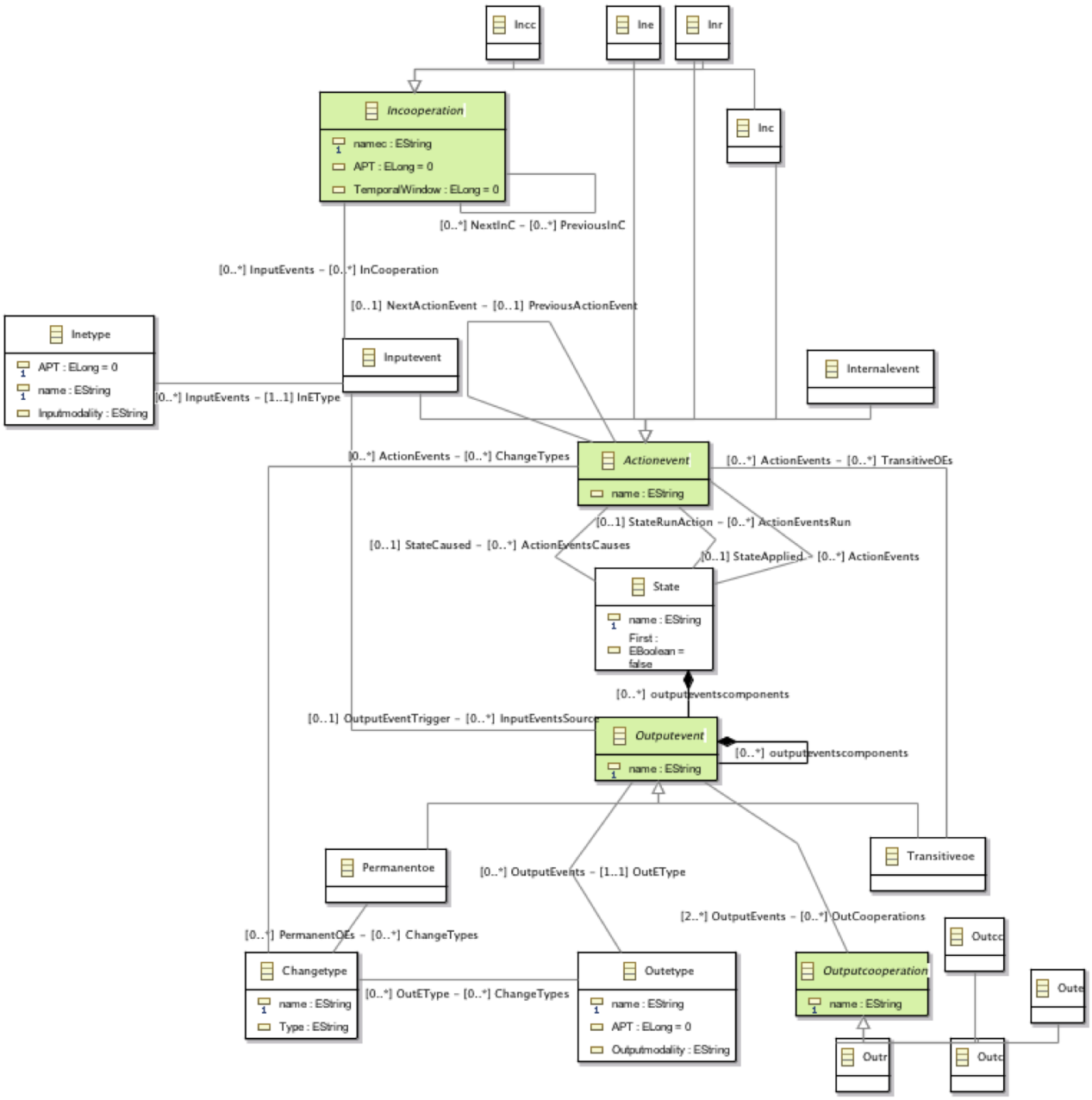


FIGURE 4.10 – La dernière version du méta-modèle du langage M4L

4.4 Conclusion

Dans ce chapitre, nous avons présenté notre proposition conceptuelle qui consiste en un langage de modélisation rigoureusement défini par un méta-modèle. Notre langage nommé M4L permet la modélisation des interfaces mobiles multimodales en entrée et en sortie. Il

permet la modélisation des évènements d'interaction à base de capteurs ainsi que des relations entre ces évènements. Nous avons présenté l'approche ascendante que nous avons suivie pour définir le méta-modèle du langage. L'approche a commencé par un méta-modèle initial (inspiré des langages existants) puis par une étape d'évolution de ce méta-modèle en fonction des applications modélisées et des problèmes détectés. Notre but était de définir des concepts pertinents et concis en s'inspirant des langages existants dans la littérature. Dans les chapitres suivants, nous allons vérifier la validité de nos choix de concepts pour la modélisation. Le gain en performance, c'est-à-dire la facilité et la rapidité de modélisation et de génération, nous permettra de valider la pertinence de notre langage et l'utilité de notre approche.

Notre langage présente encore quelques ambigüités qui peuvent être limitées en utilisant les contraintes OCL (Object Constraint Language). Par exemple, avec les concepts et associations de M4L, on ne peut pas exiger qu'un évènement en entrée, qui participe à une équivalence, ne doit pas avoir des effets propres à lui, ou qu'un état ne doit pas contenir plus d'un menu. Ainsi, nous avons défini une liste des contraintes OCL qui permettent aux concepteurs de vérifier leurs modèles avant de passer à la génération. Ces contraintes permettent non seulement de limiter les ambigüités, mais aussi de vérifier l'ergonomie des interactions (voir section 5.3.2).

Le chapitre suivant présente notre environnement de modélisation MIMIC (MobIle MultImodality Creator) qui permet de manipuler graphiquement les concepts de M4L pour modéliser les interfaces des applications mobiles multimodales.

Chapitre 5

MIMIC : un framework pour la modélisation et la génération d'interfaces mobiles multimodales

Sommaire

Dans le chapitre précédent, nous avons présenté notre langage de modélisation M4L. Les concepts de modélisation étant fixés, nous consacrons ce chapitre au framework de modélisation des interfaces mobiles multimodales que nous avons créé à base de M4L. Ce framework, nommé MIMIC (MobIle MultImodality Creator), permet la modélisation et la génération automatique d'interfaces multimodales sur différentes plateformes mobiles.

La figure 5.1 montre la fenêtre principale de MIMIC. La partie A est dédiée aux diagrammes graphiques. Elle permet de modéliser les états de l'application et de spécifier les interactions en entrée et en sortie ainsi que leurs combinaisons. La partie B est la barre d'outils (la palette) qui propose les éléments de modèle et les évènements d'interaction (la bibliothèque des évènements) à glisser et déposer dans les modèles. Finalement, la partie C représente le panel de contrôle qui affiche les propriétés des différents éléments de modèles et permet au concepteur de les modifier selon ses besoins (changer la taille des widgets, la valeur de la fenêtre temporelle, etc.).
Dans ce chapitre nous allons présenter en détail ces différentes parties.

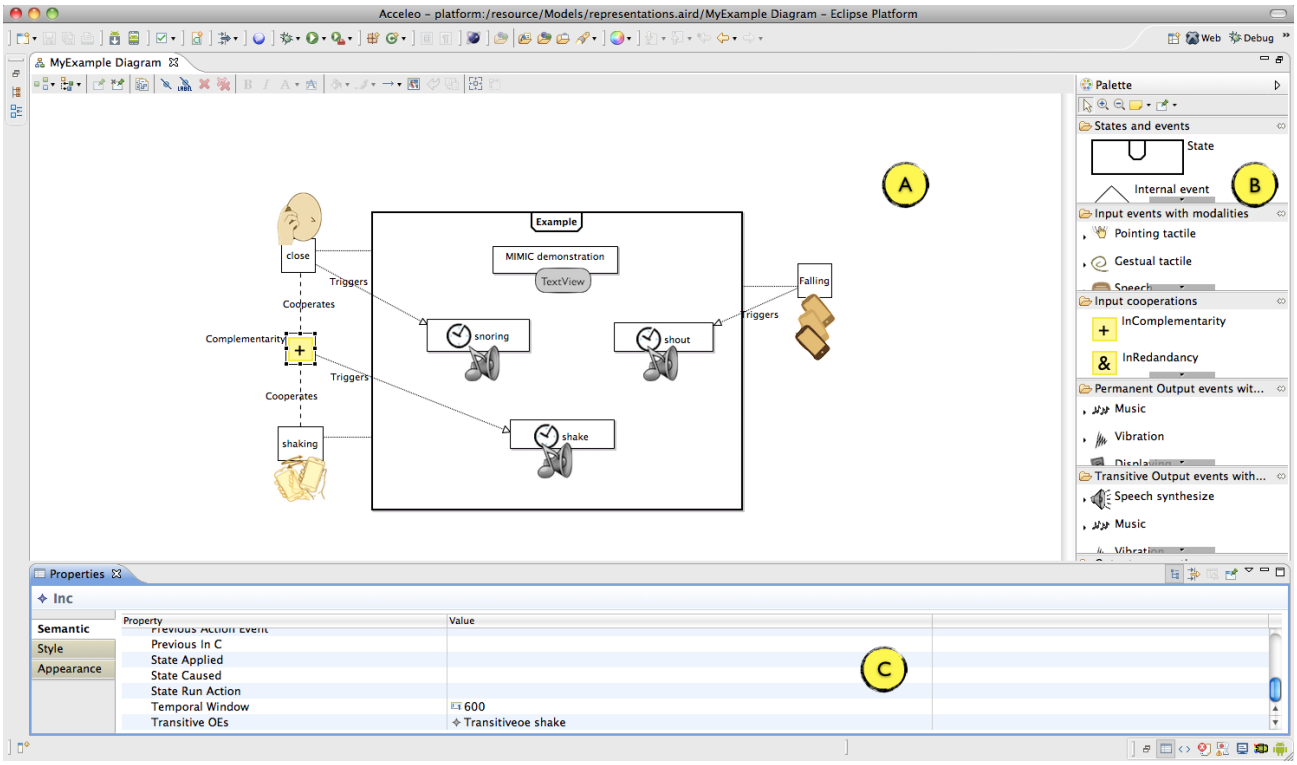


FIGURE 5.1 – La fenêtre principale de MIMIC

Après avoir exposé les outils de modélisation choisis pour notre plateforme, nous présen-

tons la notation visuelle définie pour utiliser les concepts du langage M4L. Nous présentons aussi la bibliothèque des évènements d'interaction associée à MIMIC afin de simplifier la modélisation des interactions à base de capteurs en entrée et en sortie. Finalement, nous décrivons les générateurs de code pour les différentes plateformes. Nous concluons par la description de nos implémentations des différents critères IDM au sein de MIMIC.

5.1 Outil de modélisation

5.1.1 Choix de l'environnement

Pour définir notre langage de modélisation, ainsi que notre éditeur de modèles, nous avons commencé par utiliser le méta-éditeur ModX [1], puis nous avons utilisé Obeo Designer [2].

5.1.1.1 ModX

ModX est un outil de modélisation et de méta-modélisation qui a été créé dans notre laboratoire de recherche (LIFL [3]) à Lille (2004). C'est un méta-éditeur qui permet de créer des méta-modèles et de générer leurs éditeurs correspondants avec un ou plusieurs formalismes graphiques. Il se base sur la norme MOF (Meta-Object Facility de l'OMG) et permet aussi l'implémentation des générateurs de code en JavaScript afin de générer du code à partir des modèles.
Bien moins puissant que des outils comme EMF (Eclipse Modeling Framework)/GMF (Graphical Modeling Framework), ModX est par contre un très bon outil de modélisation et il est facilement installable et utilisable. Il a été employé dans différents domaines de l'Ingenierie Logicielle (e-Learning [20], IHM multimodale [94], etc.).

5.1.1.2 Obeo Designer

Obeo est une société de conseil française qui s'intéresse à l'IDM. Elle est membre de la fondation Eclipse et contribue à de nombreux projets de modélisation. Obeo Designer est un produit basé sur Eclipse. Son objectif principal est de créer des visualisations pour les modèles en se basant sur plusieurs vues [98]. Il fournit un environnement de paramétrage pour

1. `http://www.lifl.fr/modx/` (dernière consultation le 24/06/2014)
2. `http://www.obeodesigner.com/` (dernière consultation le 24/06/2014)
3. `http://www.lifl.fr/` (dernière consultation le 24/06/2014)

configurer les points de vue, leurs représentations graphiques, les palettes d'outils, etc. Il utilise un modèle de paramétrage appelé « odesign » avec une approche interprétative pour créer dynamiquement un environnement de modélisation personnalisé à base d'un méta-modèle.

Obeo Designer se caractérise par sa simplicité d'apprentissage et d'utilisation. Il facilite la création des méta-modèles (syntaxes abstraites) et surtout des notations graphiques (syntaxes concrètes). Il se caractérise aussi par le fait qu'il se base sur les technologies solides d'Eclipse Modeling. Il est basé sur les frameworks EMF (Eclipse Modeling Framework), GEF (Graphical Editing Framework) et GMF (Graphical Modeling Framework) tel que montré dans la figure 5.2.

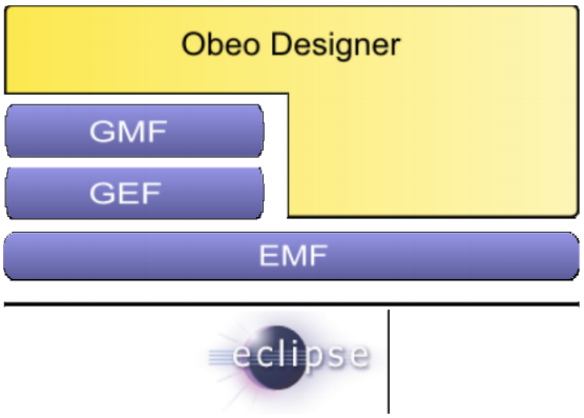


FIGURE 5.2 – L'architecture d'Obeo Designer (schéma issu de [40])

Obeo Designer cache la complexité et réduit le niveau d'expertise nécessaire pour manipuler ces frameworks afin de faciliter la tâche de création des environnements de modélisation.

Nous somme passés de ModX vers Obeo Designer pour deux raisons. La première est que ModX ne permet pas de définir des environnements de modélisation avec des formalismes graphiques riches. En revanche, Obeo Designer fournit un environnement riche pour la conception graphique des points de vue et de représentations [40]. La deuxième raison est que Obeo Designer fait partie de la communauté Eclipse, ce qui lui permet de gagner en popularité. De plus, sous Eclipse notre approche peut être facilement utilisée à l'avenir avec d'autres approches de modélisation pour les différents aspects des applications mobiles, comme l'aspect données par exemple.

5.1.2 Notation visuelle

Nous avons pris grand soin de définir une notation visuelle graphique avec une certaine efficacité cognitive pour notre langage de modélisation (syntaxe concrète). Pour cela nous avons visé le respect des critères de la physique de notation définis par Moody [74] (chapitre 2, section 2.2.3). Dans la suite nous présentons notre notation visuelle et exposons les difficultés rencontrées pour définir une notation qui respecte les différents critères.

5.1.2.1 Le choix de notation

Première notation. Nous avons tout d'abord défini une syntaxe concrète de M4L avec ModX. Cependant, il était un peu difficile de respecter les différents critères de notation visuelle (chapitre 2, section 2.2.3), car ModX ne propose pas plusieurs formalismes graphiques. Nous avons commencé par définir une représentation concrète pour chaque concept abstrait. Le tableau 5.1 présente les formes géométriques que nous avions choisies pour les différents concepts.

On a choisi trois couleurs pour identifier les concepts. Le rouge pour les interactions et combinaisons en entrée, le bleu pour les interactions et combinaisons en sortie et le vert pour les états de l'interaction. Les concepts sont donc identifiés par les couleurs ainsi que les différentes formes géographiques (expressivité visuelle). Nous avons utilisé le double codage (texte et formes géométriques) pour bien expliquer les éléments des modèles tels que montré dans l'exemple dans la figure 5.3. L'exemple illustre aussi la gestion de la complexité que l'on a utilisée et qui consiste à imbriquer les éléments du modèle : un état contient des évènements en sortie (permanents ou transitoire) et ces évènements peuvent aussi contenir d'autres évènements en sortie. Cette gestion efficace est héritée du paradigme état-transition. Cependant, nous n'avons pas respecté le fait que les symboles soient différents pour chaque concept (la notation présente peu de clarté sémantique et de discriminabilité perceptive). Les combinaisons en entrée, par exemple, ont toutes la même forme (même chose pour celles en sortie) et les interactions en entrée et en sortie ne sont différenciées qu'avec la couleur (rectangle rouge, bleu foncé et bleu clair). De plus, les symboles ne reflètent pas les contenus des concepts (pas de transparence sémantique) surtout parce que nous n'avions pas la possibilité d'ajouter des icônes avec ModX.

Nous avons ensuite essayé de trouver d'autres variables visuelles afin de mieux différencier les formes géographiques des différents concepts. Une des variables était « la bordure ». Les évènements en sortie transitoires peuvent, par exemple, être représentés par des bordures en pointillés afin d'exprimer leur volatilité (figure 5.4 : message transitoire « Hello »).

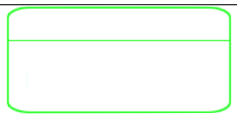
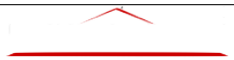
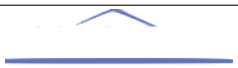
Concept	Élément graphique
State : un état de l'interaction (couleur verte)	
Input Event : un évènement en entrée de l'application (couleur rouge)	
Input Event Type : le type d'un évènement en entrée de l'application (couleur rouge)	
Input Cooperation : une combinaison entre deux ou plusieurs interactions en entrée (couleur rouge)	
Permanent Output Event : un évènement en sortie qui existe pendant toute la durée de vie d'un état d'interaction (couleur bleue)	
Transient Output Event : un évènement en sortie qui commence et se termine durant la durée de vie d'un état d'interaction (couleur bleue)	
Output Event Type : le type d'un évènement en sortie de l'application (couleur bleue)	
Output Cooperation : une combinaison entre deux ou plusieurs interactions en sortie (couleur bleue)	

TABLE 5.1 – Première syntaxe concrète de M4L

Néanmoins le pointillé peut représenter aussi la non-obligation. Nous avons également utilisé des flèches pointillées aussi afin de différencier les associations entre concepts (figure 5.5). Cependant, comme il n'y a pas beaucoup de types de flèches proposées par ModX, les associations se ressemblaient beaucoup. Avec cette notation, nous avons réalisé plusieurs exemples d'applications. Ces exemples nous ont permis de voir que cette notation est loin d'être satisfaisante et qu'elle nécessite beaucoup plus de travail afin d'être cognitivement efficace.

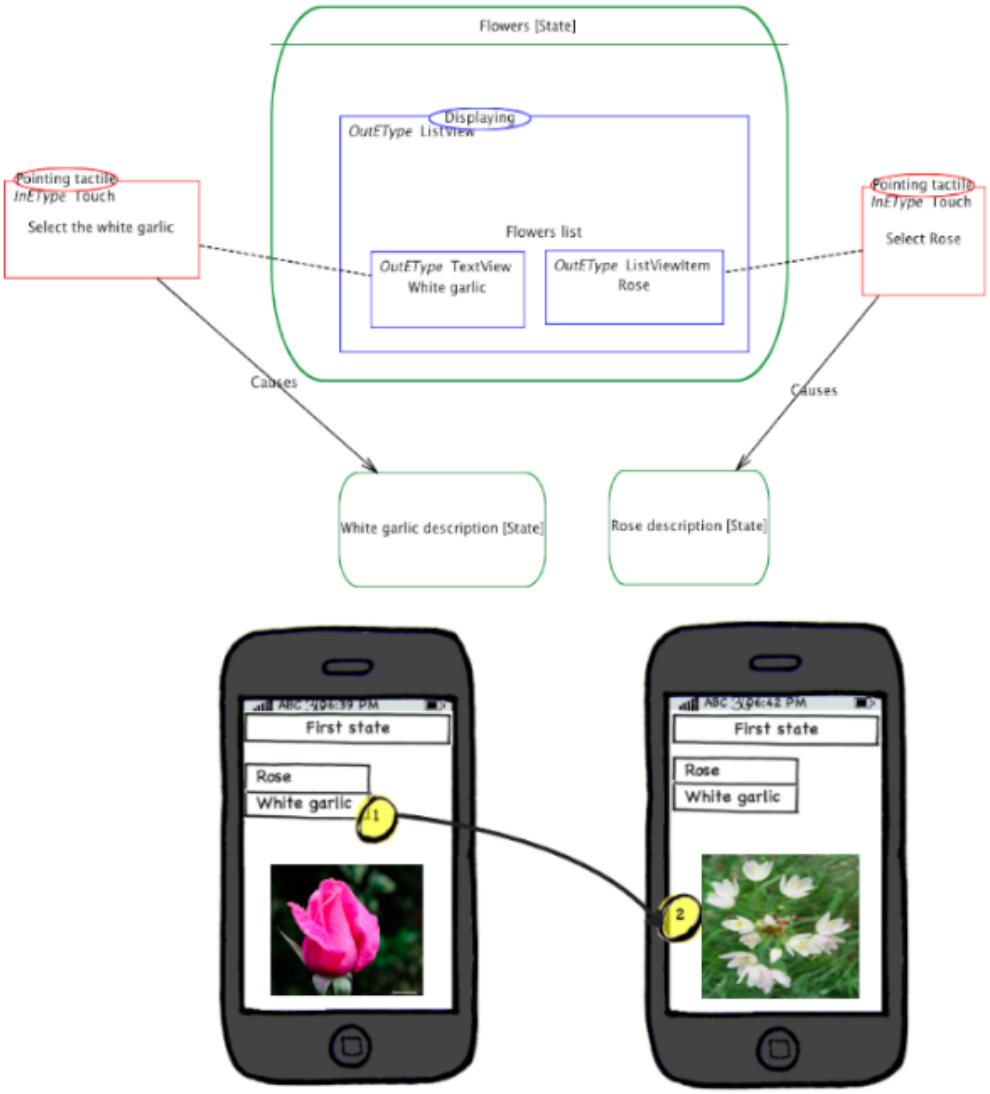


FIGURE 5.3 – Exemple de liste de fleurs avec la première notation visuelle de M4L (ModX)

Deuxième notation. Nous avons donc créé une deuxième notation sur la base de la première, mais cette fois-ci avec Obeo Designer. On avait ainsi un pouvoir d'expression plus élevé (plusieurs formalismes graphiques avec la possibilité d'intégrer des icônes). Dans cette version, nous sommes passés par plusieurs étapes avant d'arriver à une notation stable. La première étape consista à refaire la première notation sous Obeo Designer comme le montrent les figures 5.6 et 5.7. Les types d'évènements en entrée et en sortie ont été, par contre, modélisés différemment, car on avait la possibilité de les modéliser à l'intérieur des carrés qui modélisent leurs évènements.

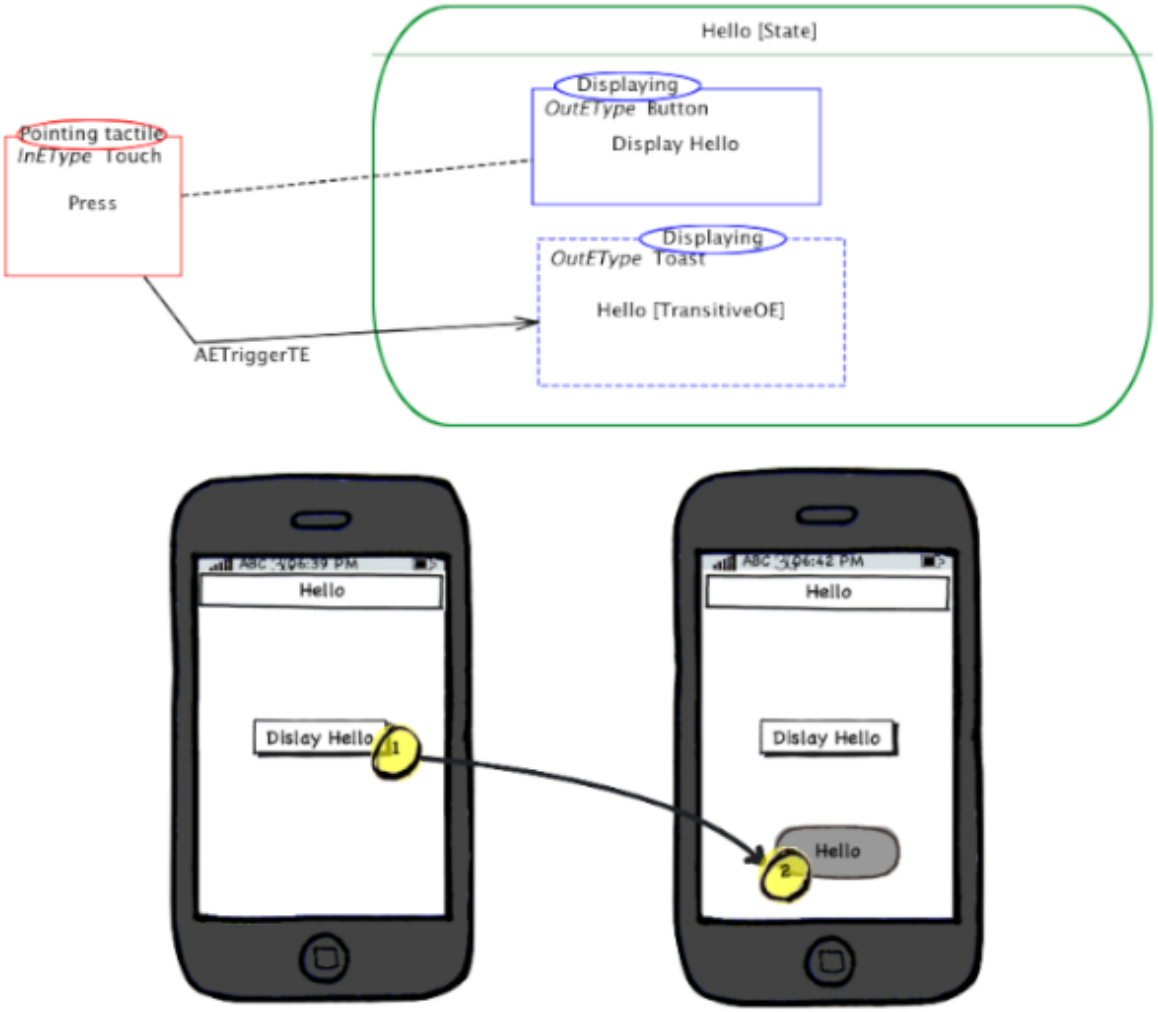


FIGURE 5.4 – Exemple de Hello Word avec la première notation visuelle de M4L (ModX)

FIGURE 5.5 – Les types de flèches possibles sous ModX

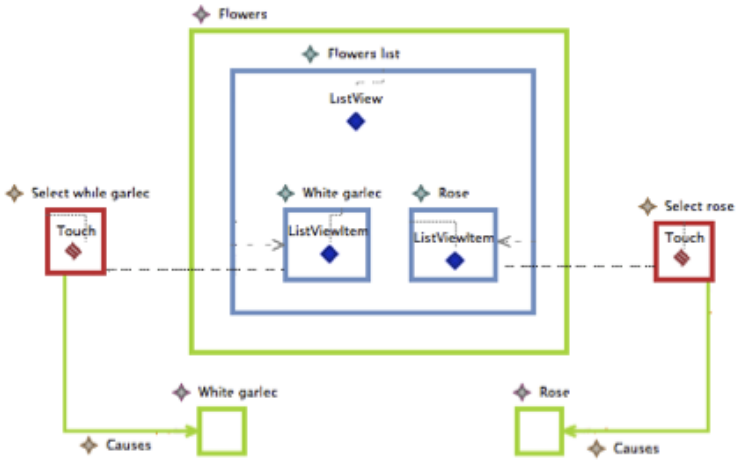


FIGURE 5.6 – Exemple de liste de fleurs avec la première notation visuelle de M4L (Obeo Designer)

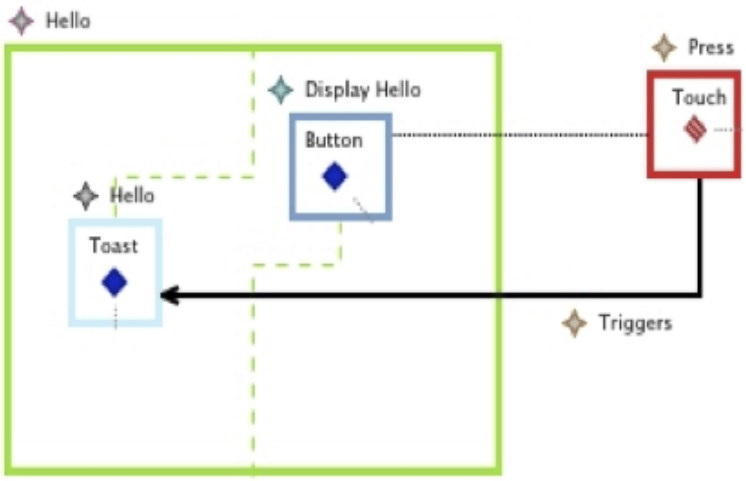


FIGURE 5.7 – Exemple de Hello Word avec la première notation visuelle de M4L (Obeo Designer)

Nous avons apporté certaines améliorations à cette notation. La première consista à ajouter des icônes. Cependant, pour choisir des icônes il nous a fallu faire des tests. Les quelques premières icônes choisies sont présentées à la figure 5.8. Chaque évènement d'interaction en entrée est modélisé par un smartphone et une flèche rouge entrante. Chaque évènement en sortie est modélisé par le même smartphone, mais avec une flèche rouge sortante.

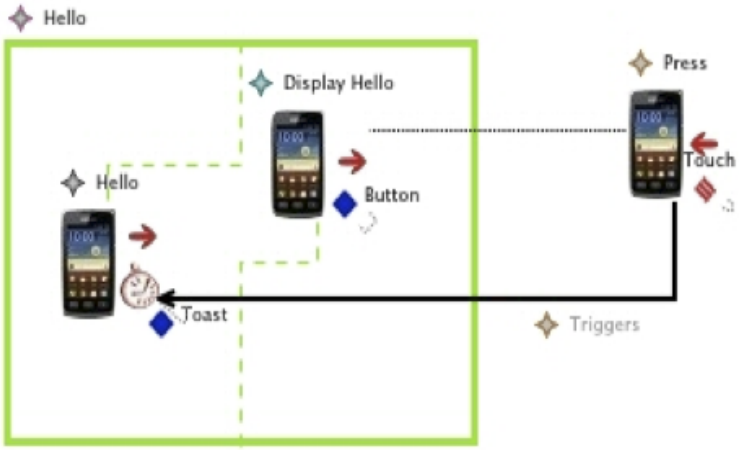


FIGURE 5.8 – Exemple de Hello Word avec des icônes

Pour faire la différence entre les évènements en sortie, les évènements transitoires ont une icône d'horloge en plus du smartphone et de la flèche, afin d'exprimer leur volatilité. Cette notation n'a pas été préférée pendant les tests effectués au sein de notre laboratoire de recherche (avec trois doctorants et trois enseignants chercheurs en IHM et en IDM ayant des connaissances en notation visuelle). Les participants disent que les icônes sont très proches les unes des autres et qu'ils n'arrivent pas à les différencier. Aussi, les types d'évènements ne sont pas clairement présentés ici, ce qui nécessite la lecture attentive des modèles (peu d'expressivité visuelle).

En se basant sur ces remarques, nous avons essayé d'améliorer encore cette notation. On a commencé par choisir deux couleurs uniquement. Le rouge en entrée et le vert en sortie (choix arbitraire). L'idée était de ne pas séparer les interactions en sortie des états de l'application puisque chaque état inclut ses propres évènements en sortie. De plus, la première vue d'un modèle avec deux couleurs uniquement permet de repérer tout de suite qu'il modélise deux grands éléments/concepts (interaction en entrée et en sortie dans notre cas). Puis, nous avons défini une forme géométrique pour le concept type d'évènement (en entrée et en sortie) : un cercle contenant le nom de type d'évènement et placé autour de l'évènement associé. Cela permet de bien cadrer les types et de les détecter plus facilement. Ensuite, nous avons défini des icônes. La première icône concerne les interactions en entrée : une flèche rouge entrante sous forme de spirale. Les combinaisons sont présentées à l'intérieur des carrés avec les symboles : + pour la complémentarité, & pour la redondance, = pour l'équivalence et //

pour la concurrence (le choix des symboles a été fait après plusieurs essais et en s'inspirant de [63]) (figure 5.9).

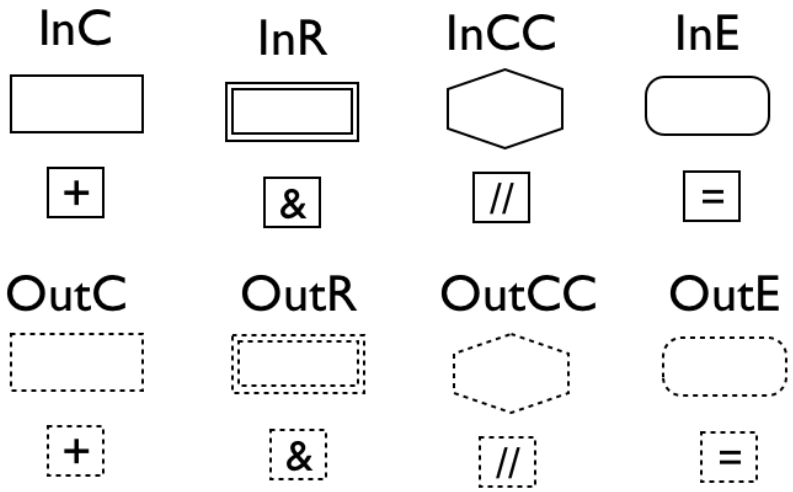


FIGURE 5.9 – Choix des symboles pour les combinaisons

Ces symboles sont rouges en entrée et verts en sortie. Les dernières icônes concernent les concepts « Internal Event » et « Change Type ». La première est représentée sous forme d'un losange tandis que la deuxième est matérialisée sous forme d'un ovale pointillé. Ainsi, chaque concept a sa propre représentation graphique différente des autres, ce qui vérifie le premier critère de la physique de notation (clarté sémantique). Finalement, nous avons choisi les flèches qui vont représenter les différentes associations (figure 5.10). La figure 5.11 montre un exemple avec cette deuxième notation.

Nous avons montré le modèle de la figure 5.12 aux participants qui ont testé notre première notation. Ils étaient plus ou moins satisfaits par rapport à la première notation. Cependant, ils avaient aussi d'autres remarques. La première remarque concerne encore la représentation des types d'évènements en entrée et en sortie.

Selon eux, cette représentation ne présente aucune transparence sémantique et donc ne permet pas d'identifier rapidement les interactions. La deuxième remarque concerne les couleurs choisies pour différencier les entrées des sorties. Un grand nombre de participants (quatre sur six) n'ont pas aimé le couplage entre le vert et le rouge dans un tel type de modèle.

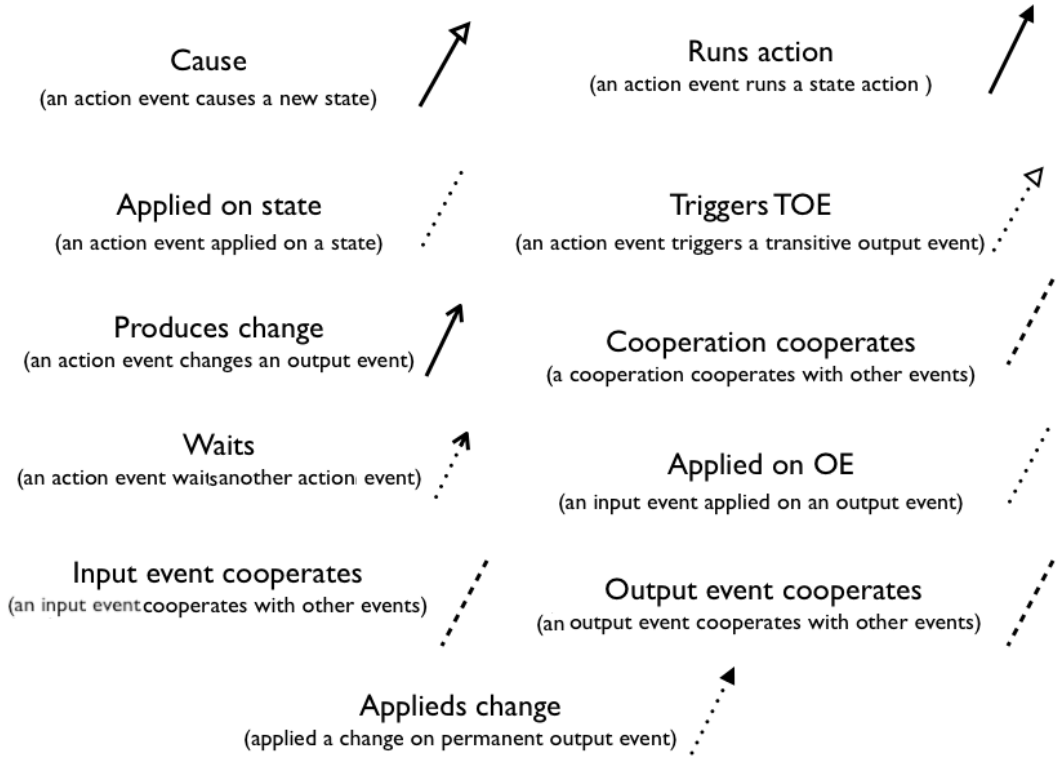


FIGURE 5.10 – Choix des formes de flèches pour les associations

Troisième notation. Pour simplifier la détection des évènements, nous avons décidé cette fois-ci de définir une icône pour chaque type d'évènement en entrée (secouage, orientation vers la droite, etc.) ou en sortie (vibration courte, bouton, image, etc.). Ainsi, la première étape d'amélioration de la deuxième version de notation était de choisir des icônes avec une transparence sémantique pour les types d'évènements.
Pour chaque type d'évènement défini dans la bibliothèque (voir section 5.1.3), nous avons défini une icône. Toutefois, il fallait aussi les normaliser avec des couleurs, car elles étaient colorées différemment (figure 5.13). Cela permet de réaliser une économie graphique (huitième critère de la physique de notation).

De nombreuses questions pourraient se poser à propos de la coloration des icônes : Est-ce que l'on reste, comme pour la deuxième notation, avec une couleur en entrée et une autre en sortie (mais en choisissant bien ces deux couleurs) ? Ou bien choisit-on une couleur par modalité d'interaction pour faciliter la détection des modalités ? Ou par mode d'interaction ? Comme nous n'avions pas de réponses pour ces questions, nous avons décidé de tester les différents cas possibles.

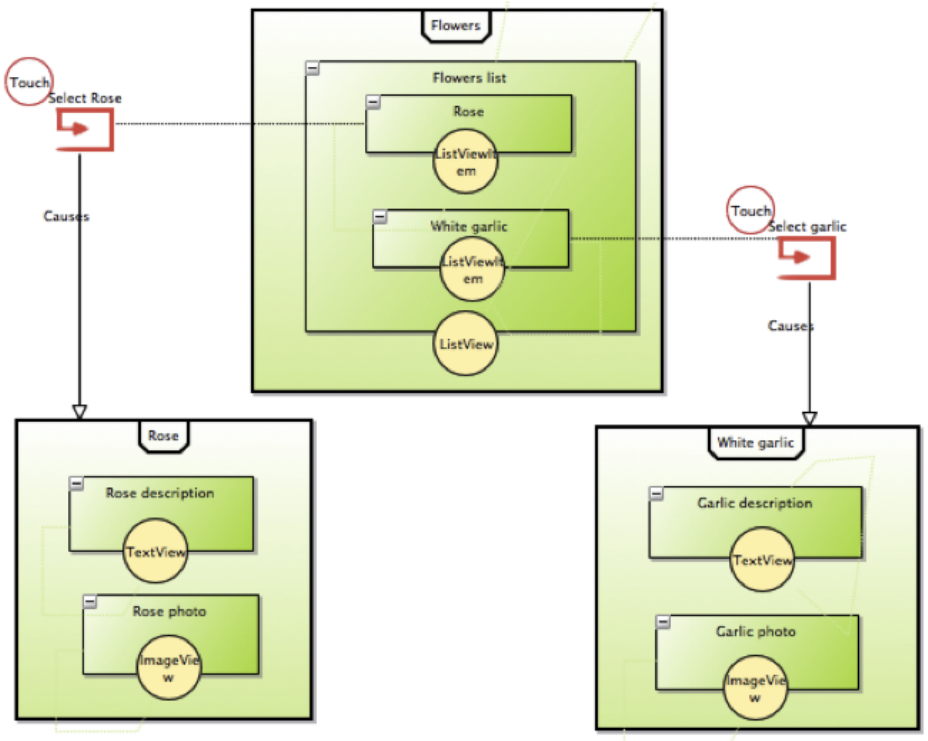


FIGURE 5.11 – Exemple de liste de fleurs avec la deuxième notation visuelle

On a commencé par choisir une couleur par modalité d'interaction. Par exemple, des types d'évènements comme le secouage, l'accélération à droite, à gauche, vers le haut et le bas seront tous colorés par la couleur de la modalité « Accélération ». Ainsi, le concepteur/lecteur de modèle pourra différencier facilement les différentes modalités d'interaction utilisées. Cela facilitera aussi la détection des conflits entre interactions (voir section 5.3.2). Le choix des couleurs était difficile, car il y a plusieurs modalités d'interaction. De plus, il fallait faire la différence entre les modalités en entrée et en sortie. Il y avait donc des couleurs qui se répétaient en entrée et en sortie ; nous avons alors réduit l'intensité lumineuse des couleurs en sortie pour faire la différence.

Les choix des couleurs pour les modalités n'ont pas été effectués au hasard (figure 5.14). Nous avons respecté les couleurs correspondant aux modes d'interactions. Par exemple, pour les types d'évènements dans le mode vocal/auditif, on utilise les couleurs proches du rouge (couleur des lèvres) et pour les types d'évènements dans le mode tactile, on utilise les couleurs proches du marron (couleurs des mains). Toutefois, il y avait des modes qui n'ont pas forcément une couleur correspondante comme par exemple le gestuel. Les figures 5.15 et 5.16 montrent les icônes colorées en entrée et en sortie respectivement.

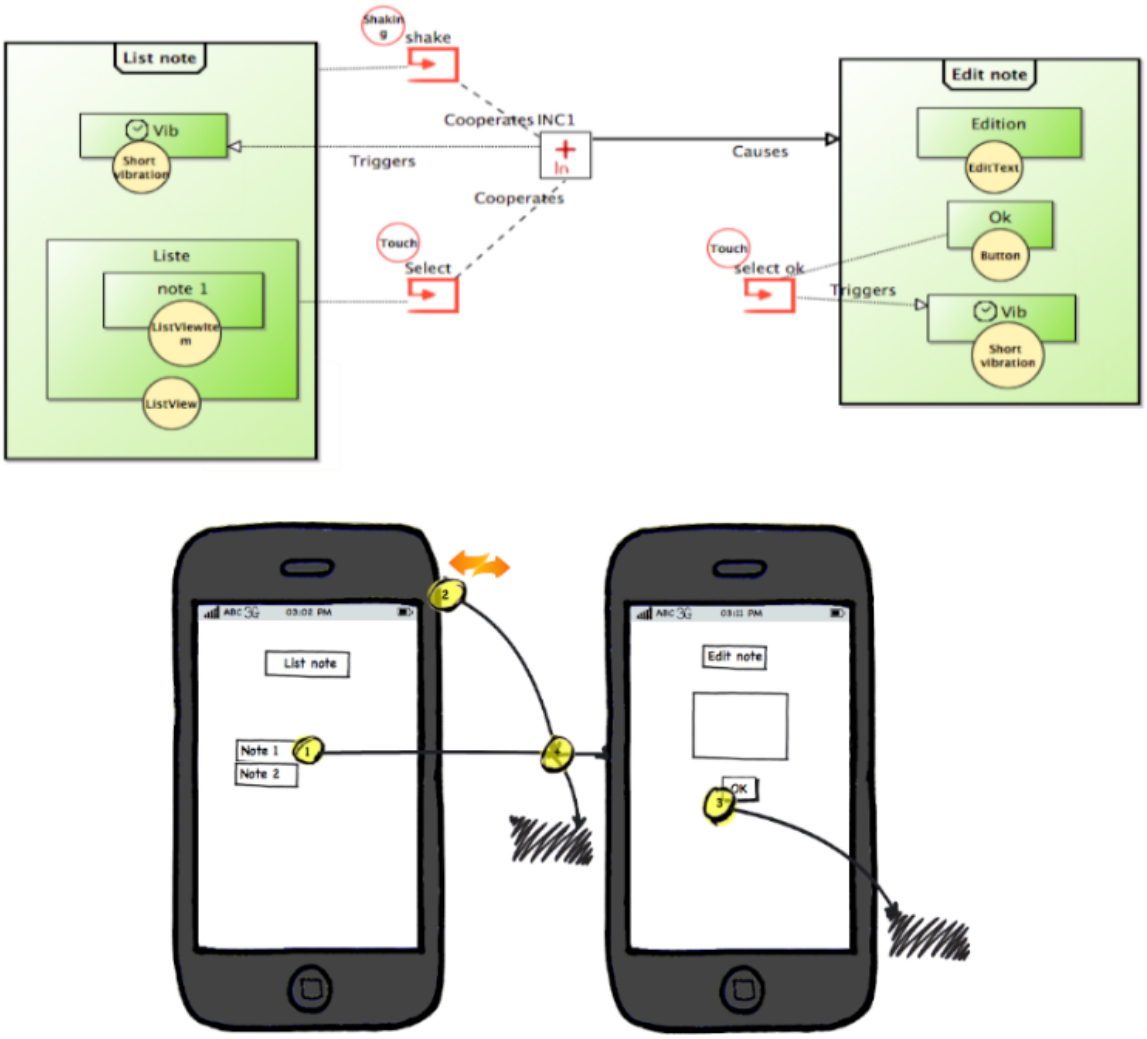


FIGURE 5.12 – Un exemple de l'application d'édition de notes modélisée avec la deuxième notation visuelle

L'exemple de l'application d'édition de notes est modélisé avec les icônes colorées (et sans les autres couleurs) dans la figure 5.17. On remarque que les couleurs ne sont pas facilement différentiables à première vue, surtout entre la couleur de la modalité vibration et la couleur des autres évènements d'affichage. Pour résoudre cela, nous avons disposé les évènements dans des rectangles avec la même couleur d'icône (figure 5.18). Les couleurs sont bien mises en avant avec la coloration des rectangles associés aux évènements. Néanmoins, pour la lecture des modèles, il est nécessaire d'avoir la légende des couleurs sous les yeux, car il y a un nombre important de couleurs légèrement différentes.

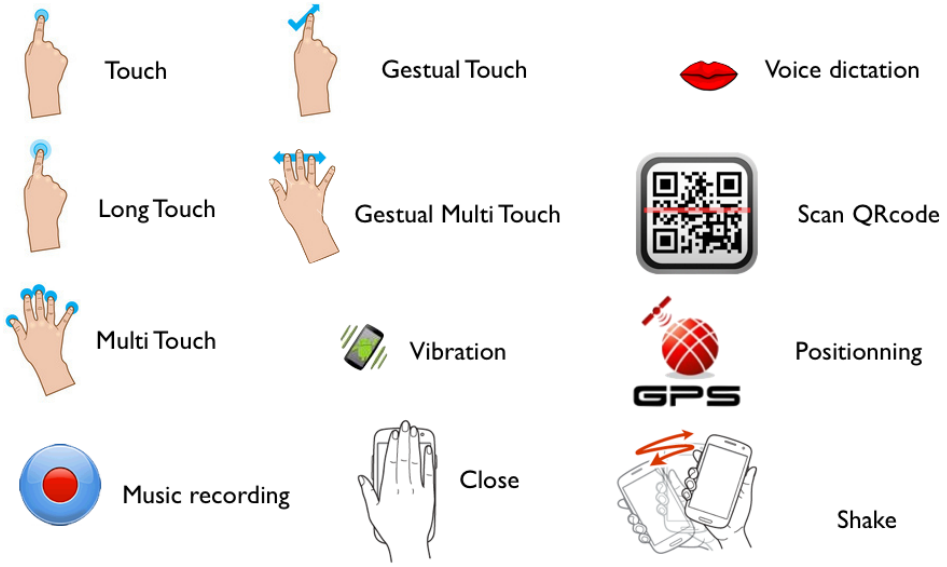


FIGURE 5.13 – Quelques icônes des types d'interaction avant le choix des couleurs de normalisation

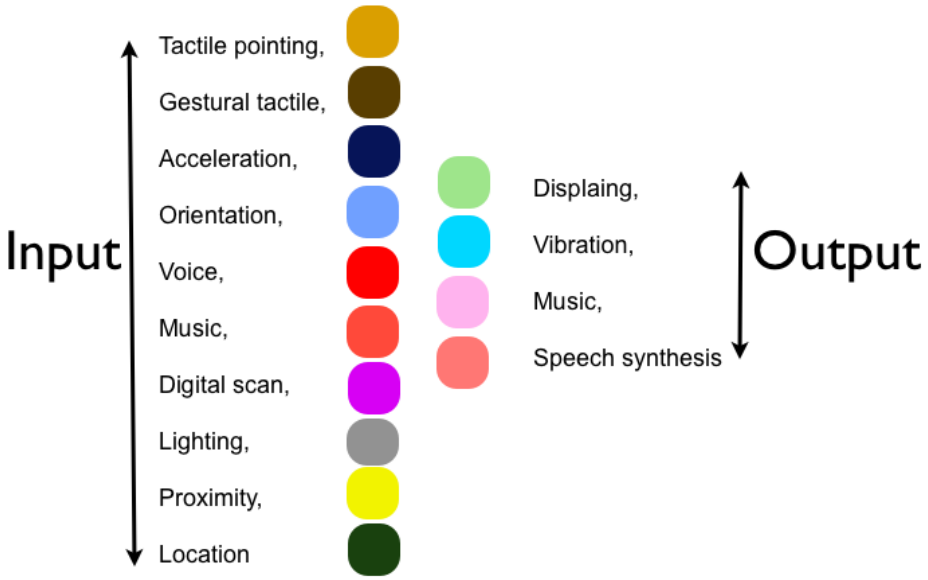


FIGURE 5.14 – Les couleurs choisies pour chaque modalité d'interaction

Ainsi, la coloration par mode d'interaction peut être plus intéressante puisque le nombre des modes d'interaction est beaucoup plus réduit par rapport à celui des modalités d'interaction. De plus, la détection des modes peut être aussi très utile pour la détection/gestion des conflits (voir section 5.3.2).

FIGURE 5.15 – Les icônes des types d'évènements en entrée, colorées selon les modalités d'interaction

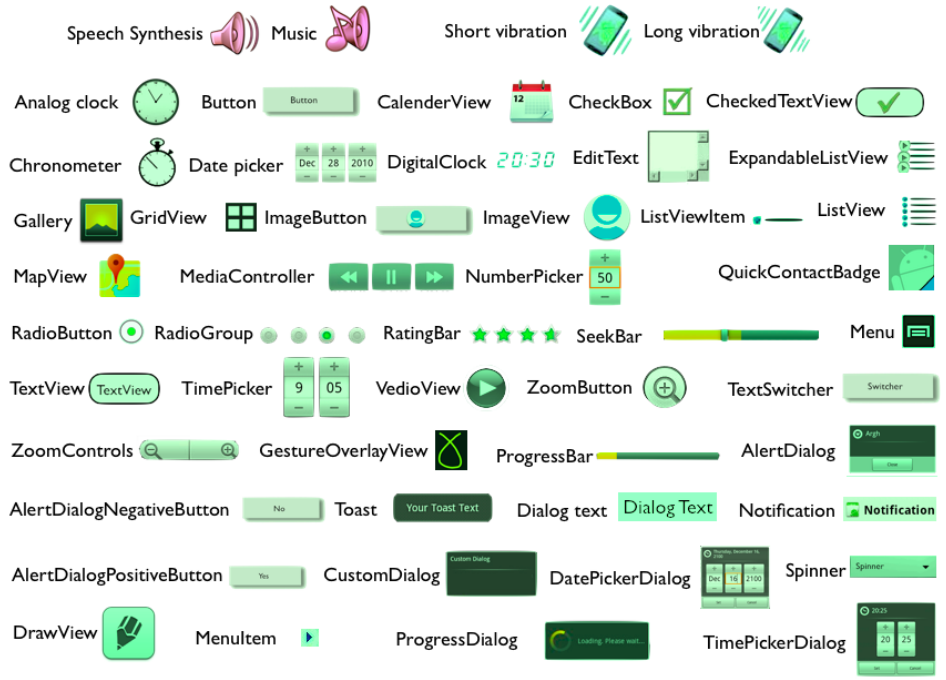


FIGURE 5.16 – Les icônes des types d'évènements en sortie, colorées selon les modalités d'interaction

Les modes d'interaction avec le mobile peuvent être décomposés en deux : 1) les sens humains (tactile, auditif et visuel) ; 2) les moyens d'expression humains (vocal et gestuel). Par conséquent, cinq couleurs, qui pourraient être par exemple les couleurs primaires et les non-couleurs (figure 5.19), suffisaient pour cette coloration.

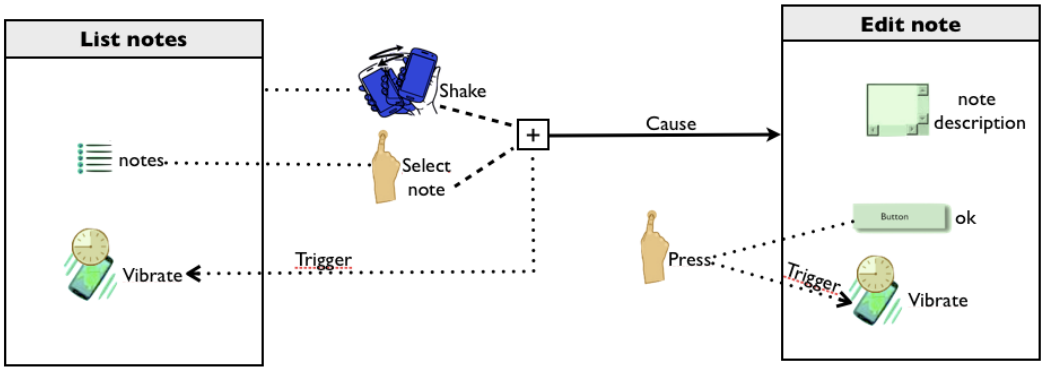


FIGURE 5.17 – Un exemple de l'application d'édition de notes modélisée avec les icônes colorées

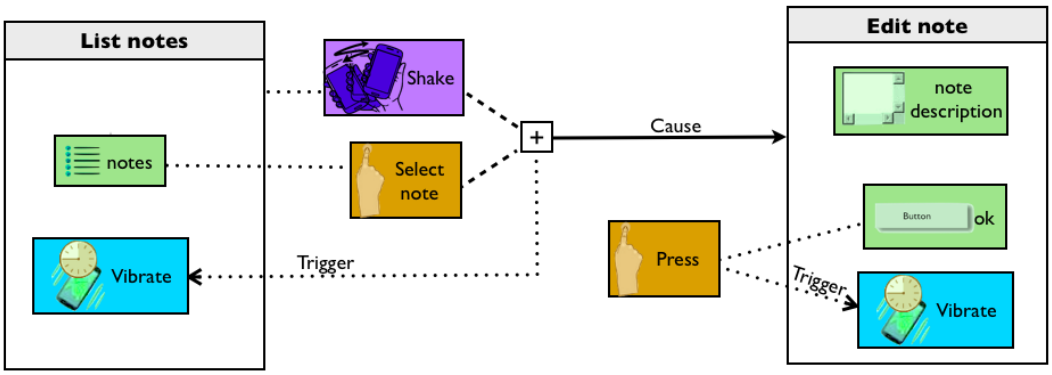


FIGURE 5.18 – Un exemple de l'application d'édition de notes modélisée avec les icônes et les arrières plans des évènements colorés

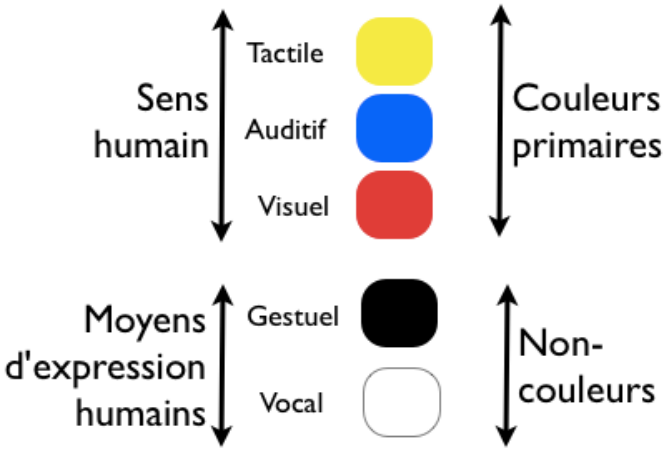


FIGURE 5.19 – Un exemple de choix de couleurs pour chaque mode d'interaction

Cependant, un nouveau problème survient. Il y a des types d'évènements qui appartiennent à plusieurs modes d'interaction. Par exemple, une vibration courte appartient aux modes : auditif, tactile et visuel. Même chose pour le scan d'un QRcode (mode gestuel et visuel), tactile gestuel (mode gestuel et tactile), etc. Pour résoudre ce problème, on peut définir des nouvelles couleurs pour les types d'évènements qui peuvent réunir plusieurs modes. Néanmoins, cela nous ramène au problème de départ (plusieurs couleurs). Par conséquent, la solution de deux couleurs (en entrée et en sortie) semble la meilleure. Il n'y aura pas d'ambiguïté entre couleurs et le lecteur pourra différencier facilement et rapidement les interactions en entrée et en sortie. En revanche, il ne pourra pas faire la différence entre les modalités et modes d'interaction à première vue. Pour résoudre ce problème, nous avons regroupé les types d'évènements par modalité d'interaction dans la palette de modélisation de MIMIC (voir section 5.3.2). Ainsi, lors de la sélection des types d'évènements, le concepteur aura déjà une idée sur les modalités et les modes d'interaction utilisés.

Pour le choix des deux couleurs, nous avons commencé par l'identification de la couleur en entrée. Le but était de choisir une couleur qui permette d'exprimer la richesse des interactions en entrée. Nous avons alors choisi le jaune brillant pour colorer les icônes des types d'évènements entrants aux applications. Puis, il a fallu choisir une couleur qui se marie parfaitement avec le jaune (pour respecter la remarque des participants au test de la deuxième notation). Le jaune se marie très bien avec les tons neutres : blanc, crème, marron, gris et noir [4]. Notre choix était plutôt dirigé vers le gris, car il est à mi-chemin entre le blanc et le noir. En outre, il est loin de la couleur utilisée en entrée (contrairement au marron et crème). Les icônes colorées sont présentées dans les figures 5.20 et 5.21.

La notation des concepts restants est présentée dans le tableau 5.2 tandis que la notation pour les associations est la même que celle de la figure 5.10. Le modèle de l'exemple de l'application « Prise de notes » avec cette troisième notation est présenté dans la figure 5.22. On a montré ce modèle, ainsi que d'autres exemples [5] aux participants qui ont testé notre première et deuxième notation. Ils étaient satisfaits cette fois-ci surtout par les icônes des types d'évènements. Ils ont trouvé que ces icônes suggèrent bien les sémantiques des différents types d'interaction. En outre, les textes qui les accompagnent (les noms des évènements) jouent un rôle important pour compléter leurs symboliques. Ils ont trouvé aussi que les icônes sont facilement distinguables les unes des autres.

4. `http://www.toutes-les-couleurs.com/couleur-jaune.php` (dernière consultation le 24/06/2014)

5. `http://www.lifl.fr/~eloualin/examples.html` (dernière consultation le 24/06/2014)

FIGURE 5.20 – Les icônes des types d'évènements en entrée colorées en jaune brillant

Ainsi, les personnes ayant testé cette notation n'avaient plus de remarque particulière, ni à propos des couleurs utilisées, ni à propos du reste des notations. L'impression générale était que cette troisième notation est la meilleure par rapport aux précédentes et qu'elle semble satisfaisante.

Du point de vue de la physique des notations, cette notation respecte la plupart des critères/principes. Chaque concept correspond à un et un seul symbole visuel et chaque symbole visuel correspond à un et un seul concept (clarté sémiotique). Les symboles fournissent une transparence sémantique et ils sont différents les uns des autres (discrimination perceptive). Les états incluent les évènements et les combinaisons en sortie pour gérer la complexité. Chaque concept possède un symbole et un nom textuel pour mieux l'expliquer (double codage). La notation utilise plusieurs variables visuelles (la forme, les couleurs, etc.) ce qui augmente l'expressivité visuelle, mais utilise un nombre important d'icônes ce qui ne contribue pas à l'économie graphique. De même, la notation n'offre pas d'adaptation cognitive (adaptation selon les catégories d'utilisateurs en utilisant des dialectes graphiques différents) ainsi que l'intégration cognitive (faciliter le passage entre diagrammes) pour deux raisons.

FIGURE 5.21 – Les icônes des types d'évènements en sortie colorées en gris

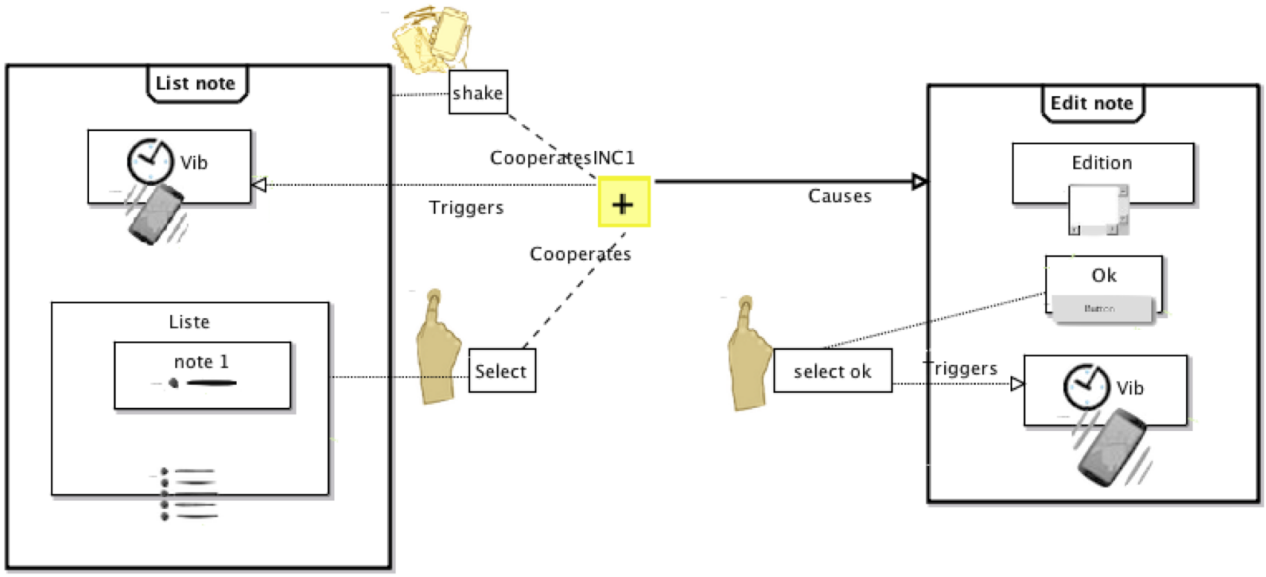


FIGURE 5.22 – L'exemple de l'application d'édition de notes modélisée avec la troisième notation visuelle

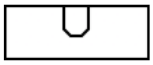
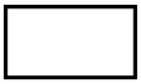
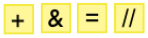
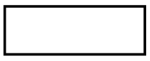
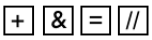
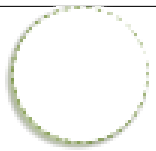
Concept	Élément graphique
State	
Input Event	
Input Complementarity, Redundancy, Equivalence and Concurrence	
Permanent Output Event	
Transient Output Event	
Output Complementarity, Redundancy, Equivalence and Concurrence	
Internal Event	
Change Type	

TABLE 5.2 – Troisième notation des concepts de M4L

La première est que l'environnement de modélisation de MIMIC ne sera fourni que pour les concepteurs/développeurs d'applications mobiles (une seule catégorie) ce qui ne nécessite pas l'adaptation de la notation. La deuxième raison est que notre approche est limitée à la modélisation et la génération des interactions multimodales. Ainsi, un seul modèle sera mo-

délisé à la fois pour chaque application mobile, ce qui ne nécessite pas l'intégration cognitive.

En plus des tests effectués dans notre laboratoire pour vérifier l'efficacité de la notation visuelle, nous avons également réalisé une autre évaluation avec des étudiants en informatique. Le but était d'évaluer l'approche générale et de recueillir leurs avis par rapport à la notation proposée. Les résultats de cette évaluation seront expliqués dans le chapitre suivant.

5.1.2.2 La définition de la notation sous Obeo Designer

Après le choix de notre notation visuelle, nous l'avons associée à notre méta-modèle. Pour faire cela, Obeo Designer nous permet de définir une vue graphique (Viewpoint Specification Project) pour les modèles issus de notre méta-modèle. On commence par les spécifier, nœud par nœud (concept par concept), et relation par relation (association par association). Chaque noeud nécessite la définition du concept associé, la notation graphique du concept (forme géométrique ou icône) et la position dans le modèle (sur le bord d'un autre concept, séparé, etc.). De même, la définition des relations nécessite la spécification de l'association et du type de flèches (couleur, style, etc.). Obeo Designer donne aussi la possibilité de définir des notations conditionnelles. Par exemple, les concepts type d'évènement en entrée et en sortie n'ont pas une seule notation graphique. La notation dépend du type de l'évènement (notation secondaire). Ainsi, pour ces concepts, on a précisé que la notation change selon l'instance créée (figure 5.23).

La figure 5.24 montre l'arborescence de la notation visuelle définie sous Obeo Designer.

Obeo Designer permet aussi la définition de la palette de l'environnement de modélisation. Cette palette facilite la tâche de modélisation graphique des interfaces multimodales en présentant aux concepteurs les concepts sous forme graphique (leurs notations visuelles).

Nous avons décomposé la palette en sept sections :

- une première pour les concepts « état » et « évènement interne » ainsi que les différentes associations,
- une deuxième pour les différents types d'évènements en entrée,
- une troisième pour les combinaisons en entrée,
- une quatrième pour les types d'évènements permanents en sortie,
- une cinquième pour les évènements transitoires en sortie,
- une sixième pour les combinaisons en sortie,
- une dernière pour les types de changement possibles.

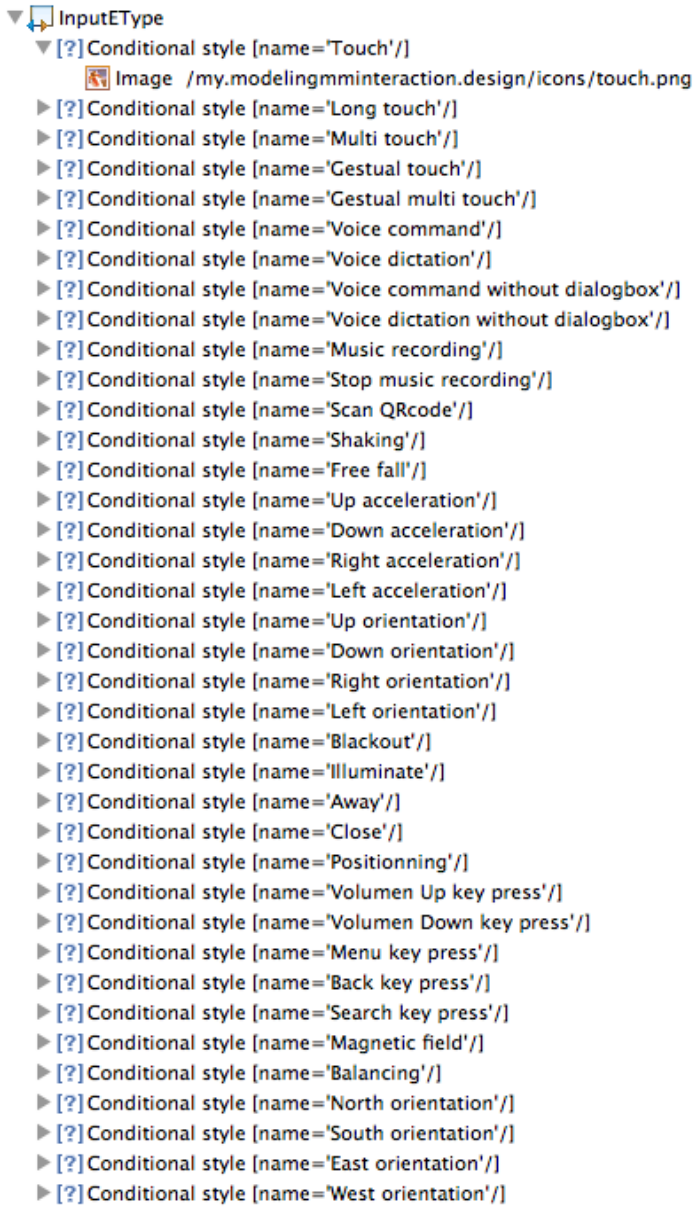


FIGURE 5.23 – La notation conditionnelle pour le concept de type d'évènement en entrée

Le but était de regrouper les concepts proches afin de faciliter la recherche de ces concepts lors de la modélisation. Chaque concept/association dans les sections est défini par une icône (icônes miniatures de leurs propres icônes dans la notation), un texte explicatif (le nom du concept/association généralement) et une définition des actions à effectuer lors de l'utilisation du concept/association dans un modèle. Ces actions permettent généralement la création des nouvelles instances d'une ou plusieurs actions et/ou associations (figure 5.25).

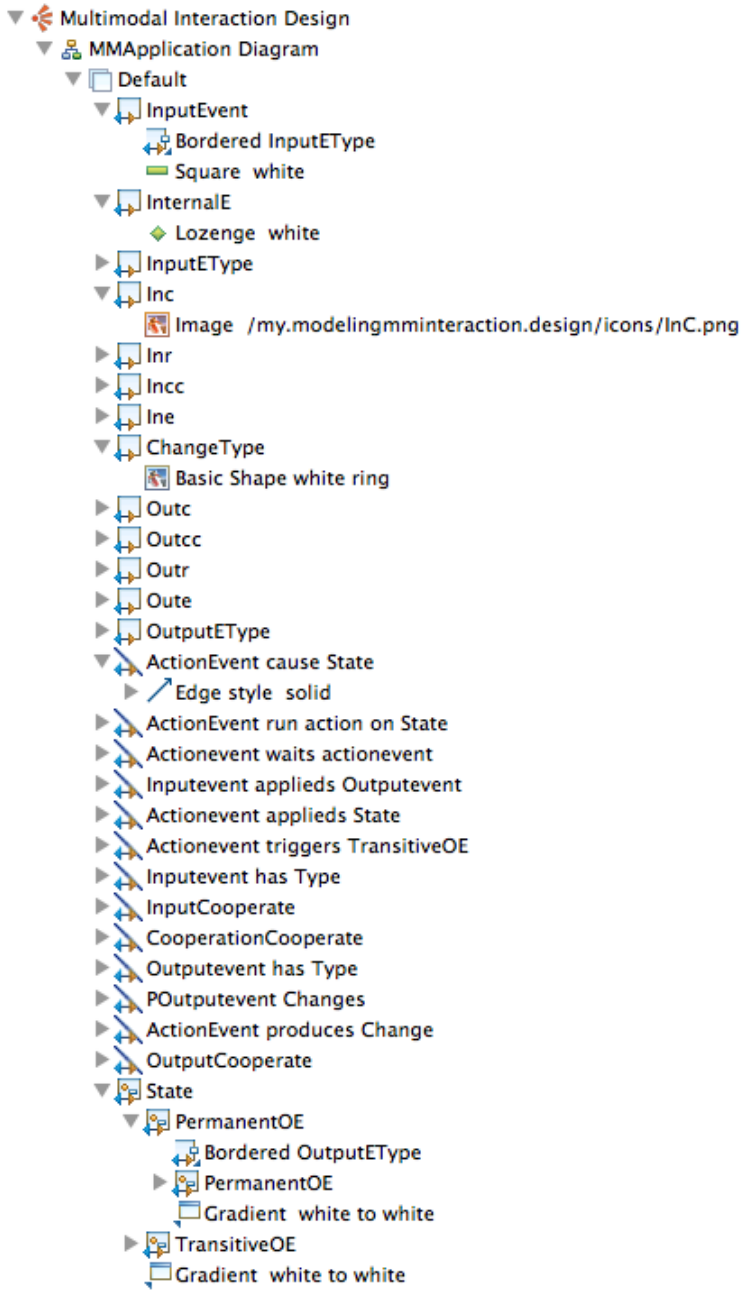


FIGURE 5.24 – La définition de la notation visuelle sous Obeo Designer

5.1.3 Bibliothèque des types d'évènements d'interaction

Lors de la définition de notre langage de modélisation M4L, nous avons rencontré le problème de précision des évènements d'interaction en entrée et en sortie (chapitre 4, section 4.2). Pour résoudre ce problème, nous avons décidé de permettre la précision des évènements au niveau modèle en utilisant une bibliothèque.

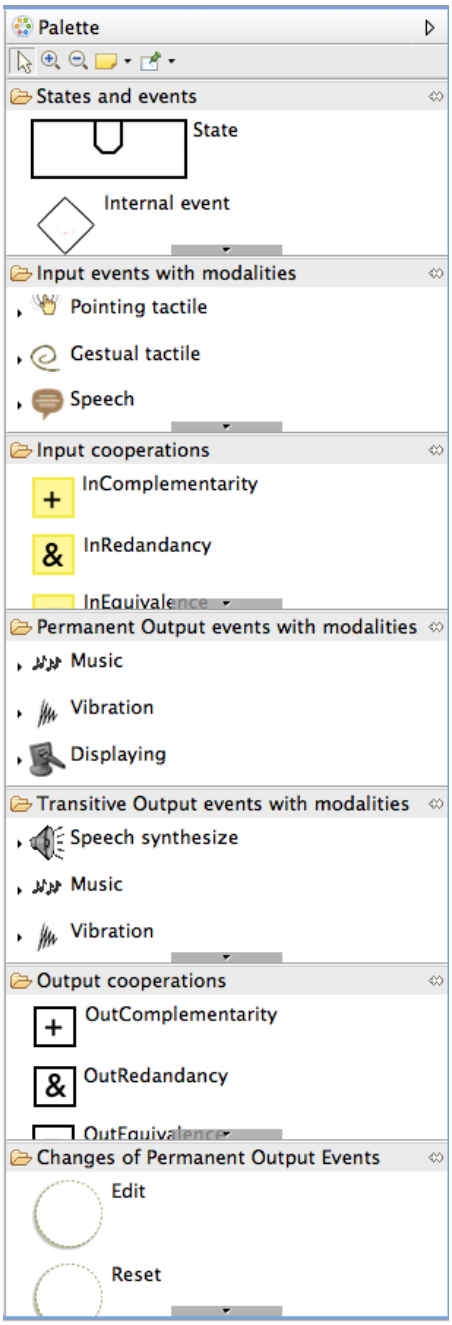


FIGURE 5.25 – Les sections de la palette de modélisation

Cette bibliothèque fournit les différents types d'évènements d'interaction qui peuvent être utilisés par un simple glisser-déposer à partir de la palette de l'environnement de modélisation de MIMIC [36]. Elle permet non seulement aux concepteurs d'identifier les types d'évènements possibles, mais leur fournit aussi des modèles réutilisables modélisant ces évènements.

Nous avons défini les éléments de la bibliothèque en deux étapes. Premièrement, nous avons identifié et testé les différents types d'interaction basés sur chaque capteur en entrée ou en sortie. Puis, en deuxième étape, nous avons préparé des modèles prêts à l'emploi qui modélisent les évènements utilisant ces différents types d'interaction. Pour créer ces modèles, nous avons adopté le principe des modèles paramétrables (des templates de modèles) qui n'encapsulent pas de code, mais créent des instances avec des propriétés pré-remplies de certains concepts et associations. La bibliothèque contient actuellement 34 types d'évènements en entrée et 54 en sortie (dont 50 sont des widgets). Bien sûr, elle n'est pas exhaustive et d'autres types peuvent être intégrés.

5.1.3.1 Types d'évènements en entrée

Dans le chapitre 2 (tableau 2.3), nous avons donné une description complète de capteurs sur mobile (pour l'année 2013-2014). Nous avons aussi listé quelques types d'interaction qui peuvent être créés à base de ces capteurs (les « types d'évènements »). Pour créer notre bibliothèque, nous avons commencé par l'implémentation et le test de ces types d'interaction. L'implémentation était principalement sous Android, car c'est la plateforme la plus répandue. Le but était surtout de vérifier la faisabilité de ces types d'interaction et de récupérer les contraintes de leurs utilisations du point de vue utilisateur (indépendamment de la plateforme). Ces contraintes d'utilisation (pas encore validées par des expérimentations avec utilisateurs finaux) nous ont aidé, plus tard, pour la définition des contraintes de modélisation et de vérification des modèles (voir la section 5.3.2). Le tableau 5.3 présente les différents types d'interaction en entrée que nous avons identifiés, ainsi que leurs notations visuelles associées.

Type en entrée	**Notation**	**Modalité**	**Combinaisons difficiles**	**Contraintes d'utilisation**
« Touch »		Pointage tactile	La complémentarité avec le vocal et le secouage nécessite une fenêtre temporelle importante, la complémentarité est impossible avec la chute libre	Utilisation des gants, utilisation des deux mains (si une ou deux sont occupées), la pluie
« Long Touch »		Pointage tactile	Même chose que « Touch »	Nécessite un retour pour arrêter (généralement une petite vibration)

« Gestural touch »		Tactile gestuel	Nécessite une augmentation de la fenêtre temporelle pour les différentes complémentarités	Nécessite un retour visuel, la reconnaissance n'est pas toujours correcte
« Gestural Multi-touch »		Tactile gestuel	Complémentarité impossible car les deux mains sont utilisées	Il faut utiliser les deux mains
Vocal		Parole	Redondance avec le tactile nécessite une fenêtre temporelle importante	Nécessite une interaction préliminaire pour être déclenché, reconnaissance pas toujours correcte
Musique (Enregistrement)		Musique	Complémentarité et redondance impossibles	Nécessite une interaction préliminaire pour être déclenché
Scan QR code		Scan (mode gestuel)	Complémentarité et redondance impossibles	Nécessite une interaction préliminaire pour être déclenché
Secouage		Accélération	Combinaison difficile avec le vocal (sauf en équivalence)	Impossible de voir l'écran au moment du secouage, nécessite un retour pour arrêter (autre que l'affichage)
Chute libre		Accélération	Impossible de coopérer en complémentarité ou redondance	Impossible de voir l'écran au moment de la chute, nécessite l'utilisation des deux mains, risque de chute réelle

Accélération vers la droite Accélération vers la gauche Accélération vers le haut Accélération vers le bas		Accélération	Combinaison difficile avec le vocal (sauf en équivalence)	Vue partielle de l'écran, mouvement de la main difficile pour certaines personnes
Orientation vers le nord Orientation vers le sud Orientation vers l'est Orientation vers l'ouest		Orientation	Peu d'opportunité de les utiliser en équivalence avec d'autres interactions	Nécessite plus de précision

Orientation vers le haut Orientation vers le bas Orientation vers la droite Orientation vers la gauche		Orientation	Combinaison difficile avec le vocal (sauf en équivalence)	Vue partielle de l'écran, mouvement de la main difficile pour certaines personnes
« Blackout »		Éclairage		Il faut généralement utiliser les deux mains - Peut être confondu avec le tactile
« Illumination »		Éclairage	Peu d'opportunité de l'utiliser en combinaison avec d'autres interactions	Difficile à effectuer (surtout à l'extérieur), il faut utiliser les deux mains
« S'approcher »		Proximité		Peut être confondu avec le tactile et surtout avec l'éclairage -Généralement, il faut utiliser les deux mains
« S'éloigner »		Proximité	Peu d'opportunité de l'utiliser en combinaison avec d'autres interactions	Pas toujours vue comme une interaction
« Positionnement »		Localisation	Très utile en concurrence et peu dans les autres combinaisons	Nécessite l'acceptation de l'utilisateur

TABLE 5.3 – Les types d'interaction en entrée

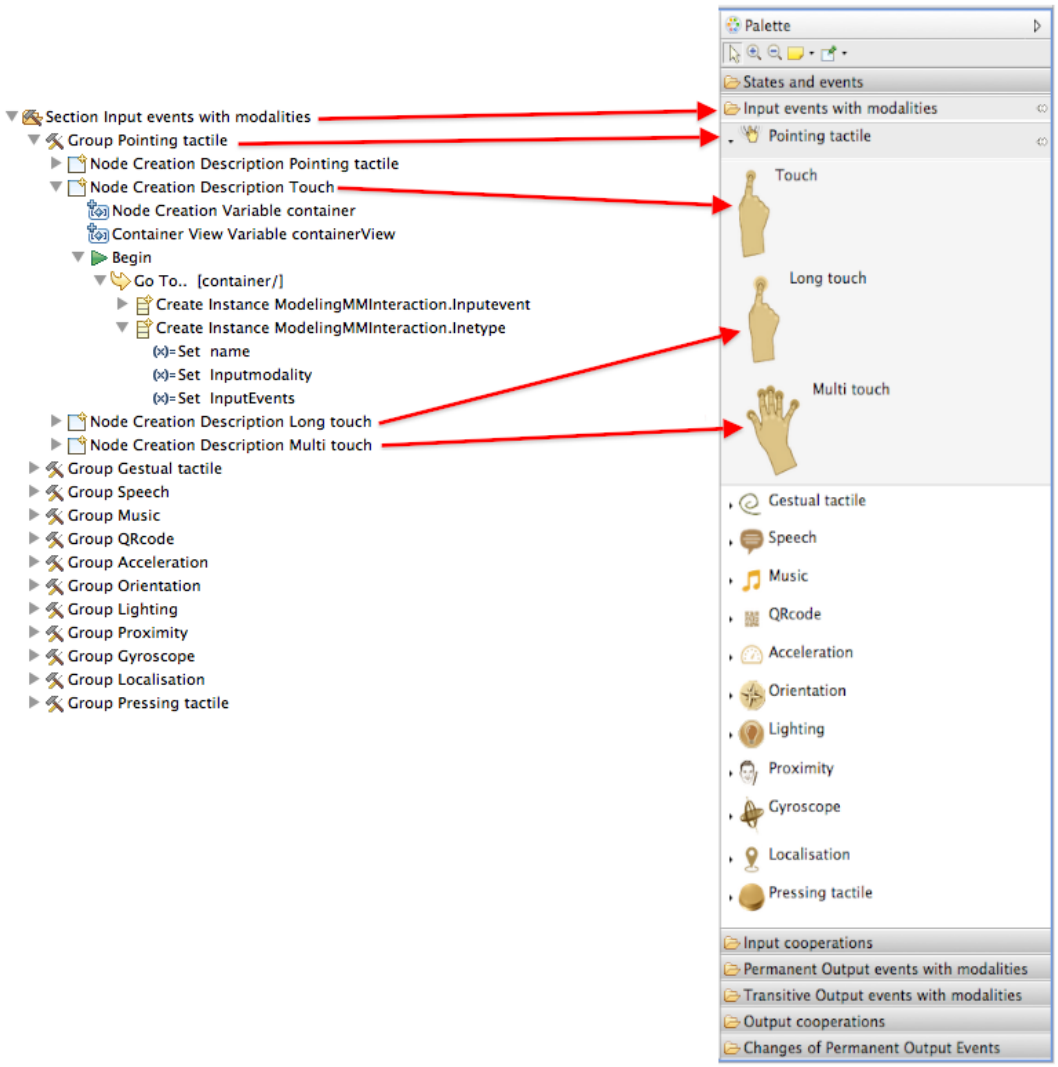


FIGURE 5.26 – La section des évènements en entrée dans la palette

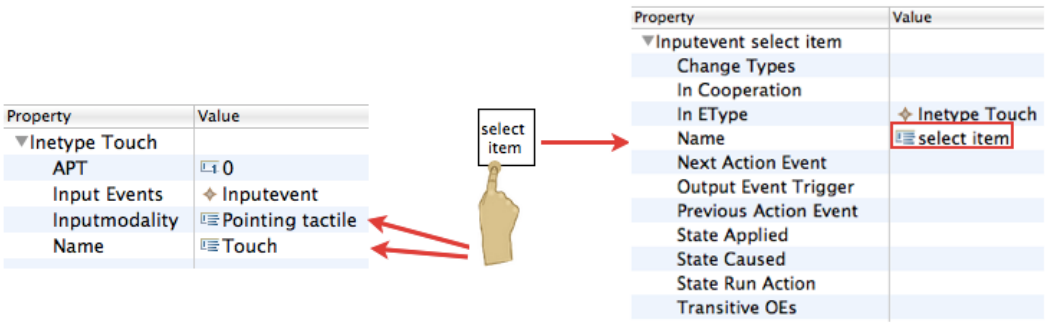


FIGURE 5.27 – Un exemple d'un évènement « Touch » en entrée

Après l'étape d'identification, nous avons ajouté les modèles paramétrables des types d'évènements dans la deuxième section de la palette (section pour les différents types d'évènements en entrée) (voir figure 5.26). Nous les avons classés par groupe de modalités d'interaction, c'est-à-dire que chaque groupe contient les types d'évènements d'une seule modalité. Ainsi, les concepteurs peuvent déterminer rapidement les modalités lors du choix des évènements pour la modélisation. L'action qui sera effectuée lors de l'utilisation d'un de ces types d'évènements dans un modèle consiste à créer un modèle paramétrable qui instancie deux concepts et une association : « Input event », « Input type » et l'association entre les deux. Les deux instances des concepts seront reliées avec l'instance de l'association et les attributs de l'instance du type d'évènement seront remplis par le nom de l'évènement et la modalité d'interaction associée (figure 5.27).
Après la sélection d'un élément de la palette, le concepteur aura la possibilité de remplir les attributs de l'instance de l'événement en entrée. Par exemple, si un concepteur sélectionne le type « Touch » à partir de la palette et l'ajoute dans le modèle, deux instances seront créées : une instance d'évènement (le carré dans la figure 5.27) et une instance du type (l'icône de la main dans la figure 5.27).

5.1.3.2 Types d'évènements en sortie

De même que pour les évènements en entrée, nous avons aussi implémenté et testé les interactions en sortie afin de vérifier leur faisabilité et de récupérer les contraintes de leur utilisation du point de vue utilisateur. Le tableau 5.4 présente les différents types d'interaction en sortie que nous avons identifiés, leurs notations visuelles associées et les contraintes de leur utilisation.

Type en sortie	**Notation**	**Modalité**	**Combinaisons difficiles**	**Contraintes d'utilisation**
« Musique »		Musique	Combinaison difficile avec la synthèse vocale	Pas utile dans des environnements bruyants, nécessite un volume éleve
Vibration courte		Vibration	/	Peut être imperceptible
Vibration longue		Vibration	/	

Synthèse vocale		Synthèse vocale	Combinaison difficile avec la musique (surtout en concurrence)	Généralement il faut afficher le texte associé (en redondance), la prononciation n'est pas toujours correcte
Widgets		Affichage	/	L'affichage nécessite de l'ergonomie

TABLE 5.4 – Les types d'interaction en entrée

Nous avons ajouté les modèles paramétrables des types d'évènements en sortie dans deux sections de la palette : la section pour les types d'évènements permanents et la section pour les types d'évènements transitoires. Comme pour les types d'évènements en entrée, ces types ont été classés par modalités d'interaction dans les deux sections (figure 5.28). L'action, qui sera effectuée lors de l'utilisation d'un de ces types dans un état d'interaction, consiste à créer un modèle paramétrable qui instancie deux concepts et une association : « Output event », « Output type » et l'association entre les deux. Les deux instances des concepts seront reliées avec l'instance de l'association et les attributs de l'instance du type d'évènement seront remplis par le nom de l'évènement et la modalité d'interaction associée (figure 5.29).

Après la sélection d'un élément, le concepteur aura la possibilité de remplir les attributs de l'instance de l'événement en sortie. Par exemple, si un concepteur sélectionne le type « Short vibration » et l'ajoute dans un état de son modèle, deux instances seront créées : une instance d'évènement (le rectangle dans la figure 5.29) et une instance du type (l'icône de vibration dans la figure 5.29). L'instance du type est déjà prête (le nom « Short vibration » et modalité « Vibration »), tandis que l'évènement nécessite l'intervention du concepteur pour le paramétrer (il ajoute le nom « vibrate » par exemple). Les deux instances sont reliées par l'association afin de spécifier que l'évènement « vibrate » est de type vibration courte (« In EType » = « Short vibration ») (l'association est définie graphiquement par le fait que le type est affiché au bord de l'évènement).

Dans cette section, nous avons défini la partie graphique (visuelle) de notre approche ainsi que la bibliothèque des types d'évènements d'interaction en entrée et en sortie. Nous reconnaissons que la définition d'une notation visuelle qui respecte les différents critères de physique des notations n'a pas été facile (un exemple avec les différentes notations est présenté dans l'annexe A).

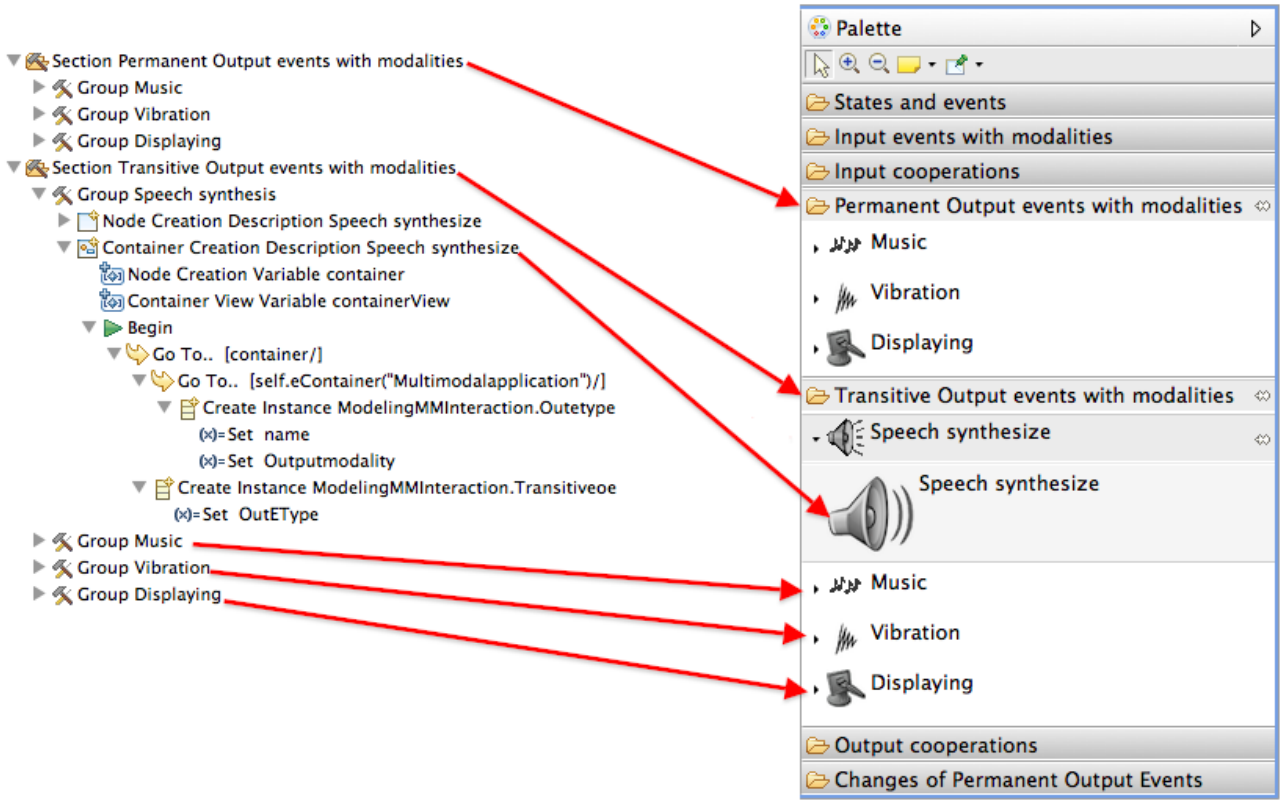


FIGURE 5.28 – Les sections des types d'évènements en sortie dans la palette

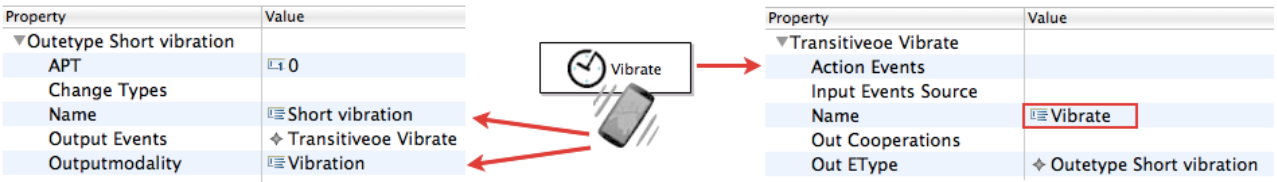


FIGURE 5.29 – Un exemple d'un évènement transitoire de vibration en sortie

La création de la bibliothèque n'était pas simple non plus, surtout lors de l'identification des différents types. Cette bibliothèque reste ouverte pour l'intégration d'autres types d'évènements basés sur les anciens ou les nouveaux capteurs.

Dans la section suivante, nous allons présenter les générateurs de code de notre approche. Ces générateurs se basent sur les modèles graphiques pour générer le code sous les différentes plateformes.

5.2 Générateurs de code

Pour la génération de code, nous avons commencé par la spécification des règles de passage des modèles sources (les modèles des applications mobiles multimodales définis avec M4L) aux modèles cibles (les applications Android, iPhone et HTML5/CSS3). Puis, nous avons choisi l'approche par template pour implémenter ces règles [12]. Celle-ci consiste à prendre un modèle template et à remplacer ses paramètres par les informations du modèle source pour définir un modèle cible. Ce choix était surtout guidé par l'outillage (Obeo Designer), car nous avons utilisé Acceleo pour la génération de code et il se base sur les templates.

5.2.1 Le générateur de code Acceleo

Acceleo est un produit open source créé par la société Obeo en 2006. Il fait aussi partie de la fondation Eclipse depuis 2009. Acceleo permet la génération de code à partir des modèles graphiques basés sur EMF (Eclipse Modeling Framework). Pour la génération, il propose un éditeur de templates sous Eclipse. Ces templates génèrent tous les types de texte (code, XML, etc.) après leur instantiation/exécution (spécification des vraies valeurs dans les « trous ») (voir l'exemple de la figure 5.30).

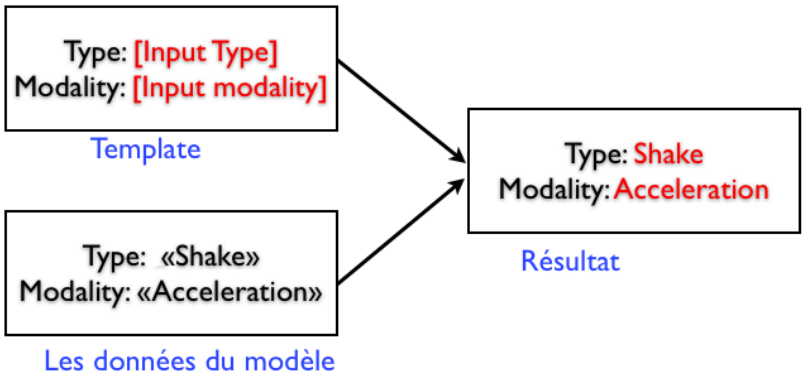


FIGURE 5.30 – Exemple d'utilisation des templates pour la génération de code

Les templates sous Acceleo sont écrits en MTL (MOF Model to Text Transformation Language) qui est un langage standardisé par l'OMG (Object Management Group) pour générer du texte à partir de modèles définis par des méta-modèles. L'utilisateur d'Acceleo commence d'abord par la création d'un projet Acceleo et la spécification du méta-modèle associé à ce projet (à travers un URI). Puis, l'utilisateur passe à la création des modules qui

correspondent à des fichiers « .mtl » et qui contiennent les templates de génération. Il écrit le code MTL pour analyser les modèles issus du méta-modèle spécifié et génère le code correspondant. La génération se fera sous forme de fichiers (l'extension doit être précisée par le développeur) après l'exécution du projet Acceleo.

Nous avons choisi Acceleo pour la création de nos générateurs pour deux raisons. Premièrement, c'est aussi un produit de la société Obeo et il convient bien pour la transformation des modèles créés sous Obeo Designer (il est intégré dans le bundle d'installation d'Obeo Designer). Deuxièmement, il se base sur le standard MTL contrairement aux autres générateurs (JET (Java Emitter Templates), XPand, etc.) qui se basent sur leurs propres langages.

5.2.2 La génération des combinaisons de modalités (fusion/fission)

Dans le chapitre 4 (section 2), nous avons présenté les différents concepts de notre langage de modélisation. Parmi ces concepts, nous avons identifié quatre natures de combinaison entre modalités en entrée et en sortie : complémentarité, redondance, équivalence et concurrence. Le choix de ces combinaisons était basé sur les propriétés CARE et TYCOON. L'assignation et le transfert n'ont pas été considérés, car ils ne traduisent pas de forme de combinaisons. L'assignation exprime l'absence de choix entre modalités [15] tandis que le transfert exprime le passage d'information entre modalité en entrée ou en sortie. Ces opérations peuvent être exprimées facilement dans les modèles en utilisant les différents concepts du langage et sans avoir besoin d'opérateurs particuliers de combinaison.
Certaines combinaisons nécessitent un traitement algorithmique pour pouvoir être utilisées sur mobile, tandis que d'autres n'en nécessitent pas. Dans cette section, nous allons présenter les algorithmes que nous avons utilisés pour gérer les différentes combinaisons.

5.2.2.1 En entrée (fusion)

En entrée, les relations InC (complémentarité) et InR (redondance) nécessitent un traitement algorithmique. Elles possèdent toutes les deux les attributs suivants (définis dans le méta-modèle de M4L) :

- Fenêtre temporelle : définit l'intervalle de temps nécessaire pour combiner les modalités (en millisecondes). Le concepteur peut la définir lui-même, sinon elle sera définie par défaut.
- Ordre : définit l'ordre chronologique que l'utilisateur doit suivre lors de combinaison des modalités. C'est un booléen « vrai » par défaut, mais le concepteur peut le changer

pour ne pas prendre en compte l'ordre d'emploi des modalités.

Ces attributs sont nécessaires pour personnaliser l'algorithme de fusion selon les besoins de chaque combinaison. L'algorithme de fusion que nous avons utilisé fusionne les données au niveau décision/sémantique (après récupération des informations à partir des entrées) et suivant une fusion basée sur les frames (frame-based fusion) [72].

Complémentarité. Pour assurer la complémentarité entre deux ou plusieurs interactions, les générateurs de MIMIC génèrent le code qui permet leur fusion. Il fusionne les données récupérées à partir de ces interactions pendant un laps de temps égal à la fenêtre temporelle. Le temps de cette fenêtre temporelle commence à se décompter dès la réalisation de la première interaction participante à la complémentarité. Dans l'exemple de la figure 5.31, le secouage du téléphone et le masquage du capteur de lumière doivent être réalisées rapidement pour permettre leur fusion. Le code de fusion commence par récupérer les données de la première interaction effectuée (« Blackout » par exemple) et attend la deuxième. Si cette dernière (le secouage) arrive pendant la fenêtre temporelle, les deux interactions seront fusionnées.

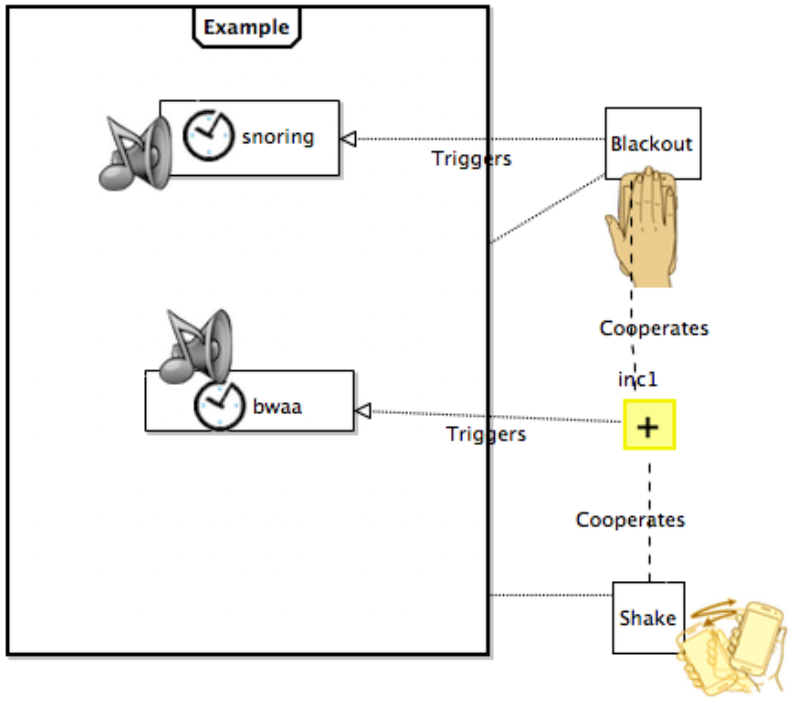


FIGURE 5.31 – Exemple d'une complémentarité entre deux interactions en entrée

Pour chaque complémentarité, une structure de fusion est créée (« frame »). Elle contient des emplacements pour les différentes interactions participant à la complémentarité (ces em-

placements sont appelés « Slots »). Chaque emplacement stocke les données reçues d'une interaction particulière. Si nous reprenons l'exemple de la figure 5.31, la structure de fusion contiendra deux emplacements (2 slots), un pour stocker le « Shake » et l'autre pour le « Blackout ». Les données sont stockées après leurs interprétations. Par exemple, les données stockées pour le « shake » ne sont pas les valeurs x, y et z de l'accéléromètre, mais plutôt l'évènement issu de ces valeurs (une structure qui indique le nom de l'évènement « Shake », son type « shaking » et le temps de son arrivée). Pour fusionner, l'algorithme suit les règles suivantes :

- À la fin de la fenêtre temporelle, si la structure est complète, une notification est envoyée aux observateurs (le patron de conception observateur/observable) qui vont effectuer le traitement (l'effet) de la complémentarité (le déclenchement du son « bwaa » dans l'exemple de la figure 5.31).
- Une fois la notification envoyée, les données seront supprimées du frame.
- Si à la fin de la fenêtre temporelle, la structure n'est pas complète, une notification est envoyée pour indiquer qu'il y a un ou plusieurs évènements qui ont été effectués sans déclencher une complémentarité. La ou les classes observateurs vont effectuer l'effet de chaque évènement séparément (s'ils ont un effet, comme le « Blackout » de la figure 5.31 - effet de déclenchement de son « snoring »-).
- Les données dans les structures incomplètes sont aussi supprimées après la notification.

Redondance. L'algorithme de redondance est très proche de l'algorithme de complémentarité. Les structures de fusion contiennent les données redondantes et l'algorithme suit presque les mêmes règles que pour la complémentarité. Cependant, si la structure de fusion n'est pas complète à la fin de la fenêtre temporelle, les évènements qui ont été réalisés ne peuvent pas déclencher leurs propres effets.
Sur mobile on ne trouve pas beaucoup d'exemples pour la redondance en entrée, car elle est trop contraignante. L'exemple le plus utilisé sur PC est l'utilisation du clavier/souris en redondance avec le vocal. Cet exemple peut être repris sur mobile en remplaçant le clavier/souris par l'écran tactile. Cependant, la fenêtre temporelle doit être importante, car la reconnaissance vocale nécessite un temps de traitement parfois important et ne permet pas le tactile en parallèle (cela commence à changer avec certains téléphones).

Équivalence. L'équivalence nécessite aussi un traitement algorithmique, mais il n'est pas aussi important que celui de la complémentarité et de la redondance. Ce traitement consiste à préciser que chaque interaction qui participe à une équivalence doit déclencher/effectuer

les effets de cette équivalence. Par exemple, dans la figure 5.32, les interactions « Shake » et « Press » sont équivalentes. Le code généré de leur équivalence va donc préciser que l'application d'un secouage du téléphone « Shake » ou d'un « Touch » (Press) sur le bouton « Validate » permettra le passage vers le deuxième état « Liste note » (l'effet de l'équivalence « Equivalence 1 »). La première interaction appliquée sera celle qui déclenchera le passage vers le deuxième état.

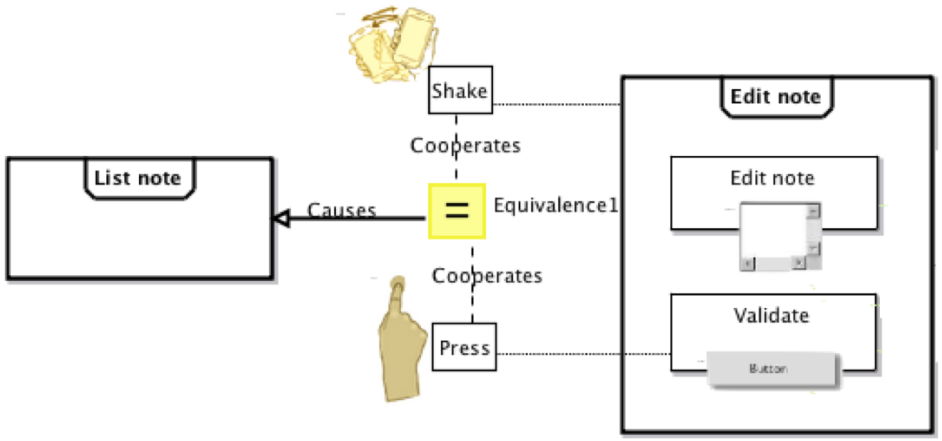


FIGURE 5.32 – Exemple d'une équivalence entre deux interactions en entrée

Concurrence. La particularité de la concurrence par rapport aux autres combinaisons est qu'elle ne possède pas d'effet sur l'interaction. Les évènements concurrents sont généralement indépendants les uns des autres et ne peuvent donc pas avoir des effets communs. Cependant, ces évènements arrivent en même temps et sont exécutés en concurrence. Sur mobile, plusieurs interactions peuvent être exécutées en concurrence sans intervention du développeur. Le système d'exploitation prend en charge le traitement de la concurrence ainsi que la gestion de leur parallélisme. L'un des exemples le plus utilisé pour la concurrence est la récupération des coordonnées GPS au moment où l'utilisateur utilise le pointage tactile pour saisir des données (figure 5.33). L'icône de concurrence n'est donc pas obligatoire dans le modèle, car elle ne permet pas de générer un code particulier. Cependant, il est très utile pour simplifier la lecture du modèle.

Les générateurs de code dans MIMIC, génèrent le code nécessaire pour effectuer les différentes combinaisons en entrée. Ils permettent aussi au développeur d'ajouter du code dans les fichiers générés. Pour faire cela, il utilise des blocs de code utilisateur qui délimitent le texte ajouté et le conservent pour les générations suivantes. Ceci est utile pour permettre à l'utilisateur d'ajouter du code tout en maintenant le reste du fichier sous le contrôle du générateur.

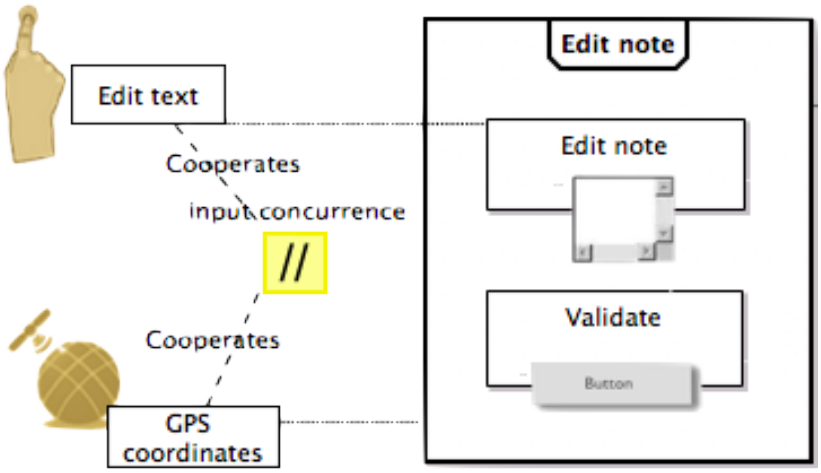


FIGURE 5.33 – Exemple d'une concurrence entre deux interactions en entrée

5.2.2.2 En sortie (fission)

En sortie, notre but n'était pas forcément de générer que des interfaces intelligentes qui prennent en compte l'environnement et/ou l'utilisateur et proposent des modalités selon les besoins (adaptation au contexte). Nous visions plutôt la génération des interfaces selon la modélisation du concepteur. Ainsi, si ce concepteur utilise des capteurs pour la détection des données sur l'environnement en entrée, il peut définir des interfaces en sortie qui déclenchent des adaptations automatiques (capteur de lumière dans la figure 5.34 par exemple). Toutefois, ce concepteur peut aussi modéliser et générer des interfaces multimodales en sortie qui ne proposent pas d'adaptations automatiques et qui nécessitent l'intervention de l'utilisateur pour choisir les modalités selon ses besoins. Certains chercheurs considèrent les interfaces qui utilisent plusieurs modalités en sortie sans adaptation au contexte comme des interfaces multimédias et pas multimodales [95]. Cependant, nous les considérons plutôt multimodales, car leur définition ne s'oppose pas à la définition de la multimodalité.

En sortie, nous modélisons les combinaisons suivantes : OutC (complémentarité), OutR (redondance), OutE (équivalence) et OutCC (concurrence). Toutefois, seules la redondance et l'équivalence nécessitent des traitements spécifiques (fusion).

Redondance. Le traitement de la redondance en sortie consiste à assurer la diffusion des mêmes données par deux ou plusieurs modalités participant à la redondance. Dans la figure 5.34, par exemple, la fission consiste à récupérer le texte de la « TextView » et à le transmettre à la synthèse vocale « Reading text » pour le faire prononcer. Ainsi, si le capteur de lumière

détecte qu'il y a beaucoup de lumière, il déclenche la synthèse vocale. La vérification de la compatibilité des interactions redondantes se fait au niveau modélisation (voir section 5.3.2).

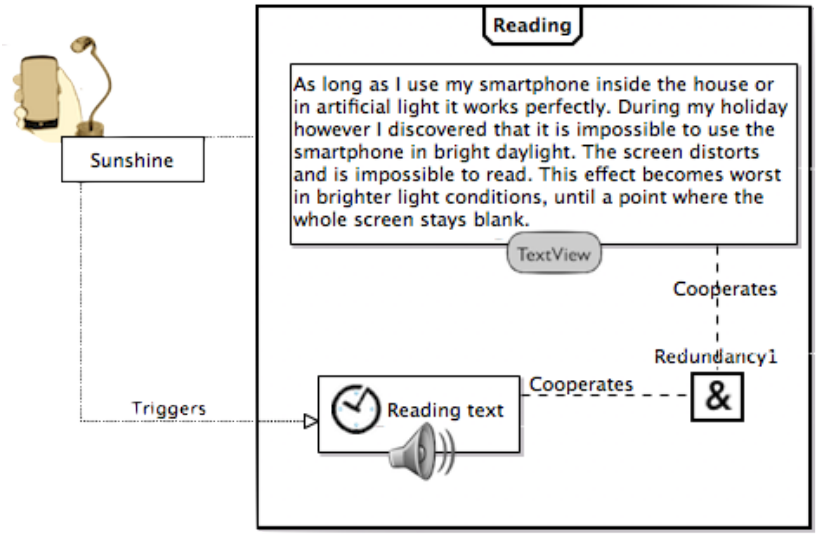


FIGURE 5.34 – Exemple d'une redondance entre deux évènements d'interaction en sortie

Équivalence. Il est impossible de définir une équivalence sans prendre en compte l'adaptation au contexte. Derrière l'équivalence il y a un choix que l'application doit faire entre les évènements d'interaction participants. Le choix entre ces évènements qui diffusent les mêmes données est effectué en fonction des informations qui sont généralement relatives à l'environnement et/ou l'utilisateur. Ainsi, c'est le concepteur qui doit définir les conditions d'utilisation des différents évènements. Par exemple, il précise que si le capteur de lumière détecte qu'il n'y a pas beaucoup de lumière dans l'environnement, il suffit d'afficher un texte, sinon il faut déclencher la synthèse vocale et cacher le texte. Le code généré dans ce cas effectuera l'équivalence selon la description du concepteur. Sinon, si le concepteur ne précise pas comment le choix doit être fait, les générateurs de code génèrent automatiquement toutes les interactions participantes à l'équivalence.

Complémentarité et concurrence. La complémentarité et la concurrence en sortie peuvent être assurées sans génération de traitements particuliers. Le concepteur modélise la complémentarité en précisant les évènements complémentaires tels que la synthèse vocale d'une question et l'affichage de deux boutons pour le choix de réponse (figure 5.35) ou l'affichage d'une notification avec une vibration pour indiquer son arrivée. De même, pour la concurrence en sortie, le concepteur définit les interactions concurrentes telles que l'affichage d'un texte avec une musique d'accompagnement, mais sans lien avec le texte. Puis, lors de la gé-

nération, les différents évènements d'interaction seront générés indépendamment (les liens entre eux sont plutôt sémantiques). Les icônes de concurrence et de complémentarité ne sont donc pas obligatoires dans les modèles puisqu'ils ne permettent pas de générer un code. Cependant, ils sont très utiles pour la lecture du modèle.

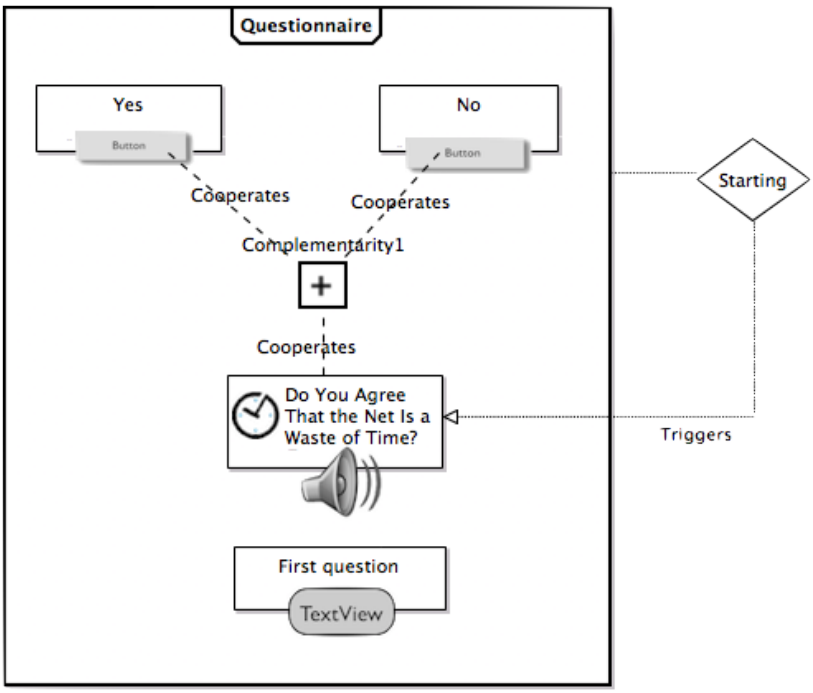


FIGURE 5.35 – Exemple d'une complémentarité entre trois évènements d'interaction en sortie

Les exemples de combinaison en sortie sont limités, car le nombre de modalités d'interaction en sortie est limité par rapport à l'entrée. De plus, certains évènements et modalités d'interaction sont déconseillés, car ils soulèvent des problèmes (contraintes d'utilisation tableau 5.4). Par exemple, les modalités sonores se basent sur le volume qui peut être réglé à zéro par l'utilisateur sans que l'application mobile en soit informée. Le bruit de l'environnement utilisateur peut aussi être élevé ce qui ne permet pas d'écouter l'évènement sonore correctement [108].

5.2.3 Générateur pour Android

Le premier générateur de code que nous avons créé était utilisé pour générer du code sur la plateforme Android. Pour faciliter l'implémentation de notre générateur, nous avons commencé par la définition des règles de génération.

Puis, nous avons implémenté le générateur avec Acceleo et réalisé plusieurs exemples d'applications Android multimodales afin de vérifier son efficacité. Certaines de ces applications ont été publiées sur Google Play.

5.2.3.1 Les règles de génération Android

Les règles de génération ou de transformation de modèle en code consistent à décrire les correspondances entre un (ou plusieurs) élément(s) du modèle source et une (ou plusieurs) instruction (s) du code. Les règles que nous avons définies pour la transformation des modèles en code pour la plateformes Android sont les suivantes :

- Chaque état d'interaction dans un modèle est transformé en une activité Android (une classe Java qui hérite de Activity).
- Le code des interactions en entrée et en sortie de chaque état est généré dans l'activité Android associée.
- Un layout (fichier XML) est également associé à chaque état d'interaction.
- Le code de positionnement et des styles des évènements affichés (les widgets) dans chaque état est généré dans le layout associé.
- Un fichier appelé « AndroidManifest.xml » est généré pour chaque modèle d'application (pour chaque application générée).
- Le manifest présente les informations essentielles sur l'application telles que les activités de l'application, les permissions nécessaires, etc.
- Des classes Java peuvent être générées pour la gestion des combinaisons (si nécessaire).

Le générateur Android suit donc ces règles pour générer des applications mobiles multimodales à partir des modèles. Le concepteur doit configurer le générateur en lui précisant le modèle de l'interface à générer ainsi que le projet Android qui va contenir les fichiers générés (les classes Java dans le répertoire « src », les fichiers de layout et menu dans le répertoire « res » et le fichier de AndroidManifest dans le répertoire racine du projet). Le projet Android peut être créé dans la même instance Eclipse de l'environnement de modélisation (après l'installation du plugin Android). Ainsi, le concepteur/développeur n'a pas besoin de changer d'environnement pour pouvoir tester l'application générée.

5.2.3.2 Générateur des « activités », « layouts », « manifests » et « menus »

Le générateur de code pour la plateforme Android est décomposé en quatre sous-générateurs :

Générateur du fichier « AndroidManifest ». Ce générateur est un module (contient un seul template) qui prend en entrée un modèle d'interface mobile multimodale et génère un fichier XML appelé « AndroidManifest » pour la description de l'application de cette interface. C'est un fichier que chaque application Android doit contenir. Il présente les informations essentielles de l'application au système d'exploitation pour qu'il puisse l'exécuter correctement. Les données essentielles générées sont les suivantes :

- Le nom du package Java de l'application Android. Il représente l'identifiant unique de l'application. Le générateur le définit à partir du nom de l'application choisi par le concepteur lors de la modélisation. S'il choisit le nom « My Application » par exemple, le package sera nommé « com.MyApplicationmma » (le « mma » à la fin est pour « Multimodal Mobile Application »). Les espaces et les caractères spéciaux sont supprimés du nom choisi par le concepteur.
- Le niveau minimum de l'API Android nécessaire pour exécuter l'application (minSdkVersion). Nous avons choisi par défaut le niveau 8 (Android 2.2.x).
- Le niveau d'API Android que l'application vise (targetSdkVersion). Là aussi nous avons choisi le niveau 11 (Android 3.0.x) par défaut. Cependant, les applications générées peuvent aussi fonctionner sous des versions supérieures d'Android.
- L'icône de l'application (android :icon) ainsi que le label qui sera affiché en bas de l'icône après installation (android :label). Pour l'icône le générateur utilise une de celles qui se trouvent par défaut dans le projet Android (répertoire « drawable ») et pour le label il utilise le nom de l'application choisi par le concepteur dans le modèle (cette fois on ne retire pas les espaces et les caractère spéciaux).
- Les activités de l'application. Le générateur déclare tous les états dans le modèle comme des activités. Il précise aussi la première activité ou le « launcher » (l'état dont l'attribut « First » est égal à « true »). Au cas où le premier état n'est pas précisé dans le modèle, le générateur considère que le premier état modélisé sera la première activité. Pour chaque activité, le générateur précise le nom (le nom de l'état dans le modèle avec suppression des espaces et caractères spéciaux) et le label (le nom de l'état sans modification).
- Les « permissions » grâce auxquelles l'application peut accéder à des parties protégées de l'API Android (l'utilisation de vibration, GPS, Internet, NFC, enregistrement de son, etc.). Pour déclarer ces permissions, le générateur cherche les interactions et actions qui nécessitent des permissions et les précise une par une. Les permissions qui sont utilisées par plusieurs interactions/actions ne sont déclarées qu'une seule fois.
- Les bibliothèques que l'application utilise doivent aussi être déclarées telles que par

exemple la bibliothèque « com.google.android.maps » pour l'utilisation des « Map View ». Le générateur déclare les bibliothèques de la même façon que les permissions.

Après la génération du fichier « AndroidManifest.xml » pour un modèle particulier, le concepteur/développeur peut ajouter du code dans les blocs protégés générés dans le même fichier. Le code ajouté ne sera pas écrasé lors de la re-génération du fichier. Cependant, généralement, les utilisateurs n'ont rien à ajouter dans le « AndroidManifest », car tout est déjà correctement généré.

Générateur des fichiers Layouts. Les layouts sont des fichiers XML qui décrivent les mises en page (structure visuelle, positionnement des widgets, etc.) des interfaces pour les différentes activités. Chaque layout doit avoir un élément racine qui peut être un « View » ou un « ViewGroup ». Les widgets et tous les éléments graphiques doivent être définis entre les balises de début et de fin de cet élément racine. Le générateur des layouts contient deux modules : un pour la génération des layouts correspondant aux activités et un autre pour la génération des layouts correspondant aux évènements transitoires personnalisés qui doivent être affichés.

Le premier module contient un template qui permet de générer un layout pour chaque état. Chaque layout est nommé par « layout_ » concaténé avec le nom de l'état (sans caractères spéciaux, en transformant les lettres majuscules en minuscules, et en remplaçant les espaces par « _ »). Il contient les éléments suivants :

- L'élément racine : par défaut nous avons le « ScrollView » (élément qui peut contenir des vues hiérarchiques et que l'utilisateur peut faire défiler) suivi d'un « LinearLayout » (élément qui organise les vues dans un ordre vertical ou horizontal). Cependant, dans le cas où l'interface contient des listes (ListView, ExpandableListView, etc.) ou des vues particulières comme « MapView » ou « GestureOverlayView », le défilement ne fonctionne pas bien, car ces éléments ont leurs propres mécanismes de défilement. Ainsi, pour résoudre ce problème, le générateur commence par chercher s'il y a un élément qui pose un problème de défilement. S'il y a au moins un élément de ce type, il retire l'élément « ScrollView » de la racine (il ne reste que « LinearLayout »).
- Les widgets de l'interface sont ajoutés entre les balises de début et de fin de la racine. La définition des widgets se fait en deux étapes. La première consiste à sélectionner les évènements permanents en sortie de l'état ayant la modalité affichage tandis que la deuxième consiste à déclarer ces évènements selon leurs types. La déclaration se fait par la définition du type d'évènement, son nom, ainsi qu'un ensemble de propriétés que le générateur définit pour chaque type. Certaines de ces propriétés sont spécifiées

par défaut tandis que la plupart d'entre elles sont spécifiées à partir du modèle. Par exemple, pour un évènement nommé « Validate » de type « Button », le générateur commence par spécifier le type et le nom de l'évènement. Puis, il spécifie les propriétés du bouton telles que la hauteur, la largeur, l'orientation par rapport à l'écran, la couleur du texte et du fond ainsi que le texte à afficher à l'intérieur du bouton. Le générateur cherche ces propriétés dans le modèle (les propriétés de l'évènement en sortie « Validate »), mais si le concepteur ne les a pas spécifiées, il met des valeurs par défaut (figure 5.36).

```
<Button
    android:id="@+id/Validate"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#FFF"
    android:orientation="horizontal"
    android:text="Validate"
    android:textColor="#000"
    android:textSize="21sp" />
```

FIGURE 5.36 – Exemple de déclaration d'un bouton dans un layout

- Au cas où aucun widget n'est défini dans un état, un « TextView » est ajouté par défaut afin d'assurer le fonctionnement de l'activité associée à cet état.
- Après la déclaration des widgets, le générateur ajoute un bloc protégé pour le code utilisateur au cas où le développeur souhaite ajouter d'autres déclarations qui ne sont pas précisées dans le modèle.

Le deuxième module contient un template qui permet de générer un layout pour chaque évènement transitoire en sortie avec un affichage personnalisé. Chaque layout contient les éléments suivants :

- L'élément racine, cette fois, est par défaut « RelativeLayout », car il est très flexible et permet de positionner chaque widget par rapport aux positions des autres widgets proches.
- Les widgets internes sont ajoutés entre les balises de début et fin de la racine. Chaque widget est déclaré avec son nom, type et propriétés (orientation, couleur, texte, source pour les images, etc.). Généralement, les widgets sont limitées à des boutons, des zones de texte ou d'édition et des images.
- Le générateur ajoute là aussi des blocs protégés pour que le développeur puisse éventuellement ajouter du code.

Les layouts correspondant aux états ainsi qu'aux évènements transitoires personnalisés

seront tous générés dans le répertoire « layout » du projet Android. Puis, ils seront reliés aux activités Java de l'application afin de spécifier leurs interfaces graphiques. Nous reconnaissons que les interfaces générées ne sont pas (toujours) jolies visuellement et optimisées graphiquement. Les widgets sont affichés les uns après les autres (dans le même « LinearLayout ») à gauche de l'écran (si le modèle ne précise pas la position), et les évènements permanents tels que les listes ou galeries de photos ne peuvent pas être personnalisés à partir du modèle (et donc ne peuvent pas être générés automatiquement). Ces manques sont relatifs au fait que la modélisation et la génération des interfaces graphiques optimisées sont des problèmes complexes et vastes [6]. Leur résolution nécessit beaucoup de travaux qui dépassent le cadre de cette thèse.

Générateur des fichiers Menus. Un menu est un fichier XML qui définit les options et les actions supplémentaires que l'utilisateur peut effectuer. Chaque activité Android peut avoir (ou non) un menu définissant un ensemble d'items (les options/actions utilisateur). Ainsi, pour pouvoir générer le code des menus, nous avons défini un module Acceleo avec un template qui prend en entrée les états de l'application et génère les fichiers XML correspondant à leurs menus. Le template commence par définir l'élément racine (< ?xml version="1.0" encoding="UTF-8" standalone="no" ?>). Puis, les items sont identifiés et déclarés les uns après les autres. Chaque item est identifié par un « id » (le nom de l'item dans le modèle sans caractères spéciaux et en transformant les lettres majuscules en minuscules), un titre (le nom de l'item dans le modèle) et une icône (une image par défaut à partir du répertoire « drawable »). Le fichier XML généré s'appelle « menu_ » concaténé avec le nom du menu et de l'activité (sans espaces et caractères spéciaux) et il est placé dans le répertoire « res/menu ». Son association avec l'activité correspondante se fait dans la partie Java (dans l'activité elle-même).

Générateur des activités (fichier Java). La génération des activités est la dernière étape pour obtenir une application Android complète à partir d'un modèle. Le générateur des activités Java représente la partie la plus importante (la plus volumineuse en lignes de code aussi) pour la génération sur la plateforme Android. Il est composé de 16 modules contenant 20 templates. Chaque module contient les templates qui génèrent le code pour une modalité d'interaction (sauf le module principal « main »). Nous avons volontairement décidé de mettre en œuvre cette décomposition en 16 modules afin de simplifier la modification du générateur ou l'ajout de nouveaux évènements ou nouvelles modalités d'interaction (surtout avec la grande évolution des interactions à base de capteurs mobiles).

Le générateur fonctionne comme suit :

- Pour chaque état du modèle, il génère une activité Java qui porte son nom (sans espaces ou caractères spéciaux).
- Dans cette activité, il génère le code qui permet de lui associer un layout (le layout généré à partir du même état) et un menu (s'il en existe un pour l'état).
- Il déclare les différentes widgets (EditText, Button, etc.) et les relie à leurs déclarations dans le layout.
- Il déclare les capteurs qui seront utilisés pendant l'interaction en entrée et en sortie. Du code est aussi généré pour vérifier si le matériel physique des capteurs existe dans le téléphone sur lequel l'application s'exécute (sinon un message est affiché pour prévenir l'utilisateur final).
- Au cas où l'application exploite des combinaisons entre évènements d'interaction (complémentarité et redondance) en entrée, une structure de fusion est créée pour chaque combinaison.
- Le traitement des interactions en entrée passe par plusieurs étapes :
  - Chaque évènement en entrée est identifié et son code de détection est généré. Par exemple, un secouage appliqué sur l'état génère dans l'activité une formule qui le détecte (avec les valeurs x, y et z de l'accéléromètre).
  - Si l'évènement ne participe à aucune combinaison, ses effets seront générés. Par exemple, si le secouage déclenche une vibration et une musique, le code de déclenchement de ces évènements est généré pour être exécuté après la détection du secouage.
  - Sinon, si l'évènement participe à une complémentarité ou une redondance, il sera utilisé pour remplir un slot dans la structure de fusion de la combinaison, et s'il participe à une équivalence, les effets de cette équivalence seront générés pour être exécutés après sa détection.
  - Pour la gestion de la complémentarité et de la redondance, des classes de fusion (« frame » et « slot », voir la section 5.2.2.1) sont générées indépendamment du modèle. Ainsi, ces classes sont générés une fois pour toutes et nous les avons déposées dans une API qui doit être ajoutée au projet Android. L'API est appelée « MMInteraction.jar ».
  - Si la fusion est réussie, les effets de la complémentarité ou de la redondance sont exécutés. Sinon, les effets des évènements sont exécutés séparément.
- Le filtrage des modalités se fait dans le module principal. Selon le type/la modalité, chaque évènement est envoyé vers le module spécialisé chargé de générer son code.

– Les classes Java seront toutes générées dans le répertoire « src » du projet Android. L'utilisateur peut ainsi exécuter directement son application, car Eclipse crée automatiquement le fichier APK (Android application Package).

Après le test de l'application, le concepteur/développeur peut modifier le modèle et regénérer le code de l'interface. Une fois satisfait, il peut ajouter le code fonctionnel de l'application. Il peut aussi améliorer l'affichage des évènements en sortie générés par défaut (la position des widgets sur l'écran, les animations, etc.).
Notre générateur pour Android a été utilisé pour générer plusieurs exemples d'application Android, qui nous ont permis de vérifier son bon fonctionnement et de corriger certaines erreurs. Quelques-unes de nos applications sont déjà présentes sous Google Play, telles que l'application « Multimodal Hello word » [6], « Multimodal Kitchen Unit » [7], « Multimodal sound recorder » [8] et le jeu « Multimodal shooter » [9]. Certaines d'entre elles ont été générées à 100% tandis que d'autres ont nécessité l'ajout du code fonctionnel. Le reste des exemples qui ne sont pas publiés sous Google play est présent dans la documentation de MIMIC [10] (et dans l'annexe B).

Les sections suivantes présentent les deux autres générateurs que nous avons créés pour MIMIC. Ils suivent presque les mêmes règles de génération ainsi que les mêmes étapes de fonctionnement que le générateur pour la plateforme Android.

5.2.4 Générateur pour Iphone

Le deuxième générateur de code que nous avons créé sous MIMIC était pour générer du code sur la plateform iPhone. Comme pour la génération sur Android, nous avons aussi défini des règles de génération pour iPhone (iOS). Puis, nous avons implémenté le générateur avec Acceleo. Toutefois, l'utilisation de ce générateur nécessite de travailler avec deux environnements différents : MIMIC (Eclipse - Obeo Designer-) et Xcode. Le premier permet de générer le code de l'interface tandis que le second permet la définition du projet de

6. `https://play.google.com/store/apps/details?id=com.Hello_worldmma` (dernière consultation le 24/06/2014)
7. `https://play.google.com/store/apps/details?id=com.MultimodalKitchenUnitConvertermma` (dernière consultation le 24/06/2014)
8. `https://play.google.com/store/apps/details?id=com.Multimodal_recordermma` (dernière consultation le 24/06/2014)
9. `https://play.google.com/store/apps/details?id=com.JeuDeTirmma` (dernière consultation le 24/06/2014)
10. `http://www.lifl.fr/~eloualin/examples.html` (dernière consultation le 24/06/2014)

l'application, puis son exécution. Cela s'explique par le fait qu'il n'y a pas de plugin pour le développement iPhone sous Eclipse (Apple oblige à utiliser Xcode).

5.2.4.1 Les règles de génération iPhone

Les règles que nous avons définies pour la transformation des modèles en code pour la plateformes iPhone sont très proches des règles de transformation pour Android :

- Chaque état d'interaction dans un modèle est transformé en un fichier d'implémentation (.m) et un fichier de déclaration (.h).
- Le fichier de déclaration (header) déclare les différents éléments visuels de l'interface tandis que le fichier d'implémentation implémente les différentes interactions en entrée et en sortie ainsi que le code de positionnement et des styles des évènements affichés (les widgets).
- Deux fichiers appelés respectivement « AppDelegate.m » et « AppDelegate.h » sont générés pour chaque application (chaque modèle d'application). Ils déclarent la première classe (le premier état du modèle) et jouent le rôle du délégué qui contrôle l'application.
- Des classes Objective-C peuvent être générées pour la gestion des combinaisons (si nécessaire).

De la même façon que le générateur Android, le générateur pour iOS suit ces règles pour générer des applications mobiles multimodales à partir des modèles graphiques. Le concepteur doit configurer le générateur en lui présentant le modèle de l'interface à générer ainsi que le projet qui va contenir les fichiers générés (un projet général sous Eclipse). Après la génération, les fichiers peuvent être mis dans un projet d'application iPhone OS sous Xcode pour pouvoir être exécutés et testés.

5.2.4.2 Générateur des .h et des .m

Le générateur de code pour la plateforme iPhone est décomposé en trois sous-générateurs :

Générateur des fichiers « AppDelegate ». Ce générateur est un module (qui contient deux templates) qui prend en entrée un modèle d'une interface mobile multimodale et génère deux fichiers « AppDelegate.m » et « AppDelegate.h ». Ces fichiers jouent presque le même rôle que le fichier AndroidManifest sous Android. Ils définissent le premier état de l'application, ainsi que les objets qui synchronisent l'interface et l'application (le contrôleur dans le modèle MVC).

Générateur des fichiers « .h ». Le fichier d'entête pour iPhone joue presque le même rôle

que le layout sous Android et il est généré de la même façon. Pour chaque état du modèle, un fichier header est généré. Il est nommé par le nom de l'état (sans caractères spéciaux ou espaces) concaténé avec « ViewController ». Il déclare les différents variables, méthodes et widgets de l'interface. Cependant, il ne décrit pas la mise en page de ces widgets (structure visuelle, positionnement, etc.) dans l'interface. La mise en page est décrite directement dans le fichier d'implémentation (.m).
Après les déclarations des widgets, un bloc de code protégé est généré pour permettre au développeur d'ajouter d'autres déclarations qui ne sont pas précisées dans le modèle.

Générateur des fichiers « .m ». Le générateur des fichiers d'implémentation est décomposé en plusieurs modules et templates qui génèrent le code pour les différentes interactions en entrée et en sortie. Il fonctionne de la même façon que le générateur des activités sous Android. Un fichier « .m » est généré pour chaque état. Il est nommé par le nom de l'état (sans caractères spéciaux ni espaces), concaténé avec « ViewController » (le même nom que le fichier d'entête associé). Il commence par importer le fichier d'entête et définir les propriétés des widgets déclarés dans le header. Puis, il déclare les capteurs utilisés dans l'interaction et gère les interactions en entrée et en sortie (détecter interactions en entrée, exécuter les effets de ces évènements, gérer les combinaisons -fusion/fission- si elles existent).

Le générateur iPhone a été utilisé pour générer les mêmes exemples d'application que ceux utilisés pour le générateur Android. Cela nous a permis d'avoir deux applications pour deux plateformes différentes à partir du même modèle graphique (le critère multiplateformes de l'IDM). Pour aller plus loin dans cette démarche, nous présentons dans la section suivante, un troisième générateur défini pour MIMIC.

5.2.5 Générateur HTML5/CSS

Le troisième générateur de MIMIC génère des applications mobiles multimodales web ou hybrides (applications qui combinent des éléments HTML5 et des éléments d'une application native). Il génère du code HTML5 (pour les pages web) et CSS3 (pour les styles des pages web). Comme pour les deux autres générateurs, nous avons défini des règles de génération et implémenté le générateur avec Acceleo. Pour tester les applications générées, le plugin « Aptana » (ou autres plugins pour le développement web) peut être installé sous l'environnement de modélisation (Eclipse-Obeo Designer). Ainsi, le concepteur/développeur utilisera le même environnement pour modéliser, générer et exécuter.

Le problème avec la génération des applications web/hybrides est que le langage HTML5 n'est pas encore stable [11]. De plus, plusieurs navigateurs ne le supportent pas complètement surtout en ce qui concerne les interactions. Certaines interactions sont implémentées différemment d'un navigateur à un autre, et d'autres sont implémentées dans certains navigateurs et pas dans d'autres. Pour ces raisons, nous avons défini Firefox comme navigateur par défaut pour exécuter nos applications générées, car c'est le seul navigateur qui traite bien les interactions sur mobile. Les autres navigateurs (Chrome, Opera, Safari, etc.) ne permettent pas les interactions à base de proximité, la vibration, la lecture de musique, l'appui prolongé sur un élément, etc.

5.2.5.1 Les règles de génération HTML5/CSS3

Les règles de transformation du modèle en code HTML5/CSS3 sont les suivantes (inspirées des règles de transformation pour Android et iPhone) :

- Chaque état d'interaction dans un modèle est transformé en une page web (HTML5) et un fichier CSS3.
- Le fichier CSS3 définit les styles des éléments visuels de l'interface (les widgets) tels que la taille, la couleur ou le positionnement (à partir du modèle ou par défaut).
- La page web implémente les différentes interactions en entrée et en sortie.
- Du code JavaScript peut être généré dans la page web pour la gestion des combinaisons (si nécessaire).

La configuration du générateur consiste à lui donner le modèle graphique de l'interface ainsi que le projet qui va contenir les fichiers générés. Le générateur suit les règles pour pouvoir générer l'application web ou hybride à partir du modèle spécifié. Puis, le concepteur/développeur peut exécuter l'application et la tester sur un ou plusieurs navigateurs (nous recommandons le navigateur Firefox).

5.2.5.2 Générateur pages HTML et CSS

Le générateur des applications web/hybrides est décomposé en deux sous-générateurs :

Générateur des fichiers CSS. Ce générateur est composé d'un seul module (contenant un template) pour générer les fichiers CSS qui décrivent les présentations des pages HTML.

11. `http://dev.w3.org/html5/decision-policy/html5-2014-plan.html` (dernière consultation le 24/06/2014)

Chaque fichier est nommé par le nom de son état (sans espaces ni caractères spéciaux) concaténé avec le mot « style ». Comme les layouts Android, les fichiers CSS commencent par la sélection des évènements permanents en sortie de l'état ayant la modalité affichage. Puis, chaque type d'évènement (Button, Image, etc.) est défini par un ensemble de propriétés. Certaines de ces propriétés sont spécifiées par défaut (« border », « cursor », etc.) tandis que la plupart d'entre elles sont spécifiées à partir du modèle (« width », « height », « orientation », « color », etc.). Le générateur des fichiers CSS ajoute (par rapport au générateur des Layouts Android) la possibilité de générer des animations lors de l'interaction avec les éléments affichés tel que « .Button :hover » ou « .ImageView :hover ». Cependant, ces animations sont définies par défaut et ne peuvent pas être spécifiées dans les modèles.

Générateur des pages HTML. Le générateur des pages web est composé de 10 modules. Il fonctionne de la même façon que les générateurs des activités Android et des fichiers d'implémentation iPhone. Une page HTML est générée pour chaque état du modèle. Elle est nommée par le nom de l'état (sans caractères spéciaux ou espaces). Dans la partie entête, elle contient le code qui permet de lui associer le fichier de style (le fichier .css de l'état -stylesheet-) ainsi que le titre de la page (le nom de l'état). Puis, la partie « corps » contient le code pour le traitement des interactions à base de capteurs (déclarer les capteurs, détecter les interactions en entrée, exécuter les effets de ces évènements, gérer les évènements en sortie, etc.). Cependant, certaines interactions ne peuvent pas être exécutées sous HTML5 comme, par exemple, le déclenchement de la reconnaissance vocale ou l'utilisation des boutons physiques du téléphone.
Il existe cependant des APIs qui permettent de les traiter. Toutefois, ces APIs sont spécifiques pour des plateformes particulières (une API pour le traitement du vocal sous Android, une autre pour l'utilisation des boutons du téléphone sous BlackBerry, etc.) alors que les applications web sont censées être multiplateformes. Nous avons ainsi préféré ne pas utiliser ces APIs et attendre que les interactions soient prises en compte par HTML5 de manière native afin de ne pas avoir un générateur d'applications web confiné à une plateforme particulière comme MMIR [96] (développement des applications web mobiles, mais pour Android uniquement).
Finalement, le traitement des combinaisons (la fusion en particulier) se fait à l'aide du code JavaScript généré qui permet de récupérer les données des évènements coopérants, de gérer le temps de la fenêtre temporelle et d'exécuter les effets de la combinaison (complémentarité ou redondance).

Le générateur HTML5/CSS3 a été utilisé pour générer la plupart des exemples d'application utilisés pour les générateurs Android et iPhone (cf. le critère multi-plateformes de l'IDM). Il a été utilisé pour générer d'autres exemples d'application web et hybrides telles que l'application « Multimodal Google », « Multimodal Google with complementarity » et le jeu « Multimodal Snake game ». Certaines d'entre elles ont été générées à 100% tandis que d'autres ont nécessité l'ajout du code fonctionnel ou graphique comme pour le jeu du serpent. Certains exemples sont disponibles dans la documentation de MIMIC [12] (et dans l'annexe B).

5.2.6 Comparaison entre les générateurs/générations

Les trois générateurs de MIMIC sont configurés et exécutés de la même façon dans le même environnement de modélisation. Ils analysent les modèles de la même façon, puis génèrent le code différemment selon la plateforme cible. Les différences entre ces générateurs sont ainsi situées au niveau des templates de génération et bien évidemment au niveau du code généré.

Les points communs et différences entre nos trois générateurs sont résumés ci-dessous :

- Pour la génération sous Android et iPhone la définition du premier état est indispensable alors qu'elle est inutile pour HTML, car la première page dans une application web peut être déterminée directement par l'URL.
- Les trois générateurs séparent le code généré pour l'interface graphique visuelle (css, layout et .h) du code pour le traitement des interactions (html, activité et .m). L'avantage de cette séparation est que l'utilisateur peut modifier l'interface graphique et la regénérer sans modifier le code des interactions.
- Pour la génération des applications web, nous avons ajouté la possibilité d'avoir des états « externes », c'est-à-dire des liens vers d'autres pages web hors application. Cela n'est pas nécessaire pour les applications natives (Android ou iPhone), car même s'ils font des appels vers d'autres applications, le code ne peut pas être généré automatiquement (il doit être ajouté manuellement selon l'application appelée).
- La déclaration des capteurs se fait presque de la même façon dans les trois générateurs. Cependant, les générateurs pour Android et iPhone génèrent juste le code qui teste si le téléphone contient ces capteurs physiquement, alors que le générateur pour HTML5

12. `http://www.lifl.fr/~eloualin/examples.html` (dernière consultation le 24/06/2014)

génère aussi le code qui teste si le navigateur autorise l'utilisation de ces capteurs.

- Certains évènements permanents en sortie avec la modalité « affichage » n'existent que pour une plateforme particulière. Par exemple, Android propose l'évènement en sortie « GestureOverlayView » qui dessine les gestes tactiles alors que la plateforme iPhone et le langage HTML5 ne proposent pas d'évènements équivalents. Ainsi, MIMIC peut proposer des évènements qui peuvent être générés sous une plateforme et pas une autre.
- Pour positionner les widgets sur les écrans générés, les trois générateurs suivent les positions indiquées dans les modèles. Toutefois, si les concepteurs ne précisent pas les positions des widgets dans leurs modèles, les générateurs se comportent différemment. Les générateurs d'Android et d'HTML5 génèrent les widgets sans indiquer leurs positions, puis lors de l'exécution les systèmes (Android et navigateurs web) les affichent verticalement les uns après les autres. Cependant, cela ne fonctionne pas pour iPhone, car si les positions exactes des widgets ne sont pas précisées, il peut les afficher les unes au-dessus des autres. Ainsi, le générateur d'iPhone utilise un système de positionnement par défaut (des variables qui sauvegardent les coordonnées des widgets et interdisent leurs réutilisations).
- Le générateur pour la plateforme Android génère plus de fichiers (activités, layouts, menus et « AndroidManifest ») par rapport au générateur pour iPhone (« AppDelegate », les fichiers de déclaration et d'implémentation) et pour HTML5 (les pages web et leurs styles). Cependant, cela ne signifie pas que le générateur pour Android génère plus de lignes de code par rapport aux autres. Par exemple, pour le modèle de la figure 5.37, les lignes de code générées par les trois générateurs sont équivalents (presque 200 lignes de code pour chaque plateforme).

Le tableau 5.5 résume les caractéristiques des trois générateurs.

Les générateurs que nous avons créés nous ont permis d'intégrer le critère IDM de génération de code (« model enactment ») dans notre approche. D'autres générateurs pour d'autres plateformes mobiles pourraient aussi être créés à l'avenir afin de souligner ce critère ainsi que l'aspect multiplateforme de MIMIC. Dans la section suivante nous présentons l'intégration des autres critères IDM dans notre approche à base de modèles.

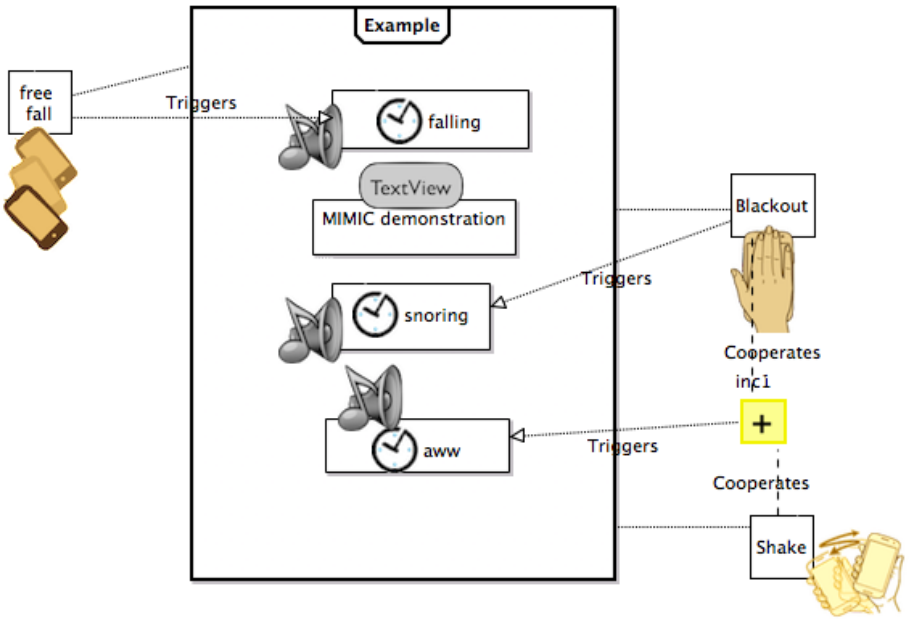


FIGURE 5.37 – Un exemple de modèle créé sous MIMIC

	Générateur pour Android	**Générateur pour HTML5/CSS3**	**Générateur pour iPhone**
Modules et templates MTL	16 modules / 20 templates	11 modules / 14 templates	14 modules / 24 templates
Environnement de génération et exécution	Eclipse	Eclipse	Eclipse + Xcode
Temps moyen de génération	20 secondes	14 secondes	14 secondes
Code généré	Java et XML	HTML5 et CSS3	Objective-C
Interactions impossibles	Aucune	Vocale et utilisation des boutons physiques du téléphone	Aucune

TABLE 5.5 – Les caractéristiques des générateurs sous MIMIC

5.3 Intégration des critères IDM

Dans la section 2.2.2, nous avons présenté l'intérêt des critères IDM (pertinence et concision des concepts, guidage de modélisation, vérification des modèles, réutilisation des modèles, génération de code) dans les approches à base des modèles. Ainsi, nous avons visé l'intégration de ces critères dans notre approche de modélisation et de génération d'interfaces mobiles multimodales.

Dans le chapitre 4, nous avons montré l'intégration du premier critère (pertinence et concision des concepts) lors de la définition de notre langage de modélisation M4L. Dans ce chapitre (section 5.1.3), nous avons montré l'intégration du quatrième critère (réutilisation de modèle) lorsqu'on a présenté la bibliothèque des types d'évènements qui donne au concepteur des modèles réutilisables. Dans les sections précédentes de ce chapitre, nous avons aussi montré l'intégration du cinquième critère (génération de code) lors de la définition des générateurs de code pour MIMIC. A présent, nous allons montrer l'intégration de deux autres critères : le guidage de modélisation et la vérification des modèles.

5.3.1 Guidage de modélisation (Model guidance)

Pour guider le concepteur et réduire son effort de modélisation, nous avons défini une notation graphique qui présente une efficacité cognitive (section 5.1.2). Nous avons aussi facilité les étapes de modélisation sous MIMIC et défini une documentation complète afin d'orienter l'utilisateur et de lui présenter des exemples de modélisation et de génération préexistants.

Outils de modélisation. Lors de la sélection des éléments de modèles à partir de la palette et de la bibliothèque des évènements, des messages sont affichés afin de guider le concepteur pour faire les bons choix. Ces messages expliquent comment utiliser/modéliser les éléments sélectionnés (indiquer l'association à utiliser entre deux évènements, expliquer l'utilité de l'élément dans le modèle, etc.) et donnent parfois des informations techniques (indiquer les évènements à base de capteurs qui peuvent être « deprecated » sur tel ou tel système mobile, indiquer la précision des capteurs utilisés, etc.) pour augmenter la qualité des modèles et des applications générées.

Documentation. La documentation que nous avons rédigée à propos de MIMIC est dis-

ponible sur le site du framework [13]. Elle permet de guider le concepteur en lui présentant des « guidelines » concernant les différentes étapes d'utilisation de notre outil MIMIC. Elle commence par l'explication des étapes d'installation (Obeo Designer, Méta-modèle M4L, environnement de modélisation MIMIC et générateurs de code). Puis, à travers un exemple, elle montre les étapes de modélisation, de vérification de modèles et de génération de code de l'application modélisée.

La documentation présente aussi dix-huit exemples de modèles et d'applications générées. Pour chaque exemple, l'utilisateur peut télécharger le modèle ainsi que le code généré (Android, iPhone et HTML5/CSS3).

Pour mieux guider le concepteur, nous avons comme perspective d'implémenter un assistant de modélisation. Il permettra de donner au concepteur des aides et des orientations selon ses besoins et sans qu'il les demande explicitement (consulter la documentation, sélectionner les éléments dans la palette pour voir les messages, etc.). Il permettra aussi de rester sur le même contexte de travail et ne pas le changer à chaque fois pour voir la documentation.

5.3.2 Vérification des modèles (Model-checking)

Dans le but d'éviter la génération de code à partir de modèles incomplets, nous avons défini des contraintes OCL et les avons intégrées dans MIMIC (voir l'annexe F). Après la modélisation, le concepteur vérifie si son modèle est bien modélisé en exécutant ces contraintes. En cas de problème, MIMIC affiche les contraintes qui ont été violées et indique les solutions possibles. Le concepteur corrige les problèmes indiqués et répète la vérification jusqu'à l'obtention d'un modèle qui respecte toutes les contraintes.
On a défini deux types de contraintes. Le premier type concerne des contraintes qui vérifient la validation des modèles par rapport au langage de modélisation M4L. Ce sont des contraintes qui ne peuvent pas être exprimées par le métamodèle et que l'on ajoute pour compléter la description du langage. Par exemple, ces contraintes sont « un évènement en entrée doit être appliqué sur un évènement en sortie ou un état », « un évènement transitoire en sortie doit être déclenché par un évènement interne ou un évènement en entrée », « une combinaison doit être entre deux évènements au moins », etc. La vérification des modèles par rapport à ces contraintes les valide syntaxiquement. Cela permet de faciliter la tâche des générateurs et d'assurer leur fonctionnement de manière correcte.

13. `http://www.lifl.fr/~eloualin/tool.html` (dernière consultation le 24/06/2014)

Le deuxième type concerne des contraintes qui vérifient l'ergonomie des interactions dans les modèles. Ce sont des contraintes qui aident le concepteur à ne pas modéliser des conflits. On peut trouver trois types de conflits : conflits entre les modes utilisateur, conflits entre les modalités et conflits entre évènements :

- Les modes gestuel et tactile peuvent être en conflit en cas de combinaison en complémentarité ou redondance si la fenêtre temporelle est très courte, car ce sont des modes qui utilisent presque les mêmes organes (mains, doigts, bras). Par exemple, utiliser la chute libre avec un « Long Touch » en complémentarité pendant une fenêtre temporelle très courte, peut être une source de conflit.
- Des combinaisons entre évènements du même mode peuvent provoquer des conflits. Par exemple, lancer une synthèse vocale et une musique en concurrence.
- Les modes visuel et auditif s'influencent s'ils sont en concurrence en sortie (effet McGurk [14], effet ventriloque, etc.).
- Les modalités qui se basent sur le même capteur peuvent être en conflit si elles coopèrent en concurrence (la proximité et l'éclairage par exemple).
- Les évènements qui possèdent la même modalité d'interaction peuvent être en conflit si la fenêtre temporelle est très courte.
- etc.

Nous avons défini un ensemble de contraintes OCL pour prévenir le concepteur des éventuels conflits qui pourraient apparaître pendant l'interaction et l'aider à modéliser des interfaces correctes. Cependant, les contraintes que nous avons définies ne vérifient pas tous les conflits possibles, car il n'est pas évident de trouver une formulation OCL pour chaque conflit. De plus, puisque la violation d'une contrainte OCL est toujours considérée comme une erreur de modélisation, certains conflits ne doivent pas être exprimés par des contraintes OCL, car leurs degrés de confusion peuvent changer d'un modèle à l'un autre. Par exemple, l'utilisation de la synthèse vocale et d'une musique concurrente en sortie peut provoquer un conflit pour l'utilisateur si la musique est très bruyante, mais elle peut ne rien provoquer si la musique est compatible avec la lecture du texte. Ainsi, dans ce cas, il revient au concepteur de détecter les modèles qui posent problèmes et de les corriger. Pour cela, la notation visuelle et la vue générale donnée par les diagrammes des interfaces peuvent l'aider à voir les conflits et à les résoudre.

14. `http://fr.wikipedia.org/wiki/Effet_McGurk` (dernière consultation le 24/06/2014)

5.4 Conclusion

Ce chapitre clôt la description de notre outil de modélisation et de génération des applications mobiles multimodales, reflétant notre contribution principale. Dans ce chapitre, nous avons commencé par décrire l'environnement de modélisation des interfaces mobiles multimodales : le choix d'outils, la notation graphique ainsi que la bibliothèque des évènements en entrée et en sortie qui sont utilisés pour la modélisation des interactions. Nous avons présenté aussi les générateurs de code associés à MIMIC. Nous avons détaillé leurs mécanismes de fusion/fission et les règles de génération qu'ils ont suivies (pour Android, iPhone et HTML5/CSS3) puis nous les avons comparées pour détecter leurs points communs. Finalement, nous avons présenté les différents critères IDM et montré comment nous les avons intégrés dans notre approche.

Dans le chapitre suivant, nous présentons les évaluations de notre langage de modélisation M4L et du framework MIMIC.

Chapitre 6

Évaluation

Sommaire

Ce chapitre est dédié à l'évaluation de nos deux contributions, M4L et MIMIC. Notre premier objectif d'évaluation consiste à montrer que le langage M4L modélise bien les différents types d'interactions mobiles multimodales en entrée et en sortie. Notre deuxième objectif d'évaluation consiste à montrer d'une part que la plateforme MIMIC influe positivement sur le développement des applications mobiles multimodales (le facilite et/ou réduit son temps), et d'autre part qu'il respecte bien les critères IDM tels que la vérification de modèle (les contraintes OCL), le guidage (la documentation et la notation visuelle) et la réutilisation (la bibliothèque des évènements).

Nous présentons tout d'abord l'évaluation de notre langage de modélisation. Ensuite, nous décrivons l'expérimentation menée auprès de développeurs mobiles, pour évaluer MIMIC. Nous présentons nos hypothèses de départ, le protocole d'expérimentation ainsi que les résultats obtenus.

6.1 Évaluation du langage : un ensemble d'exemples de modélisation

Pour évaluer notre langage de modélisation M4L, nous l'avons utilisé pour modéliser plusieurs exemples d'interfaces mobiles multimodales. Ces exemples sont les versions multimodales des applications déjà existantes issues du projet MOANO ou des différentes « appmarkets ». Ils regroupent les différents évènements d'interaction, la complexité des combinaisons entre évènements d'interaction en entrée et en sortie et ne nécessitent généralement pas beaucoup de code fonctionnel (car ce sont les interfaces qui nous intéressent particulièrement, ici). Dans cette section, nous allons présenter trois exemples d'applications : « Multimodal Kitchen Unit Converter » [1] , « Drum Pad » et « Multimodal Google ! » [2]. Les autres exemples sont disponibles dans la documentation de l'approche [3].

Multimodal Kitchen Unit Converter. Cette application aide à cuisiner. Elle permet la conversion des volumes (Pint, Quart, Tbsp, Tsp, Liter) et des poids (Lbs, Oz, Kg, Gram, Mg) pour les utiliser dans la cuisine et la pâtisserie.

1. `https://play.google.com/store/apps/details?id=com.MultimodalKitchenUnitConvertermma` (dernière consultation le 24/06/2014)
2. `http://www.lifl.fr/~eloualin/mmgoogle.html` (dernière consultation le 24/06/2014)
3. `http://www.lifl.fr/~eloualin/tool.html` (dernière consultation le 24/06/2014)

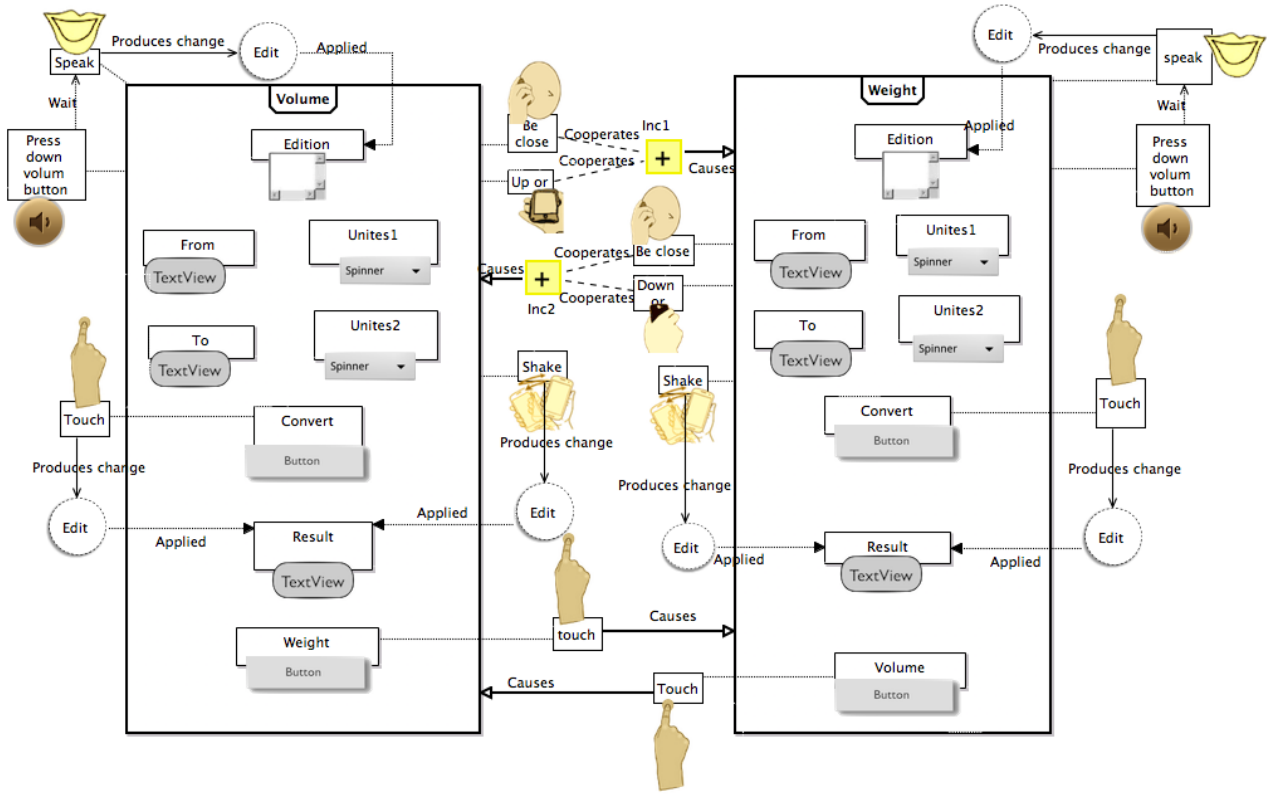


FIGURE 6.1 – Le modèle de l'application « Multimodal Kitchen Unit Converter »

En plus du « touch » (pointage tactile), l'utilisateur de l'application peut parler, secouer (accéléromètre), masquer le capteur de proximité et orienter le téléphone vers le haut ou vers le bas afin de convertir les mesures (poids et volumes). Pour entrer une valeur dans l'application, l'utilisateur peut l'écrire en utilisant le clavier tactile ou dicter la valeur (en cliquant sur le bouton « - » du volume pour l'activer) s'il ne veut pas toucher l'écran. De même, l'utilisateur peut appuyer sur le bouton « Convert » ou secouer le téléphone pour obtenir le résultat. Enfin, pour passer du volume au poids (ou l'inverse), il peut appuyer sur le bouton « Weight » (ou « Volum ») en utilisant un « touch » ou bien en orientant le téléphone vers le bas (ou vers le haut) tout en masquant le capteur de proximité. En sortie, le système affiche les résultats des conversions. Les figures 6.1 et 6.2 montrent respectivement le modèle et des captures d'écran de l'application générée.

Dans cet exemple, M4L nous a permis de modéliser facilement les complémentarités entre l'orientation du téléphone et le masquage du capteur de proximité. Il a permis également la modélisation claire du dynamisme de l'application : les interactions de l'utilisateur final et les feedbacks système (les changements qui peuvent s'appliquer sur les zones d'édition et d'affichage des résultats).

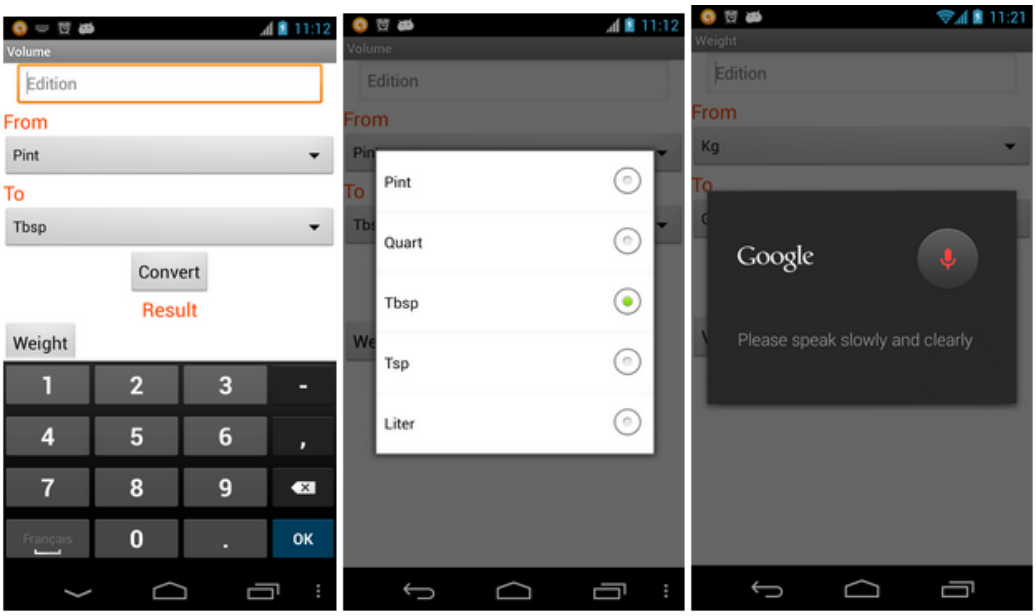


FIGURE 6.2 – Des captures d'écran de l'application « Multimodal Kitchen Unit Converter » sous Android

Ainsi, le seul code que le développeur/concepteur doit ajouter est le code fonctionnel de conversion (12% de l'application uniquement).

Drum Pad. Il s'agit d'une application pour créer de la musique avec « une batterie virtuelle » (un ensemble de fûts, cymbales, et autres percussions). L'utilisateur peut toucher (tactile) des boutons correspondant aux différentes percussions afin de créer une musique particulière, il peut secouer le téléphone et appuyer en même temps sur le bouton « + » du volume pour enregistrer sa musique et secouer en appuyant sur le bouton « - » du volume pour arrêter l'enregistrement.
Les figures 6.3 et 6.4 présentent respectivement le modèle et des captures d'écran de l'application.

Dans cet exemple, le secouage du téléphone a été modélisé une seule fois alors qu'il est utilisé dans deux complémentarités différentes (appliquées sur le même état). Cela montre un mécanisme de concision que le langage M4L propose pour réduire la complexité des modèles et augmenter leur lisibilité.
L'application a été générée à 100% sur Android. Le concepteur/développeur devait juste ajouter les sons sous forme des fichiers mp3.

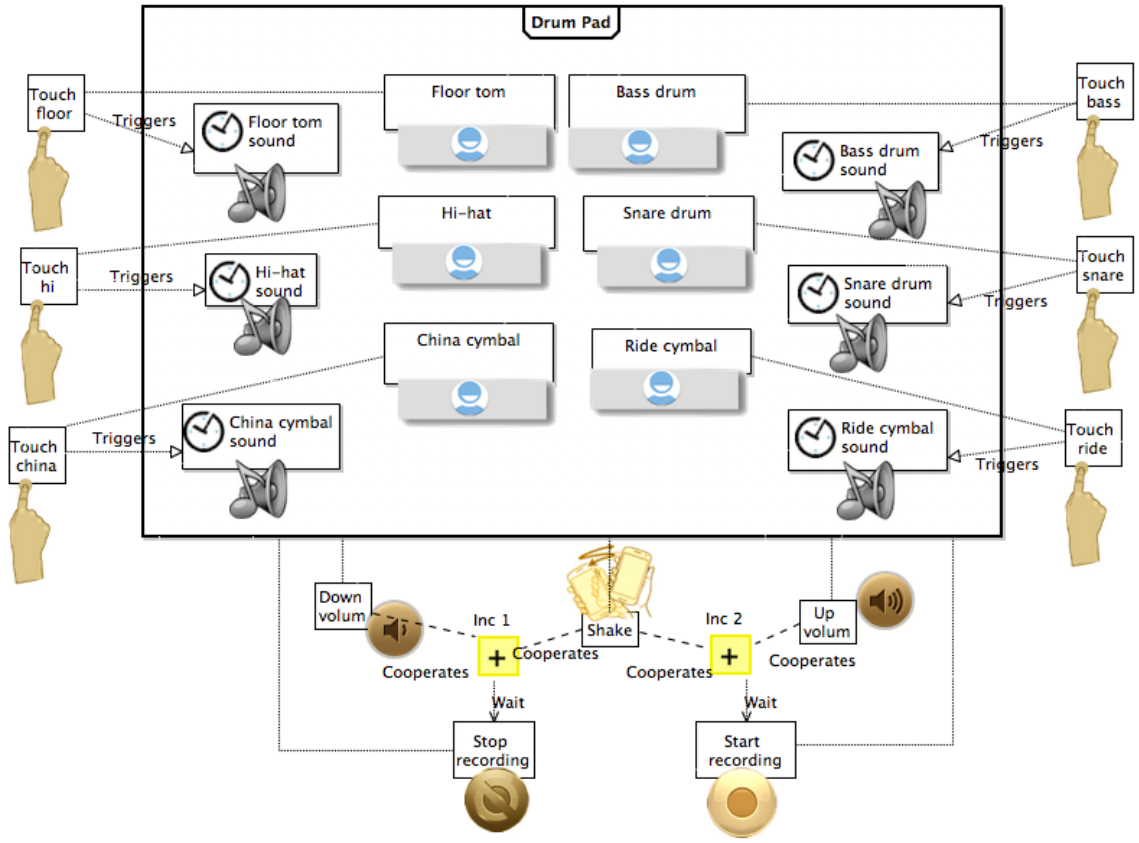


FIGURE 6.3 – Le modèle de l'application « Drum Pad »

Multimodal Google. Cette application hybride permet d'utiliser le site web mobile Google d'une façon multimodale. L'utilisateur peut écrire un mot clé puis toucher le bouton « search » ou secouer le téléphone pour chercher les résultats. Il peut aussi couvrir le capteur de proximité pour supprimer le mot clé écrit rapidement. Cependant, cette dernière interaction ne fonctionne que sous Firefox actuellement, car c'est le seul navigateur qui supporte le capteur de proximité.

Les figures 6.5 et 6.6 présentent respectivement le modèle et des captures d'écran de l'application.

Cet exemple nous a montré que M4L fournit la possibilité de modéliser des états externes de l'application. Cependant, la génération du code de passage vers ces états n'est actuellement possible que pour les applications web.

FIGURE 6.4 – Des captures d'écran de l'application « Drum Pad » sous Android

L'application « Multimodal Google » a été générée à 100% en html5 et css3. Par contre, certaines retouches ont été ajoutées dans le fichier css afin de rendre l'interface graphique plus belle.

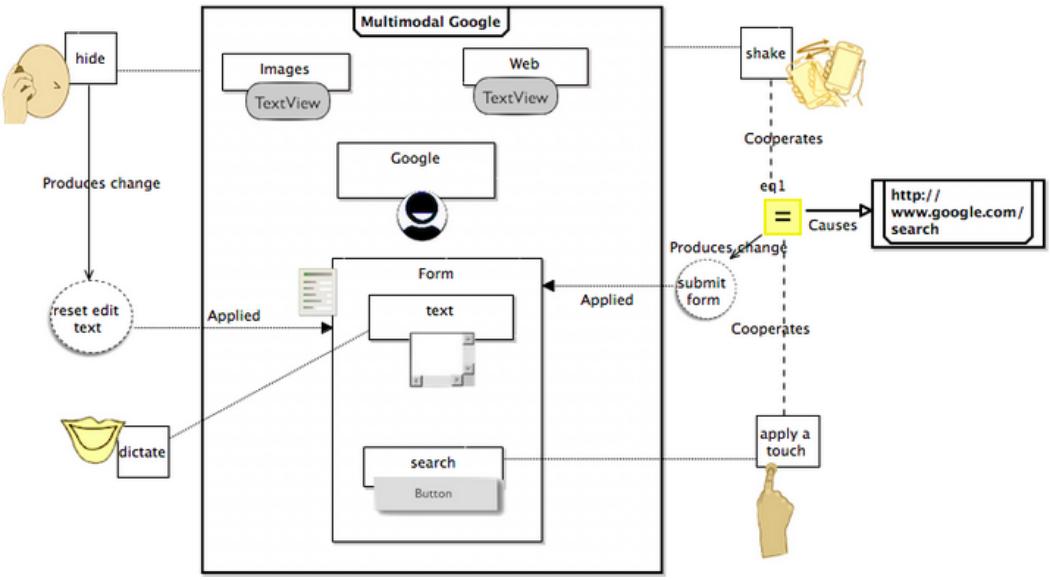


FIGURE 6.5 – Le modèle de l'application « Multimodal Google »

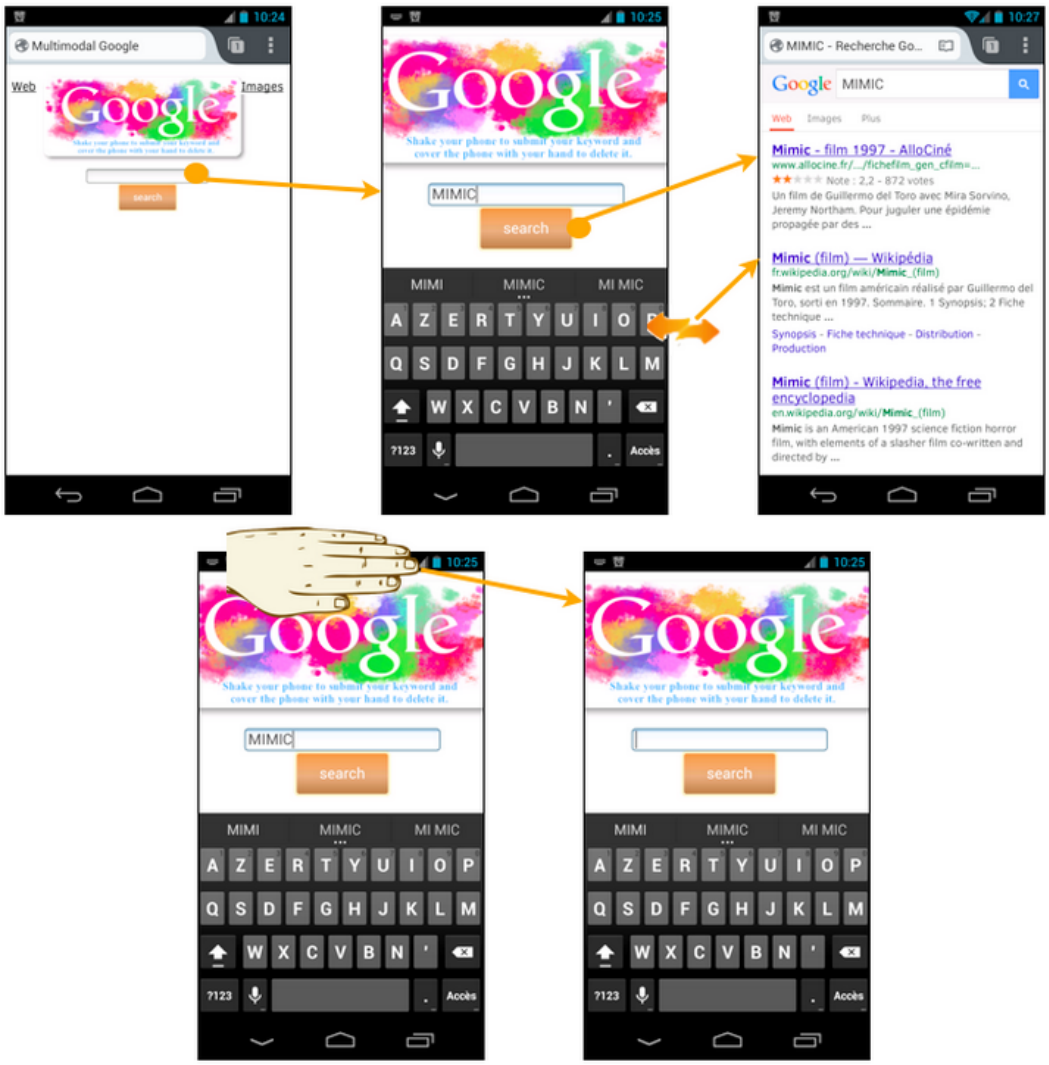


FIGURE 6.6 – Des captures d'écran de l'application « Multimodal Google » sous HTML5

Ces trois applications, ainsi que les autres exemples modélisés, nous ont permis de vérifier l'efficacité de M4L quant à la modélisation des différents types d'interaction mobiles multimodales. Elles nous ont aussi permis de détecter les points forts et faibles du langage :

Les points forts :

- M4L modélise tout le dynamisme de l'application, c'est-à-dire les évènements d'interaction en entrée et en sortie, ainsi que les retours d'information système. Cela donne une vue générale des interactions et augmente le pourcentage de génération de code de l'interface.
- Il présente un niveau d'abstraction élevé pour la gestion de l'aspect synchronisation temporelle (complémentarité, redondance, etc.).

– Il présente aussi certains mécanismes pour réduire la complexité des modèles (les états qui regroupent les évènements en sortie, la modélisation une fois des interactions utilisées plusieurs fois, etc.).

Les points faibles :

– M4L est basé sur un nombre important d'associations qui peuvent être difficiles à modéliser.
– Il ne modélise pas les aspects graphiques tels que les animations par exemple.

6.2 Évaluation du framework (modélisation et génération)

Pour évaluer notre framework MIMIC, nous avons réalisé une expérimentation avec des développeurs d'applications pour mobile. Le but de l'expérimentation était de vérifier si l'utilisation de MIMIC a un impact sur le processus de développement des applications mobiles riches de nouvelles modalités à base de capteurs mobiles et de combinaisons de modalités en entrée et/ou en sortie. Dans cette section nous allons présenter le protocole d'expérimentation suivi ainsi que les résultats obtenus.

6.2.1 Protocole d'expérimentation

L'échantillon. Vingt étudiants de l'Université Lille1, âgés de 23 à 26 ans, ont participé à l'évaluation en tant que développeurs (figure 6.7). Ils suivent des études en Master 2 informatique « E-services » [4], spécialisé dans la conception, la mise en œuvre, le déploiement et la maintenance de services numériques basés sur les nouvelles technologies de la communication. Ces étudiants ont un même niveau de programmation mobile : ils ont tous suivi un cours/TP de 24h sur le développement sous Android cette année (2013-2014).

Matériel.

– Un ordinateur pour chaque participant (vingt machines de la salle de TP des étudiants participants) avec le framework MIMIC installé.
– Un smartphone Android relié par un cable USB au PC pour chaque étudiant afin de tester les applications.
– Le sujet de l'expérimentation (dans l'annexe D)
– Un questionnaire en ligne (dans l'annexe E)

4. Le master E-services a été classé en 2013 parmi les dix meilleurs masters en France : `http://www.espacedatapresse.com/fil_datapresse/consultation_cp.jsp?idcp=2763280` (dernière consultation le 24/06/2014)

FIGURE 6.7 – Une partie des étudiants/participants durant l'expérience

Durée. Quatre heures (la durée d'un TP).

Méthode. L'expérimentation se décompose en deux parties (figure 6.8) :

- **La première partie (2h)** : la phase d'apprentissage de notre framework MIMIC
  - L'objectif de cette partie est de montrer aux participants comment réaliser un exemple d'interaction : le téléphone crie s'il est en train de tomber !
  - Puis, les étudiants consultent la documentation de MIMIC et modifient l'interaction « Free fall » par un secouage « Shake » et/ou une complémentarité entre toucher un bouton et secouer.
- **La deuxième partie (2h)** : la phase de réalisation
  - Les développeurs sont séparés en deux groupes de dix.
  - Chaque développeur du premier groupe développe un exemple d'application proposé dans le sujet en utilisant MIMIC.
  - Chaque développeur du second groupe développe la même application en utilisant le SDK Android.
  - Les participants du premier groupe répondent à un questionnaire en ligne à la fin de cette deuxième phase.

L'exemple de l'application proposée. L'application à réaliser est un petit jeu de tir (« Multimodal shooter » [5]).

5. `https://play.google.com/store/apps/details?id=com.JeuDeTirmma` (dernière consultation

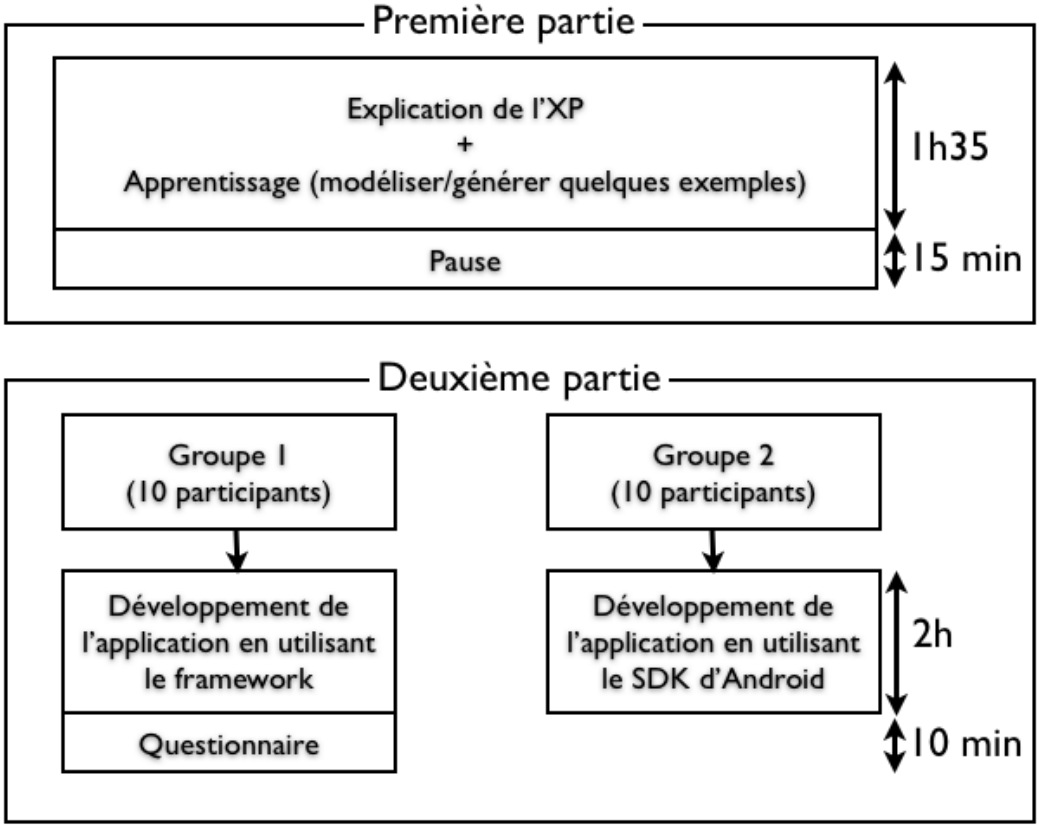


FIGURE 6.8 – Les étapes de l'expérimentation

Nous avons choisi un jeu simple, car la logique métier de l'application n'est pas importante pour cette expérimentation. Au début, le joueur a 10 balles. Pour tirer et gagner un point, il doit appuyer sur le bouton « + » du volume tout en secouant rapidement le téléphone. Avec le tir, le son d'un coup de feu se déclenche et l'application affiche une image de verre brisé. À chaque fois que le joueur tire il perd une balle et quand il n'y a plus de balle, il ne peut plus tirer. À tout moment, il peut recharger son arme en approchant sa main du capteur de proximité ou en appliquant un « Long Touch » sur l'image du verre brisé ; ce qui provoque la lecture du son de réarmement (de nouveau 10 balles disponibles) ainsi que l'effacement du verre brisé. L'application peut ressembler à la figure 6.9.

Les variables à mesurer. Pour vérifier si l'utilisation de MIMIC a un impact (positif) sur le processus de développement de l'exemple de cette expérimentation, nous avons initialement décidé de mesurer le temps et l'effort de développement pour les deux groupes.

le 24/06/2014)

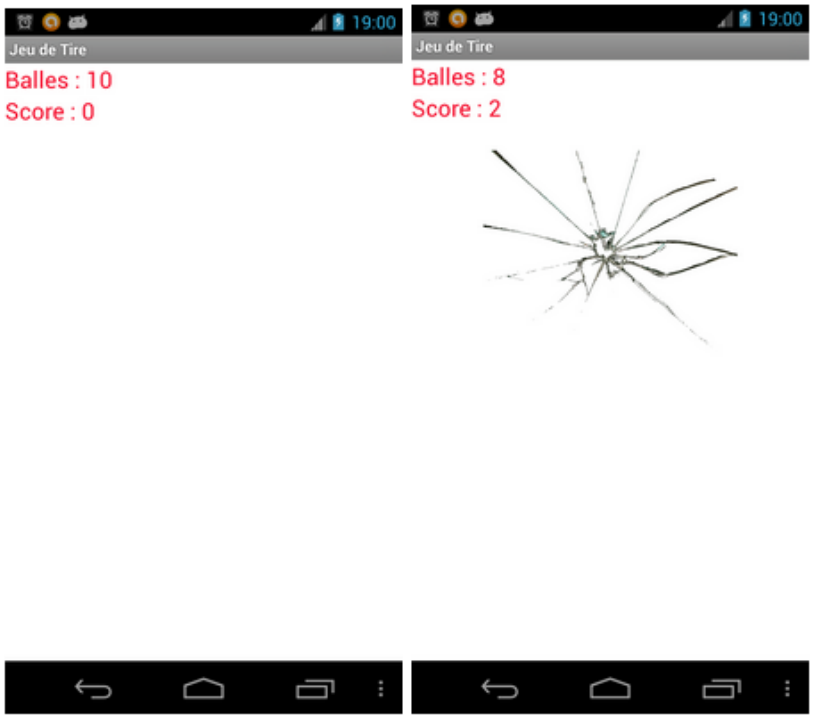


FIGURE 6.9 – Des captures d'écran de l'application à modéliser et générer

Ces deux variables nous aident à savoir si l'impact de l'utilisation de MIMIC était de faciliter et/ou accélérer le développement de l'application mobile multimodale (ce qu'on a supposé lors de la création de la plateforme). Cependant, la durée de la phase de réalisation dans cette expérimentation était limitée à deux heures, ce qui ne permet pas de donner largement le temps aux participants.

De plus, pour mesurer l'effort de développement, l'effort de programmation ainsi que l'effort de modélisation (pour les utilisateurs de MIMIC) doivent être mesurés. Toutefois, même si on considère que l'effort de programmation peut être mesuré par le nombre de lignes codées (alors que c'est un indicateur imparfait qui comporte beaucoup de défauts [6]), il est très difficile de quantifier l'effort de modélisation, car il dépend de plusieurs facteurs (déplacement des yeux, recherche des éléments de modèle, déplacement de la main avec la souris, etc.). Ainsi pour ces raisons, nous avons décidé de mesurer une autre variable qui englobe les deux variables précédentes et qui peut être facilement quantifiée : le nombre de fonctionnalités réalisées pendant le temps de développement. La mesure de cette variable pour les deux groupes nous aide à déterminer si l'utilisation de MIMIC permet de développer plus de fonctionnalités (dans un temps limité) par rapport à la programmation classique.

6. `http://fr.wikipedia.org/wiki/Ligne_de_code` (dernière consultation le 24/06/2014)

IHM	**Fonctionnel**
En entrée : 1. « Long Touch » 2. Secouage 3. Clic sur « Up volume bouton » 4. Proximité 5. Complémentarité entre secouage et bouton volume 6. Équivalence entre « Long Touch » et proximité En sortie : 1. Affichage du nombre de balles 2. Affichage du score 3. Déclenchement du son de tir après la complémentarité 4. Déclenchement du son de recharge après l'équivalence 5. Affichage de l'image après tir 6. Désaffichage de l'image au début et après recharge	1. Calcul du nombre de balles 2. Calcul du score

TABLE 6.1 – Les fonctionnalités du jeu « Multimodal shooter » (« Jeu de tir »)

Cependant, elle ne nous aide pas à savoir si le développement avec MIMIC est plutôt facile ou rapide ou les deux. Pour résoudre ce problème, nous avons posé deux questions aux participants (dans le questionnaire du paragraphe suivant) pour connaître leurs impressions concernant l'effort et le temps d'utilisation de MIMIC (par rapport au développement classique). Leurs réponses ne seront évidemment pas précises au point de les considérer comme preuve de la facilité ou de la rapidité d'utilisation de MIMIC, mais peuvent nous donner au moins des résultats préliminaires d'un point de vue utilisateur.
Nous avons mesuré le pourcentage des fonctionnalités développées par chaque participant des deux groupes. Le tableau 6.1 détaille les différentes fonctionnalités de l'application « Jeu de tir ».

Questionnaire. Le questionnaire comporte 15 questions destinées à évaluer la satisfaction des utilisateurs de MIMIC, à propos de l'implémentation des différents critères IDM (la

vérification des modèles avec les contraintes OCL, la réutilisation des modèles à partir de la bibliothèque et le guidage avec la documentation). Ces questions permettent également de collecter les impressions des participants concernant la notation visuelle des concepts/associations de modélisation, et l'effort/le temps d'utilisation de MIMIC par rapport au développement classique.

Hypothèse. L'hypothèse que nous avons fixée est la suivante :
L'utilisation de MIMIC pour le développement d'une application mobile multimodale augmente le nombre de fonctionnalités développées de cette dernière pour un même laps de temps, comparé à un développeur classique (SDK Android).

6.2.2 Analyses et interprétations des résultats

La phase d'apprentissage.

- Tous les étudiants ont réussi à modéliser et générer le premier exemple d'interaction « Free fall » (un modèle de l'application a été fourni).
- Ils ont aussi réussi à modifier une interaction par une autre, mais beaucoup d'entre eux avaient besoin d'aide de notre part pour spécifier une complémentarité entre deux évènements d'interaction.
- Beaucoup d'entre eux n'ont pas consulté la documentation. Ils se sont basés sur les exemples de modèles dans le sujet de l'expérimentation et sur les échanges entre eux.

La phase de réalisation. Pour chaque participant, nous avons calculé le pourcentage des fonctionnalités réalisées. Deux étudiants uniquement ont réalisé 100% des fonctionnalités de l'application pendant les deux heures de développement. Ces deux étudiants appartiennent au groupe des utilisateurs de MIMIC (voir tableau 6.2).

La moyenne de développement du premier groupe (avec MIMIC) est de 72,17% alors qu'elle est de 42,14% pour le deuxième groupe (avec le SDK Android). Le nombre de fonctionnalités réalisées par le premier groupe est donc plus élevé par rapport au deuxième groupe. Cela est illustré par les figures 6.10 et 6.11 qui montrent le développement des différentes fonctionnalités avec et sans MIMIC.

Pour vérifier si la différence entre les réalisations des deux groupes est significative et qu'elle n'est pas due au hasard, nous avons appliqué un test statistique de Mann-Whitney sur les résultats obtenus. Avec l'outil SPSS, le test de Mann-Whitney a révélé qu'il y a une différence significative entre les fonctionnalités réalisées par les deux groupes dans la période de deux heures et avec un risque d'erreur de 5 % ($U=22,5$, $z=-2,086$, $p=0,037 < 0,05$).

Participants	Framework	Pourcentage de réalisation	Niveau
Participant 1	Avec MIMIC	36%	Bien
Participant 2	Avec MIMIC	92,86%	Très bien
Participant 3	Avec MIMIC	71,43%	Bien
Participant 4	Avec MIMIC	92,86%	Très bien
Participant 5	Avec MIMIC	100%	Très bien
Participant 6	Avec MIMIC	100%	Très bien
Participant 7	Avec MIMIC	85,71%	Très bien
Participant 8	Avec MIMIC	71,43%	Bien
Participant 9	Avec MIMIC	57,14%	Assez bien
Participant 10	Avec MIMIC	14,28%	Bien
Participant 11	Sans MIMIC	35,71%	Assez bien
Participant 12	Sans MIMIC	42,86%	Bien
Participant 13	Sans MIMIC	35,71%	Très bien
Participant 14	Sans MIMIC	64,28%	Bien
Participant 15	Sans MIMIC	14,28%	Assez bien
Participant 16	Sans MIMIC	92,86%	Très bien
Participant 17	Sans MIMIC	21,43%	Très bien
Participant 18	Sans MIMIC	21,43%	Assez bien
Participant 19	Sans MIMIC	78,57%	Bien
Participant 20	Sans MIMIC	14,29%	Bien

TABLE 6.2 – Les pourcentages des fonctionnalités développées ainsi que le niveau de développement Android pour chaque participant

Cette différence significative est affirmée aussi par la différence importante entre les médianes des deux groupes comme le montre la figure 6.12 (avec MIMIC : Médiane = 78,57, sans MIMIC : Médiane = 35,71) avec une dispersion presque équitable par rapport à la moyenne (Avec MIMIC : Ecart type = 28,819, Sans MIMIC : Ecart type = 27,653).

Notre hypothèse est donc vérifiée, car l'utilisation de MIMIC a augmenté significativement le nombre de fonctionnalités réalisées de l'application « Jeu de tir » par rapport au développement classique sous Android (figure 6.13).

Ces résultats ont attiré notre attention sur un autre impact de l'utilisation de MIMIC qui peut être secondaire mais important. Cela concerne le niveau et la méthode de développement des participants. Dans le tableau 6.3, nous avons comparé les pourcentages des fonctionnalités réalisées par les utilisateurs de MIMIC dans l'expérimentation et leurs niveaux de développement Android. Les niveaux ont été estimés par leur enseignant d'Android, qui les a déjà évalués à plusieurs reprises (les niveaux de tous les participants sont présentés dans le tableau 6.2).

FIGURE 6.10 – Réalisation des fonctionnalités de l'application sans MIMIC

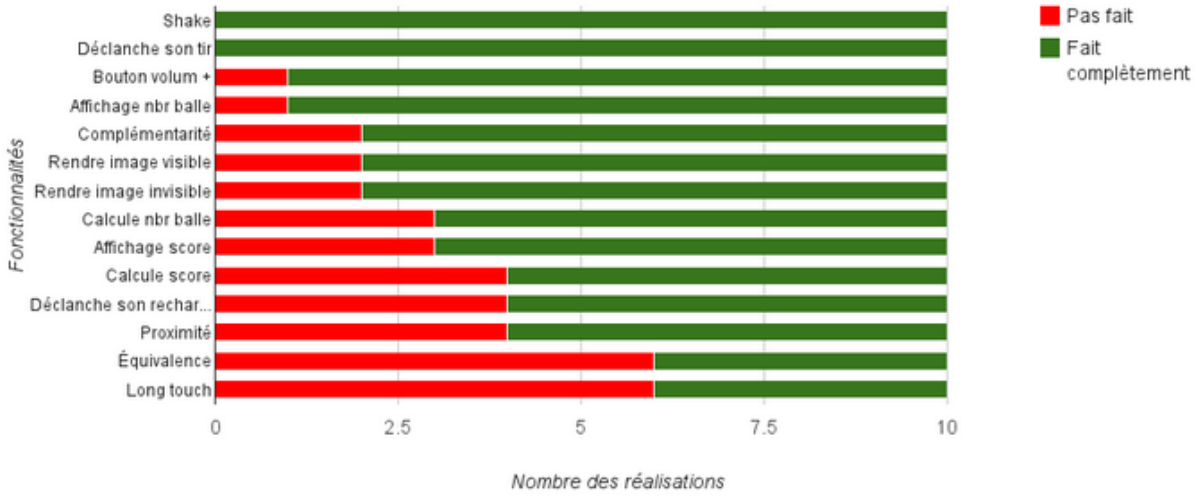


FIGURE 6.11 – Réalisation des fonctionnalités de l'application avec MIMIC

Trois participants sur cinq ayant un niveau moyen de développement Android ont réalisé un nombre important de fonctionnalités le jour de l'évaluation (les étudiants en gras dans le tableau 6.3). Cela peut être causé par le fait que l'utilisation de MIMIC a influencé positivement ces étudiants en leur facilitant la réalisation de certaines fonctionnalités avancées de l'application « Jeu de tir ».

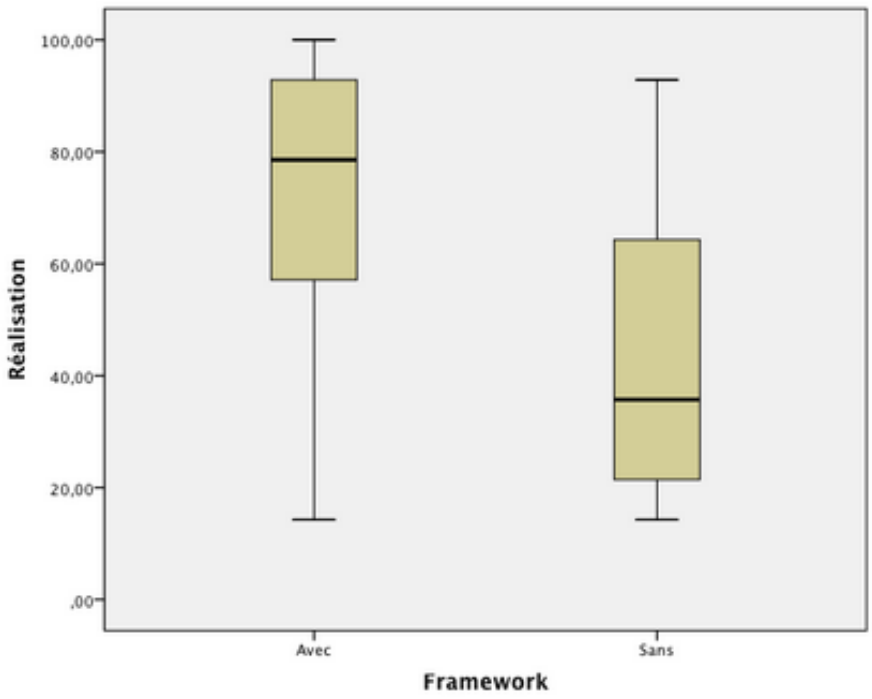


FIGURE 6.12 – Les médianes des fonctionnalités développées par les deux groupes

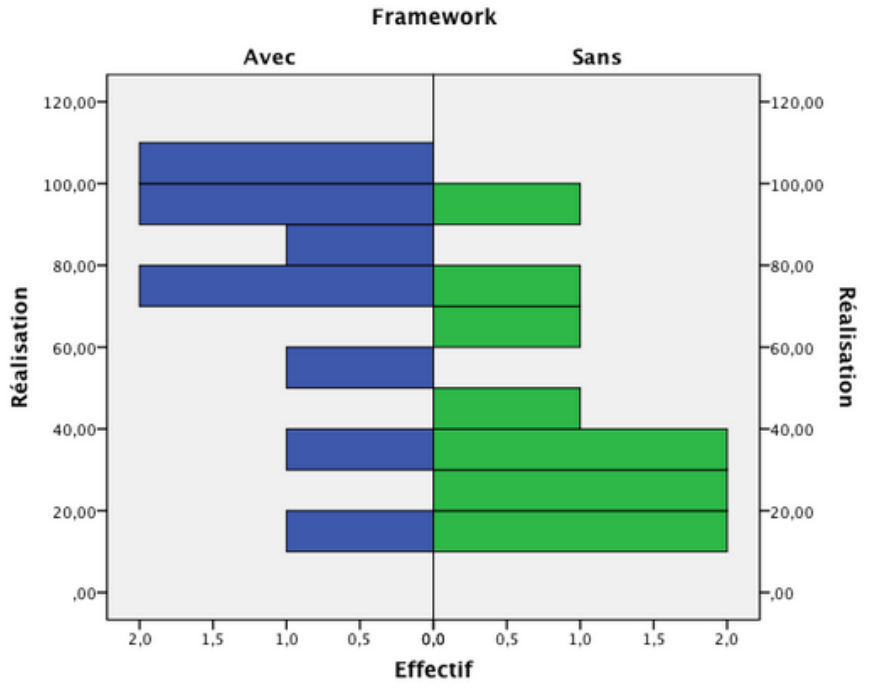


FIGURE 6.13 – Les effectifs des réalisations pour les deux groupes

Participants	Niveau	Pourcentage de réalisation
Participant 1	Bien	36%
Participant 2	Très bien	92,86%
Participant 3	**Bien**	**71,43%**
Participant 4	Très bien	92,86%
Participant 5	Très bien	100%
Participant 6	Très bien	100%
Participant 7	Très bien	85,71%
Participant 8	**Bien**	**71,43%**
Participant 9	**Assez bien**	**57,14%**
Participant 10	Bien	14,28%

TABLE 6.3 – Comparaison entre le niveau de développement et le pourcentage des fonctionnalités réalisées pour chaque participant ayant utilisé MIMIC

Le groupe des étudiants ayant utilisé la programmation classique, par exemple, avait plus de difficultés pour programmer le secouage à base de l'accéléromètre et la proximité au téléphone à base du capteur de proximité (beaucoup de cas d'échec surtout pour la proximité). En revanche, la plupart des utilisateurs de MIMIC ont réussi à les réaliser (voir figures 6.10 et 6.11). De même, beaucoup d'étudiants (même avec un niveau de programmation moyen) ont réussi à réaliser la complémentarité et l'équivalence avec MIMIC, alors que ces combinaisons n'ont été que très peu réalisées dans l'autre groupe.
De plus, en classant les fonctionnalités selon le taux de succès de réalisation (figures 6.10 et 6.11), on constate que le classement correspond à la logique de l'application pour les utilisateurs de MIMIC. La figure 6.10 montre que beaucoup de participants ont réussi à réaliser les fonctionnalités nécessaires pour la première étape de l'application (le fait de tirer), un peu moins d'entre eux sont arrivés à la deuxième étape (calcul du nombre de balles et du score) puis encore moins d'entre eux sont arrivés à la troisième étape (réarmement). Cependant, pour le groupe des utilisateurs du SDK, la figure 6.11 montre que le classement des fonctionnalités réalisées correspond plutôt à leurs niveaux de difficulté. Les participants ont tous réussi à réaliser l'affichage de nombre de balles et du score qui peut être réalisé facilement avec le SDK, un peu moins d'entre eux ont réalisé des fonctionnalités comme le secouage, changement d'affichage de l'image et le calcul du score, puis encore moins d'entre eux ont réalisé des fonctionnalités comme la complémentarité, l'équivalence et la proximité.

Nous pouvons donc conclure que l'utilisation de MIMIC a eu un impact positif sur le niveau et la méthode de développement des participants. Elle a permis aux utilisateurs de dé-

velopper l'application sans trop penser aux difficultés de réalisation des fonctionnalités (des interactions en entrée et en sortie), ce qui a facilité la tâche et permis de réaliser beaucoup de fonctionnalités dans un temps limité, même avec un niveau de développement moyen. Cependant, il est clair que cet impact n'est pas significativement révélé par cette expérimentation et que d'autres expérimentations seront encore nécessaires pour le mettre en évidence.

Le questionnaire. Les résultats du questionnaire sont les suivants :

– **Notation graphique**. 88,9% des participants ont été satisfaits de la notation visuelle (très claire : 11,1%, claire : 77,8%), mais 11,1% ne l'ont pas trouvé claire. Le choix « assez claire » n'a été choisi par aucun des participants. Ces résultats sont illustrés dans la figure 6.14. Les étudiants qui trouvent que la notation n'est pas claire ont mentionné quelques notations ambiguës (l'icône pour la proximité, les icônes de vibration courte et longue, etc.) et d'autres pas facilement identifiables (les associations, les changements qui peuvent être appliqués sur les évènements permanents, etc.).
Par conséquent, nous notons que certains éléments de la notation nécessitent d'être améliorés.

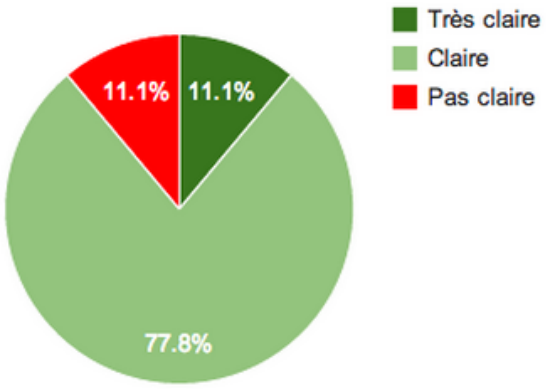


FIGURE 6.14 – Avis des utilisateurs par rapport à la notation graphique de l'environnement de modélisation

– **La tâche de modélisation**. La tâche de modélisation sous MIMIC est considérée comme « assez complexe » pour 66,7% des utilisateurs et « complexe » pour le reste (33,3%). Aucun participant n'a trouvé qu'elle est « très complexe » ou « pas complexe du tout » (figure 6.15). Ils ont mentionné que la durée d'apprentissage (2h) n'était pas suffisante pour prendre en main l'outil et que la complexité revient en grande partie au manque d'expérience.
La modélisation est donc perçue comme **assez complexe** à l'unanimité.

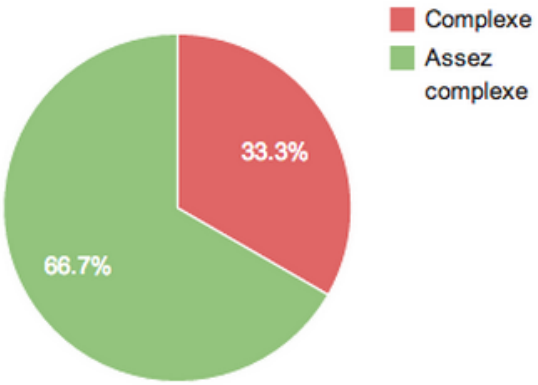


FIGURE 6.15 – Avis des utilisateurs par rapport à la tâche de modélisation sous MIMIC

– **Effort d'utilisation de notre approche (modélisation et génération)**. 33,3% et 22,2% des participants ont eu l'impression que l'utilisation de MIMIC nécessite respectivement « un peu moins » et « beaucoup moins » d'effort que celui nécessaire pour l'utilisation du SDK Android. En revanche, d'autres participants ont trouvé qu'elle nécessite « un peu plus » (22,2%) et « beaucoup plus » (22,2%) d'effort par rapport au SDK (figure 6.16).
  Les étudiants ont donc voté presque équitablement pour les 4 choix d'évaluation de l'effort nécessaire pour l'utilisation de MIMIC par rapport au SDK Android (avec un peu plus de préférence pour le « un peu moins » d'effort).

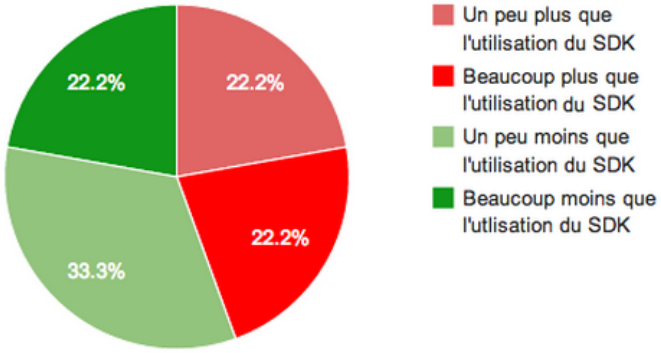


FIGURE 6.16 – Avis des utilisateurs concernant l'effort de l'utilisation de MIMIC par rapport au développement classique

– **Temps d'utilisation de l'approche (modélisation et génération)**. 44,4% des développeurs participants ont eu l'impression que l'utilisation de MIMIC pour le développement d'une application mobile multimodale nécessite « un peu plus » de temps par rapport au développement classique avec le SDK. En outre, 11% ont eu l'impression

qu'elle nécessite « beaucoup plus » de temps. Le reste des participants ont voté équitablement pour « un peu moins » (22%) et « beaucoup moins » (22%) de temps (figure 6.17).

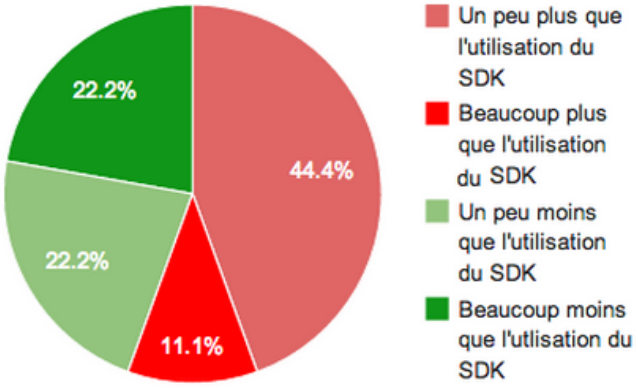


FIGURE 6.17 – Avis des utilisateurs concernant le temps de développement avec MIMIC par rapport au développement classique

– **Utilité de l'approche**. Notre approche a été considérée utile (figure 6.18) à l'unanimité des participants (88,9% : utile et 11,1% : très utile).

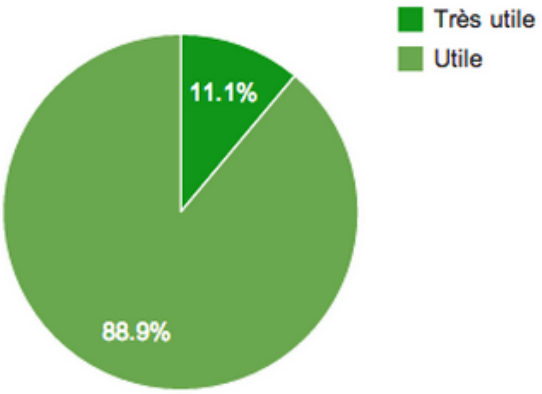


FIGURE 6.18 – Avis des utilisateurs concernant l'utilité de notre l'approche

– **Assistance de l'outil (guidage et vérification)**. 77,7% des participants ont été satisfaits de l'assistance proposée sous MIMIC (44,4% : assez satisfaisante et 33,3% : satisfaisante). En revanche, 22,2% des participants trouvent qu'elle n'est pas satisfaisante (figure 6.19). Les participants ont signalé le manque de clarté des messages d'erreur après la vérification des modèles.

Ils ont signalé aussi le manque de guidage sachant qu'ils n'ont pas lu la documentation fournie.

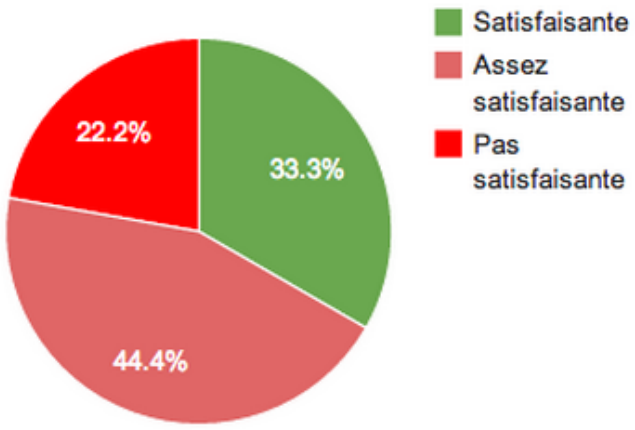


FIGURE 6.19 – Avis des utilisateurs concernant l'assistance sous MIMIC

6.2.3 Synthèse et discussion

L'utilisation de notre approche pour la réalisation de l'application mobile multimodale « Jeu de tir » avec un groupe d'étudiants du Master 2 E-service a eu quelques effets sur le processus de développement. L'effet le plus saillant concerne le nombre de fonctionnalités réalisées pendant le temps de développement. Il y avait une différence significative entre le nombre de fonctionnalités réalisées par les utilisateurs de MIMIC et les utilisateurs du SDK classique. Cela explique le large consensus des participants sur l'utilité de l'approche (figure 6.18). Cependant, cela n'explique pas si l'utilisation de MIMIC a facilité et/ou accéléré le développement des fonctionnalités de l'application. Les réponses des participants aux questions concernant la comparaison de l'effort et du temps nécessaire pour l'utilisation de MIMIC par rapport à l'utilisation du SDK (figure 6.16 et 6.17) ne donnent pas d'explications (presque équitables ; pas de consensus). Toutefois, leurs réponses concernant la tâche de modélisation laissent apparaître qu'elle est perçue comme assez complexe. Cette complexité peut être due au :

- Manque d'expérience : les étudiants n'ont passé que deux heures de prise en main du framework. À la fin de l'évaluation, ils ont tous dénoncé le manque de temps d'adaptation qui les a pénalisés.
- Assistance insuffisante : les réponses des étudiants concernant l'assistance de l'outil oscille entre « assez satisfaisante » et « satisfaisante ». Ils trouvent que MIMIC n'est

pas assez intuitif (figure 6.19) bien que la notation graphique soit claire pour la majorité d'entre eux (figure 6.14) et qu'une documentation détaillée du framework soit proposée (les participants ne l'ont pas lu, sauf quelques-uns).

Malgré la complexité de modélisation, l'utilisation de MIMIC a eu un impact positif (mais pas significatif) sur le niveau et la méthode de développement des participants. Il a permis aux utilisateurs de développer l'application sans trop penser aux difficultés de réalisation des fonctionnalités. Par conséquent, même si la modélisation sous MIMIC a présenté certaines complexités, l'approche en général (modélisation, vérification et génération de code) a facilité le développement des fonctionnalités (surtout les interactions à base de capteurs et leurs combinaisons en entrée/sortie), ce qui a permis d'avoir un nombre significativement élevé de fonctionnalités réalisées.

En ce qui concerne la rapidité de développement, le fait d'avoir les deux seuls étudiants ayant terminé leurs applications dans le groupe des utilisateurs de MIMIC peut être dû à un effet d'accélération de développement. Cependant, ce phénomène n'est pas suffisamment puissant pour mettre en évidence l'impact de MIMIC, d'autant plus que ces deux étudiants ont un très bon niveau de développement Android.

Les limites de cette expérimentation sont relatives au fait qu'elle ne concerne qu'une seule étude de cas à propos de l'utilisation de notre approche. Elle examine un ensemble de propriétés avec un seul exemple et un seul groupe de personnes dans une situation particulière. Il est ainsi difficile d'interpréter et de généraliser ces résultats sur n'importe quelle autre situation [54]. Cependant, elle nous a quand même donné des indices sur les points forts et faibles de notre approche ainsi que des variances que nous devons appliquer sur le protocole des prochaines expérimentations.
Le point fort le plus important est que l'utilisation de MIMIC facilite le développement de certaines fonctionnalités (combinaisons et interactions à base de capteurs) ce qui permet de développer plus, par rapport au développement classique. Les points faibles concernent surtout l'implémentation des critères IDM tels que le guidage et la vérification des modèles. L'expérimentation a montré que les utilisateurs ne lisent pas la documentation de la plateforme. Ils préfèrent peut-être un assistant de modélisation, qui est l'alternative classique de la documentation, pour ne pas changer à chaque fois le contexte de travail et pour faciliter la tâche de modélisation. La vérification des modèles avec les règles OCL nécessite aussi plus de soins afin d'afficher des messages d'erreur faciles à comprendre. Enfin, pour les prochaines évaluations, des améliorations doivent être appliquées sur notre protocole d'expéri-

mentation telles que l'augmentation de la durée d'apprentissage et de la période consacrée au développement, ainsi que le développement des exemples pour d'autres plateformes (iPhone, HTML5).

6.3 Conclusion

L'objectif de ce chapitre était d'évaluer le langage et la plateforme de modélisation réalisés dans le cadre de cette thèse. L'évaluation du langage de modélisation M4L a été présentée en premier. Puis, l'évaluation de la plateforme MIMIC auprès d'utilisateurs finaux (des développeurs) a été détaillée.
Concernant le langage de modélisation, l'évaluation nous a permis de vérifier sa capacité à modéliser les différents types d'interaction mobiles multimodales. Elle nous a aussi permis de détecter ses points forts et faibles. Concernant la plateforme de modélisation, l'évaluation avec des développeurs nous a montré que son utilisation augmente significativement le nombre de fonctionnalités développées dans un temps donné. Il facilite la tâche de développement, ce qui permet aux utilisateurs de développer leurs applications sans trop se focaliser sur les difficultés de réalisation des fonctionnalités (les interactions à base de capteurs, les combinaisons, etc.). L'évaluation nous a aussi montré les problèmes d'utilisation de MIMIC tels que la complexité de modélisation et l'assistance insuffisante.

Nous sommes parfaitement conscients que les évaluations réalisées ne sont pas en mesure de donner une image complète de l'apport de MIMIC. Ainsi, d'autres évaluations sont encore nécessaires pour mettre en valeur les caractéristiques de notre approche et généraliser ses impacts positifs sur le processus de développement des applications mobiles multimodales.

Chapitre 7

Conclusions et perspectives

Sommaire

Dans cette thèse, nous avons présenté nos contributions scientifiques dans le domaine de l'interaction multimodale sur mobile. Nous avons défini une approche à base de modèles pour la création d'applications mobiles multimodales. Elle comporte le langage M4L pour la modélisation des interactions mobiles multimodales en entrée et en sortie, ainsi que la plateforme MIMIC pour la génération d'applications mobiles contenant ces interactions pour Android, iOS et HTML5.

Ces travaux constituent une étape vers un développement plus facile et rapide d'applications mobiles riches de nouvelles modalités à base de capteurs et de combinaisons de modalités en entrée et/ou en sortie.

En guise de conclusion, nous résumons ci-après nos contributions, puis nous exposons un ensemble de points forts et de limites de notre approche, et identifions des perspectives envisagées pour ces travaux.

7.1 Résumé des contributions

7.1.1 Langage de modélisation M4L

Le langage M4L (Mobile MultiModality Modeling Language) représente notre première contribution dans cette thèse (chapitre 4). C'est un langage qui permet de modéliser les interactions multimodales sur mobiles en entrée et en sortie, ainsi que leurs combinaisons à travers un ensemble d'opérateurs basés sur les propriétés Tycoon et CARE. Pour le créer, nous avons été inspirés par les points positifs des différents langages existants. Nous avons aussi suivi une approche ascendante qui part des applications existantes et des contraintes d'implémentation afin de déterminer les concepts de modélisation nécessaires.

M4L possède deux principales caractéristiques : 1) il permet la modélisation de la multimodalité sur mobile à la fois en entrée et en sortie suivant le paradigme « état-transition » ; 2) il permet de modéliser les différentes interactions avec un niveau d'abstraction qui cache la complexité de leur implémentation et qui les présente avec précision afin d'augmenter la lisibilité des modèles. De plus, il fournit des concepts pertinents et concis afin de réduire la taille du langage, le rendre plus facile à apprendre et permettre la création de modèles pas inutilement vastes.

Pour évaluer M4L (chapitre 6), nous l'avons utilisé pour modéliser plusieurs exemples d'interfaces mobiles multimodales. Ces exemples regroupent les différents évènements d'interaction et la complexité des combinaisons entre interactions en entrée et en sortie. Nous avons ainsi évalué la capacité de M4L à modéliser les différents types d'interactions mobiles multimodales. Cependant, d'autres évaluations sont encore nécessaires afin d'assurer la validité globale des concepts du langage.

7.1.2 MIMIC : Outil de modélisation et de génération des interfaces mobiles multimodales

Notre deuxième contribution est MIMIC (MobIle MultImodality Creator) (chapitre 5). C'est un environnement de modélisation et de génération des applications mobiles multimodales. Il facilite la modélisation sur la base du langage M4L, propose une bibliothèque des types d'évènements d'interaction en entrée et en sortie, et permet la génération de code pour trois plateformes mobiles (Android, iPhone et HTML5/CSS3). MIMIC vérifie aussi les critères IDM tels que la vérification de modèle et le guidage. La figure 7.1 présente les différents composants de MIMIC que nous avons détaillés dans le chapitre 5.

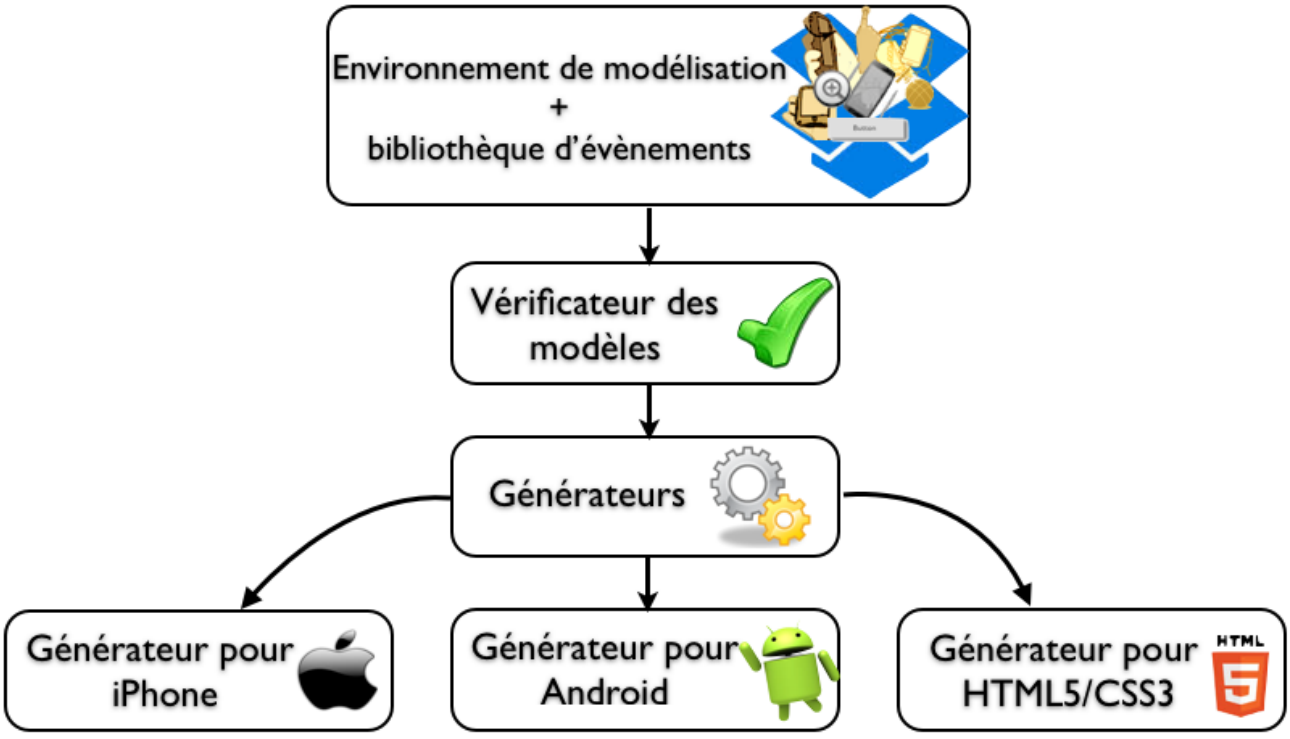


FIGURE 7.1 – Les composants de MIMIC

Les principales caractéristiques de MIMIC sont :

1. Il fournit une bibliothèque pour aider les concepteurs à modéliser les interactions multimodales. Cette bibliothèque propose des modèles réutilisables modélisant les différents types d'évènements d'interaction.
2. Il permet la vérification des modèles non seulement pour assurer leur validité par rapport au langage de modélisation M4L, mais encore pour assurer l'ergonomie des interactions modélisées et pour détecter les conflits possibles.
3. MIMIC permet actuellement la génération de code pour les trois plateformes mobiles les plus utilisées : Android, iPhone et HTML5. Il pourra aussi générer du code pour d'autres plateformes à l'avenir (en définissant d'autres générateurs).

Nous avons évalué partiellement MIMIC avec des développeurs d'applications mobiles sur Android. Le point le plus important que cette évaluation a souligné est que l'utilisation de MIMIC facilite le développement des combinaisons et des interactions à base de capteurs, ce qui permet de développer plus de fonctionnalités par rapport au développement classique. Elle change aussi la méthode de travail des utilisateurs en leur permettant de développer les applications sans trop focaliser sur les difficultés de réalisation des fonctionnalités. L'évaluation a souligné aussi que les implémentations des critères IDM, tels que le guidage et la vérification des modèles, nécessitent d'être améliorés, car les utilisateurs avaient besoin de plus d'assistance.

7.2 Originalités et points forts

L'originalité de notre travail réside dans la fertilisation croisée entre deux domaines, l'IHM et le GL. Nous avons puisé dans le domaine du Génie Logiciel l'approche IDM, qui nous a permis de définir une perspective de modélisation et de génération dédiée à la préoccupation IHM des applications mobiles multimodales. Cette perspective peut être utilisée par la suite avec d'autres perspectives de modélisation et de génération afin de traiter les autres préoccupations des applications mobiles multimodales (telles que les bases de données par exemple). Ainsi, en vertu de l'IDM, la génération complète de ces applications sera possible.

L'IDM nous a permis de modéliser graphiquement et de générer automatiquement les interfaces mobiles multimodales. Elle nous a permis aussi de vérifier les propriétés ergonomiques lors de la modélisation des interactions et des combinaisons afin de créer des interactions ergonomiques. De plus, avec la notation graphique du langage de modélisation, des

vues générales des interactions sont données à travers les modèles. Ces vues permettent aux concepteurs/développeurs de mieux voir les différentes interactions mobiles multimodales. Ils peuvent ainsi communiquer facilement entre eux, ou avec le client, et résoudre les problèmes d'interaction en cas de maintenance.

En d'autres termes, le couplage IDM et IHM dans notre approche a permis de faciliter le développement des interactions mobiles multimodales, de cacher la complexité et l'hétérogénéité sur mobile, de faciliter la gestion des conflits entre interactions, de proposer des interactions de haut niveau à base de capteurs mesurant des valeurs de bas niveau, et donner une vue complète des interactions en entrée et en sortie pour chaque application modélisée.

Avec cette approche, les développeurs peuvent proposer de nouvelles modalités/types d'interaction et combinaisons à leurs clients. Les clients peuvent aussi tester rapidement et choisir les interactions qui leur conviennent. Par conséquent, notre approche participera à la pénétration des applications multimodales mobiles.

7.3 Limites

Actuellement, MIMIC est complètement opérationnel. Cependant, nous avons noté quelques limitations technologiques et conceptuelles.

Limitations technologiques.

- L'installation de MIMIC est un processus long. Les différentes étapes d'installation sont décrites clairement dans la documentation (et dans l'annexe C). Cependant, les utilisateurs trouvent qu'elle prend beaucoup de temps. Pour résoudre ce problème, nous allons créer des bundles d'installation pour les différents systèmes d'exploitation.
- Obeo Designer est un outil payant avec une licence académique gratuite. Cela peut être un obstacle devant l'utilisation de MIMIC par les non-académiques. Ainsi, nous avons prévu de migrer vers Sirius [1] qui est la version open source d'Obeo Designer sortie en 2013.
- Le générateur pour HTML5/CSS3 ne permet pas de générer certaines fonctionnalités qui ne sont pas encore traitées par HTML5 telles que le déclenchement du vocal ou l'utilisation des boutons physiques du téléphone.

1. `http://lc.cx/Wo6` (dernière consultation le 24/06/2014)

– MIMIC n'est actuellement pas capable d'ajouter des modalités d'interaction dans des applications déjà existantes. Il faut commencer par la modélisation et la génération des interactions multimodales avec MIMIC, puis ajouter manuellement le code métier de l'application.

Limitations conceptuelles.

– Le langage de modélisation M4L ne modélise pas les aspects graphiques tels que les animations et les jeux. Il ne permet pas non plus de modéliser les positions exactes des widgets sur les écrans des applications. Ainsi, MIMIC n'arrive pas à générer des interfaces graphiques esthétiques. La résolution de ce problème nécessite la modélisation du graphique et des styles avec plus de soin. Or, les styles d'affichage sont très différents d'une plateforme mobile à une autre, et dépendent fortement des goûts des différents utilisateurs finaux. Leurs modélisations nécessitent ainsi beaucoup de travail d'abstraction qui dépasse le cadre de cette thèse.
– L'évaluation réalisée avec des développeurs mobiles nous a montré que l'assistance sous MIMIC nécessite plus d'améliorations. Nous avons donc décidé de réaliser prochainement un assistant de modélisation pour guider les concepteurs/développeurs et faciliter la tâche de modélisation.

7.4 Perspectives

Plusieurs perspectives à notre travail de recherche sont identifiées. Nous les présentons dans les points suivants.

7.4.1 D'autres évaluations de l'approche

Il semble pertinent que MIMIC soit évalué avec d'autres expérimentations, complémentaires à la première évaluation déjà réalisée. Le but sera de confirmer que MIMIC est un outil efficace pour faciliter et accélérer le développement d'applications mobiles multimodales.
Un travail à court terme sera donc d'organiser d'autres expérimentations avec des développeurs d'applications mobiles pour les différentes plateformes. Ils auront la possibilité de développer plusieurs exemples d'applications avec et sans MIMIC afin d'identifier les effets de son utilisation. Les exemples peuvent être basés sur des applications multimodales développées avec les outils existants en interaction multimodale comme SMUIML / HephaisTK [32], OpenInterface [99], etc.

Nous envisageons aussi de tester les applications générées auprès d'utilisateurs finaux. Ces tests nous permettront de juger la résistance des algorithmes utilisés pour identifier les interactions à base de capteurs par rapport aux utilisations différentes (puisque chaque utilisateur peut effectuer les interactions à sa manière).

7.4.2 Génération pour d'autres plateformes et d'autres versions

À court terme, nous envisageons de créer des générateurs de code pour d'autres plateformes mobiles telles que Windows Phone. Nous envisageons aussi la génération de code pour les différentes versions de ces systèmes qui peuvent être très différentes. Le but sera de confirmer l'aspect multi-plateforme de MIMIC à grande échelle et de l'enrichir encore pour le rendre plus complet.
Après cette étape, des évaluations et des comparaisons entre générateurs devront être réalisées.

7.4.3 Reverse engineering

Nous avons identifié plusieurs pistes de recherche pour enrichir notre approche. La première piste concerne l'utilisation de la « rétro-ingénierie ».
La rétro-ingénierie est le processus d'analyse d'un système pour identifier ses composants et leurs relations afin de les représenter dans une autre forme avec un niveau d'abstraction élevé [21]. Dans notre cas, l'utilisation de la rétro-ingénierie nous permettra d'identifier les interactions multimodales à partir d'un code existant et de créer le modèle graphique correspond sous MIMIC. Ainsi, les développeurs pourront utiliser des applications mobiles existantes, retrouver leurs modèles correspondants, ajouter, supprimer ou modifier leurs interactions et combinaisons détectées puis générer le code nécessaire (sans pour autant écraser le code existant). Le premier avantage de cette approche sera de permettre aux développeurs d'utiliser leurs applications déjà développées sans l'aide de MIMIC. Ils obtiendront automatiquement les modèles graphiques des interfaces de ces applications, ce qui facilitera leur compréhension, la maintenance et la génération pour d'autres plateformes. De plus, ils auront la possibilité de les enrichir avec d'autres interactions/combinaisons et de générer automatiquement le code correspondent.
Le deuxième avantage de cette approche sera au profit de MIMIC. La création automatique des modèles à partir du code, permet de détecter les interactions qui sont inconnues pour MIMIC. Ainsi, la plateforme peut les ajouter dans sa bibliothèque d'évènements d'interac-

tions et dans ses générateurs. Pour ajouter une interaction dans la palette de l'environnement de modélisation, il suffira de demander au développeur de donner un nom à l'évènement d'interaction, de lui associer une icône et une modalité d'interaction (choisir à partir d'une liste ou en ajouter une). MIMIC ajoutera ensuite le code correspondent à cet évènement dans les générateurs. Cependant, le code ne sera probablement disponible que pour une seule plateforme mobile (la plateforme de l'application dans laquelle le framework a détecté cette interaction). Par conséquent, la génération de cet évènement ne sera pas multi-plateforme sauf si MIMIC détecte son code encore une fois sous une autre plateforme ou après une intervention de notre part (intervenir pour compléter le code sous les autres plateformes).
Cette perspective nous semble primordiale pour aboutir à une approche auto-évolutive de développement des applications mobiles multimodales, surtout avec la grande évolution des capteurs mobiles.

7.4.4 Résoudre les problèmes ergonomiques de la multimodalité sur mobiles

Notre deuxième piste de recherche concerne l'utilisation ergonomique des interactions à base de capteurs. Dans la partie vérification des modèles (Model-checking - chapitre 5), nous avons essayé de prendre en compte certains aspects ergonomiques de l'interaction multimodale sur mobile. Nous avons défini des règles de vérification pour aider les concepteurs à détecter les conflits d'interaction. Cependant, certains aspects ergonomiques ne peuvent pas être détectés ou résolus avec des règles de vérification. Par exemple, les développeurs ne trouvent généralement pas comment informer les utilisateurs d'applications mobiles des interactions à effectuer. L'utilisation du tactile est évidente, car les utilisateurs peuvent voir les points d'interaction sur l'écran (bouton, liste, etc.). Par contre, pour les interactions à base de capteurs comme le secouage du téléphone ou le changement d'orientation, aucune indication n'est affichée. Il faut donc trouver un moyen pour en informer les utilisateurs. Actuellement, il existe deux solutions. La première consiste à décrire les interactions dans la partie descriptive de l'application sur les « App-stores ». Par exemple, sous Android lors de la publication d'une application, nous avons la possibilité de la décrire, d'ajouter des vidéos et des captures d'écran. Ainsi, nous pouvons utiliser cet espace pour indiquer les interactions à base de capteurs ainsi que les combinaisons de cette application (complémentarité, redondance, etc.). Le problème de cette solution est que la partie descriptive n'est pas toujours lue par les utilisateurs lors du téléchargement d'applications. De plus, cette solution ne peut pas être appliquée pour les applications web mobiles qui sont accessibles directement par un URL

(pas besoin de téléchargement). La deuxième solution consiste à ajouter des assistants dans l'application pour expliquer les interactions. La figure 7.2 montre un exemple d'assistant sur Samsung Galaxy S3 qui explique l'interaction pour prendre une capture d'écran.

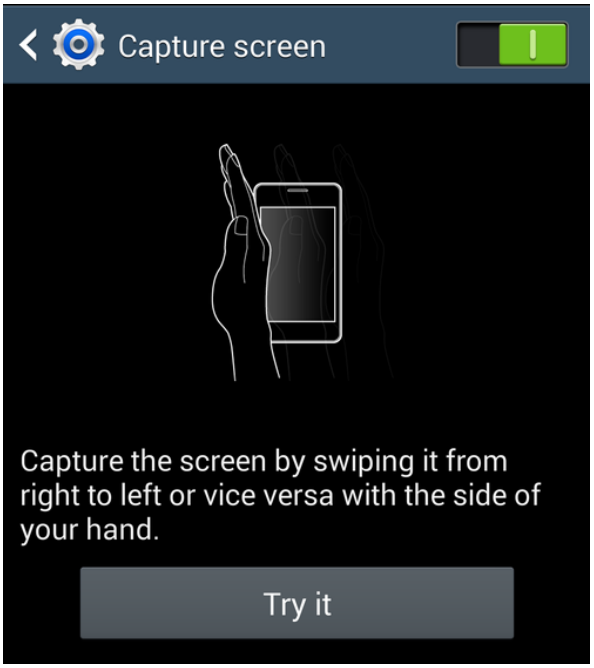


FIGURE 7.2 – Explication de l'interaction nécessaire pour prendre une capture d'écran sous Samsung Galaxy S3

Le problème de cette solution est que les assistants viennent parfois perturber les utilisateurs, surtout s'il y a plusieurs interactions à expliquer (même s'il y a toujours la possibilité d'arrêter l'apparition de ces assistants).
Par conséquent, nous devons proposer d'autres solutions plus adéquates pour faciliter l'apprentissage de l'interaction multimodale sous mobile.

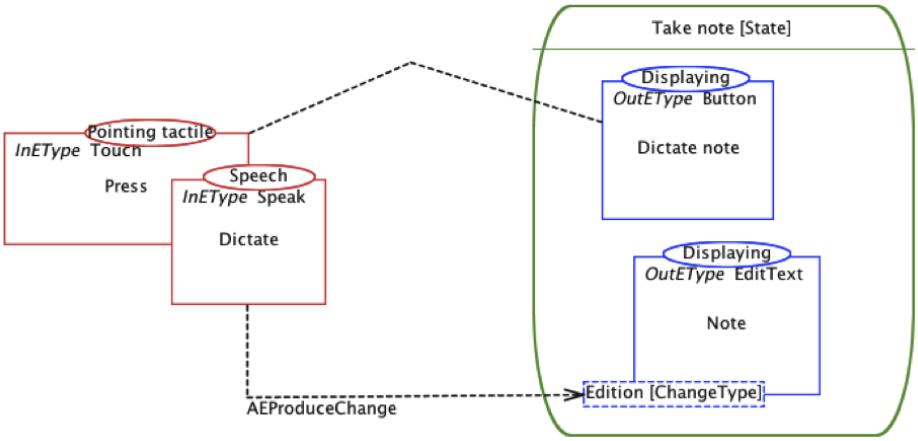
Beaucoup d'autres problèmes ergonomiques liés à l'utilisation des capteurs mobiles et de la multimodalité sont identifiés. Par exemple, le problème de définition des interactions intuitives pour les divers groupes d'âge [79], les problèmes liés à la confidentialité des informations détectées par les capteurs mobiles [22], les problèmes de réalisation des combinaisons [107], etc. Leur résolution constitue un vaste programme de recherche d'un point de vue utilisateur.

Annexe A

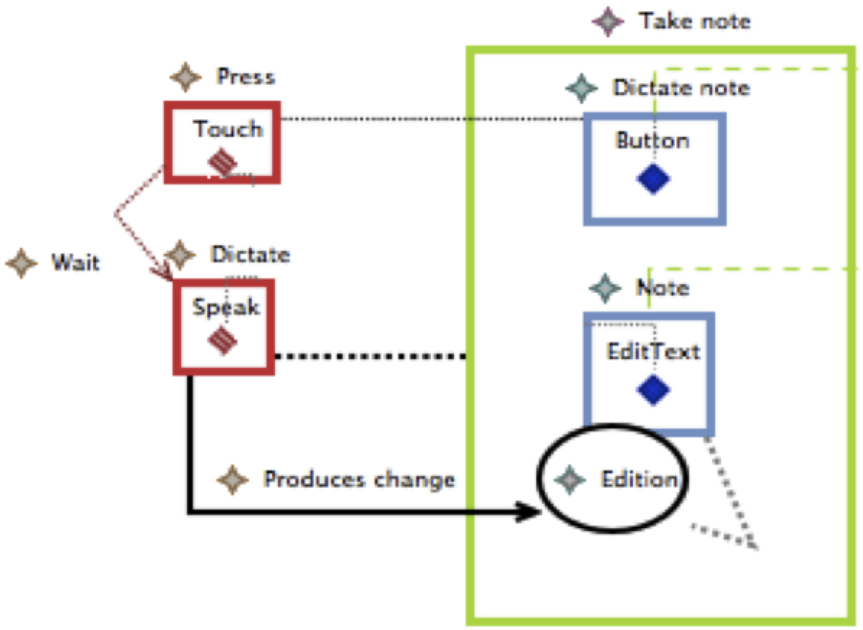
Exemple de l'application/interaction « Dicter une note » modélisée avec les différentes notations de M4L (chapitre 4)

Sommaire

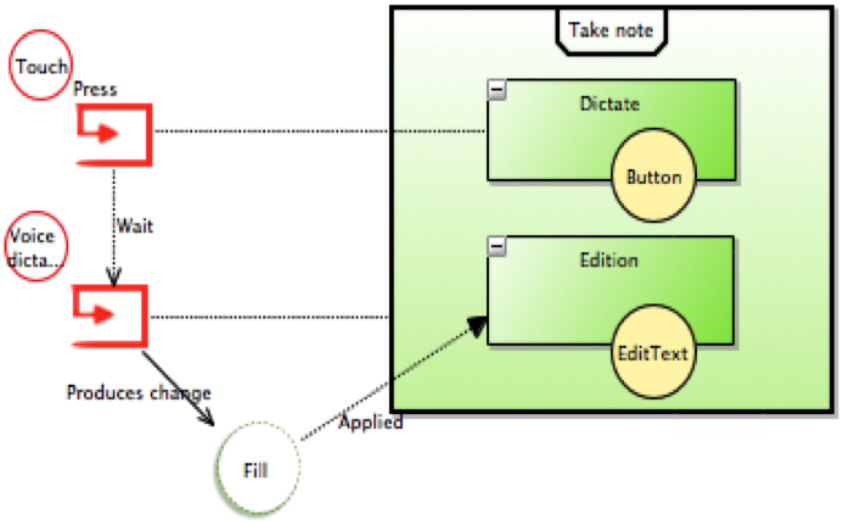
A.1 Version 1



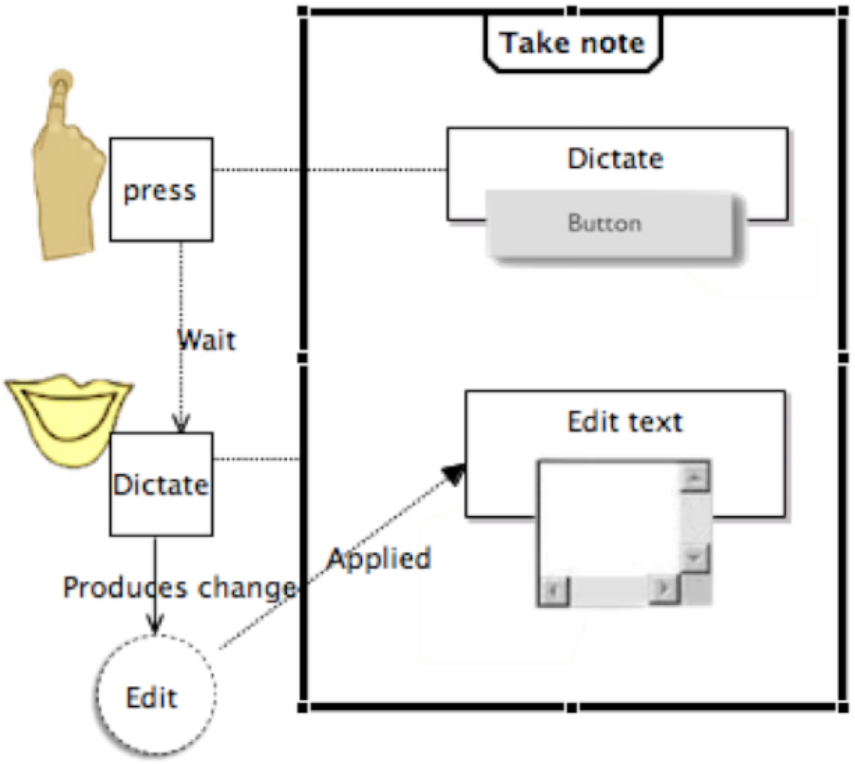
A.2 Version 2



A.3 Version 3



A.4 Version 4



ANNEXE A. EXEMPLE DE L'APPLICATION/INTERACTION « DICTER UNE NOTE 198 » MODÉLISÉE AVEC LES DIFFÉRENTES NOTATIONS DE M4L (CHAPITRE 4)

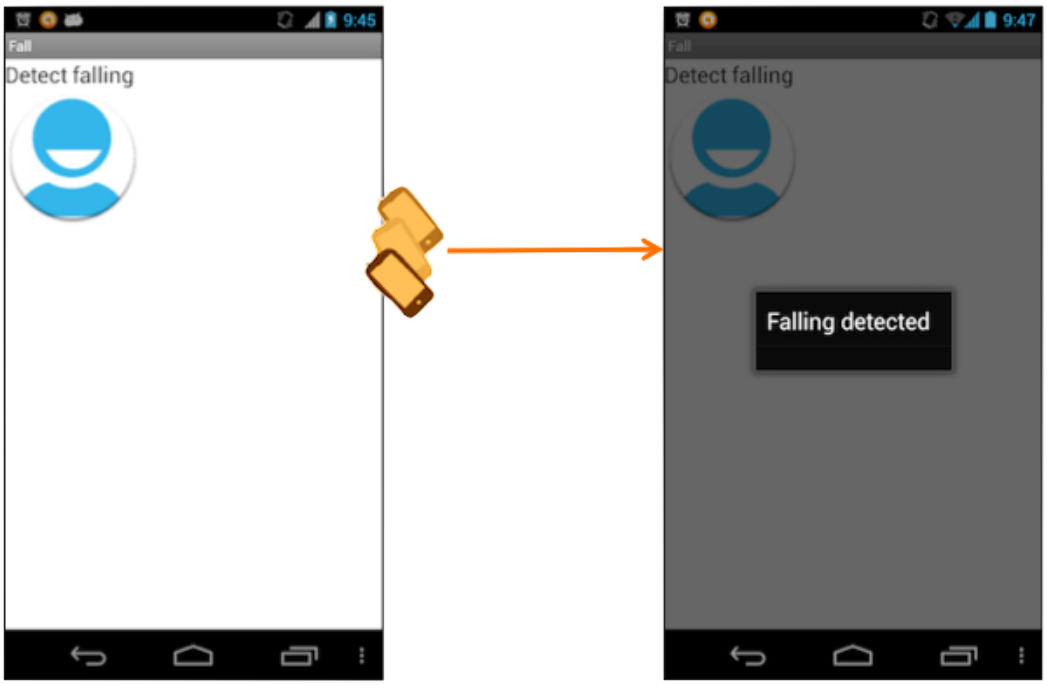
Annexe B

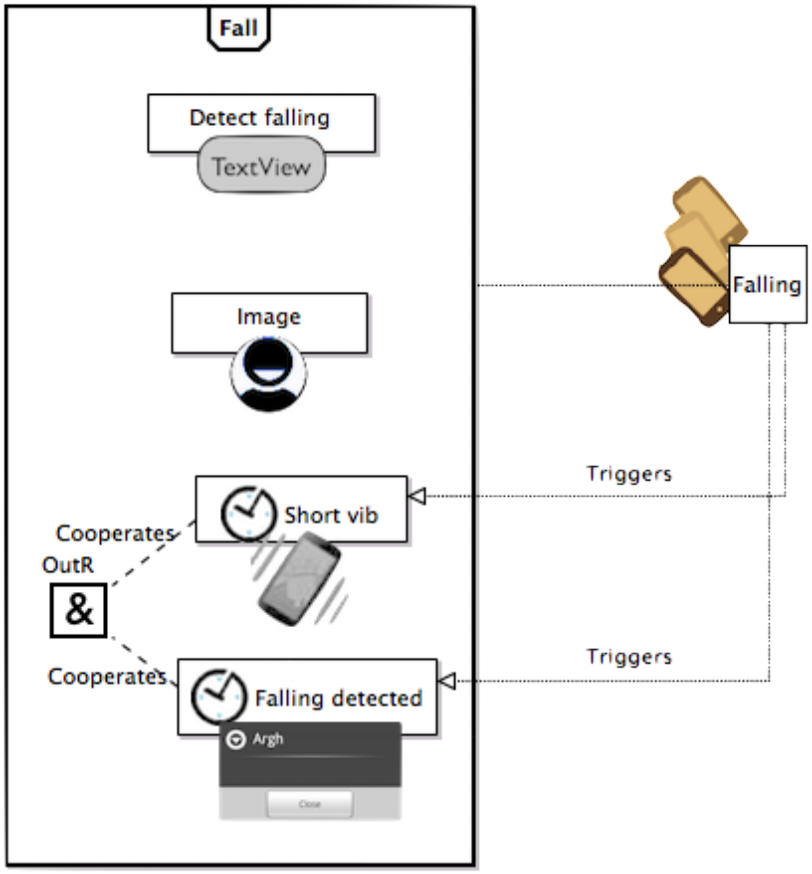
Des exemples d'applications modélisées avec et générées sous MIMIC

Sommaire

B.1 Exemple 1 : « Chute libre »

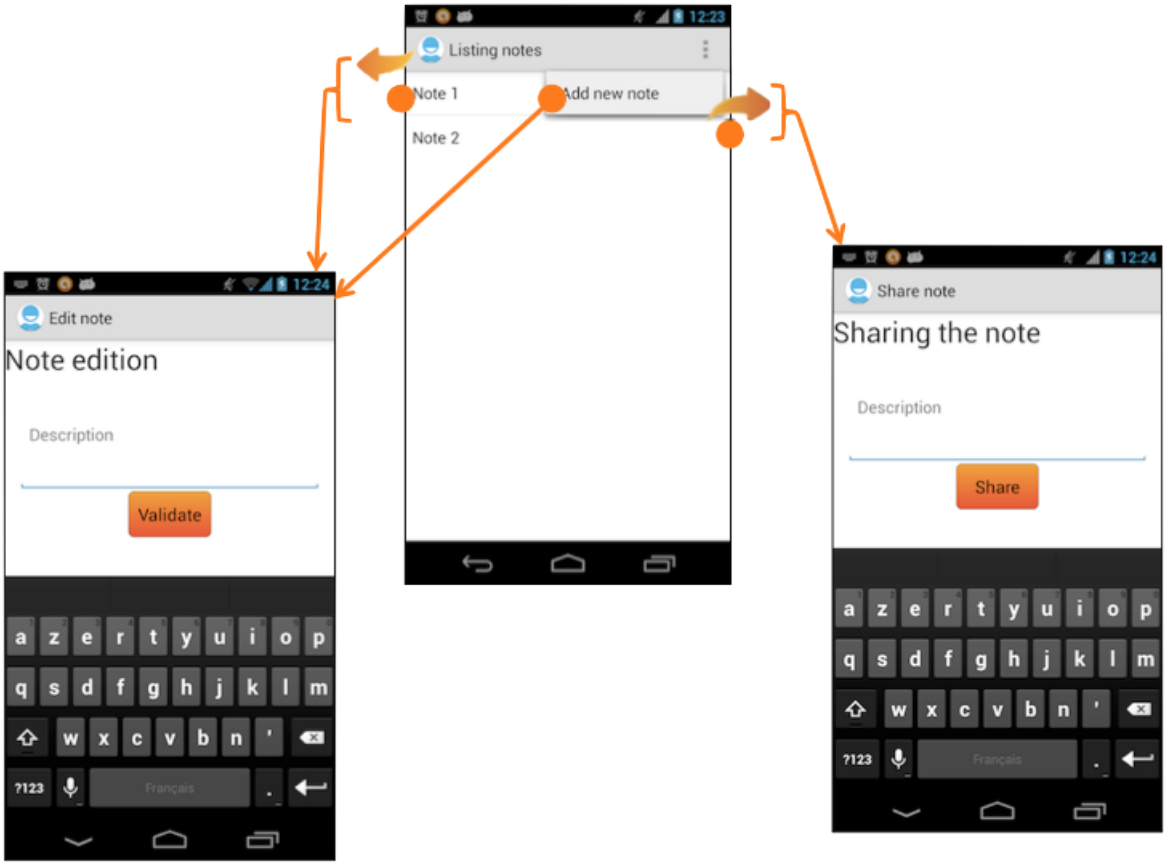
Cet exemple décrit une application qui affiche une alerte (« Falling detected ») et déclenche une vibration courte si elle détecte que le téléphone tombe.

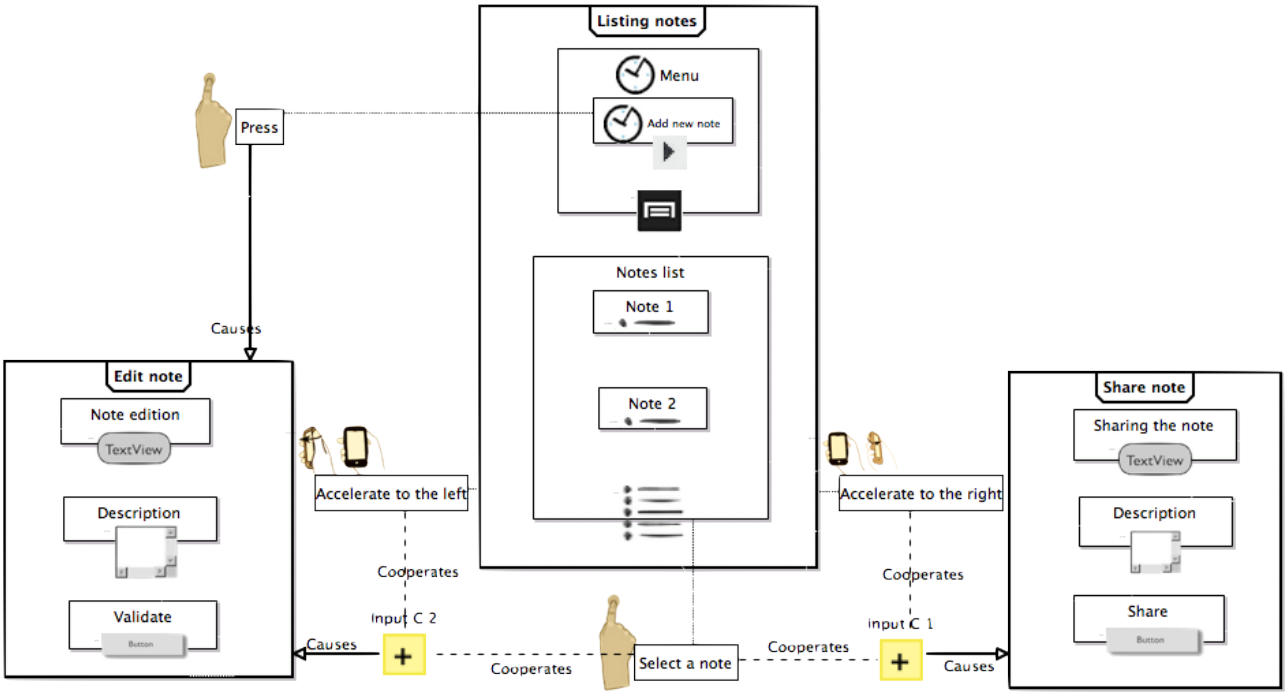




B.2 Exemple 2 : « Prise de notes »

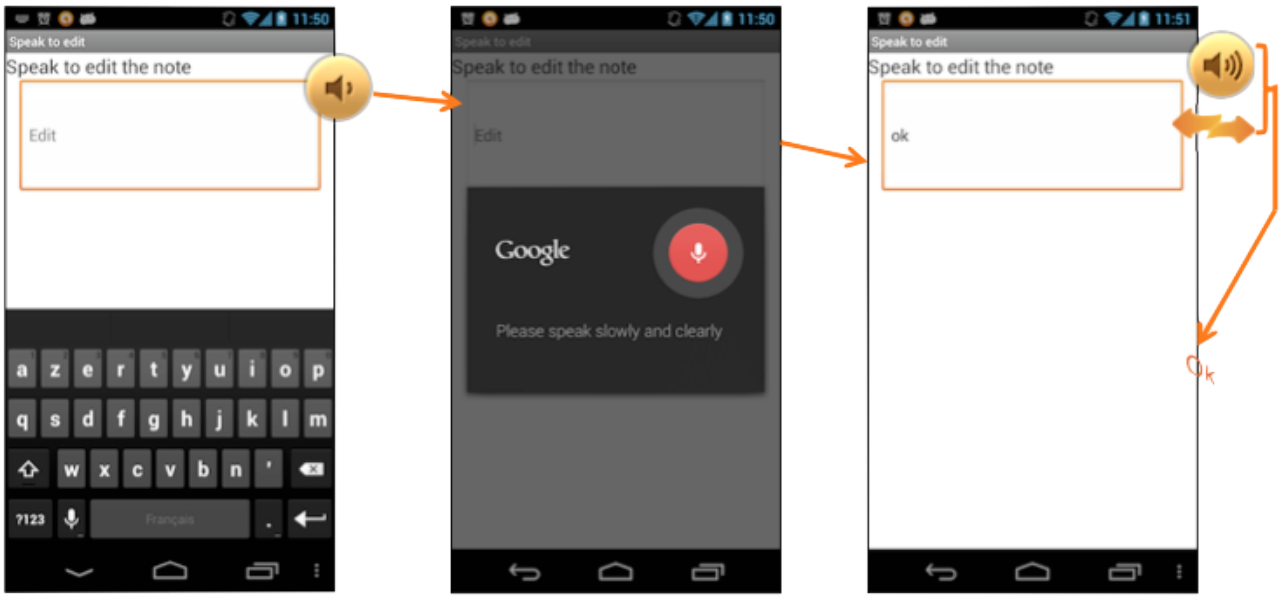
Cette application permet de consulter, éditer et partager des notes. Pour consulter une note, l'utilisateur la choisit à partir de la liste affichée dans le premier écran. Pour éditer une nouvelle note, il sélectionne la commande « Add new note » à partir du menu de l'application. Pour éditer une note déjà existante, il utilise une complémentarité entre le pointage tactile (sélection de la note) et l'accélération (vers la gauche). Finalement, pour partager une note existante, il utilise une autre complémentarité entre le pointage tactile (sélection de la note) et l'accélération (vers la droite).

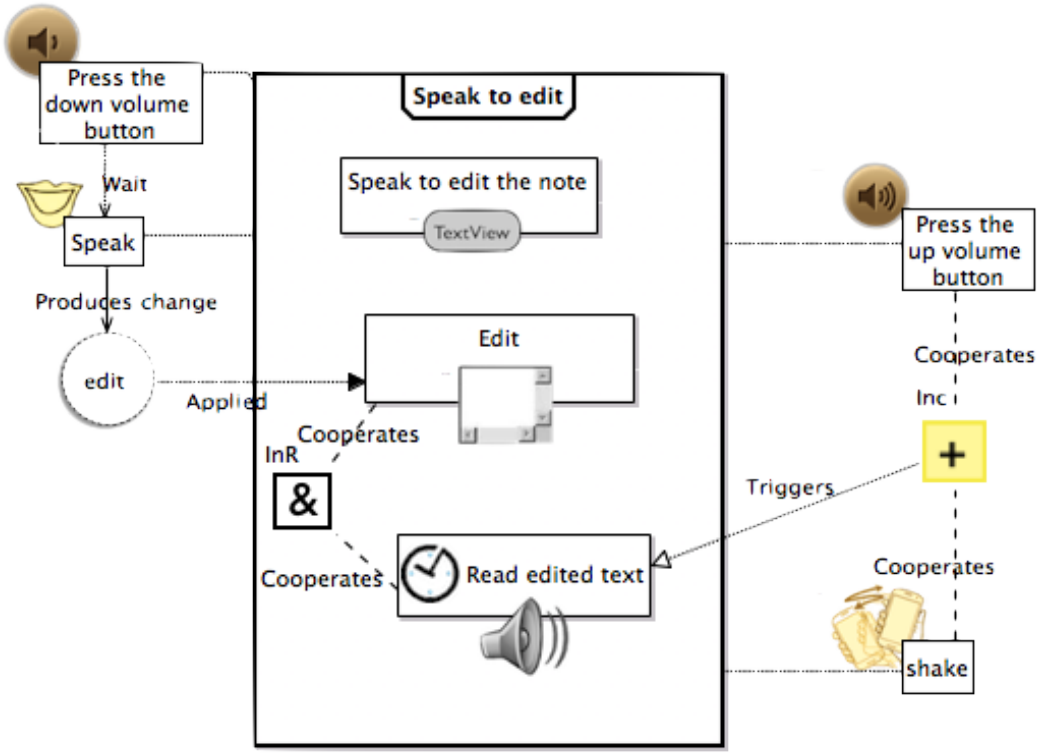




B.3 Exemple 3 : « Éditer en vocal et lire en synthèse vocale »

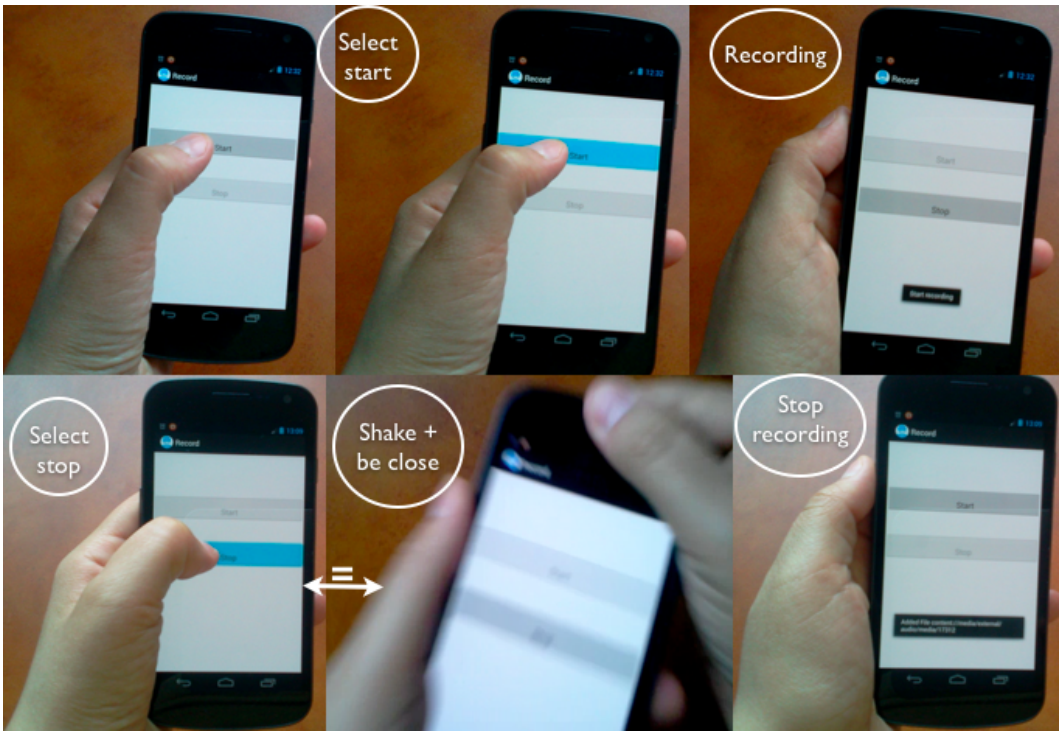
Cette application permet à l'utilisateur d'éditer du texte en vocale (le déclenchement de la reconnaissance vocale se fait par un clic sur le bouton physique de réduction de volume). Elle permet aussi de lire le texte édité en utilisant la synthèse vocale. Pour faire cela, l'utilisateur clique sur le bouton physique d'augmentation de volume et secoue le téléphone en même temps.

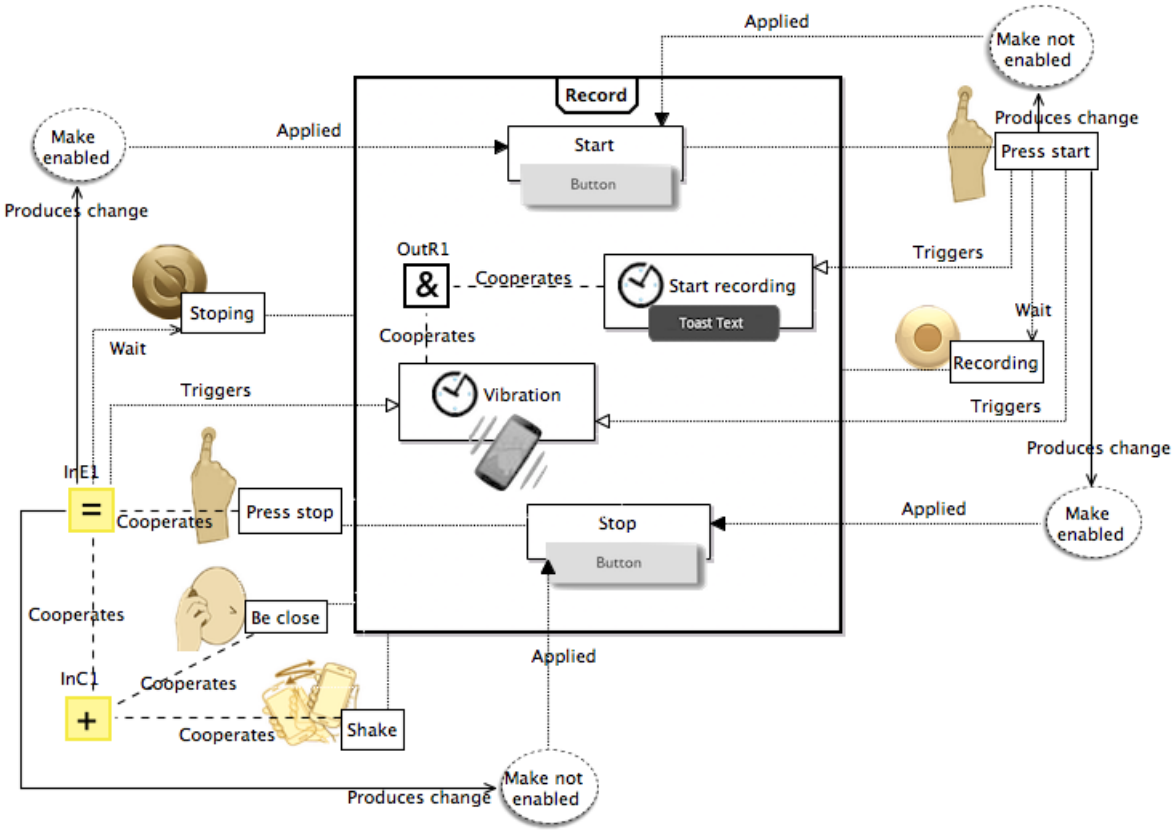




B.4 Exemple 4 : « Enregistrement de musique »

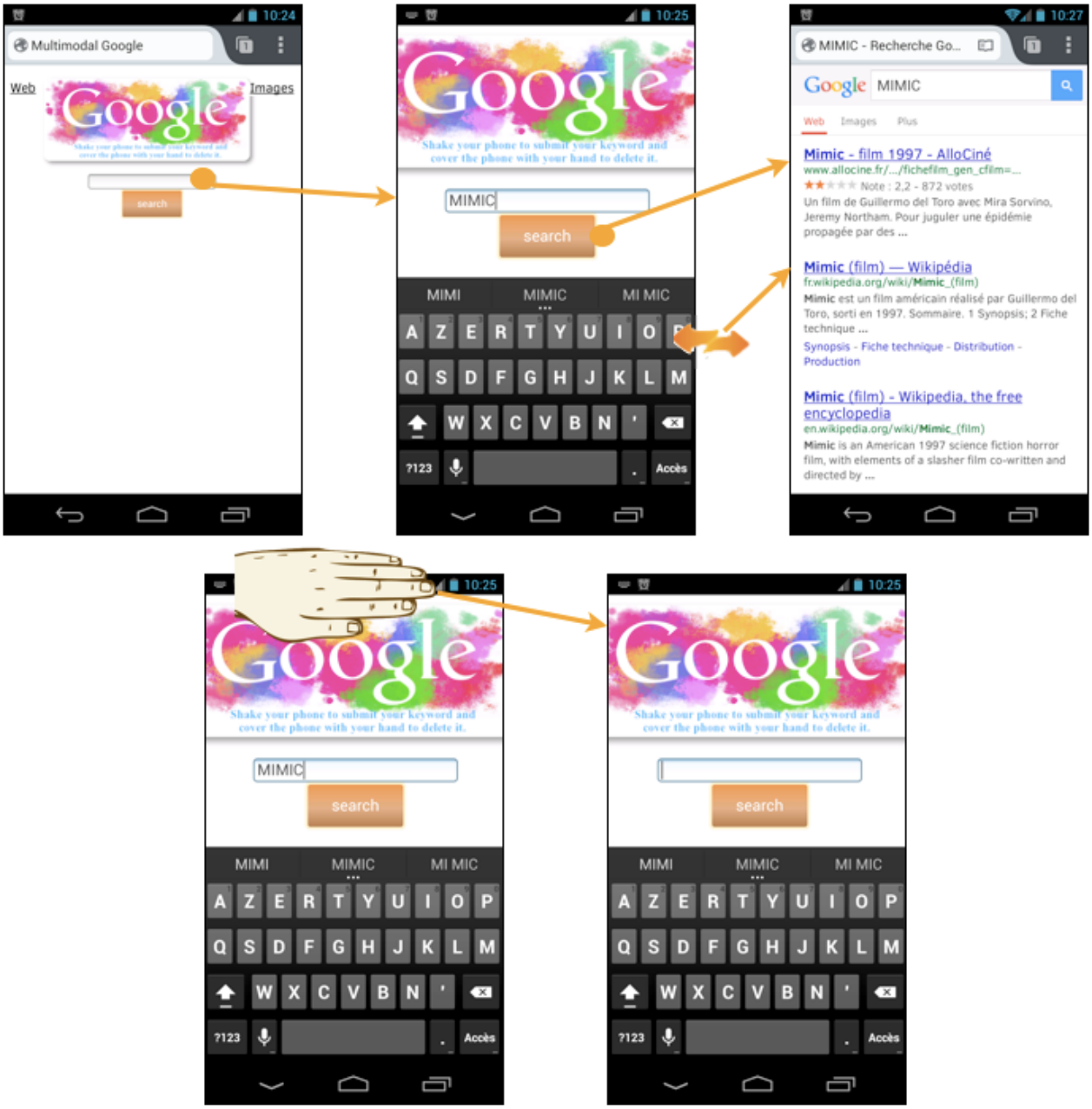
L'application dans cet exemple est un outil pour enregistrer du son en utilisant différentes modalités d'interaction. Pour commencer l'enregistrement, l'utilisateur clique sur le bouton « Start ». L'application démarre l'enregistrement, émet une vibration courte, affiche un message rapide pour indiquer le début de l'enregistrement et désactive le bouton « Start ». Pour arrêter, l'utilisateur peut choisir entre un clic sur le bouton « Stop » ou un secouage du téléphone tout en couvrant le capteur de proximité. Ce dernier choix peut être appliqué si l'utilisateur porte des gants par exemple. L'application arrête l'enregistrement, vibre, affiche l'adresse du fichier enregistré et active le bouton « Start ».

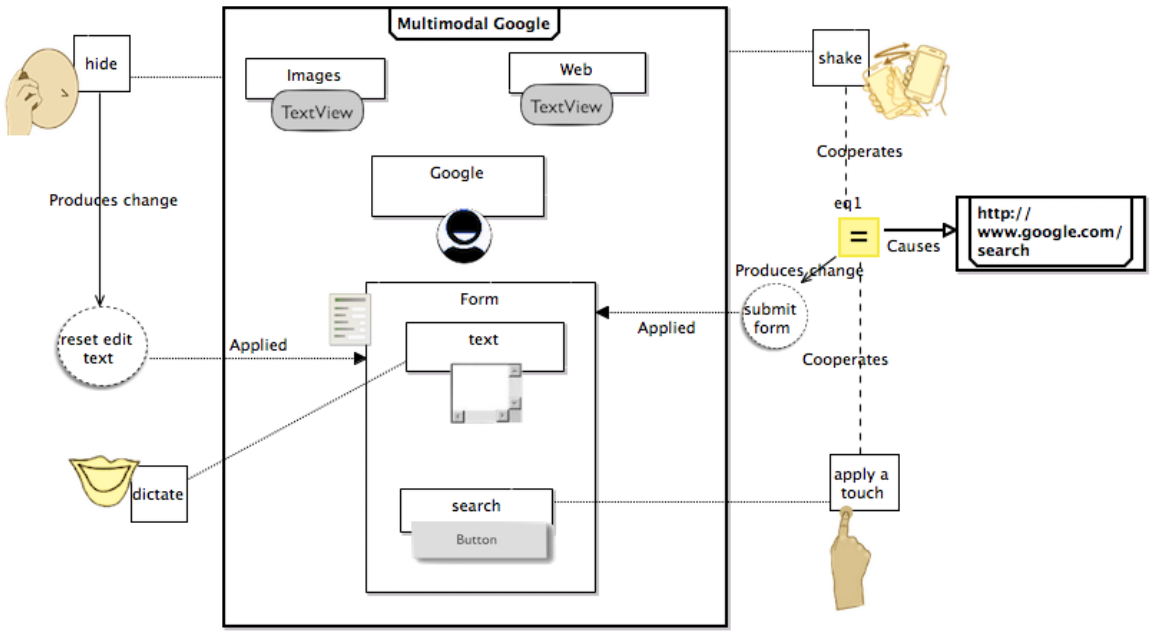




B.5 Exemple 5 : « Google multimodal »

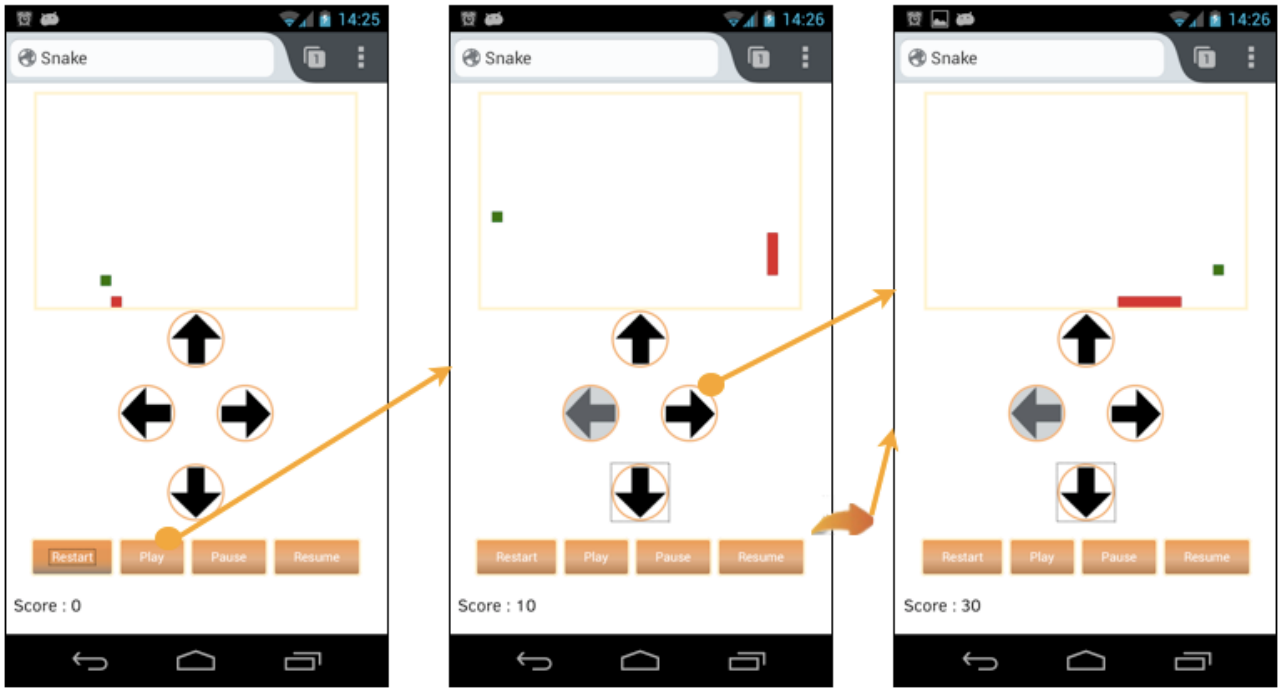
Cette application web permet d'utiliser le moteur de recherche Google d'une façon multimodale. L'utilisateur peut écrire le mot-clé et secouer le téléphone pour valider la recherche. Il peut également couvrir le capteur de proximité pour supprimer rapidement le mot-clé (cette interaction n'est possible qu'avec le navigateur Firefox).

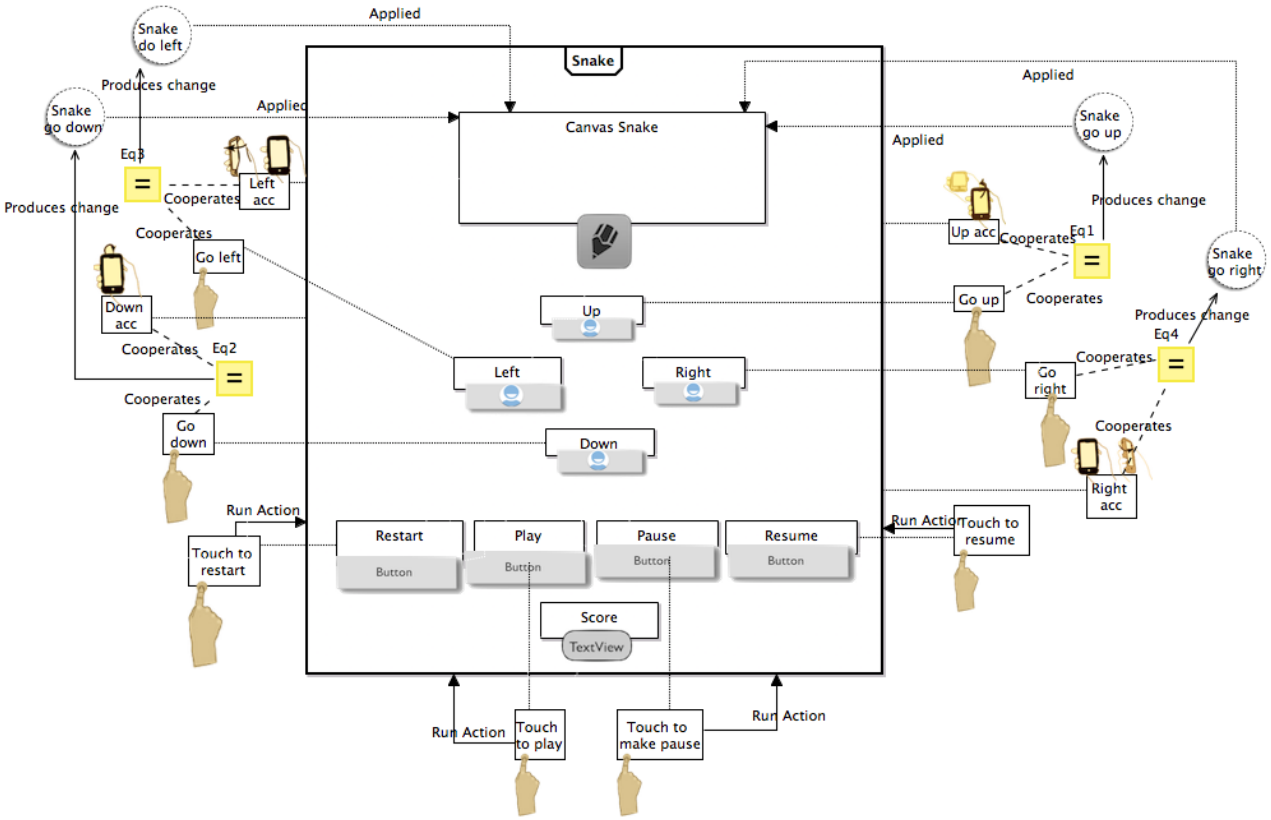




B.6 Exemple 6 : « Jeu de serpent multimodal »

Pour orienter le serpent dans ce jeu, l'utilisateur peut utiliser le pointage tactile (avec les boutons tactiles) ou l'orientation du téléphone (vers le haut, le bas, la gauche ou la droite).





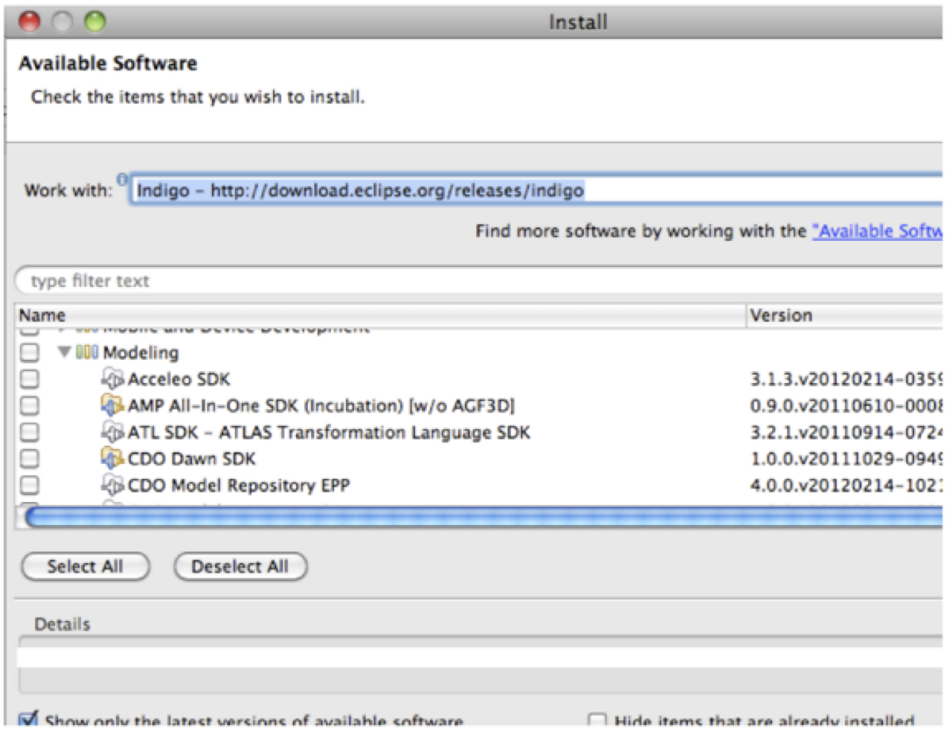
Annexe C

La documentation de MIMIC

How to install the framework?

1. Download ObeoDesigner (from Bundle)

2. Install OCL (Object Constraint Language) : start Eclipse (ObeoDesigner), then select Help > Install New Software. Click Add, in the top-right corner. In the Add Repository dialog that appears, enter the following URL for the Location : http://www.obeo.fr/download/eclipse/juno-3.8 and click Ok. In the Available Software dialog, select “Modeling” and then “OCL”. Click Next. Read and accept the license agreements, then click Finish.



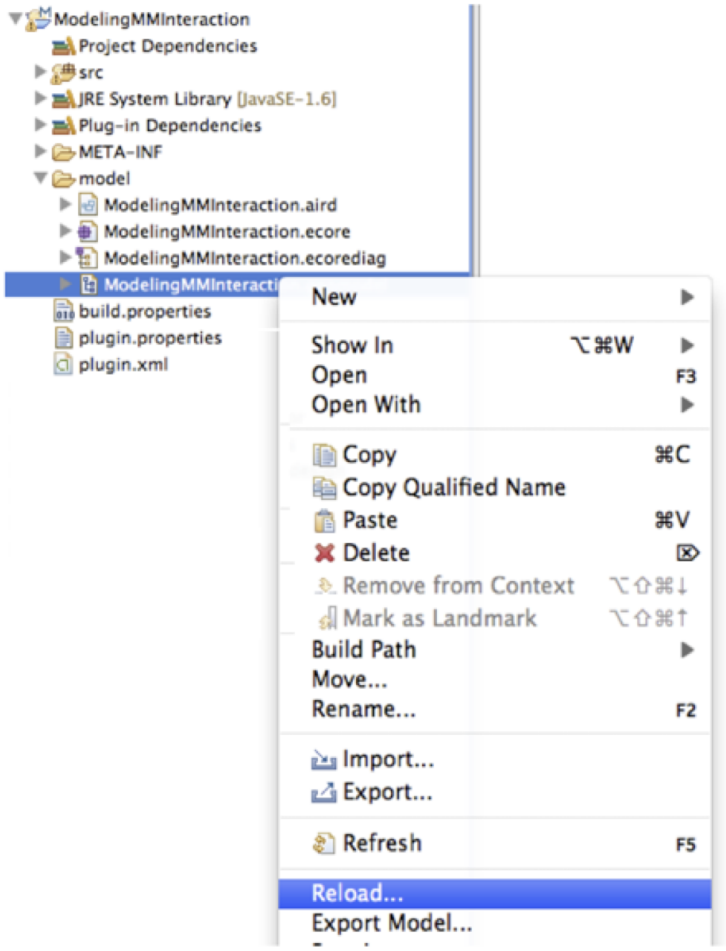
3. Install Xtext 2.4.1 (OCL editor is implemented with Xtext) from : http://download.eclipse.org/modeling/tmf/xtext/updates/composite/releases (install Xtext Runtime and Xtext UI)

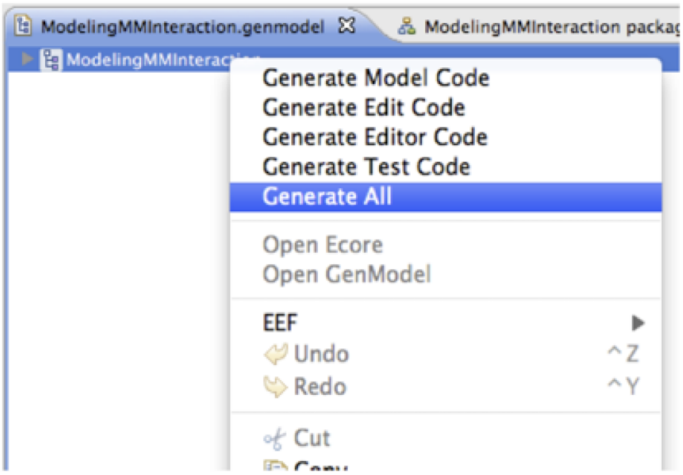
4. Install Android plugin : Help > Install New Software > Add : https://dl-ssl.google.com/android/eclipse/ (if you have trouble acquiring the plugin, try using "http" in the Location URL, instead of "https"). When the

installation completes, restart Eclipse. Then, follow this tutorial to configure the plugin.

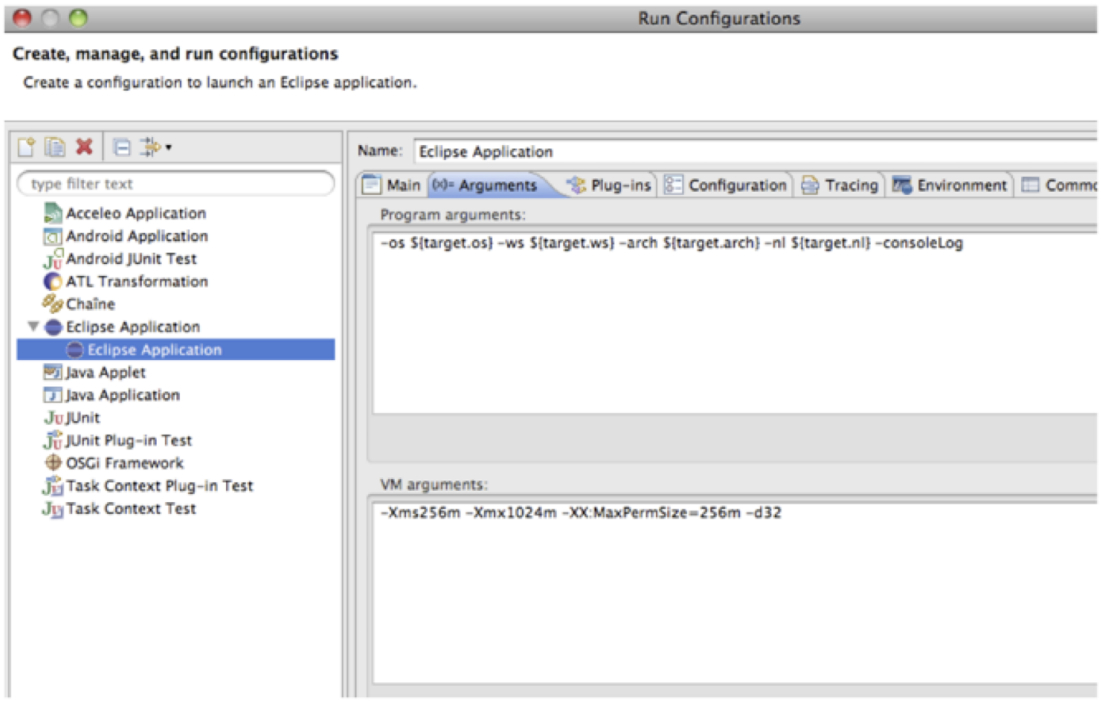
5. Import the meta-model ModelingMMInteraction.zip (18 dawnloads) of the model-based approach DSML as well as its visual notation my.modelingmminteraction.design.zip (14 dawnloads) project (File > Import > General > Existing Projects into Workspace) (projects should be already in the workspace).

6. Open the meta-model project "ModelingMMInteraction" > model folder > right click on "ModelingMMInteraction.genmodel" > Reload > follow instructions > right click on the opening file > select "Generate all"





7. In order to launch a new eclipse application click on Run > Run Configurations and double click on Eclipse Application to get a New_configuration. You have to specify VM arguments in this new configuration as follows: "-Xms256m -Xmx1024m -XX:MaxPermSize=256m -d32" (for Windows: "-Xms256m -Xmx1024m -XX:MaxPermSize=256m") > Run

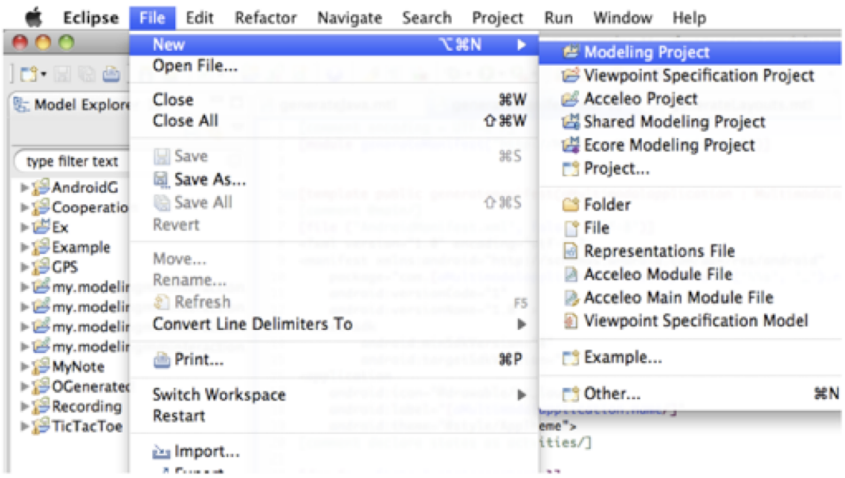


8. In the second eclipse instance you can import the Android generator Generator.zip (15 dawnloads) . Import "my.modelingmminteraction.acceleo.Java", "my.modelingmminteraction.acceleo.Layouts", "my.modelingmminteraction.acceleo.Manifest" and "my.modelingmminteraction.acceleo.Menu". You can also import the HTML5/CSS3 generator Generatorhtml.zip (19 dawnloads) . Import "my.modelingmminteraction.acceleo.Html5" and "my.modelingmminteraction.acceleo.Css"

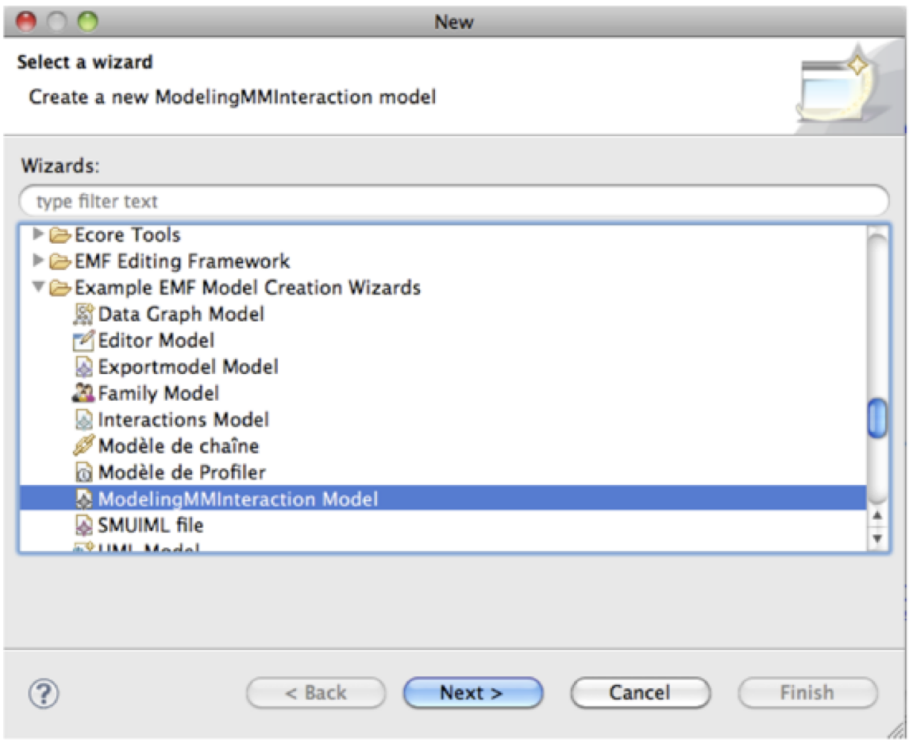
9. To creat HTML5 projects you can install the Aptana plugin for Eclipse: Help > Install New Software > Add : https://download.aptana.com/studio3/plugin/install (if you have trouble acquiring the plugin, try using "http" in the Location URL, instead of "https")

A quick start

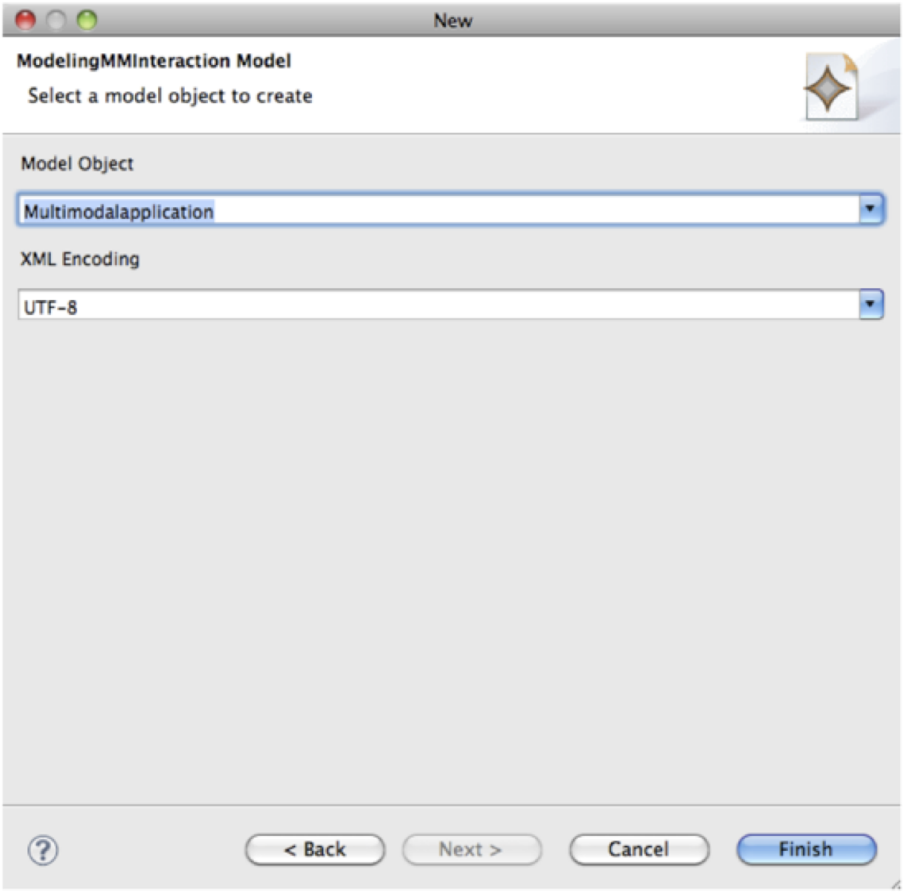
1. From the second eclipse instance : File > New > Modeling project (chose a name, for example "MobileMultimodality")



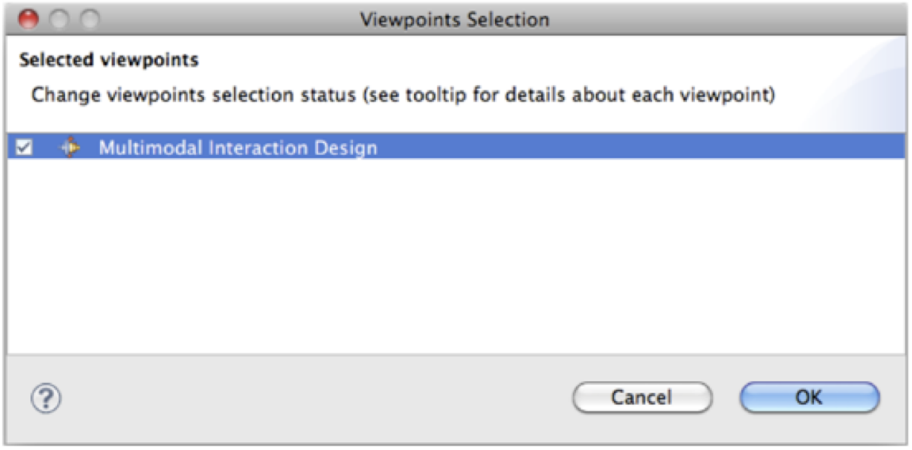
2. Right click on the created modeling project > New > other. Then select the "ModelingMMInteraction" Model from the Example EMF Model Creation Wizards.



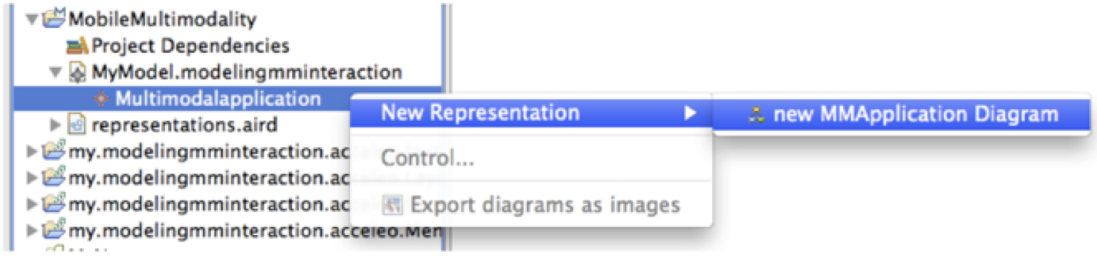
3. Select "Multimodalapplication" as the root Domain Class of your Domain Model (then select the model name, for example "MyModel")



4. To create a graphical designer, right click in the modeling project > Viewpoints selection > select "Multimodal Interaction Design"

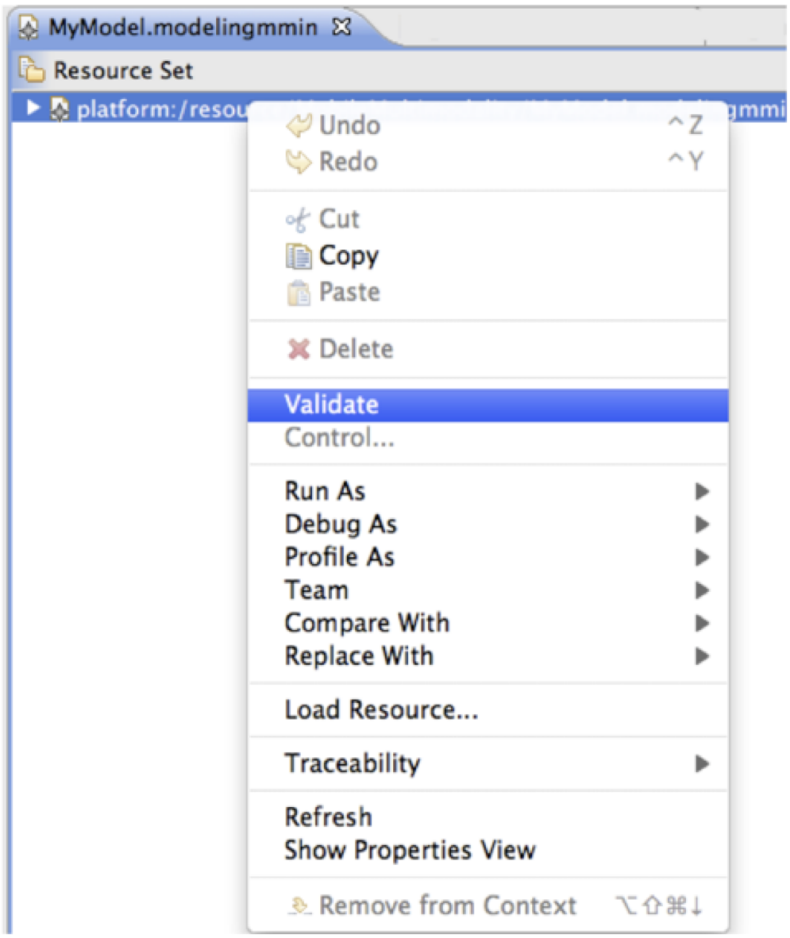


5. Then right-click on the created model > select New Representation / new MMApplication diagram (then select a graphic représentation name)

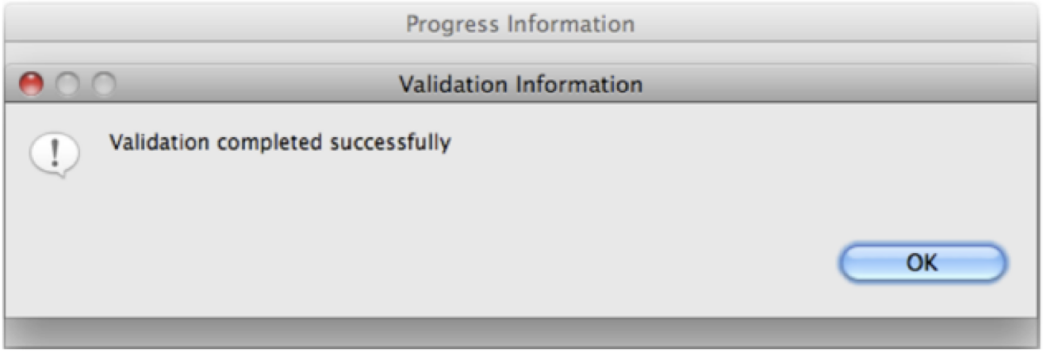


Model checking

1. After modeling, you can validate the model using OCL constraints. Double click on the model (not the representation) > right click on the first element displayed on the opening window > Validate

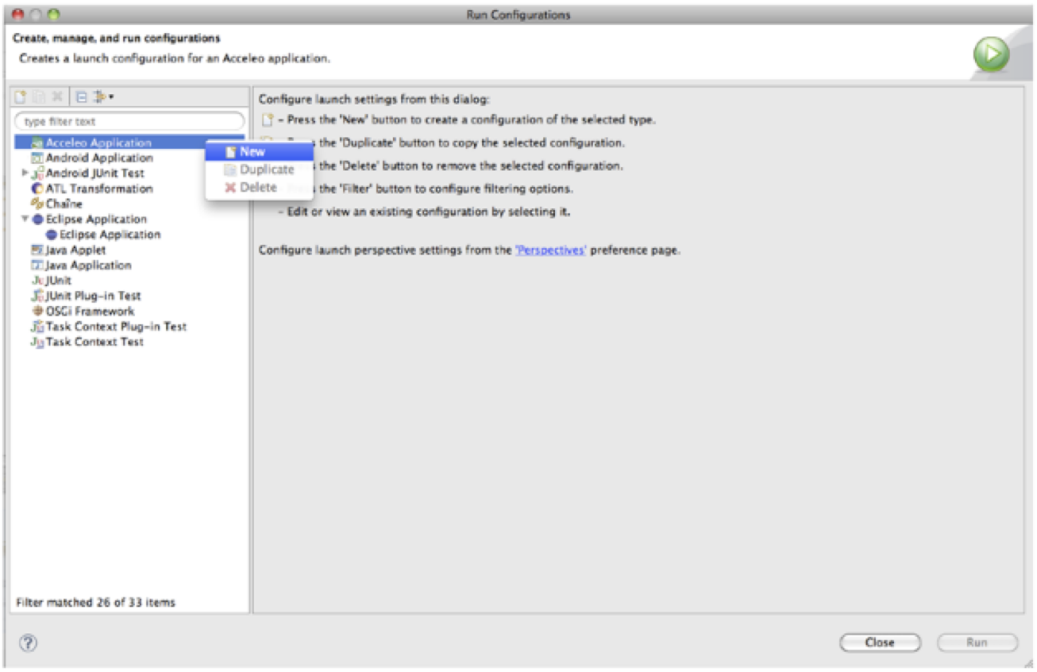


2. The validation detects errors or tells you that the model is validated



Code generation

1. After model validation, you can generate code of the modeled application. You start by configuring the four generators (Java, Layout, Manifest, Menu). Click on Run > Run configuration > select Acceleo Application > right click > New



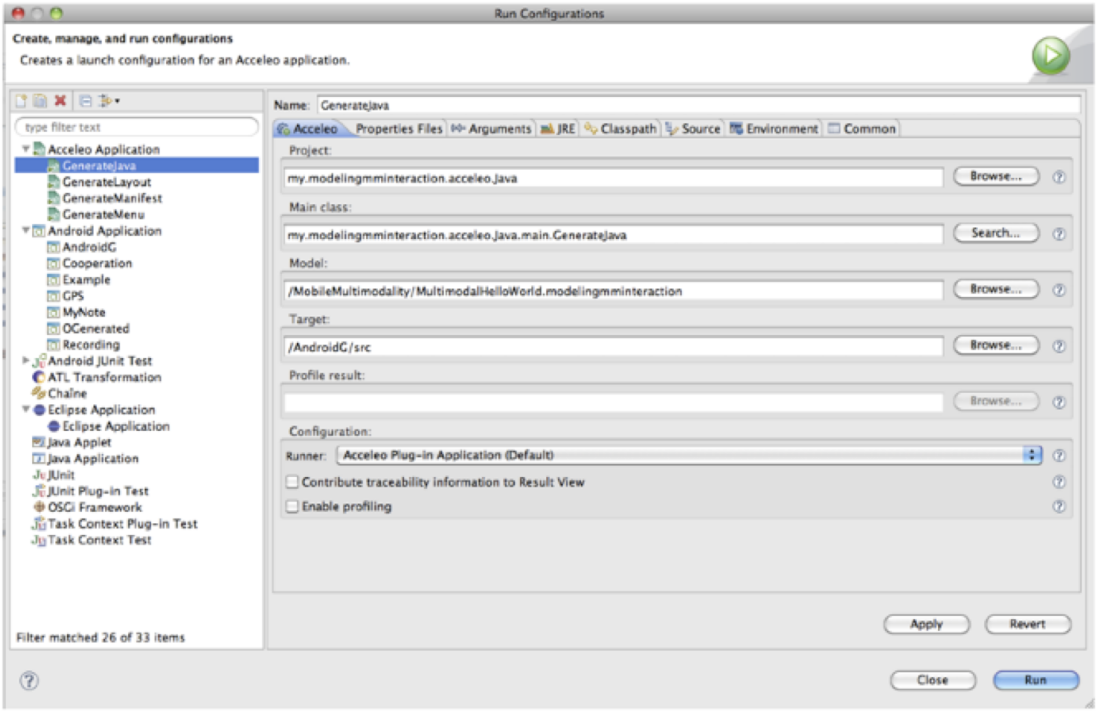
2. For each generator you have to create a new Acceleo application where you specify the Acceleo project, the main class of the project, the model to transfer and the target folder that will contain the generated code.

3. The acceleo project is “my.modelingmminteraction.acceleo.Java” for Java generator,
“my.modelingmminteraction.acceleo.Layouts” for Layout generator,
“my.modelingmminteraction.acceleo.Manifest” for Manifest generator and
“my.modelingmminteraction.acceleo.Menu” for Menu generator.

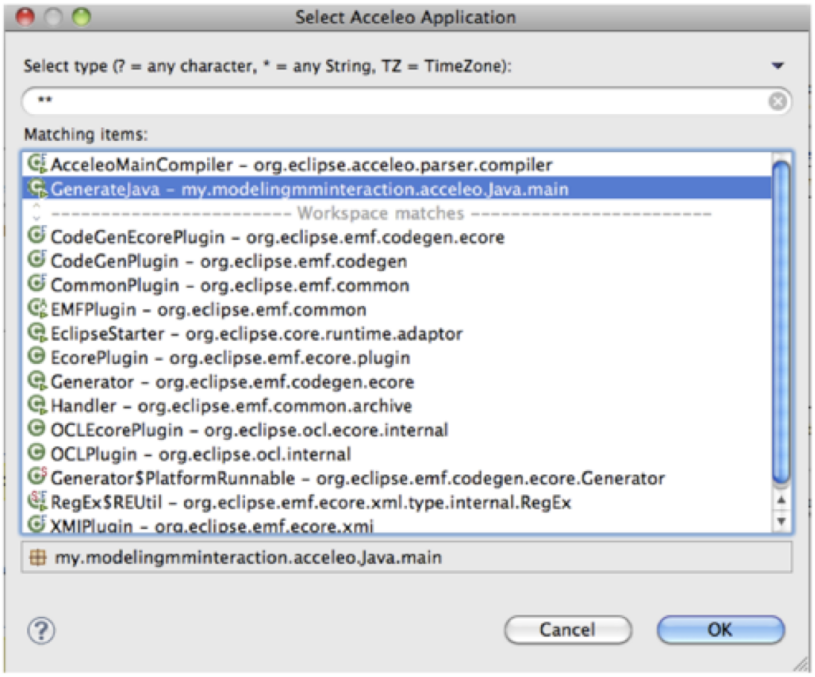
4. The Main class is “my.modelingmminteraction.acceleo.Java.main.GenerateJava” for Java generator,
“my.modelingmminteraction.acceleo.Layouts.main.GenerateLayouts” for Layout generator,
“my.modelingmminteraction.acceleo.main.GenerateManifest” for Manifest generator and
“my.modelingmminteraction.acceleo.Menu.main.GenerateMenu” for menu generator

5. The Model, ie the model you created to model your application (MyModel, for example),

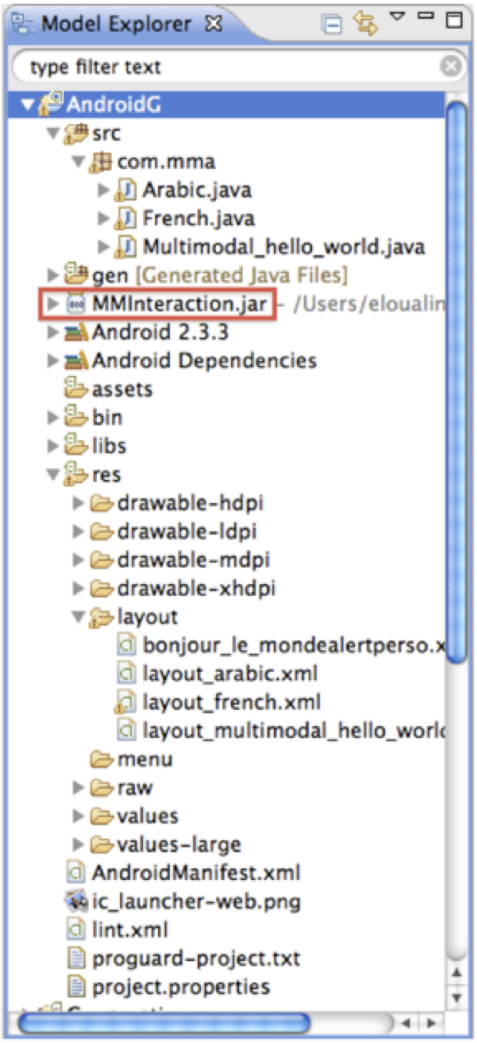
6. The Target, ie the Android project (for Java you specify the “src folder”, for Layout “res/layout”, for Manifest “the root project” and for Menu “res/menu”).



7. To run generators : right click on each generator > run as > select the acceleo application > Ok (the generated file will be in the specified Android project)



8. Finally and in order to execute the generated interface you add this API (MMInteraction.jar) to the Android project.



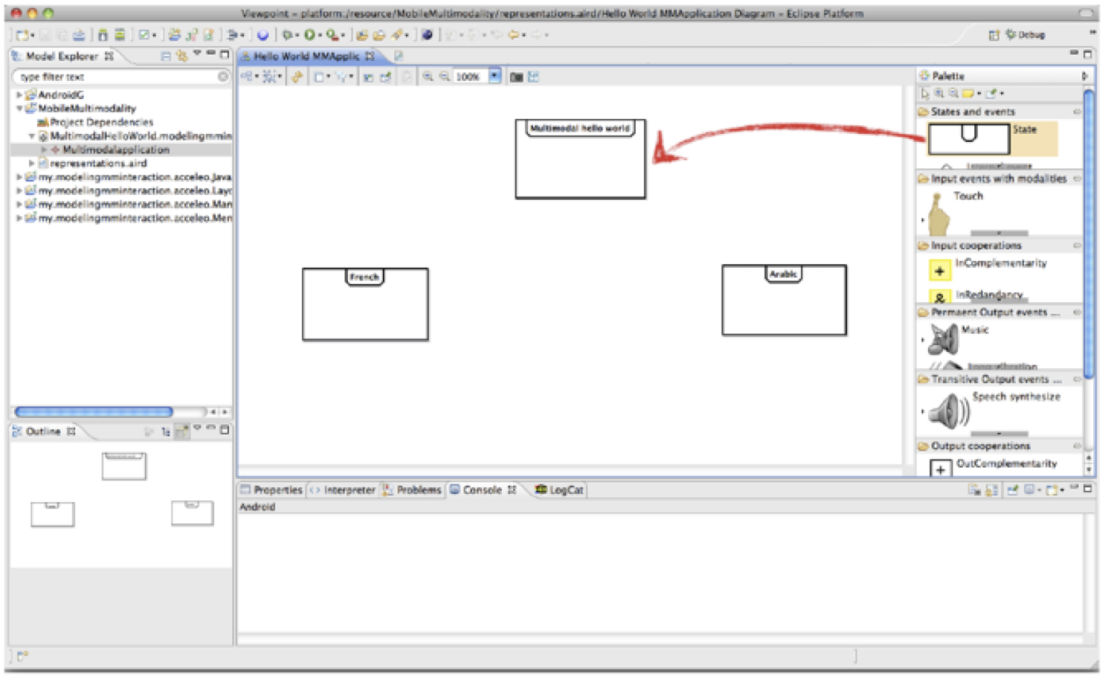
Complete example

This section explains how to model your first graphical multimodal mobile application using our framework. We are based on a simple example called "Multimodal hello world". The user can manipulate the application using several interaction modalities (tactile gestures, shaking the phone, making long touches) and the system responds using three output modalities (displaying, speech synthesis, vibration). Using the tactile gestures, the user writes A to get the Arabic translation of Hello world, then to return to the previous screen, she/he applies a long touch on the displayed text and shakes the phone quickly (input complementarity) or uses the back button of Android. Writing F on the first activity will send the user to the French translation, where, in addition, she/he can trigger the translation speech synthesis by applying a long touch on the text.

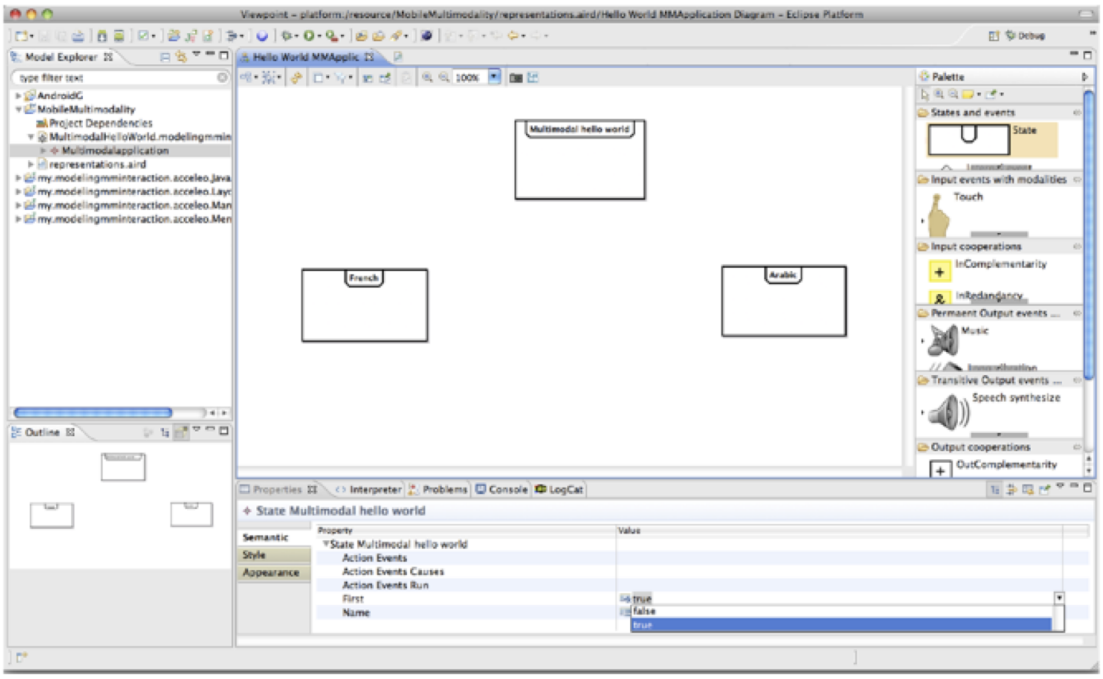
The model creation

1. First, you create the modeling project, the model that we call “MultimodalHelloWorld” and its graphic representation called “Hello world MMApplication Diagram” (as shown in Quick start).

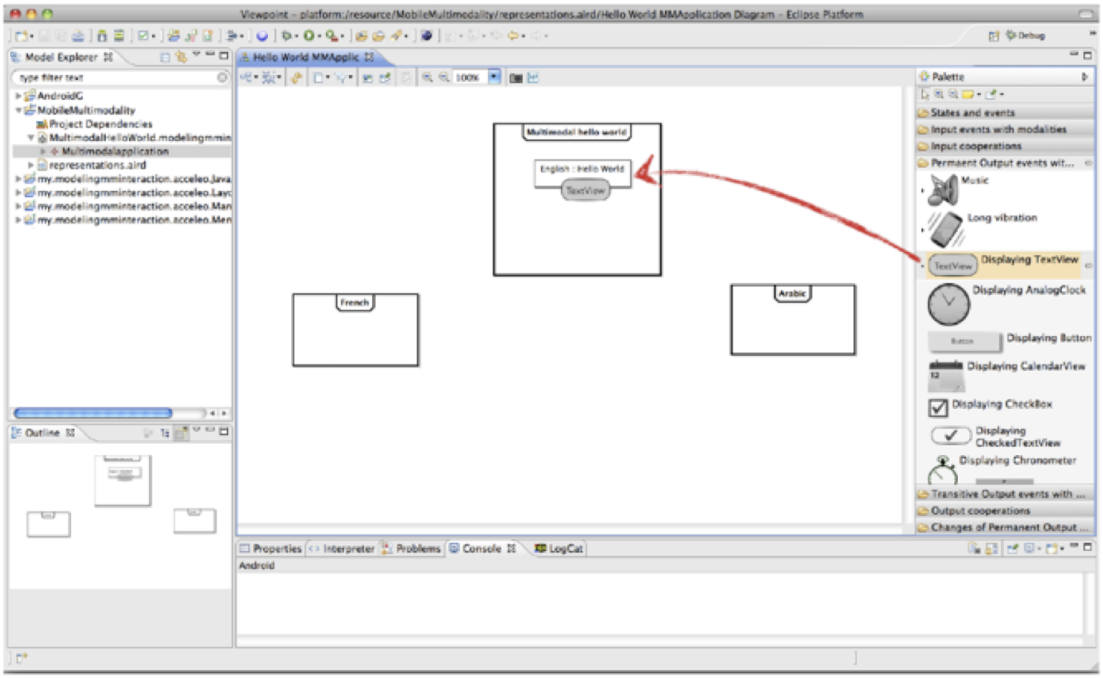
2. On the modeling space you start by defining the applications states. You drag and drop the “State” concept from the palette and gives them names: “Multimodal hello world”, “Arabic” and “French” (to write the state's name you have just to select it and start writing).

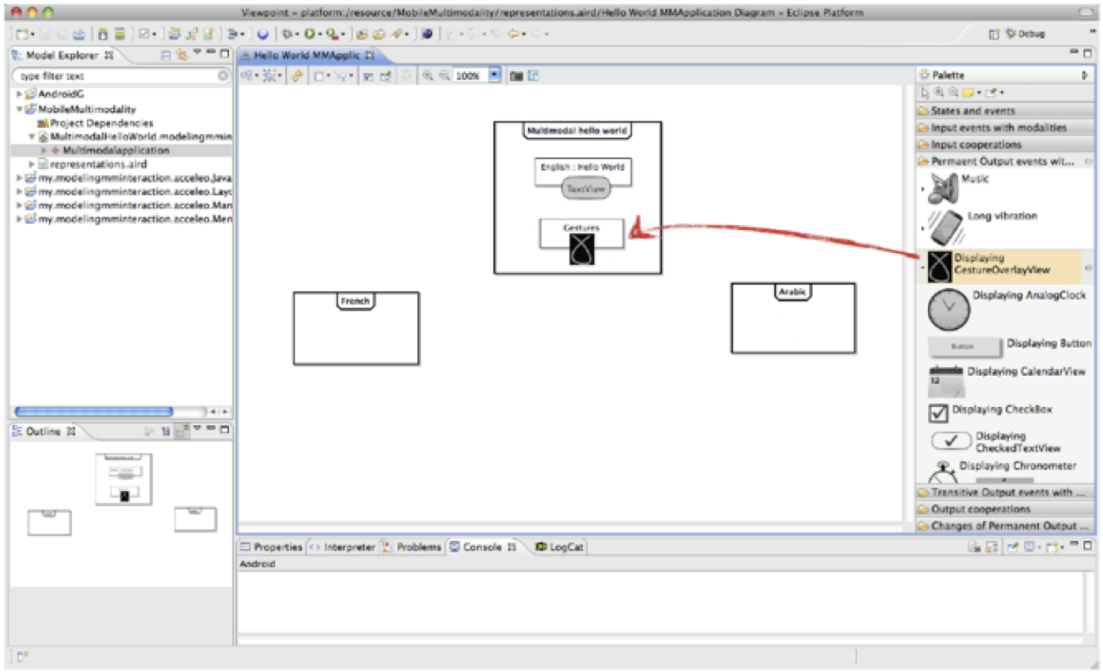


3. You have to specify the first state by selecting it and then changing its “First” property to true.

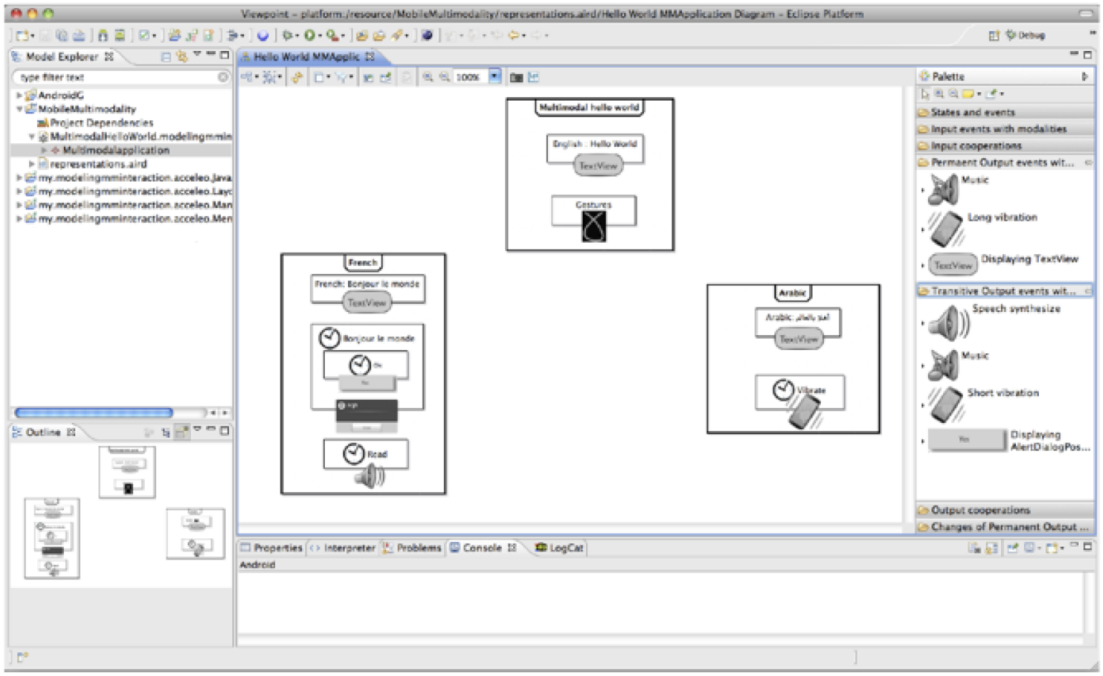


4. Then, you define the output events of the different states. First, for the "Multimodal hello world" you add a "Text View" permanent output event as the state title ("English : Hello World") and "Gesture Overlay View" that allows application users to make tactile gestures.

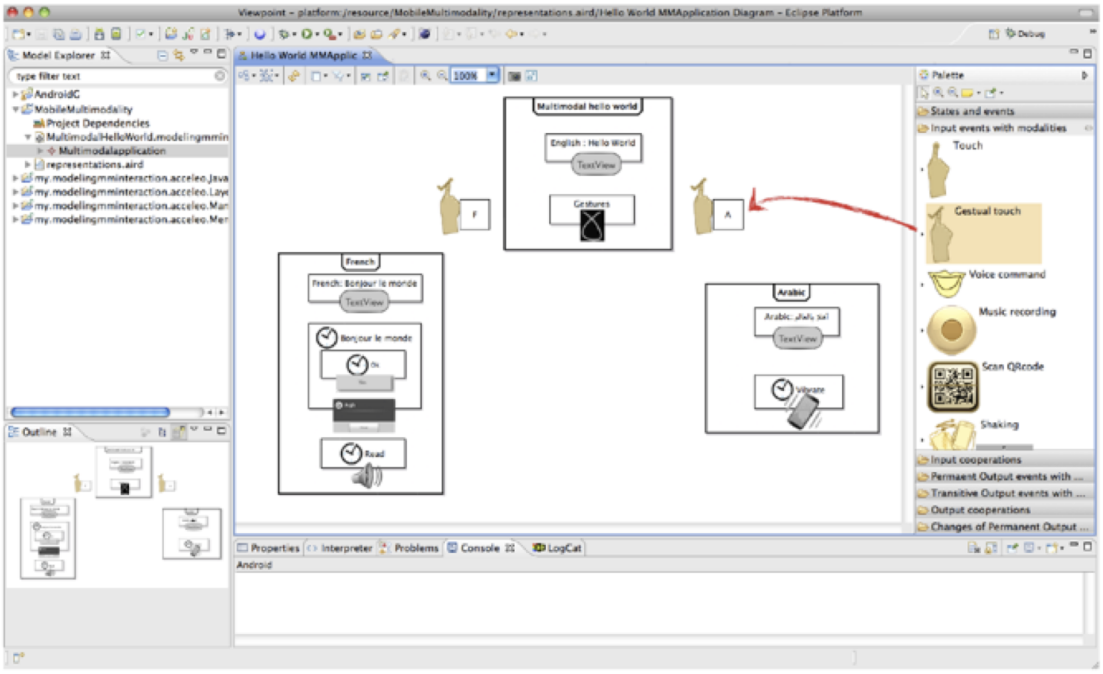


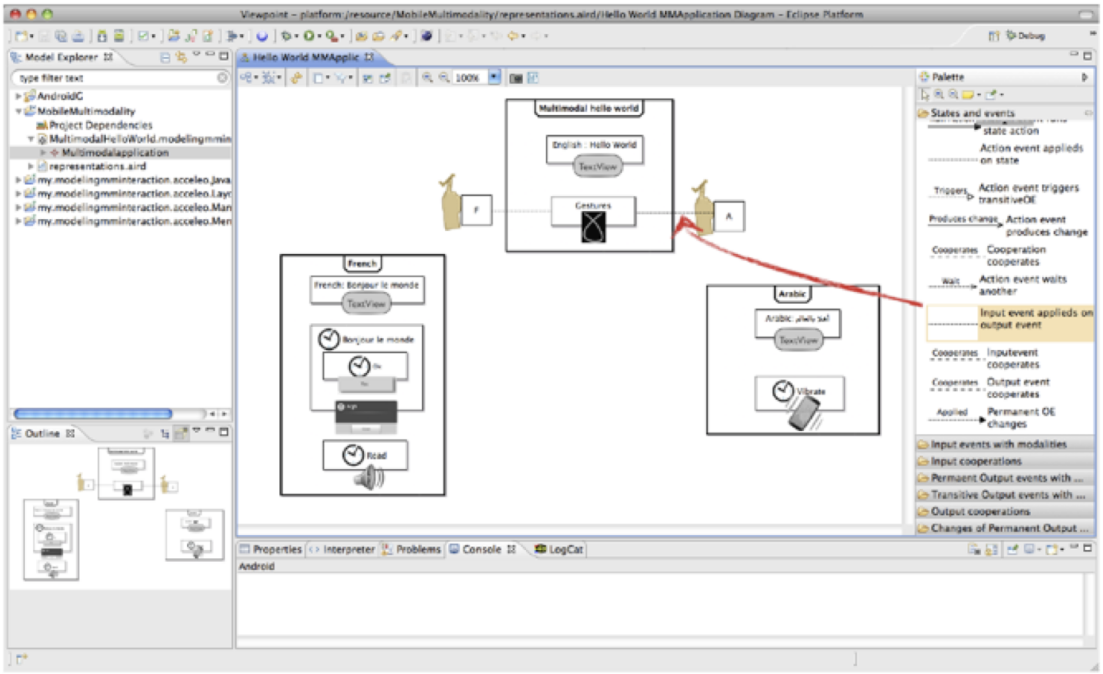


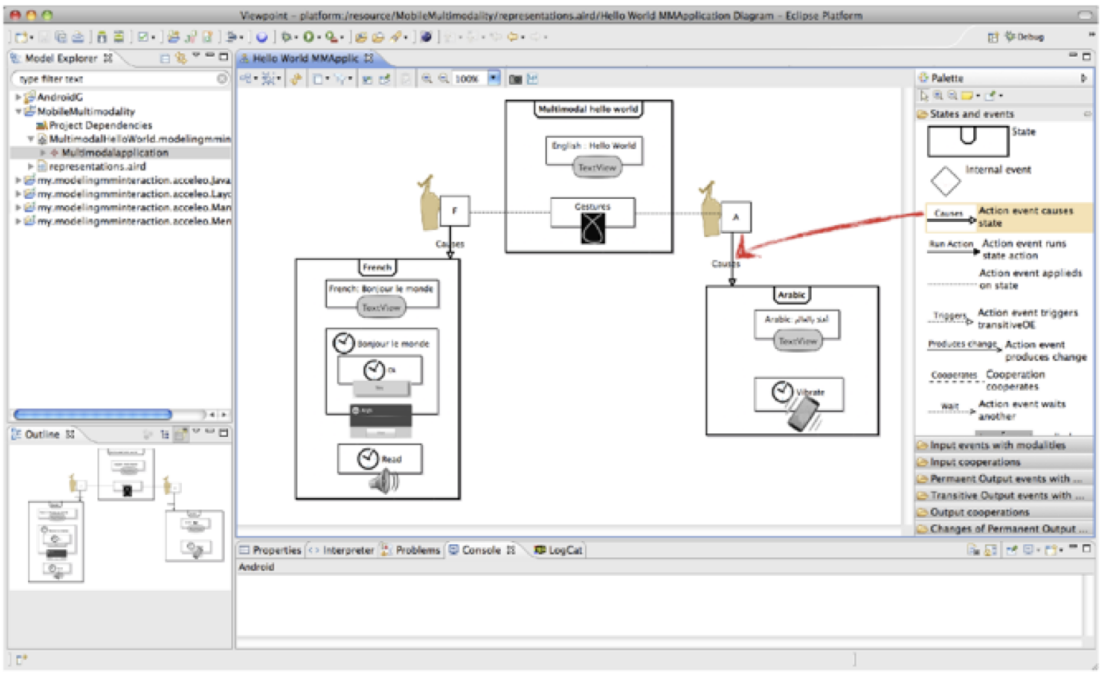
5. In the same way, you add the other output events (permanent ou transitives) from the palette to the others states.



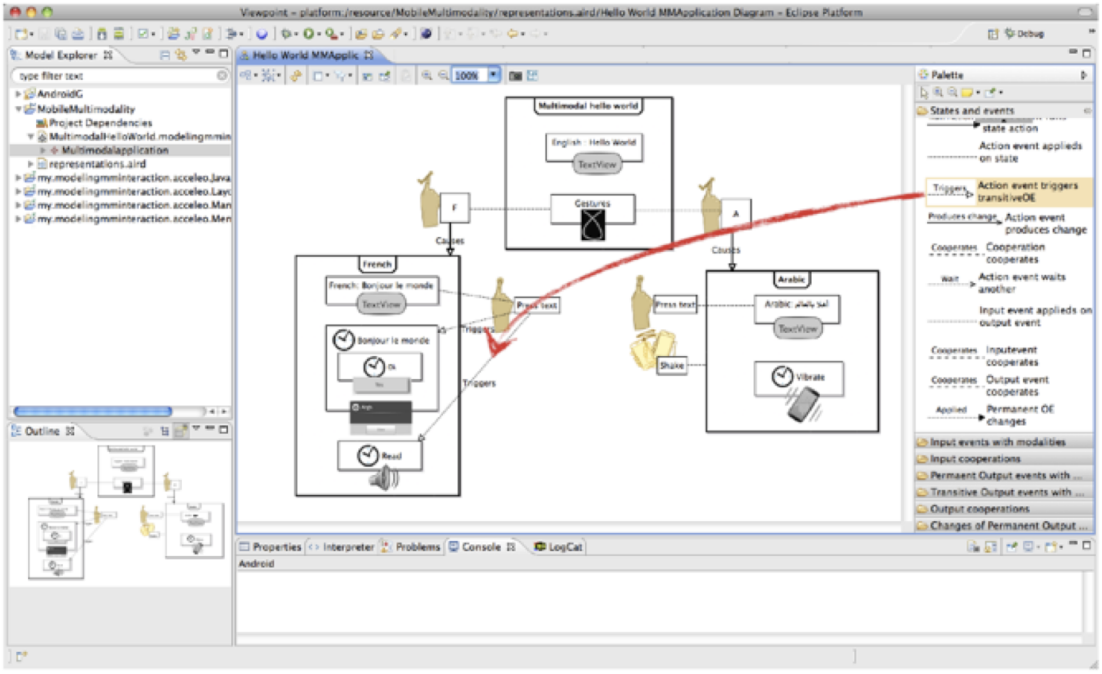
6. You add now the input event that will be applied to states or output events in order to allow interaction. The first input events are the tactile gestures applied on the “Gesture” output event to get the “Arabic” and “French” states. First, you add them to the model. Second, you associate them to the “Gesture” output event using the “Input event applied on output event” association and the two other states using the “Cause” association.



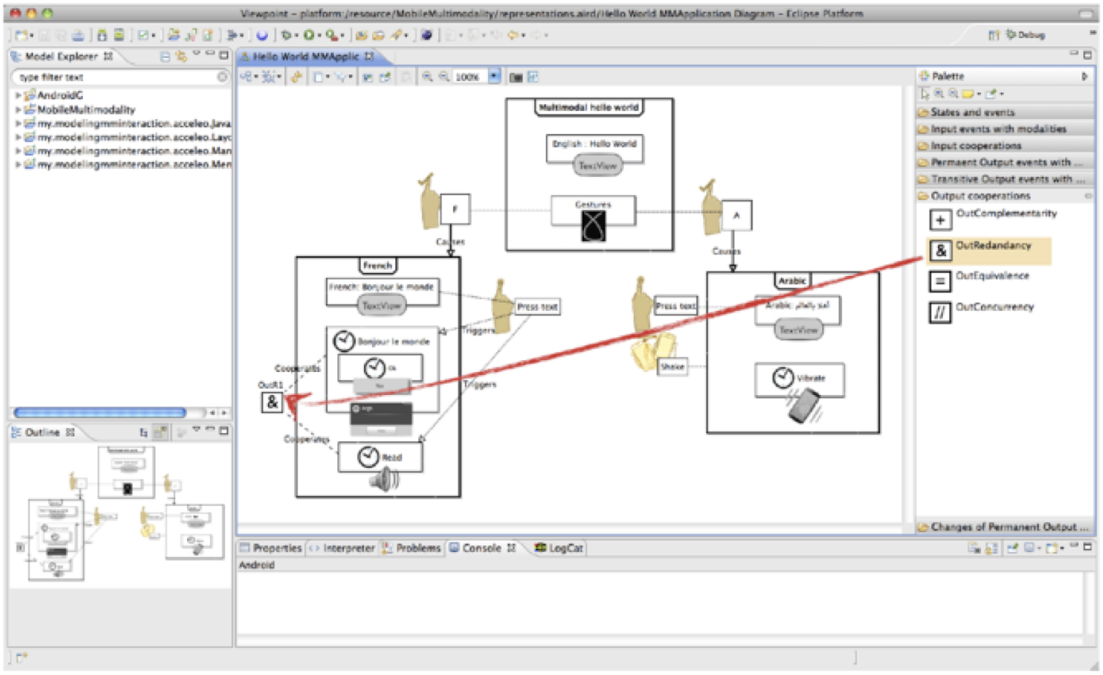




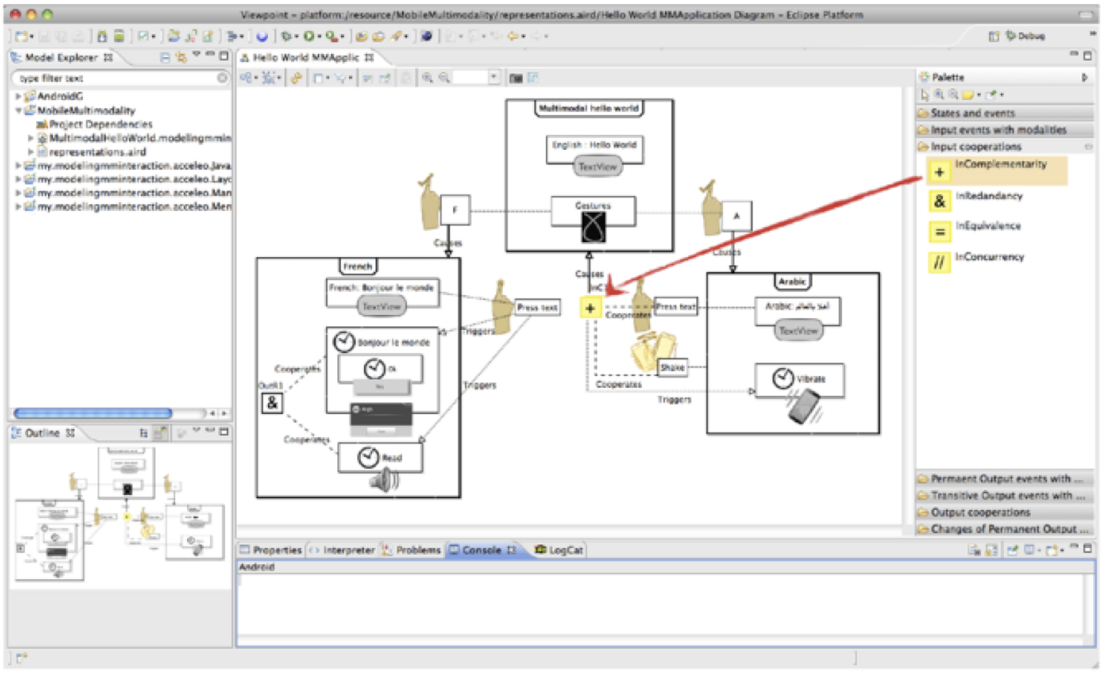
7. In the same way, you add the other input events from the palette : the long touches applied on “French: Bonjour le monde” and “Arabic: أهلا بالعالم” and the shake applied on the “Arabic” state (using the “Action event applied on state” association). The first long touch triggers the “Alert (Bonjour le monde)” as well as the “Speech synthesis (Read)” based on the “Action event triggers transitiveOE” association.



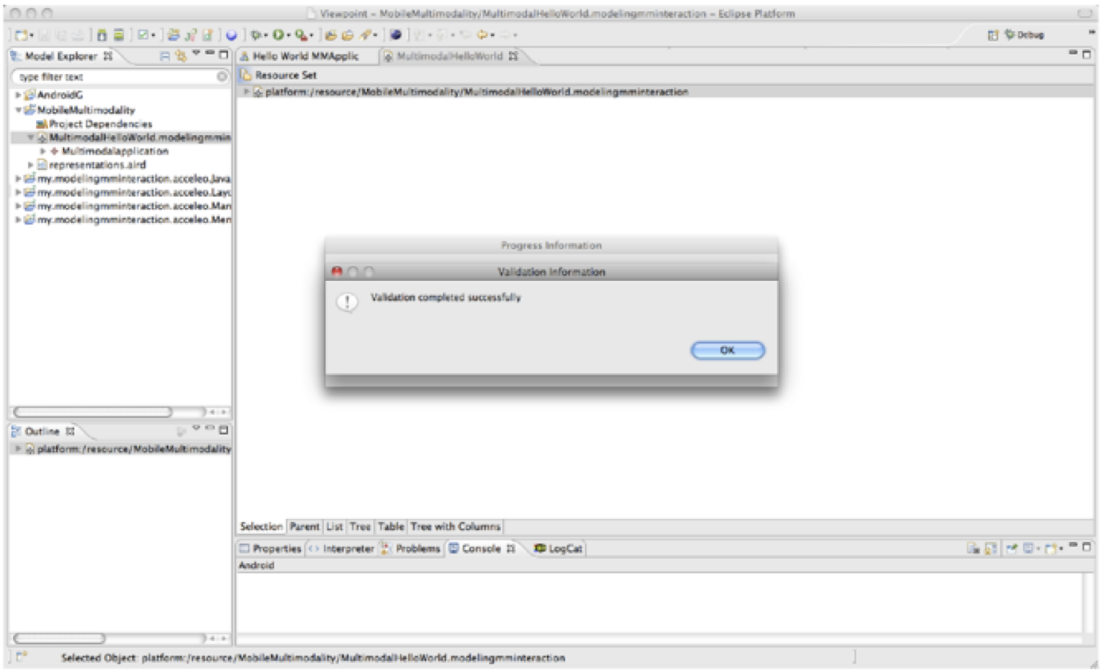
8. Now, you have to add cooperations between events. The single output combination of this example is an Output redundancy between the displayed alert and speech synthesis (“French” state). The association needed between the redundancy and output events is “Output event cooperates”.



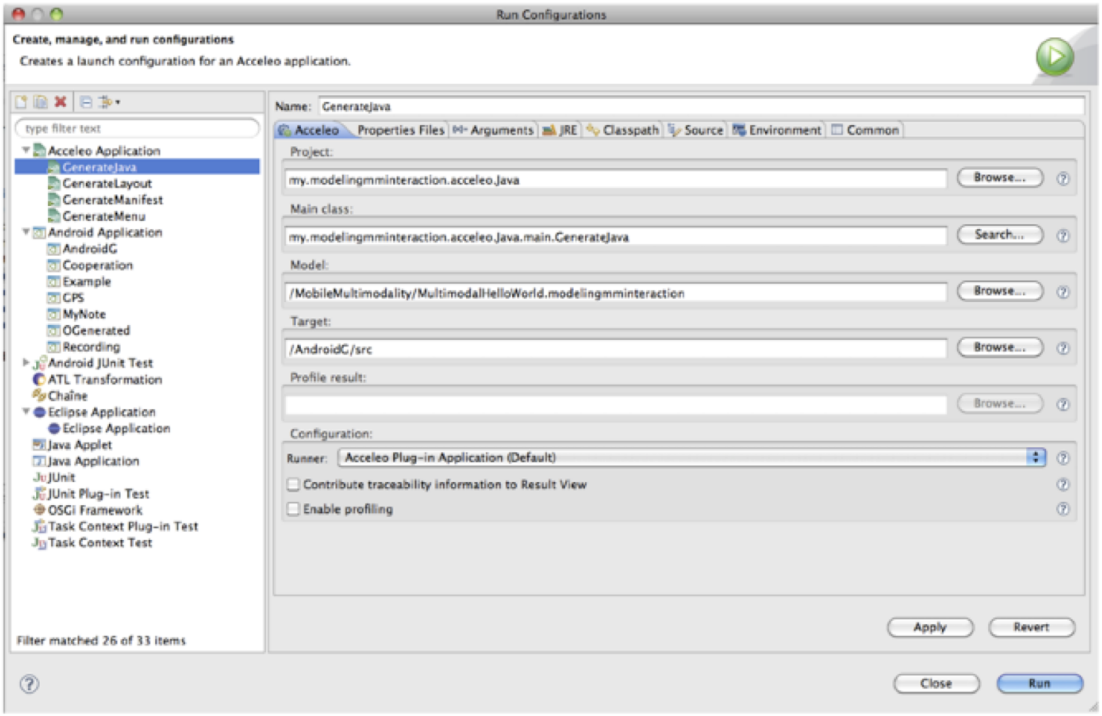
9. The input complementarity is between shake and long touch causes a transition to the first state and triggers a vibration. The order between cooperative input events is important for input complementarity. If you start by connecting the long touch to the input complementarity and then the shake, the user has to make a long touch then a shake quickly to allow the complementarity.



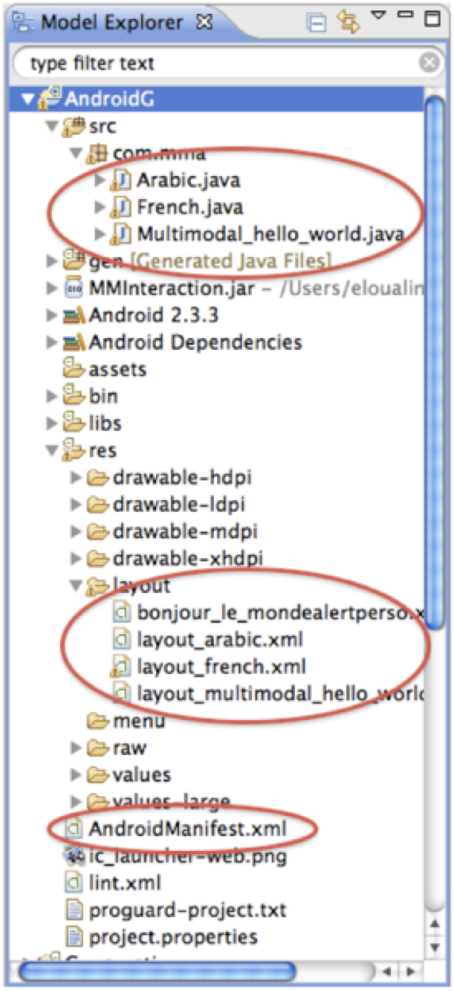
10. Now the model is complete, but you must still check it (see shown in Model checking section). If you find errors you must correct them before proceeding to the generation.



11. To generate the interface, you start by configuring generators (as shown in Code generation section). Then you run generator one after the other (do not forget to add the API (MMInteraction.jar) to Android project).



Code generated



The generated application "Multimodal Hello World" is presently available on the Google Market : https://play.google.com/store/apps/details?id=com.Hello_worldmma&feature=search_result#?t=W251bGwsMSwyLDEsImNvbS5IZWxsb193b3JsZG1tYSJd
A video is available on YouTube giving a detail about the modeling and generation of the "Multimodal Hello World" application : http://www.youtube.com/watch?v=QP5fVqo5lxE

Annexe D

Le sujet de l'évaluation

TWA- Android : TP
Modélisation et génération d'applications Android multimodales

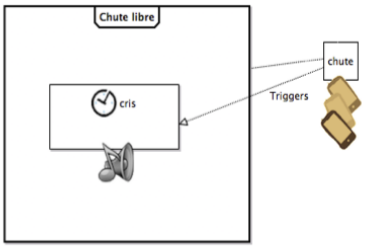
Le but de ce TP est d'évaluer un framework de création d'applications mobiles multimodales. Une application est multimodale si elle intègre deux ou plusieurs modalités (techniques ou manières) d'interaction en entrée et/ou en sortie. Par exemple, une application qui intègre d'une part le tactile et le vocal en entrée et d'autre part l'affichage et la synthèse vocale en sortie est multimodale.

Le travail se compose de trois parties :

1. **Apprentissage du framework** :

Dans cette partie nous allons faire de petits exemples avec le framework afin d'apprendre la modélisation des applications multimodales et leurs générations sous forme de projets Android.

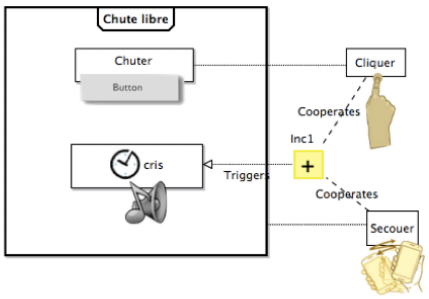
- L'objectif de cette première étape est de modéliser et générer une petite application de « Chute libre » : si le téléphone est en train de tomber, il crie ! Voici le modèle à réaliser :



- Pour modéliser (et générer) cette application, consultez (sans faire les exercices) la documentation du framework (http://www.lifl.fr/~eloualin/tool.html) :
    - Pour cela, lisez les étapes de modélisation et génération en choisissant le lien « Complete example » (http://www.lifl.fr/~eloualin/completeexample.html), puis consultez les exemples d'applications (surtout les exemples 2, 3, 4, 6, 7, 9, 10, 13, 14, 15) en choisissant le lien « Other examples » (http://www.lifl.fr/~eloualin/examples.html).

    - Réalisez ensuite le modèle de l'application « Chute libre » (cf. figure ci-dessus) et générez l'application. Testez-la ☺
- Maintenant, modifiez le modèle « Chute libre » en changeant la « chute » par une autre modalité d'interaction comme un secouage (« shake ») par exemple. Testez cette nouvelle version.

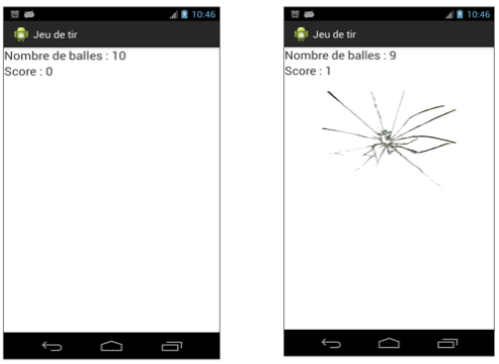
- Enfin, changez la « chute » par une complémentarité entre « secouer le téléphone » et « toucher un bouton ». Testez l'application. Voici le modèle :



2. **Réalisation** :

- Vous allez être séparé en deux groupes. Chaque développeur du premier groupe développe un exemple d'application multimodale en utilisant le framework. Les développeurs du deuxième groupe développent la même application en utilisant le SDK Android uniquement (ADT, Android Studio...).

- L'application à réaliser est un petit jeu de tir. Au début, le joueur a 10 balles. Pour tirer et gagner un point, il doit cliquer sur le bouton "+" du volume et secouer rapidement le téléphone (dans un délai d'une seconde entre ces deux évènements). Avec le tir, on joue le son d'un coup de feu et l'application affiche une image de verre brisé. À chaque fois que le joueur tire, il perd une balle et quand il n'y a plus de balle, il ne peut plus tirer. À tout moment, il peut recharger son arme en approchant sa main du capteur de proximité on en appliquant un « long touch » sur l'image du verre brisé ; ceci provoque la lecture du son de réarmement (et on affiche de nouveau 10 balles disponibles) ainsi que l'effacement du verre brisé. Voici à quoi peut ressembler l'application :



- Nous allons mesurer le temps de réalisation de l'application pour les deux groupes. **Vous avez 2 heures maximum**. Ecrivez maintenant ci-dessous vos noms et prénoms, et l'heure actuelle. Quand vous aurez fini, testé et validé l'application, vous écrirez l'heure de fin.
    - Nom prénom : ……...
    - Heure de début : ……
    - Heure de fin : ……..…..
- Quand vous avez terminé, envoyez vos projets Android de l'application "Jeu de tir" (code source et APK) sous la forme d'un zip, par mail à elouali_nadia1@yahoo.fr et jeanclaude.tarby@gmail.com.
    - Attention, le mail de l'Université limite la taille à 5 Mo. Si votre mail est trop volumineux, utilisez un autre moyen (dropbox, Google drive…) et indiquez-nous par mail comment récupérer votre travail.
- Les utilisateurs du framework doivent aussi rendre le projet de modélisation de l'application.
    - Pour cela, joignez au zip votre « Modeling project » qui contiendra le modèle de « Jeu de Tir ».

3. **Questionnaire** (uniquement pour les utilisateurs du framework) :
Merci de répondre au questionnaire en ligne suivant (10 minutes) : https://docs.google.com/forms/d/1ehWU8K7syAzsfrck-CmFnbosrxwuSF0Q1zweNQokBoU/viewform

Remarque : La version électronique du sujet et les ressources d'applications (sons et images) dont vous aurez besoin pour le développement sont disponibles sur : http://www.lifl.fr/~eloualin/Evaluation.zip

Annexe E

Le questionnaire de l'évaluation

Questionnaire pour la création des interfaces mobiles multimodales

Merci de prendre quelques minutes afin de répondre à ce questionnaire concernant le framework que vous avez utilisé pour créer des interfaces mobiles multimodales.

* Required

1. **Sex** *
   Mark only one oval.
   - ◯ Féminin
   - ◯ Masculin

2. **Age** *
   .....

3. **Expertise en développement mobile** *
   Mark only one oval.
   - ◯ Expert
   - ◯ Compétent
   - ◯ Débutant avancé

4. **Préférez-vous utiliser des frameworks pour la génération de code?** *
   Mark only one oval.
   - ◯ Oui
   - ◯ Non
   - ◯ Je ne sais pas

5. **Pourquoi ?**
   .....
   .....
   .....
   .....
   .....

6. **Avez-vous déjà utilisé des frameworks pour la génération de code ?** *
   Code Java, JavaScript…
   Mark only one oval.
   - ◯ Oui
   - ◯ Non

7. **Si oui, lesquels ?**

.....

.....

.....

.....

.....

8. **De manière globale, comment trouvez-vous le framework que vous avez utilisé pour développer des interfaces mobiles multimodales ?** *
Mark only one oval.

- ◯ Très utile
- ◯ Utile
- ◯ Pas très utile
- ◯ Pas utile

9. **Comment évaluez-vous l'assistance (aide, vérification, guidage) du framework pendant la modélisation ?** *
Mark only one oval.

- ◯ Très satisfaisant
- ◯ Satisfaisant
- ◯ Assez satisfaisant
- ◯ Pas satisfaisant

10. **Comment trouvez-vous les symboles et les icônes de la palette du framework ?** *
Mark only one oval.

- ◯ Très claire
- ◯ Claire
- ◯ Assez claire
- ◯ Pas claire

11. **Pouvez-vous nous indiquer les icônes que vous ne trouvez pas claire ?**

.....

.....

.....

.....

.....

12. **Comment évaluez-vous la tâche de modélisation avec le framework ?** *
Mark only one oval.

- ◯ Très comlexe
- ◯ Complexe
- ◯ Assez complexe
- ◯ Pas complexe

13. **Pouvez-vous évaluer l'effort que vous avez fournie pour créer l'application "Jeu de tir" ?** *
Mark only one oval.

- ◯ Beaucoup plus d'effort (beaucoup plus que celui nécessaire pour l'utilisation du SDK)
- ◯ Un peu plus d'effort (un peu plus que celui nécessaire pour l'utilisation du SDK)
- ◯ Un peu moins d'effort (un peu moins que celui nécessaire pour l'utilisation du SDK)
- ◯ Beaucoup moins d'effort (beaucoup moins que celui nécessaire pour l'utilisation du SDK)

14. **Pouvez-vous évaluer le temps que vous avez passé pour créer l'application "Jeu de tir" ?** *
Mark only one oval.

- ◯ Beaucoup plus de temps (beaucoup plus que celui nécessaire pour l'utilisation du SDK)
- ◯ Un peu plus de temps (un peu plus que celui nécessaire pour l'utilisation du SDK)
- ◯ Un peu moins de temps (un peu moins que celui nécessaire pour l'utilisation du SDK)
- ◯ Beaucoup moins de temps (beaucoup moins que celui nécessaire pour l'utilisation du SDK)

15. **Merci de donner votre appréciation générale par rapport à l'utilisation du framework**
Remarques, idées, suggestions...

.....

.....

.....

.....

.....

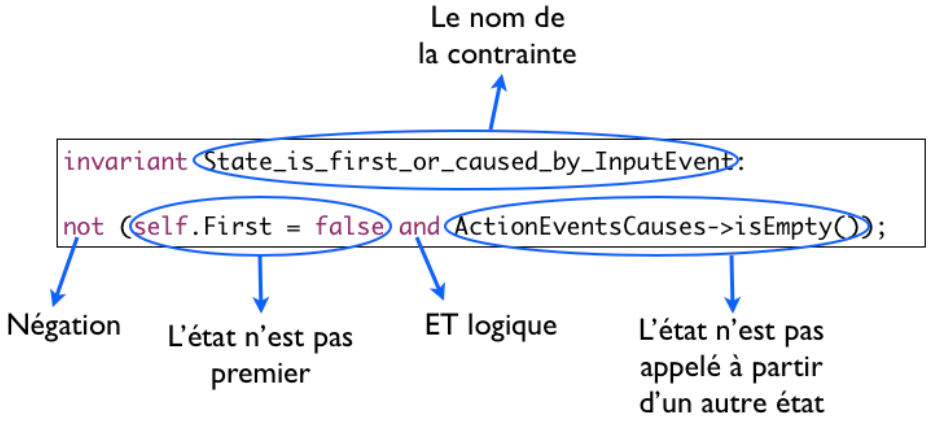
Annexe F

Les contraintes OCL syntaxiques et ergonomiques

Sommaire

F.1 Les contraintes OCL syntaxiques et ergonomiques

– Un état d'interaction doit avoir l'attribut « First » égale à « true » ou être appelé par d'autres états.



– Les noms des évènements (en entrée, en sortie ou interne) sont obligatoires.

```
invariant Name_is_obligatory_for_inputevent_and_internelevent:
      if  self.oclIsTypeOf(Inputevent)  or  self.oclIsTypeOf
(Internalevent) then self.name <> null else true endif;
```

– Un évènement en entrée doit être reçu par un évènement en sortie ou un état d'interaction.

```
invariant InputEvent_must_be_applied_on_state_or_OutputEvent:
      not (OutputEventTrigger.oclAsSet()->isEmpty() and
StateApplied.oclAsSet()->isEmpty());
```

– Un évènement interne doit être reçu par un état d'interaction.

```
invariant InternatEvent_must_be_applied_on_a_state: not
StateApplied.oclAsSet()->isEmpty();
```

– Un évènement transitoire en sortie doit être déclenché par un évènement en entrée ou un évènement interne.

```
invariant
TransitiveOutputEvent_have_to_be_declanched_by_an_InputEvent_o
r_an_InternatEvent:
      if self.oclContainer().oclIsTypeOf(Transitiveoe) or
self.OutEType.name = 'Menu' then self.ActionEvents->size() = 0
else self.ActionEvents->size() > 0 endif;
```

– Pour déclencher un évènement transitoire en sortie ou une action système, l'évènement déclencheur (évènement internet, en entrée ou combinaisons d'évènements) doit être appliqué sur le même état d'interaction que l'évènement ou l'action.

```
invariant
To_trigger_a_TransitiveOutputEvent_InternalEvents_have_to_be_a
pplied_on_the_same_state:
      if self.ActionEvents->select(oclIsKindOf
(Internalevent))->collect(oclAsType(Internalevent))->select
((StateApplied <> null))->size() > 0 then self.ActionEvents-
>select(oclIsKindOf(Internalevent))->collect(oclAsType
(Internalevent))->select(StateApplied <> null)->forAll
(StateApplied.name = self.oclContainer().oclAsType
(State).name) else true endif;
invariant
To_trigger_a_TransitiveOutputEvent_InputEvents_have_to_be_appl
ied_on_the_same_state:
      if self.ActionEvents->select(oclIsKindOf(Inputevent))-
>collect(oclAsType(Inputevent))->select((StateApplied <>
null))->size() > 0 then self.ActionEvents->select(oclIsKindOf
(Inputevent))->collect(oclAsType(Inputevent))->select
(StateApplied <> null)->forAll(StateApplied.name =
self.oclContainer().oclAsType(State).name) else if
self.ActionEvents->select(oclIsKindOf(Inputevent))->collect
(oclAsType(Inputevent))->select((OutputEventTrigger <> null))-
>size() > 0 then self.ActionEvents->select(oclIsKindOf
(Inputevent))->collect(oclAsType(Inputevent))->select
(OutputEventTrigger <> null)->forAll
(OutputEventTrigger.oclContainer().oclAsType(State).name =
self.oclContainer().oclAsType(State).name) else true endif
endif;
```

– Les évènements participant à une combinaison doivent être appliqués sur le même état.

```
invariant
InputEvents_of_an_InputCooperation_have_to_be_applied_on_the_s
ame_State:
      self.InputEvents->select(OutputEventTrigger <> null or
StateApplied <> null)->forAll(ine1 : Inputevent, ine2 :
Inputevent | ine1.StateApplied.name =
ine2.OutputEventTrigger.oclContainer().oclAsType(State).name
or ine1.OutputEventTrigger.oclContainer().oclAsType
(State).name = ine2.StateApplied.name or
ine1.OutputEventTrigger.oclContainer().oclContainer
().oclAsType(State).name = ine2.StateApplied.name or
ine1.StateApplied.name = ine2.OutputEventTrigger.oclContainer
().oclContainer().oclAsType(State).name or
ine1.StateApplied.name = ine2.StateApplied.name or
ine1.OutputEventTrigger.oclContainer().oclAsType(State).name =
ine2.OutputEventTrigger.oclContainer().oclAsType(State).name
or ine1.OutputEventTrigger.oclContainer().oclContainer
().oclAsType(State).name =
ine2.OutputEventTrigger.oclContainer().oclAsType(State).name
or ine1.OutputEventTrigger.oclContainer().oclAsType
(State).name = ine2.OutputEventTrigger.oclContainer
().oclContainer().oclAsType(State).name or
ine1.OutputEventTrigger.oclContainer().oclContainer
().oclAsType(State).name =
ine2.OutputEventTrigger.oclContainer().oclContainer
().oclAsType(State).name);
```

– Une application doit avoir au moins un état initial (dont l'attribut « First » égale à « True »).

```
invariant
MultimodalMobileApplication_must_have_one_first_State:
      if self.statescontents->size() > 0 then
self.statescontents->select(First = true)->size() = 1 else
self.statescontents->select(First = true)->size() = 0 endif;
```

– Un état d'interaction ne peut avoir qu'un seul évènement en sortie de type « Menu ».

```
invariant Only_one_options_menu_by_state:
      self.outputeventscomponents->collect(OutEType)-
>select((name = 'Menu'))->size() <= 1;
```

– Un évènement en sortie de type « Menu » ne peut être en relation d'inclusion qu'avec des évènements en sortie de type « MenuItem ».

```
invariant Menu_contains_MenuItem_only:
      self.OutEType.name = 'Menu' and
self.outputeventscomponents->size() > 0 implies
self.outputeventscomponents->collect(OutEType)->select(name
<> 'MenuItem')->size() = 0;
```

– Un évènement en sortie de type « Gallery » ne peut être en relation d'inclusion qu'avec des évènements en sortie de type « ImageView ».

```
invariant Gallery_contains_ImageViews_only:
      self.OutEType.name = 'Gallery' and
self.outputeventscomponents->size() > 0 implies
self.outputeventscomponents->collect(OutEType)->select(name
<> 'ImageView')->size() = 0;
```

– Un évènement en sortie de type « ListView » ne peut être en relation d'inclusion qu'avec des évènements en sortie de type « ListViewItem ».

```
invariant ListView_contains_ListViewItems_only:
      self.OutEType.name = 'ListView' and
self.outputeventscomponents->size() > 0 implies
self.outputeventscomponents->collect(OutEType)->select(name
<> 'ListViewItem')->size() = 0;
```

– Un évènement en sortie de type « RadioGroup » ne peut être en relation d'inclusion qu'avec des évènements en sortie de type « RadioButton ».

```
invariant RadioGroup_contains_RadioButton_only:
      self.OutEType.name = 'RadioGroup' and
self.outputeventscomponents->size() > 0 implies
self.outputeventscomponents->collect(OutEType)->select(name
<> 'RadioButton')->size() = 0;
```

– Un évènement en sortie de type « GridView » ne peut être en relation d'inclusion qu'avec des évènements en sortie de type « ImageView » et « TextView ».
– Un évènement en sortie de type « AlertDialog » ne peut être en relation d'inclusion qu'avec des évènements en sortie de type « AlertDialogNegativeButton », « AlertDialogPositiveButton » ou « DialogText ».

```
invariant GridView_contains_ImageViews_or_TextView_only:
      self.OutEType.name = 'GridView' and
self.outputeventscomponents->size() > 0 implies
self.outputeventscomponents->collect(OutEType)->select((name
<> 'ImageView' and name <> 'TextView'))->size() = 0;
```

```
invariant
AlertDialog_contains_AlertDialogNegativeButton_or_AlertDialo
gPositiveButton_or_DialogText_only:
      self.OutEType.name = 'AlertDialog' and
self.outputeventscomponents->size() > 0 implies
self.outputeventscomponents->collect(OutEType)->select((name
<> 'AlertDialogNegativeButton' and name <>
'AlertDialogPositiveButton' and name <> 'DialogText'))->size
() = 0;
```

– Un évènement en sortie de type « Notification » ne peut être en relation d'inclusion qu'avec des évènements en sortie de type « DialogText ».

```
invariant Notification_contains_DialogText_only:
      self.OutEType.name = 'Notification' and
self.outputeventscomponents->size() > 0 implies
self.outputeventscomponents->collect(OutEType)->select(name
<> 'DialogText')->size() = 0;
```

– Un évènement en sortie de type « Form » ne peut être en relation d'inclusion qu'avec des évènements en sortie de type « EditText », « Button », « CheckBox », « CheckedTextView », « NumberPicker », « RadioButton », « RadioGroup », « TextView », « TimePicker » ou « Password ».

```
invariant
Form_contains_EditText_or_Button_or_CheckBox_or_CheckedTextV
iew_or_NumberPicker_or_RadioButton_or_RadioGroup_or_TextView
_or_TimePicker_or_Password_only:
      self.OutEType.name = 'Form' and
self.outputeventscomponents->size() > 0 implies
self.outputeventscomponents->collect(OutEType)->select((name
<> 'EditText' and name <> 'Button' and name <> 'CheckBox'
and name <> 'CheckedTextView' and name <> 'NumberPicker' and
name <> 'RadioButton' and name <> 'RadioGroup' and name <>
'TextView' and name <> 'TimePicker' and name <>
'Password'))->size() = 0;
```

– Un changement doit être déclenché par un évènement en entrée ou interne.

```
invariant
Changes_have_to_be_producted_by_an_InputEvent_or_InternelEve
nt: self.ActionEvent.oclAsSet()->size() > 0;
```

– Un changement de type « Edit » ne peut être appliqué que sur les évènements en sortie ayant la modalité « Displaying ».

```
invariant
Change_Edit_have_to_be_applied_on_displayed_events_only:
       self.Type = 'Edit' implies self.PermanentOEs->collect
(OutEType)->select(Outputmodality <> 'Displaying')->size() =
0;
```

– Un changement de type « Play » ou « Pause » ne peut être appliqué que sur les évènements en sortie ayant la modalités « Music ».

```
invariant
Changes_Pause_Play_Media_have_to_be_applied_on_Music_Only:
       self.Type = 'Pause-Play media' implies
self.PermanentOEs->collect(OutEType)->select(Outputmodality
<> 'Music')->size() = 0;
```

– Un changement de type « Stop » ne peut être appliqué que sur les évènements en sortie ayant la modalités « Vibration », « Speech synthesis » ou « Music ».

```
invariant
Changes_Stop_Media_and_Vibration_have_to_be_applied_on_Music
_or_Speech_or_Vibration_only:
       self.Type = 'Stop media-vibration' implies
self.PermanentOEs->collect(OutEType)->select((Outputmodality
<> 'Music' and Outputmodality <> 'Vibration' and
Outputmodality <> 'Speech synthesize'))->size() = 0;
```

– Un changement de type « Submit » ou « Reset » ne peut être appliqué que sur les évènements en sortie de type « Form ».

```
invariant Change_Submit_have_to_be_applied_on_Form_only:
	self.Type = 'Submit' implies self.PermanentOEs-
>collect(OutEType)->select(name <> 'Form')->size() = 0;
	invariant
Change_Reset_have_to_be_applied_on_Form_only:
	self.Type = 'Reset' implies self.PermanentOEs-
>collect(OutEType)->select(name <> 'Form')->size() = 0;
```

– Un changement de type « Change to visible », « Change to invisible », « Change to enabled » ou « Change to not enabled » ne peut être appliqué que sur les évènements en sortie ayant la modalité « Displaying ».

```
invariant
Change_Visibility_have_to_be_applied_on_displayed_events_onl
y:
	self.Type = 'Change to visible' or self.Type =
'Change to invisible' or self.Type = 'Change to enabled' or
self.Type = 'Change to not enabled' implies
self.PermanentOEs->collect(OutEType)->select(Outputmodality
<> 'Displaying')->size() = 0;
```

F.2 Les contraintes OCL ergonomiques

– Un état d'interaction ne peut avoir qu'un seul évènement permanent en sortie ayant la modalité « Music ».

```
invariant Only_one_permanent_Music_by_state:
      self.outputeventscomponents->select(Permanentoe)-
>select(p : Permanentoe | (p.OutEType.name = 'Music'))-
>size() <= 1;
```

– Une complémentarité entre des évènements en entrée ayant la modalité « Pointage tactile » et d'autres ayant la modalité « Acceleration » doit avoir une fenêtre temporelle importante (l'attribut TemporalWindow >= 5000 ms).

```
invariant InputComplementarity_between_tactile_acceleration:
      if (self.InputEvents->select(i : Inputevent |
(i.InEType.Inputmodality = 'Pointing tactile'))->size()>0) then
         (if (self.InputEvents->select(i : Inputevent |
(i.InEType.Inputmodality = 'Acceleration'))->size()>0) then
(self.TemporalWindow >=5000) else true endif) else true
endif;
```

– Une complémentarité avec un évènement en entrée ayant la modalité « Speech » doit avoir une fenêtre temporelle importante (l'attribut TemporalWindow >= 5000 ms).

```
invariant InputComplementarity_between_tactile_acceleration:
      if (self.InputEvents->select(i : Inputevent |
(i.InEType.Inputmodality = 'Speech'))->size()>0) then
         (if (self.InputEvents->select(i : Inputevent |
(i.InEType.Inputmodality = 'Acceleration'))->size()>0) then
(self.TemporalWindow >=5000) else true endif) else true
endif;
```

Bibliographie

[1] USIXML metamodèle. http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/. Dernière consultation le 24/06/2014

[2] XDK Intel exemple. http://www.greymatter.com/corporate/hardcopy-article/straight-talking/. Dernière consultation le 24/06/2014

[3] Abdallah El Ali, A. : Minimal mobile human computer interaction. Ph.D. thesis, University van Amsterdam, Pays-Bas (2013)

[4] Aebi, F. : Multimodal fusion on mobile devices. Seminar on Multimodal Interaction on Mobile Devices (2012)

[5] Ait-Ameur Y., A.S.I.B.M. : Modélisation et validation formelle d'IHM. Rapport technique (2005)

[6] Angly, O. : Benchmarking d'algorithmes pour la génération automatique de layout d'interfaces graphiques. Mémoire de master. université catholique de louvain (2011)

[7] Avouac, P.A. : Plateforme autonomique dirigée par les modèles pour la construction d'interfaces multimodales dans les environnements pervasifs. Ph.D. thesis, Université de Grenoble, France (2013)

[8] Avouac, P.A., Lalanda, P., Nigay, L. : Autonomic management of multimodal interaction : Dynamo in action. In : Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS '12, pp. 35–44. ACM, New York, NY, USA (2012). DOI 10.1145/2305484.2305493. URL http://doi.acm.org/10.1145/2305484.2305493

[9] Ballagas, R., Memon, F., Reiners, R., Borchers, J. : istuff mobile : Rapidly prototyping new mobile phone interfaces for ubiquitous computing. In : Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '07, pp. 1107–1116. ACM, New York, NY, USA (2007). DOI 10.1145/1240624.1240793. URL http://doi.acm.org/10.1145/1240624.1240793

[10] Bellik, Y. : Interface multimodales : Concepts, modèles et architectures. Ph.D. thesis, Université Paris XI, France (1995)

[11] Bellik, Y., Teil, D. : Définitions Terminologiques pour la Communication. Proceedings of Interface Homme Machine (IHM'92) (1992)

[12] Blanc, X., Salvatori, O.C., Desfray, P. : MDA en action : ingénierie logicielle guidée par les modèles. Architecte logiciel. Eyrolles, Paris (2005). URL http://opac.inria.fr/record=b1101908

[13] Bolt, R.A. : Put-that-there : Voice and gesture at the graphics interface. SIGGRAPH Comput. Graph. **14**(3), 262–270 (1980). DOI 10.1145/965105.807503. URL http://doi.acm.org/10.1145/965105.807503

[14] Bordegoni, M., Faconti, G., Feiner, S., Maybury, M.T., Rist, T., Ruggieri, S., Trahanias, P., Wilson, M. : A standard reference model for intelligent multimedia presentation systems (1997)

[15] Bouchet, J. : Ingénierie de l'interaction multimodale en entrée : Approche à composants ICARE. Ph.D. thesis, Université de Grenoble, France (2006)

[16] Bourguet, M.L. : A toolkit for creating and testing multimodal interface designs. Proceedings of user interface software and technology (UIST) (2002)

[17] Bourguet Marie-Luce, J.C. : Design and usability evaluation of multimodal interaction with finite state machines : A conceptual framework. Journal on Multimodal User Interfaces pp. 53–60 (2008)

[18] Brunette, W., Sodt, R., Chaudhri, R., Goel, M., Falcone, M., Van Orden, J., Borriello, G. : Open data kit sensors : A sensor integration framework for android at the application-level. In : Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, MobiSys '12, pp. 351–364. ACM, New York, NY, USA (2012). DOI 10.1145/2307636.2307669. URL http://doi.acm.org/10.1145/2307636.2307669

[19] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J. : A unifying reference framework for multi-target user interfaces. INTERACTING WITH COMPUTERS **15**, 289–308 (2003)

[20] Caron, P.A., Blay-Fornarino, M., Le Pallec, X. : La contextualisation de modèles, une étape indispensable à un développement dirigé par les modèles ? Revue RSTI - L'Objet, Numéro Spécial : Ingénierie Dirigée par les Modèles, 18 pages pp. – (2007). URL http://hal.archives-ouvertes.fr/hal-00731357

[21] Chikofsky, E.J., Cross II, J.H. : Reverse engineering and design recovery : A taxonomy. IEEE Softw. **7**(1), 13–17 (1990). DOI 10.1109/52.43044. URL http://dx.doi.org/10.1109/52.43044

[22] Christin, D., Reinhardt, A., Kanhere, S.S., Hollick, M. : A survey on privacy in mobile participatory sensing applications. J. Syst. Softw. **84**(11), 1928–1946 (2011). DOI 10.1016/j.jss.2011.06.073. URL http://dx.doi.org/10.1016/j.jss.2011.06.073

[23] Clerckx, T., Winters, F., Coninx, K. : Tool support for designing context-sensitive user interfaces using a model-based approach. In : Proceedings of the 4th International Workshop on Task Models and Diagrams, TAMODIA '05, pp. 11–18. ACM, New York, NY, USA (2005). DOI 10.1145/1122935.1122939. URL http://doi.acm.org/10.1145/1122935.1122939

[24] Combemale, B. : Ingénierie dirigée par les modèles (IDM). État de l'art (2008)

[25] Coutaz, J. : Pac, an object oriented model for dialog design. Proceedings Interact **87**, 431–436 (1987)

[26] Coutaz, J., Nigay, L. : Les propriétés CARE dans les interfaces multimodales. IHM'94 pp. 7–14 (1994)

[27] Coutaz, J., Nigay, L., Salber, D., Caelen, J. : The MSM framework : A design space for multisensory-motor systems. pp. 231–241. Springer-Verlag (1993)

[28] Cutugno, F., Leano, V.A., Rinaldi, R., Mignini, G. : Multimodal framework for mobile interaction. In : Proceedings of the International Working Conference on Advanced Visual Interfaces, AVI '12, pp. 197–203. ACM, New York, NY, USA (2012). DOI 10.1145/2254556.2254592. URL http://doi.acm.org/10.1145/2254556.2254592

[29] De Moura, C., Taillard, J., Guyomarch, F., Dumoulin, C. : Adaptation des Templates UML pour la modélisation de composants paramétrables : application à Gaspard2. In : 4èmes Jounées sur l'Ingénierie Dirigée par les Modèles (IDM 08). Mulhouse, France (2008). URL http://hal.inria.fr/inria-00494977

[30] Demumieux, R., Losquin, P. : Gather customer's real usage on mobile phones. In : Proceedings of the 7th International Conference on Human Computer Interaction with Mobile Devices &Amp ; Services, MobileHCI '05, pp. 267–270. ACM, New York, NY, USA (2005). DOI 10.1145/1085777.1085828. URL http://doi.acm.org/10.1145/1085777.1085828

[31] Diaw S., L.R.C.B. : État de l'art sur le développement logiciel dirigé par les modèles. Technique et Science Informatiques, Ingénierie Dirigée par les Modèles pp. 505–536 (2010)

[32] Dumas, B. : Frameworks, description languages and fusion engines for multimodal interactive systems. Ph.D. thesis, Laboratoire d'informatique de Grenoble, Université de Fribourg (2010)

[33] Dumas, B., Solórzano, M., Signer, B. : Design guidelines for adaptive multimodal mobile input solutions. In : Proceedings of the 15th International Conference on Human-computer Interaction with Mobile Devices and Services, MobileHCI '13, pp. 285–294. ACM, New York, NY, USA (2013). DOI 10.1145/2493190.2493227. URL http://doi.acm.org/10.1145/2493190.2493227

[34] Dupuy-Chessa, S. : Modélisation en interaction homme-machine et en système d'information : À la croisé des chemins. Habilitation à diriger des recherches, Université de Grenoble (2011)

[35] El Kouhen A., D.C.G.S.B.P. : Spécifiez vos éditeurs de diagrammes à l'aide de composants réutilisables. Proceedings of 2ème Conférence en Ingénierie Logiciele (CIEL'13) (2013)

[36] Elouali, N., Le Pallec, X., Rouillard, J., Tarby, J.C. : Mimic : Leveraging sensor-based interactions in multimodal mobile applications. In : CHI '14 Extended Abstracts on Human Factors in Computing Systems, CHI EA '14, pp. 2323–2328. ACM, New York, NY, USA (2014). DOI 10.1145/2559206.2581222. URL http://doi.acm.org/10.1145/2559206.2581222

[37] Elouali, N., Liabeuf, D., Le Pallec, X., Rouillard, J., Tarby, J.C. : Model-driven Evolution for multimodal mobile Geographic Information Systems. ERCIM News - Journal of the European Research Consortium for Informatics and Mathematics **88** (2012). URL http://hal.inria.fr/hal-00808318

[38] Elouali, N., Rouillard, J., Le Pallec, X., Tarby, J.C. : Multimodal Interaction : a Survey from Model Driven Engineering and Mobile perspectives. Journal on Multimodal User Interfaces (2013). URL http://hal.inria.fr/hal-00823086. A paraître

[39] Elouali, N., Tarby, J.C., Le Pallec, X., Rouillard, J. : Approche IDM pour le développement d'applications mobiles multimodales. In : A. Etien (ed.) 9ème édition de la conférence MAnifestation des JEunes Chercheurs en Sciences et Technologies de l'Information et de la Communication - MajecSTIC 2012 (2012). Nicolas Gouvy, Villeneuve d'Ascq, France (2012). URL http://hal.inria.fr/hal-00780187

[40] Etienne J., B.J. : Viewpoints creation using Obeo Designer or how to build eclipse DSM without being an expert developer ? Obeo Designer Whitepaper (2010)

[41] Foster, M.E. : State of the art review : Multimodal fission. Deliverable 6.1, COMIC project (2002). URL http://groups.inf.ed.ac.uk/comic/documents/deliverables/Del6-1.pdf

[42] Foucault, C., Micaux, M., Bonnet, D., Beaudouin-Lafon, M. : Spad : A bimanual interaction technique for productivity applications on multi-touch tablets. In : CHI '14 Extended Abstracts on Human Factors in Computing Systems, CHI EA '14, pp. 1879–1884. ACM, New York, NY, USA (2014). DOI 10.1145/2559206.2581277. URL http://doi.acm.org/10.1145/2559206.2581277

[43] France, R., Rumpe, B. : Model-driven development of complex software : A research roadmap. In : 2007 Future of Software Engineering, FOSE '07, pp. 37–54. IEEE Computer Society, Washington, DC, USA (2007). DOI 10.1109/FOSE.2007.14. URL http://dx.doi.org/10.1109/FOSE.2007.14

[44] Frank, U. : Some guidelines for the conception of domain-specific modelling languages. In : EMISA, pp. 93–106 (2011)

[45] Gellersen, H., Kortuem, G., Schmidt, A., Beigl, M. : Physical prototyping with smart-its. IEEE Pervasive Computing **3**(3), 74–82 (2004). DOI 10.1109/MPRV.2004.1321032. URL http://dx.doi.org/10.1109/MPRV.2004.1321032

[46] Ghezzi, C., Jazayeri, M., Mandrioli, D. : Fundamentals of Software Engineering, 2nd edn. Prentice Hall PTR, Upper Saddle River, NJ, USA (2002)

[47] Hammoudi, S. : Contribution à l'étude et à l'application de l'ingénierie dirigée par les modèles. Habilitation à diriger des recherches (2010)

[48] Jacquet, C. : Présentation opportuniste et multimodale d'informations dans le cadre de l'intelligence ambiante. Ph.D. thesis, Université Paris XI, France (2006)

[49] K., K., N.D., W. : Multimodal interfaces to mobile terminals -a design-for-all approach. Proceedings of User interfaces (2010)

[50] Karlson A. K., B.B.B. : Understanding single-handed mobile device interaction. Rapport technique (2006)

[51] Kaufmann, B., Buechley, L. : Amarino : A toolkit for the rapid prototyping of mobile ubiquitous computing. In : Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services, MobileHCI '10, pp. 291–298. ACM, New York, NY, USA (2010). DOI 10.1145/1851600.1851652. URL http://doi.acm.org/10.1145/1851600.1851652

[52] Kelly, S., Tolvanen, J.P. : Domain-Specific Modeling : Enabling Full Code Generation. Wiley (2008)

[53] Ketabdar, H., Lyra, M. : Activitymonitor : Assisted life using mobile phones. In : Proceedings of the 15th International Conference on Intelligent User Interfaces, IUI '10, pp. 417–418. ACM, New York, NY, USA (2010). DOI 10.1145/1719970.1720050. URL http://doi.acm.org/10.1145/1719970.1720050

[54] Kitchenham, B., Pickard, L., Pfleeger, S.L. : Case studies for method and tool evaluation. IEEE Softw. **12**(4), 52–62 (1995). DOI 10.1109/52.391832. URL http://dx.doi.org/10.1109/52.391832

[55] Kleppe, A.G., Warmer, J., Bast, W. : MDA Explained : The Model Driven Architecture : Practice and Promise. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2003)

[56] König, W.A., Rädle, R., Reiterer, H. : Squidy : A zoomable design environment for natural user interfaces. In : CHI '09 Extended Abstracts on Human Factors in Computing Systems, CHI EA '09, pp. 4561–4566. ACM, New York, NY, USA (2009). DOI 10.1145/1520340.1520700. URL http://doi.acm.org/10.1145/1520340.1520700

[57] Krasner, G.E., Pope, S.T. : A cookbook for using the model-view controller user interface paradigm in smalltalk-80. J. Object Oriented Program. **1**(3), 26–49 (1988). URL http://dl.acm.org/citation.cfm?id=50757.50759

[58] Landay, J.A., Myers, B.A. : Sketching interfaces : Toward more human interface design. Computer **34**(3), 56–64 (2001). DOI 10.1109/2.910894. URL http://dx.doi.org/10.1109/2.910894

[59] Landragin, F. : Physical, semantic and pragmatic levels for multimodal fusion and fission. In : Proceedings of seventh international workshop on computational semantics (IWCS-7, pp. 346–350 (2007)

[60] Lane, N.D., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T., Campbell, A.T. : A survey of mobile phone sensing. Comm. Mag. **48**(9), 140–150 (2010). DOI 10.1109/MCOM.2010.5560598. URL http://dx.doi.org/10.1109/MCOM.2010.5560598

[61] Lange, C., Chaudron, M. : Managing Model Quality in UML-Based Software Development. In : Software Technology and Engineering Practice, 2005. 13th IEEE International Workshop on, pp. 7–16 (2005). DOI 10.1109/STEP.2005.16

[62] Larkin, J.H. : Why a diagram is (sometimes) worth ten thousand words. Cognitive Science pp. 65–99 (1987)

[63] Le Bodic L De Loor P, K.J. : Umar : a modeling of multimodal artifact. Proceedings of Human Computer Interaction (HCI) (2005)

[64] Le Pallec, X., Dupuy-Chessa, S. : Intégration de métriques de qualité des modèles et des langages dans l'outil ModX. In : Actes de la Conférence en Ingénierie du Logiciel (CIEL), p. 6p. Rennes, France. URL http://hal.archives-ouvertes.fr/hal-00757420

[65] Lenat, D.B., Guha, R.V., Pittman, K., Pratt, D., Shepherd, M. : Cyc : Toward programs with common sense. Commun. ACM **33**(8), 30–49 (1990). DOI 10.1145/79173.79176. URL http://doi.acm.org/10.1145/79173.79176

[66] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V. : Usixml : a language supporting multi-path development of user interfaces. pp. 11–13. Springer-Verlag (2004)

[67] Lo, K.W.K., Tang, W.W., Leong, H.V., Chan, A., Chan, S., Ngai, G. : i*Chameleon : A unified web service framework for integrating multimodal interaction devices. Pervasive Computing and Communications Workshops, pp. 106–111 (2012)

[68] Martin, J.C. : TYCOON : Six primitive types of cooperation for observing, evaluating, and specifying cooperations. Technical Report- American association for artificial intelligence FS pp. 61–66 (1999)

[69] Marvie, R. : Séparation des préoccupations et méta-modélisation pour environnements de manipulation d'architectures logicielles à base de composants. Ph.D. thesis, Université de lille, France (2002)

[70] McNab, T., James, D.A., Rowlands, D. : iphone sensor platforms : Applications to sports monitoring. Procedia Engineering **13**, 507–512 (2011)

[71] Mechkour, S. : Smuiml editor : graphical tool for modeling multimodal interaction. Mémoire de master. université de fribourg, switzerland (2011)

[72] Meixner, G., Paterno, F., Vanderdonckt, J. : Past, present, and future of model-based user interface development. i-com **10**(3), 2–11 (2011). URL http://dblp.uni-trier.de/db/journals/icom/icom10.html#MeixnerPV11

[73] Minsky, M. : Matter, Mind and Models. In : Proceedings of IFIP Congress 65, pp. 45–49 (1965). URL http://groups.csail.mit.edu/medg/people/doyle/gallery/minsky/mmm.html

[74] Moody, D. : The "physics" of notations : Toward a scientific basis for constructing visual notations in software engineering. IEEE Trans. Softw. Eng. **35**(6), 756–779 (2009). DOI 10.1109/TSE.2009.67. URL http://dx.doi.org/10.1109/TSE.2009.67

[75] Moody, D., Hillegersberg, J. : Software language engineering. chap. Evaluating the Visual Syntax of UML : An Analysis of the Cognitive Effectiveness of the UML Family of Diagrams, pp. 16–34. Springer-Verlag, Berlin, Heidelberg (2009). DOI 10.1007/978-3-642-00434-6_3. URL http://dx.doi.org/10.1007/978-3-642-00434-6_3

[76] Mottier, C. : Application de domotique sur la plateforme android. Rapport de stage (2009)

[77] Myers, B., Hudson, S.E., Pausch, R. : Past, present, and future of user interface software tools. ACM Trans. Comput.-Hum. Interact. **7**(1), 3–28 (2000). DOI 10.1145/344949.344959. URL http://doi.acm.org/10.1145/344949.344959

[78] Myers, B.A., Rosson, M.B. : Survey on user interface programming. In : Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '92, pp. 195–202. ACM, New York, NY, USA (1992). DOI 10.1145/142750.142789. URL http://doi.acm.org/10.1145/142750.142789

[79] Naumann, A., Wechsung, I., Hurtienne, J. : Multimodality, inclusive design, and intuitive use. Is Prior Experience the Same as Intuition in the Context of Inclusive Design (2009). URL http://www.joernhurtienne.com/iuui/Prior_Experience/Position_Papers_files/NaumannWechsungHurtienne.pdf

[80] Navarre, D., Palanque, P., Dragicevic, P., Bastide, R. : An approach integrating two complementary model-based environments for the construction of multimodal interactive applications. Interact. Comput. **18**(5), 910–941 (2006). DOI 10.1016/j.intcom.2006.03.002. URL http://dx.doi.org/10.1016/j.intcom.2006.03.002

[81] Nigay, L., Coutaz, J. : Multifeature systems : The CARE properties and their impact on software design. In : Multimedia Interfaces : Research and Applications, chapter 9. AAAI Press (1997)

[82] Obrenovic, Z., Starcevic, D. : Modeling multimodal human-computer interaction. Computer **37**(9), 65 (2004)

[83] Oulasvirta, A., Reichel, A., Li, W., Zhang, Y., Bachynskyi, M., Vertanen, K., Kristensson, P.O. : Improving two-thumb text entry on touchscreen devices. In : Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '13, pp. 2765–2774. ACM, New York, NY, USA (2013). DOI 10.1145/2470654.2481383. URL http://doi.acm.org/10.1145/2470654.2481383

[84] Oviatt, S. : Multimodal interactive maps : Designing for human performance. Hum.-Comput. Interact. **12**(1), 93–129 (1997). DOI 10.1207/s15327051hci1201\&2_4. URL http://dx.doi.org/10.1207/s15327051hci1201&2_4

[85] Oviatt, S. : Ten myths of multimodal interaction. Commun. ACM **42**(11), 74–81 (1999). DOI 10.1145/319382.319398. URL http://doi.acm.org/10.1145/319382.319398

[86] Palanque, P.A., Schyn, A. : A model-based approach for engineering multimodal interactive systems. In : M. Rauterberg, M. Menozzi, J. Wesson (eds.) INTERACT. IOS Press (2003). URL http://dblp.uni-trier.de/db/conf/interact/interact2003.html#PalanqueS03

[87] Paternò, F., Mancini, C., Meniconi, S. : Concurtasktrees : A diagrammatic notation for specifying task models. In : Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction, INTERACT '97, pp. 362–369. Chapman & Hall, Ltd., London, UK, UK (1997). URL http://dl.acm.org/citation.cfm?id=647403.723688

[88] Pfaff G. E., e. : User interface management systems. In : Proceedings of the Workshop on User Interface Management Systems (1983)

[89] Porta, D., Sonntag, D., Nebelrath, R. : New business to business interaction : Shake your iphone and speak to it. In : Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services, MobileHCI '09, pp. 59 :1–59 :2. ACM, New York, NY, USA (2009). DOI 10.1145/1613858.1613931. URL http://doi.acm.org/10.1145/1613858.1613931

[90] Romero, J.R., Rivera, J.E., Durán, F., Vallecillo, A. : Formal and tool support for model driven engineering with maude (2007)

[91] Roque, R.V. : Openblocks : an extendable framework for graphical block programming systems. Ph.D. thesis, Massachusetts Institute of Technology (2007)

[92] Roques, P. : Uml 2 : Modéliser une application web. Les cahiers du programmeur. eyrolles (2008)

[93] Roudaut, A. : Conception et évaluation de techniques d'interaction pour dispositifs mobiles. Ph.D. thesis, Telecom ParisTech, France (2010)

[94] Rouillard, J., Tarby, J.C., Le Pallec, X., Marvie, R. : From Metamodeling to Automatic Generation of Multimodal Interfaces for Ambient Computing. International Journal On Advances in Software **3**(3&4) (2011). URL http://hal.inria.fr/hal-00808273. 1&2 publisher = IARIA 1&2 publisher = IARIA

[95] Rousseau, C., Bellik, Y., Vernier, F., Bazalgette, D. : A framework for the intelligent multimodal presentation of information. Signal Process. **86**(12), 3696–3713 (2006). DOI 10.1016/j.sigpro.2006.02.041. URL http://dx.doi.org/10.1016/j.sigpro.2006.02.041

[96] Ruß, A. : Mmir framework : Multimodal mobile interaction and rendering. In : M. Horbach (ed.) GI-Jahrestagung, *LNI*, vol. 220, pp. 2702–2713. GI (2013). URL http://dblp.uni-trier.de/db/conf/gi/gi2013.html#Russ13

[97] Samaan, K. : Prise en compte du modèle d'interaction dans le processus de construction et d'adaptation d'applications interactives. Ph.D. thesis, Université de Lyon, France (2005)

[98] Sandven, A. : Metamodel based code generation in dpf editor. Mémoire de master. université de bergen (2012)

[99] Serrano, M., Nigay, L., Lawson, J.Y.L., Ramsay, A., Murray-Smith, R., Denef, S. : The Openinterface Framework : A tool for multimodal interaction. In : CHI '08 Extended Abstracts on Human Factors in Computing Systems, CHI EA '08, pp. 3501–3506. ACM, New York, NY, USA (2008). DOI 10.1145/1358628.1358881. URL http://doi.acm.org/10.1145/1358628.1358881

[100] Sigchi : A Metamodel for the Runtime Architecture of An Interactive System. The UIMS Workshop Tool developers pp. 32–37 (1992)

[101] Sinha, A.K., Landay, J.A. : Capturing user tests in a multimodal, multidevice informal prototyping tool. In : Proceedings of the 5th International Conference on Multimodal Interfaces, ICMI '03, pp. 117–124. ACM, New York, NY, USA (2003). DOI 10.1145/958432.958457. URL http://doi.acm.org/10.1145/958432.958457

[102] Sottet, J.S. : Méga-ihm : malléabilité des interfaces homme-machine dirigées par les modèles. Ph.D. thesis, Université de Grenoble, France (2008)

[103] Sunwoo, J., Yuen, W., Lutteroth, C., Wünsche, B. : Mobile games for elderly healthcare. In : Proceedings of the 11th International Conference of the NZ Chapter of the ACM Special Interest Group on Human-Computer Interaction, CHINZ '10, pp. 73–76. ACM, New York,

NY, USA (2010). DOI 10.1145/1832838.1832851. URL http://doi.acm.org/10.1145/1832838.1832851

[104] Szyperski, C. : Component Software : Beyond Object-Oriented Programming, 2nd edn. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002)

[105] Tang, W.W., Lo, K.W., Chan, A.T., Chan, S., Leong, H.V., Ngai, G. : I*Chameleon : A scalable and extensible framework for multimodal interaction. In : CHI '11 Extended Abstracts on Human Factors in Computing Systems, CHI EA '11 (2011)

[106] Thevenin, D., Calvary, G., Coutaz, J. : La plasticité en interaction homme-machine (2000)

[107] Turk, M. : Multimodal interaction : A review. Pattern Recognition Letters **36**(0), 189 – 195 (2014). DOI http://dx.doi.org/10.1016/j.patrec.2013.07.003. URL http://www.sciencedirect.com/science/article/pii/S0167865513002584

[108] Vernier, F. : La multimodalité en sortie et son application à la visualisation de grandes quantités d'information. Ph.D. thesis, Université de Grenoble, France (2001)

[109] Winkler, T., Ide, M., Wolters, C., Herczeg, M. : Wewrite : 'on-the-fly' interactive writing on electronic textiles with mobile phones. In : Proceedings of the 8th International Conference on Interaction Design and Children, IDC '09, pp. 226–229. ACM, New York, NY, USA (2009). DOI 10.1145/1551788.1551836. URL http://doi.acm.org/10.1145/1551788.1551836

[110] Xiao, B., Girand, C., Oviatt, S. : Multimodal integration patterns in children. In : IN PROC. OF ICSLP'02, pp. 629–632 (2002)

[111] Xiao, B., Lunsford, R., Coulston, R., Wesson, M., Oviatt, S. : Modeling multimodal integration patterns and performance in seniors : Toward adaptive processing of individual differences. In : the Fifth International Conference on Multimodal Interfaces, pp. 265–272. ACM Press (2003)