

UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS
ÉCOLE DOCTORALE STIC
SCIENCES ET TECHNOLOGIES DE L'INFORMATION
ET DE LA COMMUNICATION

T H È S E

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice - Sophia Antipolis

Mention : INFORMATIQUE

Présentée et soutenue par

André KALAWA

Migration des applications vers les tables interactives par recherche d'équivalences

Thèse dirigée par Michel RIVEILL

préparée au sein du laboratoire I3S, Equipe RAINBOW

soutenue le 30 août 2013

Jury :

<i>Rapporteurs :</i>	Daniel HAGIMONT	- Professeur, INPT/ENSEEIHT de Toulouse
	Jacky ESTUBLIER	- Professeur, LIG Grenoble
<i>Directeur :</i>	Michel RIVEILL	- Professeur, Université de Nice Sophia Antipolis
<i>Co Encadrante :</i>	Audrey OCCELLO	- Docteur, Université de Nice Sophia Antipolis
<i>Président :</i>	Mireille BLAY-FORNARINO	- Professeur, Université de Nice Sophia Antipolis

Résumé

Dans le domaine du génie logiciel pour les Interactions Homme Machine (IHM), la migration des interfaces utilisateurs (UI) est un moyen pour réutiliser des applications sur des plateformes ayant des modalités d’interactions différentes des environnements de départ. Les approches existantes de migration des UI sont manuelles dans le cadre des approches spécifiques, elles sont automatiques dans le cadre des services d’adaptation des UI aux contextes d’usage, ou elles sont semi automatiques dans le cadre d’une migration flexible dirigée par un concepteur.

Dans cette thèse nous nous intéressons à la migration semi automatique des UI vers une cible comme une table interactive dans l’objectif de transformer des UI Desktop en UI qui favorisent la collaboration et l’utilisation des objets tangibles. Les tables interactives sont des plateformes qui disposent des instruments d’interactions permettant de décrire des UI tangibles et multi-utilisateurs. En considérant que le noyau fonctionnel (NF) des applications de départ peut être réutilisé sur les cibles sans changement, les UI des applications sont caractérisées par la dimension des dialogues entre les utilisateurs et le système, la dimension de la structure et du positionnement des éléments graphiques et la dimension du style des éléments visuels. La migration d’une UI dans ces conditions consiste à transformer ou à recréer les différentes dimensions d’une UI de départ pour la cible tout en considérant les critères de conception des UI pour les tables interactives.

Nous proposons dans cette thèse un modèle d’interactions abstraites pour établir les équivalences entre les dialogues et la structure des UI indépendamment des modalités d’interactions des plateformes source et cible. Les primitives d’interactions et la structure des composants graphiques permettent de décrire des opérateurs d’équivalences pour retrouver et classer les éléments graphiques équivalents en prenant en compte les guidelines des tables interactives. Nous proposons aussi des règles de substitution et de concrétisation pour accroître l’accessibilité des éléments graphiques et favoriser l’utilisation des objets tangibles.

Mots clés : migration des interfaces utilisateurs, équivalences des plateformes, modalités d’interactions, critères de conception, guidelines

Abstract

In software engineering, in the field of human computer interaction (HCI), the migration of user interface (UI) is a way to reuse existing applications on platforms with different interactions modalities. The existing approaches for UI migration can be manual (for specific applications), they can be automatic (for services which adapt UI based on context aware), or they can be mix of the previous - semi automatic (providing a flexible migration process driven by the person in charge).

This thesis proposes a semi automatic process for migration of UI from a desktop to interactive table for the purpose of transforming the UI of desktop to support further collaboration and usage of tangible objects. The interactive tables are platforms with interactions instruments which allow the description of tangible and multi users UIs. Considering that the functional core (FC) of source applications can be reused on target platform without transformation, any UI can be characterized with three dimensions : the first dimension concerns the dialogues between the users and the system, the second dimension concerns the structure and the layout of graphical components, and the third dimension concerns the visual style of graphical elements. In this context, the problematic regarding the UI migration is how to transform or re inject these different dimensions of source UI into the target, while considering the UI design criteria for interactive tables.

This thesis proposes an abstract interactions model for establishing equivalences (independent of modalities of interactions) between the source and the dialogue and structure of the target. The primitives of interaction and the structure of graphical components are used to describe equivalence operators to find and to rank equivalent elements on interactive tables. Furthermore, this thesis proposes substitution and concretization rules to increase the accessibility of graphical elements and to facilitate the usage of tangible objects. The ranking process and the transformation rules are based on guidelines for UI migration to interactive tables which are interpreted from design criteria.

Keywords : user interface migration, équivalence of plateform, design criteria, guidelines

Table des matières

I Introduction	3
1 Introduction	5
1.1 Contexte	5
1.2 Enjeux de la migration	6
1.3 Contribution de la thèse	7
1.4 Plan du manuscrit	7
2 Migration des applications	9
2.1 Motivations	9
2.2 Scénario & Problèmes	10
2.2.1 Cas d'une application desktop	10
2.2.2 Problèmes liés à la migration du dialogue	12
2.2.3 Problèmes liés à la migration de la structure et du positionnement	13
2.2.4 Problèmes liés à la migration du style	15
2.2.5 Espace des problèmes liés la migration vers les tables interactives	16
2.3 Pérимètres de la migration des UI vers les tables interactives	17
2.3.1 Hypothèse de travail	17
2.3.2 Pistes de migration	18
2.4 Objectifs	19
II Domaine d'étude	21
3 Tables interactives et Migration des UI	23
3.1 Modèle d'interactions d'une table interactive	23
3.1.1 Les dispositifs matériels d'interactions d'une table interactive	24
3.1.2 Bibliothèques graphiques des tables interactives	27
3.1.3 Modalités d'interactions	29
3.1.4 Modèle d'interactions abstraites pour la migration des UI	31
3.2 Principes de conception des UI pour les tables interactives	33
3.2.1 Propriétés caractéristiques des tables interactives	33
3.2.2 Critères ergonomiques de conception des UI	35
3.2.3 Recommandations pour la migration des UI vers les tables interactives	38
3.3 Synthèse	43
4 Approches de migration des UI	45
4.1 Introduction	45
4.2 Approches spécifiques de migration des UI	46
4.2.1 Migration manuelle	46
4.2.2 Portage des applications existantes sur des tables interactives	48
4.3 Approches de migration basées sur les modèles de l'UI	51
4.3.1 Approches de migration automatiques des UI	52
4.3.2 Approche semi automatique de migration des UI	57

4.4	Synthèse et objectifs	62
4.4.1	Synthèse	62
4.4.2	Objectifs	64
III	Méthodes	65
5	Modélisation des UI	67
5.1	Introduction	67
5.2	Primitives d'interactions	68
5.2.1	Les primitives d'interactions en entrée	68
5.2.2	Les primitives d'interactions en sortie	72
5.3	Modèles de composants graphiques	73
5.3.1	Un modèle de types de composants graphiques	75
5.3.2	Un modèle d'instance d'une UI	81
5.3.3	Synthèse des modèles abstraits	90
5.4	Opérateurs d'équivalences	91
5.4.1	Opérateurs d'équivalences	91
5.4.2	Opérateurs d'équivalences & types de données	95
5.5	Synthèse	98
6	Mécanismes de migration des UI	99
6.1	Introduction	99
6.2	Prise en compte des guidelines	100
6.2.1	Conception assistée des UI	101
6.2.2	Interprétation des guidelines pour la migration	102
6.2.3	Utilisation effective des guidelines	104
6.3	Transformations du modèle de l'UI source	108
6.3.1	Transformation des groupes d'éléments graphiques	108
6.3.2	Transformation d'un élément	119
6.4	Classement des éléments équivalents	124
6.4.1	Conformité des composants graphiques aux guidelines	124
6.4.2	Charge de travail	127
6.4.3	Algorithme de classement	129
6.5	Synthèse	133
IV	Expérimentations	135
7	Validations du processus	137
7.1	Introduction	137
7.2	Implémentation du processus de migration	138
7.2.1	Abstraction de l'UI source	139
7.2.2	Proposition d'une version des UI migrées	143
7.2.3	Personnalisation des UI proposées	145
7.2.4	Génération de l'UI finale	148
7.2.5	Remarques	149
7.3	Évaluation	149

TABLE DES MATIÈRES

1

7.3.1	Critères d'évaluation de l'approche proposée	150
7.3.2	Résultats	151
7.3.3	Interprétation des résultats	154
V	Conclusions et Perspectives	157
8	Conclusions et perspectives	159
8.1	Synthèse	159
8.2	Perspectives	161
8.2.1	A moyen terme	161
8.2.2	A long terme	161
Appendices		169
A	Application des modèles de l'UI et des PI	171
A.1	Comment décrire une instance du modèle de type de composants graphiques ?	171
A.1.1	Identification d'un Widget à partir d'un Assembly	171
A.1.2	Identification des comportements	171
A.1.3	Remarque	172
A.2	Abstraction des UI finales	172
A.2.1	Identification des UIComponent à partir d'un fichier XAML	172
A.2.2	Identification des Containers à partir d'un fichier XAML	173
A.2.3	Identification de la classe Content à partir d'un fichier XAML	173
A.2.4	Identification de la classe ImplementedEvent à partir d'un fichier XAML	174
A.2.5	Exemple d'UIStructure XAML	174
A.2.6	Exemple de UIBehavior C#	176
A.2.7	Exemple d'AbstractView C#	177
A.3	Primitives d'interactions	180
A.3.1	Liste des Widgets	180

Première partie

Introduction

CHAPITRE 1

Introduction

1.1 Contexte

Le nombre grandissant de plateformes et surtout leurs très grandes variétés de dispositifs d’interactions dont ils disposent telles que les smartphones, les tablettes, les terminaux tactiles ou les tables interactives ont généralisé l’usage des applications ayant des modalités d’interactions [NC93] beaucoup plus sophistiquées qu’une simple utilisation du clavier et de la souris. Les tables interactives par exemple offrent la possibilité de mettre en œuvre des applications réellement multi-utilisateurs mêlant interactions tactiles et objets tangibles¹ sur une surface de prêt de 1m².

Le domaine du génie logiciel pour les Interactions Homme Machine (IHM) propose plusieurs approches pour la conception d’une application. En particulier la tendance actuelle est de concevoir les Interfaces Utilisateurs (UI) selon les modèles du Framework de Référence Cameleon (CRF) [CCB⁺02] identifiant différents niveau d’interface de la plus abstraite, indépendante des dispositifs d’interactions disponibles sur la plateforme à un niveau concret permettant la mise en œuvre de l’UI selon la modalité d’interaction souhaitée disponible sur la plateforme cible. Si les UI de l’application sont construites selon cette approche, alors la migration des applications entre terminaux ayant des modalités différents a déjà été partiellement abordé en particulier dans [Pat11]. Notre travail aborde la possibilité de faire migrer entre plateformes ayant des modalités d’interactions différentes des applications qui n’ont pas été prévue selon les différents modèles de référence de Cameleon.

La migration des UI est une activité de génie logiciel qui implique la transformation des différents aspects qui constituent une UI existante tels que les interactions (ou le dialogue entre l’utilisateur et le système) qu’il faut nécessairement préserver pour que l’utilisateur puisse toujours accomplir les mêmes tâches, les structures (organisations et types de données des éléments graphiques), le positionnement et les styles des composants graphiques qui doivent être adaptés pour être conformes aux spécificités de la plateforme cible et toujours satisfaire les utilisateurs finaux dans les choix de configuration qui ont pu être fait sur la source

Au delà des problèmes liés à des différences possibles entre les environnements d’exécution des plateformes source et cible [TKB78] qui nécessite le portage du code, les différences des modalités d’interactions entre la source et la cible impliquent nécessairement la prise en compte des critères usuels de conception des UI. La transformation de l’UI de départ doit être guidée non seulement par les critères ergonomiques de conception mais aussi par les dispositifs d’interactions disponible sur la plateforme cible qu’il peut être intéressant d’utiliser.

Évidemment, en complément de la volonté de rendre disponible des dispositifs d’interactions nouveaux, peut-être plus intuitifs, les critères ergonomiques de conception constituent un ensemble de principes à respecter pendant la mise en œuvre. Ils permettent de garantir l’utilisabilité d’une UI. Par exemple, l’utilisation d’une application pour desktop sur une table interactive sans aucune adaptation pose des problèmes d’utilisabilité, car la simulation du clavier et de la souris n’est pas le meilleur

1. Les interfaces utilisateurs tangibles (TUI) permettent à une application de pouvoir interagir avec des objets physiques directement manipulable par les utilisateurs [IU97]. Les objets tangibles sont les objets pouvant être manipulé par une interface utilisateur tangible.

mode d’interaction en terme d’utilisabilité. Si l’application le permet, on peut même imaginer une utilisation multi-utilisateurs avec une nouvelle UI à déduire de l’UI d’origine.

1.2 Enjeux de la migration

Dans le cadre de la migration des applications pour desktops vers les tables interactives par exemple, nous notons que les dialogues, la structure et le positionnement des éléments des UI graphiques doivent prendre en compte l’éventualité d’avoir plusieurs utilisateurs mais surtout la possibilité d’utiliser des objets tangibles. Travailler sur ces deux exemples va nous permettre d’avoir une réelle évolutivité des UI entre le terminal source et le terminal cible et permet de favoriser la collaboration dans un espace de travail co-localisé et multi-utilisateurs offert par la table interactive et donc de l’utiliser pleinement.

Nous faisons l’hypothèse que le cœur de l’application peut être migré sans modification et qu’il ne faut agir qu’au niveau de l’UI qu’il faut bien évidemment faire migrer. Cette migration, d’un terminal source vers un terminal cible donné doit permettre de conserver dans la mesure du possible les dialogues, la structure et les positionnements relatifs des éléments de l’interface, le respect du style des éléments graphiques des UI de la source mais bien évidemment proposer une adaptation pour utiliser au mieux, sans perdre les utilisateurs, les nouveaux moyens d’interaction mis à la disposition des utilisateurs.

La transformation des dialogues d’une UI desktop vers une cible collaborative peut-elle garantir à chaque utilisateur des interactions qui favorisent la collaboration ? De nombreux éléments sont à prendre en compte. En effet, chaque dialogue ou message d’erreur ne doit pas perturber les activités des autres utilisateurs ; par exemple les “feedbacks” ou les messages d’erreurs destinés à un utilisateur ne doivent pas bloquer un autre utilisateur si une réponse est attendue. Par ailleurs la transformation des dialogues peut-elle assurer que chaque dialogue reste cohérent avec l’intention et les actions des utilisateurs ? Par exemple la modification d’une valeur par deux utilisateurs distincts peut les perturber. Il faut en effet décider si le résultat est la somme des deux modifications, la moyenne, le min ou le max. Évidemment le contexte et la nature de la valeur peuvent guider sur le choix à effectuer.

La transformation des dialogues sur les UI et les fonctionnalités de l’application source peuvent provoquer de nombreux changements ? En effet, la transformation des dialogues des UI desktops pour les tables interactives peuvent par exemple impliquer des modifications pour prendre en compte la présence de plusieurs utilisateurs ou d’objets tangibles.

La transformation de la structure et du positionnement des éléments d’une UI pour desktop vers une table interactive consiste à assurer l’accessibilité aux différents utilisateurs des éléments graphiques pertinents². Par ailleurs la transformation des aspects structurels d’une UI doit elle aussi préserver au mieux l’utilisabilité de l’UI. En effet les solutions de transformations proposées doivent produire des UI conformes aux spécificités de la plateforme cible et satisfaisantes pour les utilisateurs finaux.

Pour terminer, la transformation du style a nécessairement un impact sur la collaboration ou sur l’utilisation des objets tangibles. Généralement, chaque application adopte une charte graphique qu’il faut soit respecter, soit rénover.

Les questions soulevées par les transformations des différents aspects des UI lors d’une migration sont usuellement traitées de différentes manières. En premier lieu, il est possible d’adopter une approche manuelle [WGM08]. Celle-ci permet d’avoir, au prix d’un travail minutieux, une UI conforme aux attentes des utilisateurs finaux car les transformations sont flexibles et donc la qualité de l’UI produite dépend directement des compétences de celui qui effectue la migration manuelle. Mais cette

2. Un menu par exemple

approche est difficilement réutilisable, à moins de la consigner dans un document, que ce soit pour d'autres applications ou pour d'autres personnes car elle nécessite une bonne connaissance des critères de transformation des différents aspects et des technologies des UI des plateformes source et cible.

Il est aussi possible d'adopter une approche automatique basée ou non sur des modèles abstraits [Bes10, PZ10] qui permet une transformation des différents aspects des UI. Bien que ces approches automatiques soient réutilisables, elles sont nécessairement moins flexibles car le concepteur qui ne peut pas intervenir pendant la transformation, adapte l'UI produite dans un second temps.

Entre ces deux approches, il est aussi possible d'adopter une approche semi automatique[MR97] qui présente de nombreux inconvénients car non seulement elle induit un travail supplémentaire pour la personne en charge de la migration qui guide la transformation mais en permettant plus de flexibilité, il est assez difficile de capitaliser sur le travail effectuer si la personne en charge de la migration change. Néanmoins, cette approche permet une transformation interactive des différents aspects de l'UI. L'intérêt d'une flexibilité dans l'approche de migration d'UI permet d'avoir des UI migrées proches des attentes des utilisateurs finaux.

1.3 Contribution de la thèse

Cette thèse a pour premier objet d'étudier la migration des UI entre plateformes ayant des dispositifs d'interactions différents. Nous avons fait le choix de cibler en particulier la migration des UI depuis une station possédant clavier et souris vers une table interactive. Nous souhaitons que cette migration se fasse au moindre coût pour les concepteurs ou les développeurs tout en prenant en compte les spécificités de la cible et, bien évidemment, les critères ergonomiques usuels utilisés pour la conception des UI.

La solution que nous proposons repose sur un processus semi automatique de migration d'UI. Celui-ci comporte plusieurs étapes interactives pour que les concepteurs ou les développeurs puissent effectuer des choix conformes aux critères de conception qu'ils souhaitent privilégiés. Nous avons fait le choix de l'interactivité, basé sur des choix simples pour garantir simultanément la réutilisabilité, minimiser les coûts mais aussi pour accroître la flexibilité et garantir ainsi l'UI la plus pertinente.

Il est primordial de prendre en compte, lors de la migration des interfaces utilisateurs, les critères ergonomiques de conception. Nos travaux utilisent des concepts issus de plusieurs domaines de recherche.

Dans le domaine de l'utilisabilité, nous nous sommes intéressés aux travaux qui modélisent les critères ergonomiques de conception pour les traduire en règles opérationnelles et utilisables pendant la conception. L'objectif est de pouvoir intégrer ces règles dans la plateforme de migration dans le but de réduire la charge de travail des personnes en charge de la migration.

Dans le domaine de l'ingénierie des modèles, nous nous sommes intéressés aux travaux permettant de modéliser une plateforme dans le but d'effectuer la migration à un niveau abstrait : de concept à concept. L'objectif est de rendre notre travail réutilisable si la source et la cible évoluent. Cette approche nous permet d'abstraire non seulement les UI mais aussi les interactions. Nous proposons ainsi un modèle d'interactions basées sur les primitives d'interactions[KODPR12] pour décrire les actions atomiques qui constituent les dialogues entre l'utilisateur et le système.

1.4 Plan du manuscrit

Notre manuscrit est structuré de la manière suivante :

Le chapitre 2 présente l'espace des problèmes liés à la transformations des différents aspects d'une UI. Il délimite le périmètre de la migration des UI d'un poste fixe vers une table interactive. Il fixe nos objectifs.

Le chapitre 3 présente une étude du modèle d'interactions des tables interactives dans le but d'identifier les spécificités et les critères ergonomiques de conception à intégrer pour la migration des UI vers cette cible.

Le chapitre 4 est un état de l'art des approches de migration des UI. Dans ce chapitre nous décrivons les critères nécessaires pour atteindre nos objectifs et nous évaluons les différentes approches à l'aide de ces critères. Ce chapitre se termine par une synthèse des différentes approches présentées.

Le chapitre 5 propose un modèle d'UI qui prend en compte deux aspects des UI : leurs interactions et leur structure. Les objectifs de ce modèle d'UI sont de décrire les UI à migrer indépendamment des plateformes mais aussi de décrire des opérateurs d'équivalences entre les dispositifs d'interactions des plateformes source et cible.

Le chapitre 6 présente les mécanismes de transformation de la structure et des interactions de l'UI source vers la cible. L'objectif de ce chapitre est de décrire la prise en compte effective des critères ergonomiques de conception sous forme de guidelines par les mécanismes de transformation.

Le chapitre 7 présente le prototype que nous avons réalisé. Il constitue une preuve de concept des mécanismes proposés. Plusieurs applications ont été migrées afin de valider les éléments des différents modèles, mettre en évidence le respect des critères ergonomiques et surtout mettre en évidence les bénéfices que l'on peut retirer d'une telle approche.

Le chapitre 8 est une conclusion. Elle rappelle les objectifs que nous souhaitions atteindre, met en évidence nos contributions et leurs apports. Elle donne aussi quelques éléments sur des travaux complémentaires qui pourraient être menés pour parfaire nos travaux.

Migration des applications vers les tables interactives

Sommaire

2.1 Motivations	9
2.2 Scénario & Problèmes	10
2.2.1 Cas d'une application desktop	10
2.2.2 Problèmes liés à la migration du dialogue	12
2.2.3 Problèmes liés à la migration de la structure et du positionnement	13
2.2.4 Problèmes liés à la migration du style	15
2.2.5 Espace des problèmes liés la migration vers les tables interactives	16
2.3 Périmètres de la migration des UI vers les tables interactives	17
2.3.1 Hypothèse de travail	17
2.3.2 Pistes de migration	18
2.4 Objectifs	19

DANS le but de décrire les différentes étapes et les problèmes lier à la migration des interfaces Hommes-Machines en général, et vers une table interactive en particulier, nous avons pris le parti d'utiliser un exemple “fil rouge”. Les motivations du choix de l'étude de cette cible sont décrites dans la section 2.1, puis la section 2.2 décrit pas à pas l'application initialement conçue pour un desktop que nous souhaitons mettre en œuvre une table surface. La section 2.4 présente les objectifs de cette thèse.

2.1 Motivations

Les tables interactives comme les desktops, les smartphones ou les tablettes peuvent être utilisées dans divers domaines d'activités [KLL⁺09]. Une table interactive peut servir comme plateforme de jeux, pour la consultation de photos, de cartes, de dessins, pour noter les idées lors d'une session de brainstorming, etc. Les tables interactives permettent de mettre en œuvre un espace de travail collaboratif.

Néanmoins, on remarque que les tables interactives ne se sont pas démocratisées comme les tablettes ou les smartphones. Évidemment, le prix de vente en est une des raisons peut constituer un frein à l'achat. Il est par exemple d'environ 6600€ pour la version 2.0 commercialisées par Microsoft. Même si nous observons une nette baisse du prix par rapport à la première version celui-ci constitue indiscutablement un frein à l'achat. Par ailleurs, la faible diffusion de cette plateforme fait que les applications pour les tables interactives sont loin d'être aussi nombreuses que pour les tablettes ou les smartphones. Selon des données provenant de Distimo³, en mars 2012 nous avons noté plus de 350.000 applications développées aux Etats-Unis pour des desktops, 150.000 applications pour les

3. Distimo : www.distimo.com

smartphones et presque 100.000 spécifiquement pour les tablettes⁴ (cf figure 2.1) et aucune pour les tables interactives.

Pour accroître le nombre d'applications disponibles et toucher un éventuel public, nous pensons que la migration des applications existantes peut être une piste sérieuse si la migration utilise effectivement les divers dispositifs d'interactions disponible sur la table. Dans cette thèse, nous étudions par conséquent les divers problèmes posés par le processus de migrations en espérant pouvoir en complément enrichir les applications d'une dimension collaborative et tangible.

Nous présentons donc à la section 2.2 les problèmes liés à la migration d'une application desktop vers une table interactive puis les objectifs de nos travaux. Les hypothèses et les différentes approches permettant de faire migrer des UI sont présentés dans la section 2.3. L'ensemble de ce chapitre nous permet de décrire le périmètre de nos travaux.

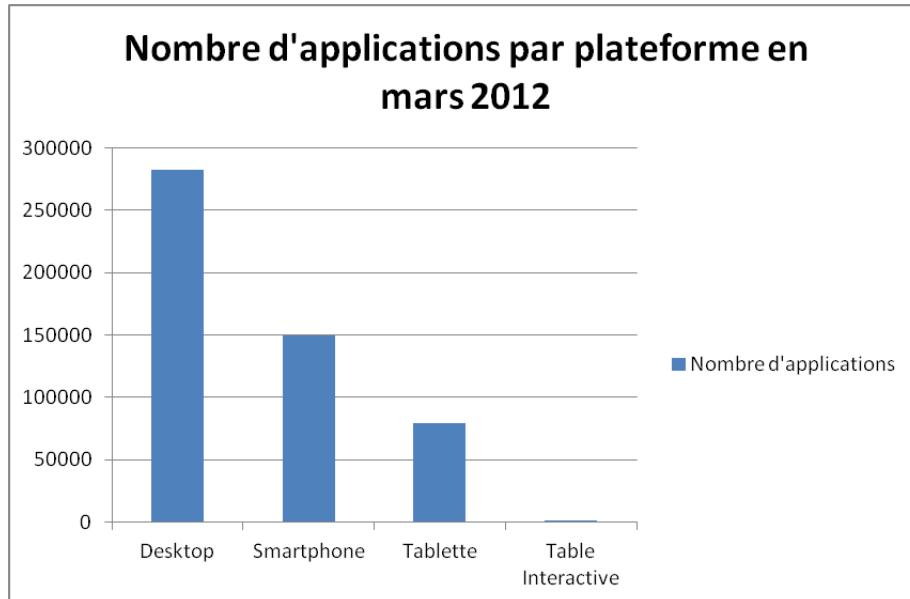


FIGURE 2.1 – Applications disponibles sur les marchés de téléchargement par plateformes en mars 2012 aux États Unis, source Distimo

2.2 Problèmes liés à la migration des UI vers une table interactive

Cette section décrit le fonctionnement d'une application permettant l'élaboration de bande dessinée que nous souhaitons faire migrer vers une table interactive. Dans un second temps, nous présentons les problèmes liés à cette migration et les différents objectifs que nous souhaitons atteindre sont décrits à la fin de cette section.

2.2.1 Cas d'une application desktop

Considérons l'exemple suivant : soit une application conçue pour un desktop qui permet l'élaboration des bandes dessinées (BD) telle que *Comics Book Application* (CBA). Cette application est

4. En considérant Windows Phone 7 Marketplace, Google Play, Nokia Ovi Store, Apple Mac App Iphone, Apple App Store - iPad, Apple Mac App Store, Amazon Appstore et BlackBerry App World

conçue pour être utilisée par un dessinateur de BD qui est assis devant son écran en utilisant sa souris et son clavier. La fenêtre principale de l'application CBA décrite par la figure 2.2 permet d'identifier trois zones majeures qui sont la zone des menus correspondant aux composants graphiques situés en haut de la fenêtre principale, la zone d'espace de travail qui contient les cadres d'une page de bande dessinée et la zone de légende correspondant aux groupes, formulaires et listes à gauche de la fenêtre.

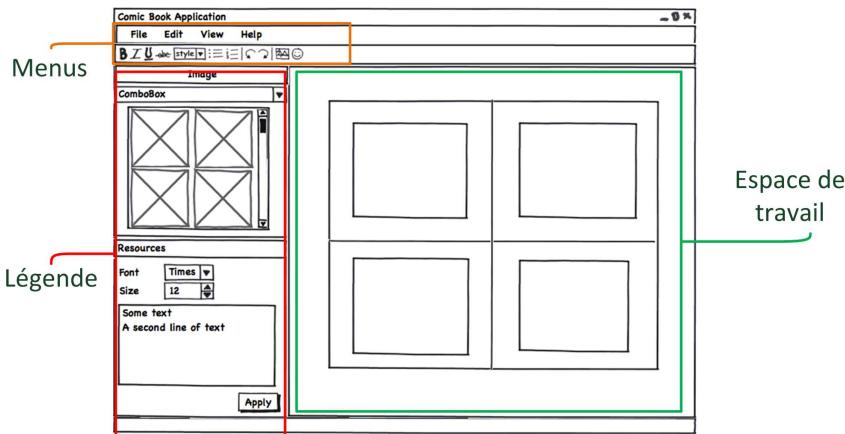


FIGURE 2.2 – Description de la fenêtre principale de l'application CBA

Nous considérons de manière globale que les applications à migrer doivent respecter une décomposition minimale permettant de séparer aisément l'UI et le Noyau Fonctionnel (NF). Si cette hypothèse est vérifiée, les différents aspects des UI des applications peuvent être décrites de manière générale comme suit :

- le **dialogue** entre l'utilisateur et le système permet d'organiser les interactions et les comportements d'une UI graphique,
- la **structure et le positionnement** des éléments graphiques qui décrivent l'organisation et les types de données d'une UI,
- le **style** pour décrire les tailles, les couleurs et les polices d'une UI graphique.

Nous considérons ces trois aspects comme autant de dimensions qui permettent de construire l'espace des problèmes lié à la migration d'une UI prévue pour un desktop pour que l'application s'exécute sur une table interactive. Pour chacune de ces dimensions nous étudions les problèmes soulevés par l'utilisation des objets tangibles/physiques comme moyen d'interactions. Dans le cadre des tables interactives, une **interaction tangible** consiste à utiliser des objets physiques (généralement en contact avec l'écran ou la surface d'affichage) comme des dispositifs de manipulation directes ou comme des moyens d'accès à des fonctionnalités. Pour une application "conception de bandes dessinées" (CBA) sur une table interactive, il est envisageable d'utiliser un stylo pour écrire dans les bulles de dialogue d'une bande dessinée. Dans ce cas le stylo est un **objet tangible** de manipulation directe. Par ailleurs, il est également possible que les concepteurs des bandes dessinées souhaitent utiliser un cube comme **objet tangible** dans le but sélectionner des images (représentant des personnages par exemple). Chaque face du cube sera associée à une catégorie de liste d'images ou d'objets graphiques utilisables dans une bande dessinée.

Nous étudions aussi les problèmes soulevés par la transformation d'une UI mono-utilisateur en UI multi-utilisateurs co-localisées. Dans le cadre des tables interactives, les applications peuvent naturellement être utilisées par plusieurs personnes ce qui est plus difficilement imaginable sur un smartphone. Il est alors nécessaire de re-concevoir l'UI de l'application qui est alors destinée à plusieurs

personnes (UI multi-utilisateurs) mais qui partagent le même écran (utilisateur co-localisé). L'application “conception de bandes dessinées” que l'on souhaite faire migrer sur table interactive peut être utilisée simultanément par plusieurs créateurs de bandes dessinées qui ainsi collaborent pour concevoir ensemble une nouvelle BD.

Dans les sections 2.2.2, 2.2.3 et 2.2.4, nous illustrons ces problèmes pour les différents aspects de l'application CBA décrite par la figure 2.2 ci-dessus.

2.2.2 Problèmes liés à la migration du dialogue

Le dialogue est une dimension des UI de départ qui varie fortement dans le cadre de la migration vers les tables interactives. En effet le nombre d'utilisateurs et les modalités d'interactions changent entre la source et la cible. Dans ce cadre, la transformation du dialogue de départ soulève des questions par rapport à l'utilisation des objets tangibles, à l'équivalence des interactions et au nombre d'utilisateurs. Cette section expose ces questions.

2.2.2.1 Utilisation des objets tangibles comme moyen d'interactions

Dans le cadre de l'application CBA, les dessinateurs de BD peuvent utiliser différents objets (cube, disque, etc.) pour afficher un menu de manière contextuelle dans le but d'effectuer une tâche de changement de couleur ou de police d'un texte. De manière générale, il est indispensable d'identifier dans l'UI départ les composants graphiques ou les fonctionnalités que les utilisateurs utiliseront par des **interactions tangibles**.

Les objets physiques peuvent aussi servir d'instruments de manipulations directes ou pour entrer des données. L'exemple d'un stylo pour sélectionner des options ou pour écrire dans le cadre de l'application CBA est envisageable. Dans ce cas, la migration des dialogues impacte à la fois l'UI et le NF. En effet, les **interactions tangibles** peuvent aussi nécessiter l'ajout dans le NF des traitements inexistant dans l'application de départ et la transformation de l'UI. Par exemple l'écriture à main levée avec un stylo à main levée peut nécessiter l'ajout d'un module de reconnaissance d'écriture si la plateforme cible n'offre pas cette fonctionnalité.

L'utilisation des objets tangibles comme moyen d'interaction pour les UI de la cible soulève un sous-ensemble de la dimension des problèmes liés à l'**équivalence des interactions** entre la source et la cible.

2.2.2.2 Équivalences des interactions entre la source et de la cible pendant la migration

Pour établir des équivalences entre les plateformes source et cible, il est indispensable d'étudier les instruments qui permettent aux utilisateurs de dialoguer avec les UI sur ces plateformes. En effet les différents instruments d'interactions offrent différentes modalités aux utilisateurs des applications. Dans le cas de la migration de l'application CBA du desktop vers une table interactive, il est indispensable de transposer par les interactions du clavier et de la souris sur les tables par des interactions tactiles ou par un objet tangible.

La transposition des dialogues de la source sur une table interactive impose des transformations des interactions initiales présentes entre l'UI desktop et l'utilisateur et entre l'UI et le noyau fonctionnel. Dans le but d'assurer la cohérence des dialogues après la migration, il est alors indispensable de s'assurer que les nouveaux dialogues permettent toujours une utilisation de l'application. Pour ce faire, nous pensons que les équivalences d'interactions doivent :

- Garantir que l'ensemble des fonctionnalités de l'application desktop restent accessibles sur les tables interactives après la migration. En effet, une migration ne doit pas altérer les fonctionnalités de l'application initiale sauf si la personne en charge de la migration le souhaite.
- Assurer que les dialogues modifiant les données d'une application prennent en compte les interactions de plusieurs utilisateurs.

2.2.2.3 Transformation du dialogue de la source pour une plateforme multi-utilisateurs et co-localisée

Dans le cadre de la migration, ce problème consiste aussi à transformer une application mono-utilisateur en une application multi-utilisateurs pour les tables interactives au cas où le nombre d'utilisateurs est supérieur à un. La migration dans ce cas impacte à la fois l'UI et le NF.

En ce qui concerne les transformations au niveau de l'UI, comment éviter d'avoir un dialogue gênant pour les autres utilisateurs ? Dans le cadre de l'application CBA par exemple les feedbacks, les alertes ou les messages d'erreurs provoqués par un utilisateur ne doivent pas être bloquants pour d'autres. Par ailleurs, comment décrire le couplage entre les utilisateurs et les dialogues ? En effet, une tâche⁵ ou une activité est exprimée par ensemble de dialogues entre l'utilisateur et le système pour une application desktop. Sur une table interactive, une tâche peut être effectuée par plusieurs utilisateurs de manière concurrente ou en collaboration. Dans ce cas, comment transformer une UI pour prendre en compte les tâches concurrentes ou les tâches effectuées en collaboration ? Comment surtout garantir que le noyau fonctionnel gère de manière cohérente la multiplicité des interfaces et les différents utilisateurs.

En ce qui concerne les transformations au niveau du NF, la migration de l'application CBA par exemple implique que les tâches de sauvegardes de fichiers ou de modifications des données puissent être faites par les utilisateurs en préservant la cohérence des données de l'application. Comment transformer le NF des applications mono-utilisateur pour prendre les dialogues **multi-utilisateurs** ?

Généralement les spécifications conceptuelles des applications à migrer ne sont pas disponibles. Il est indispensable de retrouver la description des tâches d'une application à partir de son code source par exemple pour transformer les dialogues. Comment retrouver et représenter la description des dialogues d'une application afin de les transformer ? Dans notre cadre de migration, si l'extraction des tâches est difficile ou impossible à partir du code source de l'application à migrer, est-il possible de transformer les dialogues sans description des tâches ? Quels sont les limites d'une transformation des dialogues non basée sur les tâches ?

2.2.3 Problèmes liés à la migration de la structure et du positionnement

Nous pensons que la migration des UI vers les tables interactives soulève des problèmes liés à la transformation de sa structure et du positionnement de ses éléments. En effet, la structure et le positionnement sont des caractéristiques des UI qui se déclinent de différentes manières suivant les plateformes et les utilisateurs. La transformation de la structure et du positionnement dans le cadre des tables interactives a pour objectif de favoriser l'accessibilité des éléments graphiques et de façon globale à l'utilisabilité des UI migrées.

La transformation de la structure et du positionnement des UI de départ soulève des questions sur les équivalences entre les éléments graphiques qui composent l'UI source et ceux proposés par la cible, le regroupement et le positionnement des éléments graphiques pour favoriser la collaboration.

5. Nous considérons les tâches et les activités impliquant les utilisateurs dans cas

2.2.3.1 Équivalences entre les éléments graphiques

Dans le cadre de l'application CBA par exemple, les menus ou les listes d'éléments peuvent être présentés sur les tables interactives dans le but d'avoir des interactions directes et intuitives. Les éléments graphiques proposés par les tables interactives permettent d'atteindre ce but. Par exemple, une liste contenant des items textuels peut être remplacée par une liste d'éléments d'images associées aux items sur la table interactive. L'ensemble des éléments de la liste textuelle doit être transposé sur la table interactive dans une liste équivalente.

De manière globale, les équivalences entre les éléments graphiques de la source et de la cible doivent préserver les données qu'ils contiennent. La multitude d'éléments graphiques et des plate-forme implique de mettre en place une solution équivalente entre les éléments graphiques automatisable.

2.2.3.2 Favoriser la collaboration par un regroupement et un positionnement adéquat des éléments graphiques

La fenêtre principale de l'application CBA est conçue en respectant la métaphore du bureau qui permet de décrire les applications pour les ordinateurs personnels [App95]. Les menus sont placés en haut à des positions fixes, les outils ou les légendes sont toujours placés à gauche ou à droite et la zone de travail au centre. Cette structuration en zones permet aux utilisateurs des desktops de développer des réflexes quelque soit l'application dans une approche similaire.

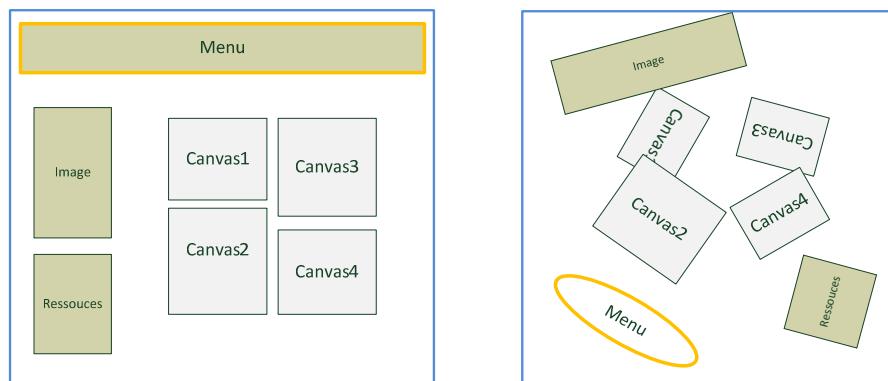


FIGURE 2.3 – Migration de la structure d'une UI pour desktop sur une table interactive

Sur une table interactive, cette structuration n'est pas recommandée car l'accessibilité des composants graphiques dépend directement de la position de l'utilisateur autour de la table : le haut pour un utilisateur peut être le bas d'un autre. La table interactive ne possède pas d'axe bas-haut immédiat et il est donc nécessaire de repenser la disposition de différentes fenêtre. Par exemple, pour l'application CBA, les différentes zones de la structure de départ doivent pouvoir être conservées mais chaque zone n'est plus associée à un espace géographique spécifique de l'écran. Sur la droite de la figure 2.3, les différentes zones de l'UI CBA de départ n'ont plus de position fixe sur une table interactive. On peut même imaginer dupliquer certaines zones pour qu'elle deviennent accessibles aux utilisateurs.

Les transformations de la structure et du positionnement de l'UI a pour objectif de favoriser l'**accessibilité**. Pour ce faire, il est important de déterminer les **groupes d'éléments graphiques** à transformer.

Dans le but de concevoir des UI utilisables, l'ingénierie des IHM préconise de prendre en compte des critères ergonomiques (ou des heuristiques) pendant la conception des UI [Sca86, NM90]. Nous considérons dans cette thèse que le respect des critères ergonomiques de conception pendant la migration permet de favoriser l'utilisabilité des UI obtenues. Dans ce cadre, il reste à définir comment on peut prendre en compte ces critères ergonomiques pendant la migration des UI. Il est donc important de les identifier et de les intégrer aux processus de migration pour pouvoir ultérieurement affiner les propositions d'UI et émettre des recommandations utilisables par la personne en charge de la migration.

2.2.3.3 Remarque

- La migration de la structure et du positionnement sont des transformations uniquement au niveau des UI des applications à migrer.

2.2.4 Problèmes liés à la migration du style

Dans notre cadre, la migration du style consiste à modifier l'aspect visuel des différents éléments graphiques de l'UI d'arrivée. Nous avons identifié trois caractéristiques des éléments graphiques pour décrire la modification du style.

- La **taille** des éléments graphiques est importante sur une table interactive car les dimensions des composants graphiques peuvent favoriser ou non un travail commun. Selon ce que l'on souhaite : véritable travail en commun ou chacun doit être capable de visualiser immédiatement l'activité d'un autre utilisateur ou au contraire, un travail plus individuel ou l'activité de l'un ne doit pas perturber l'activité de l'autre ; il sera nécessaire d'ajuster la taille des composants graphiques. En effet des composants graphiques trop larges peuvent gêner les autres utilisateurs et au contraire, des composants trop petit peuvent rendre difficile le travail commun.
- La **couleur** et la **police des textes** sont des propriétés subjectives car elles peuvent ne pas être acceptées par tout le monde. Cependant dans le cadre de la migration nous pensons que leur choix doivent être cohérents entre les éléments graphiques d'une UI. La cohérence du style consiste à s'assurer que les éléments graphiques de même type doivent avoir des couleurs ou des polices “compatibles” par exemple (respect d'une charte graphique).

Les tables interactives ont une surface d'affichage plus grande qu'un desktop ou une tablette. La taille des éléments d'une UI doit nécessairement être adaptée en fonction de cette surface mais aussi en fonction du nombre d'utilisateurs prévus afin de faciliter la participation de tous.

2.2.4.1 Cohérence globale du style des UI migrées

Il est important que les couleurs, les polices, les icônes, les images ou la taille des éléments graphiques soient définies suivant des recommandations pour assurer une cohérence globale. L'homogénéité de caractéristiques⁶ pour chaque type d'éléments graphiques constitue la **cohérence globale** à un style. Il ne s'agit pas d'un problème spécifique aux tables interactives mais la taille de l'écran disponible pose d'une manière nouvelle le respect des chartes graphique et des recommandations de style.

6. Les couleurs, les polices, les icônes, les images ou la taille des éléments graphiques

2.2.5 Espace des problèmes liés la migration vers les tables interactives

Les problèmes liés à la migration que nous avons étudiés à la section 2.2 peuvent être représentés à l'aide d'un espace à trois dimensions décrit par la figure 2.4.

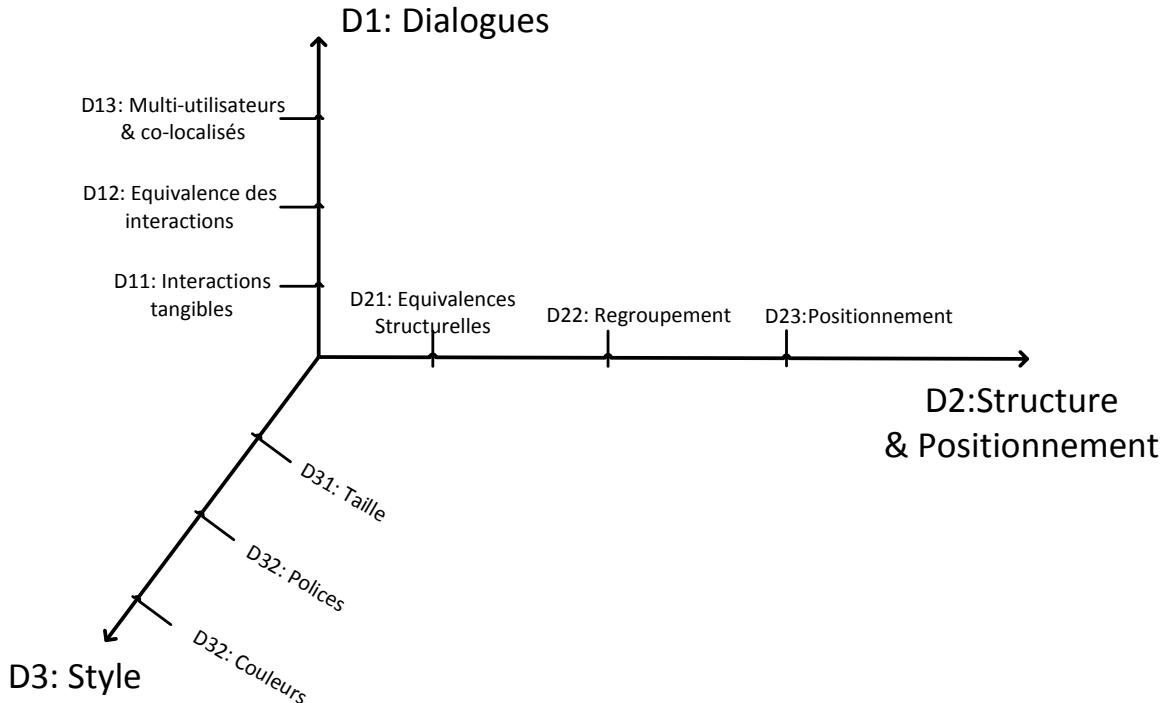


FIGURE 2.4 – Espace problèmes de la migration des UI vers les tables interactives

La dimension **D1** correspond aux problèmes de migration des dialogues. Elle consiste à transformer l'UI et porter le NF des applications. Il est indispensable d'avoir des processus génériques pour prendre en compte plusieurs applications et réduire les coûts de la migration. Elle regroupe trois sous-dimensions.

- La dimension **D11** correspondant aux problèmes liés à la transformation des dialogues par ajout des **interactions tangibles** aux applications de départ.
- La dimension **D12** correspond aux problèmes d'**équivalence des interactions** des UI migrées.
- La dimension **D13** correspondant aux problèmes liés à la transformation des dialogues mono-utilisateur en **multi-utilisateurs co-localisés**.

Ensuite la dimension **D2** correspond aux problèmes de transformation de la structure et du positionnement des éléments graphiques. Elle consiste à transformer l'UI de l'application source. Dans ce cas, il est important d'avoir un processus qui prend en compte les critères ergonomiques de conception afin de garantir l'utilisabilité. La généricité du processus de transformation est importante. Elle regroupe deux sous-dimensions.

- La dimension **D21** correspondant aux problèmes liés à l'**équivalence structurelle** des éléments graphiques de la source et de la cible.
- La dimension **D22** correspondant aux problèmes de **regroupement** des éléments graphiques des UI migrées.
- La dimension **D23** correspondant aux problèmes de **positionnement** des éléments graphiques des UI migrées.

Enfin la dimension **D3** correspond aux problèmes liés à la transformation du style de l'UI départ pour favoriser leur accessibilité. Dans le but de réduire la charge de travail des concepteurs et d'assurer l'homogénéité, il est important d'utiliser des styles génériques. Cependant la flexibilité de la migration du style permet aussi de personnaliser les UI cibles. Elle regroupe deux sous-ensembles de problèmes :

- La dimension **D31** correspondant aux problèmes liés à la **taille** des éléments graphiques sur un espace de travail partagé.
- La dimension **D32** correspondant aux problèmes liés à la **police** des textes des UI migrées.
- La dimension **D33** correspondant aux problèmes liés à la **couleurs** des éléments graphiques des UI migrées.

2.3 Périmètres de la migration des UI vers les tables interactives

Dans notre contexte, les applications à migrer sont des systèmes interactifs (SI) conçus en respectant un modèle d'architecture et identifiant un noyau fonctionnel (NF) et des interfaces utilisateurs (UI). Les modèles d'architecture sont des patrons de conception logiciel, ils préconisent des stratégies de répartition des services qui se traduisent par un ensemble de constituants logiciels. Il existe plusieurs modèles d'architecture qui permettent la séparation UI-NF.

- Le modèle d'architecture de ARCH [Dev92] par exemple, permet une séparation entre le NF, le Contrôleur de Dialogue (CD) et la Présentation (P). Elle permet aussi la migration des composants de l'UI (CD et P) sans modification des composants du NF.
- Le modèle d'architecture PAC-Amodeus [Nig94] est basé sur ARCH, il permet de définir plusieurs contrôleurs de dialogue pour une application grâce aux agents PAC [Nig94].
- Le modèle d'architecture MVC [KP⁺88] permet la conception des SI réutilisables. Il divise les applications en trois types de composants : le modèle (M), la vue (V) et le contrôleur (C). Le modèle est une représentation du domaine d'une application, il peut contenir des données, des services, etc. et il fait partie du NF d'une application. La vue est la structure de l'UI d'une application, elle est constituée des éléments d'une bibliothèque graphique. Le contrôleur est une interface entre le modèle, la vue et les dispositifs d'interactions en entrée. Cette architecture permet une migration de la vue sans modification du modèle et du contrôleur.

Nous présentons à la section 2.3.1 notre hypothèse de travail. La section 2.3.2 présente les pistes de migration des UI et la section 2.4, les objectifs de notre travail.

2.3.1 Hypothèse de travail

Les applications à migrer ont une décomposition fonctionnelle minimale qui comprend une interface utilisateur (UI) et un noyau fonctionnel (NF). Cette décomposition identifie clairement la partie qu'il faut migrer (l'UI) et la partie qui sera réutilisée sur la plateforme cible (le NF). Dans cette thèse, pour ne pas aborder simultanément tous les problèmes de migration et parce que ceux-ci ont été abondamment étudiés par ailleurs, nous considérons que nous pouvons réutiliser le NF de l'application source sur la plateforme cible sans modifier son code. Il offre donc les mêmes fonctionnalités. Il est évident que si ce NF devrait lui-même être migrer, quel qu'en soit les raisons, cela doit être effectivement traité, éventuellement par les propositions décrites dans [TKB78, TKR10] du NF par exemple.

Les deux hypothèses simplificatrices que nous faisons sont现实的. La plupart des applications interactives adoptent aujourd'hui une architecture proche du modèle MVC très largement répandu permettant d'identifier très clairement la partie UI et la partie NF de l'application. Par ailleurs, la table surface que nous avons utilisé - une des rares tables interactives disponibles - utilisent le même système d'exploitation que celui utilisé par la très grande majorité des desktops rendant inutile le

portable du NF. Si celui-ci s'avérait nécessaire, il serait alors possible d'utiliser une solution à base de machine virtuelle.

Il est évident que notre approche conserve les dialogues existants entre le NF et l'UI et que nous n'avons que très peu abordé les questions relatives à l'évolution de ceux-ci lors de la migration.

2.3.2 Pistes de migration

A partir des hypothèses précédemment exposées, la migration des UI des applications existantes vers une nouvelle plateforme tout en conservant le NF peut se faire de plusieurs manières :

- Une **nouvelle conception de l'UI** pour la plateforme cible sans tenir compte de l'UI source.
- Une **déduction de l'UI** à partir des spécifications du NF de départ.
- Une **adaptation de l'UI** de départ par rapport à la plateforme cible.

2.3.2.1 Nouvelle conception de l'UI

Cette approche consiste à concevoir une nouvelle UI pour la plateforme cible sans nécessairement tenir compte de l'UI de départ. Cette nouvelle conception permet de prendre en compte les spécificités de la plateforme d'arrivée et d'intégrer ses critères de conception des UI (chaque plateforme est “livrée” avec un guide du bon usage). Les approches de conception des UI telles que DIANE+ [Bar88] ou MUSE [LL09] et les approches de conception basées sur des modèles [MPV11] peuvent être utilisées pour concevoir la nouvelle UI. DIANE+ et MUSE permettent d'inclure dans la conception de l'UI des critères ergonomiques [Van97].

L'avantage d'une nouvelle conception est de construire une UI sur mesure, prenant en compte l'ensemble des nouveaux besoins liés à la plateforme cible. Cependant le temps pris pour la conception de cette nouvelle UI comprend nécessairement le temps pris pour la transformation des dialogues de l'UI source et les temps nécessaire à la conception de la structure, du positionnement et du style de la nouvelle UI. Ces différents temps constituent des coûts de mise en œuvre non négligeables. Cette piste de solution, ne capitalisant pas sur le travail déjà effectué, demandant des compétences métiers fortes et un temps non négligeable de mise en œuvre, ne nous permet pas de **réduire les coûts de mise en œuvre** des applications pour les tables interactives.

2.3.2.2 Déduction de l'UI à partir du NF

La deuxième piste de migration préconise de s'appuyer sur le NF de l'application de départ, particulièrement sur les liens entre ce NF et l'UI. Il est en effet possible de déduire une UI en se basant sur la représentation abstraite des liens entre le NF et l'UI. Il existe des travaux qui préconisent la génération des UI à partir des descriptions de services Web par exemple [KKM03]. Par ailleurs l'approche ALIAS [JOF11] propose des modèles pour une représentation abstraite du NF et des liens entre NF et UI qui peuvent être utilisés pour déduire les UI à partir des NF. La déduction des UI en se basant sur les NF implique la déduction à partir du NF des éléments suivants :

- les composants graphiques qui composent l'UI,
- la structure et le positionnement (layout) de ses composants graphiques,
- les dialogues et les enchaînement des fenêtres et des activités de l'UI.

Les travaux sur la génération d'une UI à partir du NF [TC02] montrent qu'en plus du NF, il est indispensable d'avoir une spécification des tâches sous forme d'un contrôleur de dialogue et d'adaptateur du NF (dans une architecture ARCH). Le NF ne permet usuellement pas de déduire la structure et les différents types de regroupements des composants graphiques (fenêtres, panels, etc.). Les UI

obtenues par une déduction à partir du NF sont **utilisables** à condition d'avoir une description des tâches pour permettre l'organisation des dialogues et de sa structure.

2.3.2.3 Adaptation de l'UI par recherche d'équivalences

La troisième piste de migration est celle qui préconise de s'appuyer sur les spécifications de l'UI à migrer et de les adapter à la plateforme d'arrivée. Ces spécifications sont décrites par des modèles abstraits. Les modèles servent à décrire les différentes dimensions des UI tels que les dialogues décrits par l'UI, le placement ou le layout des éléments de l'UI mais aussi les styles de présentations des caractéristiques visuelles (tailles et couleurs des textes, etc.). Les différents modèles abstraits des UI à migrer peuvent être retrouvé par abstraction à partir du code source par exemple [BS08, BV02, GPF10, TS99].

L'adaptation des UI à migrer pour les tables interactives peut se faire en transformant partiellement ou en totalité les trois dimensions identifiées.

- L'adaptation des dialogues consiste à produire des dialogues équivalents à partir de l'UI de départ et en prenant en compte les caractéristiques de la cible : introduction d'interactions tangibles et présence potentielle de plusieurs utilisateurs.
- L'adaptation de la structure consiste à produire pour la plateforme cible une structure équivalente de l'UI de départ en prenant en compte les problèmes liés au regroupement et au positionnement des composants graphiques sur les tables interactives pour lesquels l'axe haut-bas dépend de la position de l'utilisateur. Selon les cas, la structure et le positionnement de départ peuvent être ignorés ou réutilisés partiellement pendant la migration. Dans le cas où ils sont ignorés, il est indispensable de proposer des outils pour les réintroduire facilement.
- En ce qui concerne le style, il est indispensable de le réintroduire pour la cible, en respectant la charte graphique et en favorisant l'accessibilité des éléments graphiques. En effet, la taille, la couleur ou la police des textes des tables interactives sont différentes de celles des desktops. Le style de la cible peut se baser sur un modèle de style lié aux tables interactives.

L'introduction manuelle du positionnement et du style permet d'avoir une approche de migration des UI plus de flexibilité mais avec un coût de mise en œuvre non négligeable. Au contraire, une adaptation automatique du positionnement et du style présente un coût de migration faible mais au prix d'une moindre flexibilité et un faible respect des critères ergonomiques de la cible.

2.4 Objectifs de la migration des UI vers les tables interactives

Nous avons fait le choix dans cette thèse d'explorer la troisième piste de migration (cf. section 2.3) et d'appliquer nos travaux aux tables interactives qui sont des plateformes innovantes provocant un changement assez important dans la conception des UI. En effet, il ne s'agit pas uniquement de trouver les composants similaires entre les toolkits de la plateforme source et de la plateforme cible mais d'introduire au sein de l'UI de nouveaux dispositifs d'interactions par l'utilisation d'interfaces tangibles et de prendre en compte la réelle possibilité qu'ont des utilisateurs de pouvoir enfin partager de manière intuitive, pour peu que l'UI soit bien construite, une même application. La migration d'une application existante sans avoir à reconcevoir l'UI est à même d'enrichir le catalogue d'application disponible sur les tables interactives par un abaissement important du coût de mise en œuvre de ces applications.

Ces arguments nous ont amenés à nous poser deux questions :

- d'une part, nous devons adapter les différentes dimensions d'une UI fondamentalement conçue pour une personne en une UI collaborative et tangible,

- et d'autre part, il est nécessaire d'inclure dans le processus de migration la prise en compte de critères ergonomiques de conception[Sca86, NM90], liés à la plateforme cible pour réduire effectivement le coût de la migration tout en concevant une UI réellement utilisable.

La réponse à ces deux questions doit nous permettre d'atteindre les objectifs que nous nous sommes fixés dans le cadre de cette thèse. Pour ce faire, la deuxième partie du document présente les spécificités des tables interactives qui constituent notre domaine d'étude pour identifier les critères de conception. Nous considérons que ces critères sont les critères ergonomiques de conception qui doivent être affinés en recommandations lors de la migration des UI vers les tables interactives.

Deuxième partie

Domaine d'étude

Tables interactives et Migration des UI

Sommaire

3.1 Modèle d’interactions d’une table interactive	23
3.1.1 Les dispositifs matériels d’interactions d’une table interactive	24
3.1.2 Bibliothèques graphiques des tables interactives	27
3.1.3 Modalités d’interactions	29
3.1.4 Modèle d’interactions abstraites pour la migration des UI	31
3.2 Principes de conception des UI pour les tables interactives	33
3.2.1 Propriétés caractéristiques des tables interactives	33
3.2.2 Critères ergonomiques de conception des UI	35
3.2.3 Recommandations pour la migration des UI vers les tables interactives	38
3.3 Synthèse	43

NOUS présentons dans ce chapitre les spécificités des tables interactives de manière générale afin d’identifier les recommandations pour la migration des UI. Pour atteindre cet objectif, nous étudions dans la section 3.1 les instruments d’interactions ou de dialogue qui sont à la fois des dispositifs matériels et logiciels. Puis les modèles permettant de décrire les dialogues indépendamment des instruments d’une plateforme sont étudiés. La section 3.2 présente les principes de conception en décrivant les critères ergonomiques de conception et les recommandations pour la migration des UI vers les tables interactives. La section 3.3 synthétise les principaux travaux de cette étude sur les tables interactives dans le but de mettre en évidence les principes qui doivent être respectés pour migrer une UI vers cette plateforme cible.

3.1 Modèle d’interactions d’une table interactive

Les instruments d’interactions constituent l’ensemble des dispositifs matériels et logiciels d’une table interactive qui permettent d’interagir avec un SI. Les dispositifs matériels d’interactions (cf. figure 3.1) sont des moyens d’interactions en entrée (actions) ou en sortie (réactions et feedback). Les bibliothèques graphiques encapsulent les paradigmes de manipulation des différents dispositifs présent et permettent de décrire des interfaces utilisateurs.

Dans cette section nous nous appuierons sur différentes tables interactives : Microsoft PixelSense, TangiSense, DiamondTouch, etc. Cette diversité nous permet de caractériser les dispositifs d’interactions (cf section 3.1.1), les bibliothèques graphiques (cf section 3.1.2) et les modalités d’interactions des tables interactives (cf section 3.1.3).

Nous étudions ces trois tables interactives dans le but de caractériser les différents types d’UI qu’il est possible de mettre en œuvre en fonction des instruments d’interactions. Ces trois tables interactives décrivent des UI tactiles, des UI tangibles et des UI collaboratives.



FIGURE 3.1 – Modèle d’interactions instrumentale d’une table interactive

3.1.1 Les dispositifs matériels d’interactions d’une table interactive

Dans ce paragraphe nous étudions les instruments d’interactions de trois tables interactives représentatives. Nous étudions d’abord DiamondTouch [DL01] qui est l’une des premières tables interactives utilisées dans un cadre non expérimental, elle fut industrialisée par MERL [Mit], nous l’étudions car elle comporte une bibliothèque graphique DiamondSpin qui permet de décrire des interactions multi-utilisateurs, collaboratives et tactiles. La seconde table étudiée est metaDESK [UI97] qui n’accepte des interactions que via des objets tangibles. La troisième table étudiée est la famille Microsoft PixelSense 1.0 et 2.0 [Mic11] qui possèdent une bibliothèque graphique complète qui permet de construire des interactions collaboratives, tangibles et tactiles.

L’étude effectuée dans cette section a pour objectifs de caractériser les différentes catégories de dispositifs d’interactions.

3.1.1.1 DiamondTouch

La table interactive DiamondTouch est multi-touche et multi-utilisateurs. Elle possède une surface tactile capacitive pour les interactions en entrée et un projecteur pour les interactions en sortie qui sont reliés à un ordinateur. Sa surface tactile fonctionne grâce à un réseau d’antennes qui transmet des signaux très faible. Lorsqu’un utilisateur touche la surface tactile, des signaux capacitifs sont associés à un récepteur en partant du point de contact entre l’utilisateur et la surface et en passant dans son corps. Les récepteurs sont associés aux différents utilisateurs de la table par l’intermédiaire d’un tapis placé sur leur chaise. La figure 3.2 présente les différents composants de la table DiamondTouch. Les récepteurs (receiver) sont des circuits électroniques qui déterminent les coordonnées du point de la surface touchée par un utilisateur. Le transmetteur (transmitter) est constitué de plusieurs antennes de la surface tactile, il est couplé de manière capacitif aux récepteurs de chaque utilisateur.

Cette table est aussi multi-utilisateurs, car elle permet simultanément plusieurs touches de différents utilisateurs. Chaque touche peut être associée à un utilisateur précis grâce à son tapis. Par sa capacité à reconnaître les différents utilisateurs, cette table permet de décrire des UI collaboratives qui permettent de partager la même UI et donc d’accéder simultanément à la même application. Il est aussi possible, de créer pour chaque utilisateur son propre espace de travail personnel. L’utilisation de la table DiamondTouch impose une répartition géographique fixe des utilisateurs autour de la table. Elle ne supporte pas les interactions tangibles car elle ne peut pas détecter des objets ou des tags.

Pour faire migrer l’UI de l’application CBA vers la table DiamondTouch impose de connaître le nombre d’utilisateurs de l’UI migrée car chaque utilisateur doit être en contact avec un tapis (assis ou debout). Sur la figure 3.2, les tapis sont placés sur les chaises des utilisateurs pour servir de récepteur, par contre la surface d’affichage (ou la table) est un transmetteur.

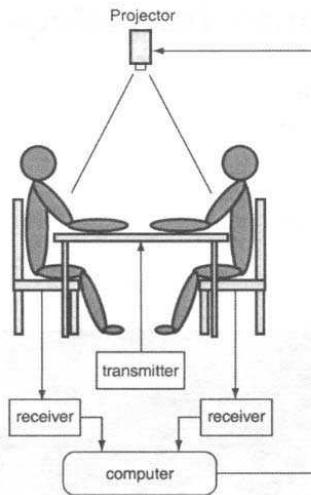


FIGURE 3.2 – Table interactive DiamondTouch

3.1.1.2 metaDESK

C'est la première prototype de table interactive conçue en 1997 par Tangible Media Group du MIT Media Lab pour comprendre le conception des interface utilisateur tangible ou Tangible User Interface (TUI). Les moyens d'interactions de la table metaDESK sont constitués d'objets tangibles, des caméras infrarouges, d'une caméra vidéo et d'un vidéo projecteur. Ces objets permettent à chaque utilisateur de manipuler des objets virtuels associés aux objets physiques et par conséquent de construire des TUI. La reconnaissance des objets tangibles se fait par la caméra vidéo et les cameras infrarouges en utilisant une méthode de reconnaissance et de tracking des objets sur la table. La table metaDESK est utilisée dans le cadre d'une application de localisation géographique ; les objets physiques dans ce cas représentent des bâtiments, des ponts, etc. Il existe aussi des tables interactives conçues pour d'autres usages ; les tables AudioPad [PRI02] et Xenakis [BCL⁺08] par exemple disposent des TUI pour composer et écouter de la musique. Les objets tangibles dans ce cadre ont des formes proches des boutons d'une console de mixage pour faciliter les interactions des utilisateurs.

Contrairement à la table DiamondTouch, la table metaDESK ne limite pas le nombre d'utilisateurs et ceux-ci ne sont pas assignés à un emplacement fixe et peuvent se déplacer autour de la table. La table metaDESK ne permet pas aux différents utilisateurs d'avoir un espace personnel comme DiamondTouch.

Ulmer et Ishii [IU97] définissent un TUI comme un système interactif qui utilise des objets physiques pour représenter et utiliser des informations digitales. Le mapping entre le monde physique et digital peut se faire en représentant les différents composants graphiques d'une UI graphique à l'aide d'objets concrets. Par exemple, les concepteurs de la table metaDESK [IU97], associent (cf. figure 3.3) une lentille à une fenêtre, un plateau à un menu, etc.

Dans le cadre de la migration d'une UI desktop vers la table metaDESK, toutes les interactions de l'UI de départ doivent être adaptées ou émulées par l'intermédiaire d'objets tangibles. Par exemple, le formulaire *Ressources* de l'application CBA doit être affiché et rempli uniquement par l'utilisation des objets tangibles. Il en est de même pour la sélection de la taille ou de la police qui peuvent être émulées avec des objets circulaires en les faisant tourner par exemple. Par ailleurs, l'émulation d'un clavier à l'aide d'un objet tangible pour la saisie des textes est possible mais n'est pas facile

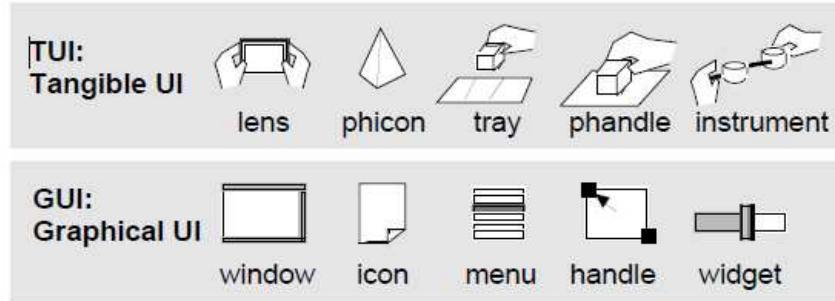


FIGURE 3.3 – Instanciation physique des éléments GUI dans TUI

à utiliser [USJ⁺08]. Toutes les applications ne se prêtent pas à être utilisée sur des tables interactives uniquement tangibles comme metaDESK ou TangiSense [KLL⁺09]. Elles sont en généralement utilisées pour des applications de réalité augmentée ou pour des applications de cartographie.

3.1.1.3 Microsoft PixelSense

La première version de la table Microsoft PixelSense est constituée d'un vidéo projecteur placé sous la surface tactile pour les interactions en sortie, de cinq caméras infrarouges pour détecter les formes des objets ou des doigts des utilisateurs, d'un clavier virtuel et des dispositifs sonores. La deuxième version est équipée d'un écran à cristaux liquides (LCD) équipé de la technologie PixelSense qui permet une reconnaissance des interactions en entrée sans usage de caméras. Cette technologie détecte les doigts, les tags ou les objets tangibles grâce à plusieurs dispositifs de rétro éclairage qui produisent une lumière infrarouge. Cette lumière est d'abord réfléchie sur les objets ou les doigts posés sur la surface tactile. Ensuite, un capteur LCD est utilisé pour recréer l'image de ce qui est posé sur la surface à partir de la lumière réfléchie. Enfin l'image est interprétée par des techniques d'analyse et le résultat est envoyé à une unité centrale.

Les versions 1 et 2 des Microsoft PixelSense permettent des UI tactiles multi-contacts. Elle supporte jusqu'à 50 contacts simultanément et permet donc de décrire des UI multi-utilisateurs dans la limite des contacts supportés. Elle permet aussi de décrire des UI tangibles par la reconnaissance des tags et la forme des objets. L'utilisation de tags permet de ne pas limiter les objets à utiliser dans la description des UI. Les tags sont des codes barres à deux dimensions qui sont facilement identifiables par la table surface. La Surface offre deux types tags : Identity tags (tags avec une plage de valeurs sur 128 bits) et Byte tags(tags avec une plage de valeurs sur 8 bits). Les tags peuvent être utilisés pour différentes actions :

- reconnaître des objets physiques ou les distinguer parmi plusieurs,
- déclencher une commande ou une action, par exemple : afficher un menu ou une application par exemple,
- pointer et orienter une application par un objet tagué, par exemple dans le but de sélectionner un élément.

Tableaux blancs interactifs La version 2.0 des tables PixelSense peut être utilisée comme un tableau blanc interactif. Les tableaux blancs interactifs[GMA05] (TBI) constituent aussi des tables. Le TBI Magic Table [Ber03] est constitué d'une surface blanche, d'un vidéo projecteur et d'une caméra relié à une unité centrale. La caméra est couplée à un composant logiciel capable d'interpréter les symboles de la surface blanche. Les interactions en entrée sont possibles avec des jetons (qui sont aussi

des objets tangibles) pour créer, déplacer, redimensionner ou supprimer des éléments graphiques. Les jetons sont associés à des gestes prédéfinis qui seront détectés par la caméra.

Magic Table est aussi utilisé pour numériser facilement le contenu d'un tableau blanc. Dans le cadre des UI graphiques, il peut être utilisé simultanément par plus d'une personne en manipulant des jetons. Contrairement aux Microsoft PixelSense, le nombre d'objets tangibles différents utilisables par le TBI MagicTable est limité au nombre de formes détectables par sa camera.

3.1.1.4 Constat

Les tables interactives ont de manière générale deux types de dispositifs physiques d'interactions (cf tableau 3.1) : les dispositifs d'interactions en entrée qui peuvent être tactiles et/ou tangibles et les dispositifs d'interactions en sortie qui sont des surfaces d'affichage. Ces surfaces sont de tailles variables et de différentes formes (rectangulaire, circulaire, etc.). Elles peuvent être disposées de manière horizontale (pour DiamondTouch, metaDESK, Microsoft PixelSense 1.0 et 2.0) ou de manière verticale (pour Microsoft PixelSense 2.0 ou les tableaux blancs interactifs).

Le nombre d'utilisateurs et leurs dispositions autour de la surface d'affichage sont des éléments qui permettent de caractériser les interactions des tables interactives. En effet ces deux caractéristiques impactent la conception des UI car une UI destinée à plusieurs personnes doit permettre l'accessibilité des différentes fonctionnalités à tous les utilisateurs. Elle doit aussi prendre en compte le partage des éléments de l'UI (menus, visualisation des contenus, etc.). Il est important aussi de savoir si les utilisateurs ont ou non un espace qui leur est propre.

Certaines tables interactives possèdent des équipements matériels leur permettant de concevoir des UI multi-utilisateurs, co-localisées, tactiles et tangibles. Si l'UI de l'application à migrer ne possède pas ces caractéristiques, il est alors nécessaire de résoudre les différents problèmes soulevés par l'introduction des ces nouvelles contraintes résultant de l'utilisation d'une table interactive et en particulier de décider du nombre d'utilisateurs qui pourront interagir avec l'application et de leurs dispositions par rapport à la surface d'affichage.

Tables Interactives	Dispositifs d'interactions d'entrée	Dispositifs d'interactions de sortie
Diamond Touch	Écran Capacitif	Vidéo Projecteur
metaDesk	Objet Tangible, Camera Infrarouge, Caméra Vidéo	Vidéo Projecteur
Microsoft PixelSense V1&2	Technologie PixelSense (V2), Caméra Infrarouge(V1), Tags, Objets Tangibles	Écran LCD

TABLE 3.1 – Synthèse des dispositifs physiques d'interactions pour 3 tables interactives

3.1.2 Bibliothèques graphiques des tables interactives

Les bibliothèques graphiques sont des boîtes à outils logiciels qui contiennent des éléments pour construire des UI graphiques et d'utiliser l'ensemble des possibilités offertes par le terminal que ce soit en entrée (reconnaissance d'objets tangibles, tag, multi-touch, multi-utilisateurs) ou en sortie (affichage). Nous présentons donc dans cette section, les principaux éléments que l'on retrouve dans les bibliothèques graphiques associées aux tables étudiées.

3.1.2.1 DiamondSpin

DiamondSpin [SVFR04] est une bibliothèque graphique pour table interactive qui offre des composants graphiques adaptés à une utilisation de la table par plusieurs utilisateurs. Les composants graphiques sont basés sur JavaSwing et sont compatible avec OpenGL. Les composants graphiques de cette bibliothèque permettent une manipulation directe des documents visuels en utilisant des doigts, un stylet ou un clavier comme des moyens d'interactions. Cette bibliothèque graphique permet aussi de créer et de gérer des espaces privés associés à chaque utilisateur d'une application collaborative et co-localisée. Les composants graphiques *DSContainer*, *DSPanel*, *DSWindow*, *DSFrame* (cf. figure 3.4) permettent par exemple d'avoir un container qui regroupe un ensemble de composants graphiques qu'un utilisateur peut déplacer en fonction de sa position.



FIGURE 3.4 – Instances des containers DiamondSpin

3.1.2.2 Surface SDK 1.0 et 2.0

[Mic12a] sont des API et des boîtes à outils permettant de développer des applications pour une table interactive Microsoft (1.0 et 2.0). Ils font partie du Framework .Net et offrent des bibliothèques graphiques pour concevoir des UI WPF [Mic12b] et XNA [Mic12c]. Ces bibliothèques graphiques permettent la reconnaissance des formes d’objets, l’utilisation des tags, la gestion des 50 points de contacts simultanés, la détection de l’orientation des touches, etc. La construction d’une UI avec ce SDK est bien plus aisé qu’avec la bibliothèque graphique DiamondSpin car elle évite par exemple au programmeur la gestion du déplacement des éléments ou la gestion de la rotation des composants graphiques. En effet un *ScatterView* (Figure 3.5) définie le déplacement ou la rotation de manière intrinsèque.

3.1.2.3 Constat

La bibliothèque Surface SDK permet de décrire des TUI à l'aide de tags. De manière générale, les bibliothèques graphiques des tables interactives offrent des composants graphiques qui ont des interactions (rotation, redimensionnement, déplacement, tags, etc.) adaptées pour la description des UI pour les tables interactives. Elles permettent d'affiner les caractéristiques des tables interactives en



FIGURE 3.5 – Exemple de ScatterView

précisant si une table interactive supporte ou non des interactions tangibles, si elle a des composants graphiques accessibles par tous les utilisateurs par exemple.

Les bibliothèques graphiques servent aussi pour spécialiser une table interactive dans un domaine applicatif précis. Dans ce cas, les bibliothèques graphiques offrent des composants graphiques spécifiques à des domaines d'application. Par exemple la table metaDesk est uniquement destinée à la cartographie, à ce titre elle dispose des composants graphiques propres pour afficher facilement un plan. Cependant, DiamondSpin et les Surface SDK 1.0 & 2.0 sont des bibliothèques graphiques indépendantes des domaines applicatifs. Dans le cadre de la migration des UI desktop ces dernières offrent des composants graphiques équivalents à la source.

3.1.3 Modalités d'interactions

Nigay [Nig94] propose la définition d'une modalité d'interaction comme un couple $\langle d, l \rangle$ constitué d'un dispositif d'interactions et d'un langage d'interactions :

- d désigne un dispositif physique. Par exemple, une souris, une caméra, un écran, un haut-parleur,
- l dénote un langage d'interaction constitué d'un ensemble structuré de signes permettant de décrire les fonctions de communication. Il peut reposer sur un langage pseudo-naturel, un graphe ou une table de correspondance.

La migration des UI vers les tables interactives implique bien souvent un changement des dispositifs d'interactions et impose donc de définir des équivalences entre les modalités d'interactions des plateformes de départ et celles des tables interactives. Dans cette section nous abordons la problématique liée aux changements des modalités d'interactions des UI à migrer. En effet comment décrire les interactions de l'UI de départ à l'aide des modalités d'interactions de la plateforme d'arrivée ? Pour répondre à cette question nous étudions au paragraphe 3.1.1 les problèmes liés aux changements de modalités d'interactions pendant la migration. Puis au paragraphe 3.1.2 nous présentons quelques modèles d'interactions abstraites qui permet de décrire les interactions indépendamment des modalités d'interactions.

3.1.3.1 Changement de modalité d'interactions

Considérons comme plateforme de départ un desktop composé d'un écran comme moyen d'interactions en sortie et d'un clavier et d'une souris comme moyen d'interactions en entrée.

- La modalité d’interactions en sortie M1 du desktop est décrite par l’écran et par le langage de description des UI 2D (L1), $M1 = \langle \text{Ecran}, L1 \rangle$. Le langage L1 correspond aux composants graphiques tels que labels, champ de texte, boîte de dialogue, etc. d’une bibliothèque graphique.
- La modalité d’interactions en entrée M2 du desktop est décrite par le clavier et par le langage des commandes (L2), $M2 = \langle \text{Clavier}, L2 \rangle$. Le langage L2 décrit pour chaque action les tâches réalisables à l’aide d’un clavier sur une UI graphique, ces actions sont par exemple : copier avec Ctrl+C, couper avec Ctrl+X, coller avec Ctrl+V, saisir un texte avec les touches alphanumériques, valider avec la touche entrée, etc.
- La modalité d’interactions en entrée M3 du desktop est décrite par la souris et par le langage de manipulation directe d’une UI 2D (L3), $M3 = \langle \text{Souris}, L3 \rangle$. Le langage L3 décrit les actions réalisables à l’aide d’une souris sur une UI graphique, ces actions sont par exemple : cliquer et valider pour sélectionner un composant graphique, cliquer et déplacer pour sélectionner un texte, déplacer un élément, redimensionner, etc.

Considérons maintenant que la table interactive cible est composée d’un écran tactile, d’un clavier virtuel et de la reconnaissance des objets physiques comme moyens d’interactions.

- La modalité d’interactions en sortie M’1 de la table interactive est décrite par l’écran tactile et par le langage de description des UI 2D (L’1), $M'1 = \langle \text{Ecran Tactile}, L'1 \rangle$. Le langage L’1 correspond à la bibliothèque graphique de la table interactive qui contient les composants graphiques tels que labels, champ de texte, images, fenêtre, etc.
- La modalité d’interactions en entrée M’2 de la table interactive est décrite par l’écran tactile et par le langage de manipulation tactile d’une UI 2D (L’2), $M'2 = \langle \text{Ecran Tactile}, L'2 \rangle$. Le langage L’2 décrit les actions utilisateurs sur un écran tactile. Ces actions sont par exemple : toucher avec un doigt pour activer ou sélectionner un élément, toucher avec deux doigts et déplacer pour agrandir, réduire, tourner des composants graphiques, etc.
- La modalité d’interactions en entrée M’3 de la table interactive est décrite par le clavier virtuel et par le langage de commande (L’3), $M'3 = \langle \text{Ecran Tactile}, L'3 \rangle$. Le langage L’3 décrit les actions utilisateurs réalisables avec un clavier virtuel tel que saisir un texte.
- La modalité d’interactions en entrée M’4 de la table interactive est décrite par l’écran tactile et par le langage de manipulation des objets tangibles (L’4), $M'4 = \langle \text{Objets Tangibles}, L'4 \rangle$. Le langage L’4 décrit les actions utilisateurs sur un écran tactile à l’aide des objets tangibles. Ces actions sont par exemple : poser un objet pour afficher un menu ou un formulaire, déplacer un objet physique pour déplacer l’objet virtuel associé, tourner un objet physique pour sélectionner une fonctionnalité, etc.

Les langages d’interactions L1 et L’1 permettent de décrire les interactions en sortie des UI des applications source et cible. Ces langages peuvent être modélisés en se basant sur des boîtes à outils indépendantes des dispositifs d’interactions en sortie. Crease dans [Cre01] propose une boîte à outils décrivant des widgets multimodales et indépendantes des dispositifs d’interactions en sortie. Les widgets de Crease [Cre01] restent cependant liées aux dispositifs d’interactions en entrée tels que le clavier et la souris. Les langages de description des interactions en sortie peuvent être modélisés indépendamment des dispositifs de sortie en prenant compte les différentes modalités de sortie. Cependant ces modèles peuvent être liés aux dispositifs d’interactions en entrée.

Les langages d’interactions L2, L3, L’2, L’3 et L’4 quant à eux permettent de décrire et d’interpréter les interactions en entrée des utilisateurs des plateformes de départ et d’arrivées. Ces langages d’interactions permettent d’associer à chaque action de l’utilisateur un comportement ayant un sens dans l’UI de l’application. Les actions utilisateurs telles que cliquer, sélectionner, Ctrl+C, poser un objet, etc. dépendent des dispositifs d’interactions tandis que les comportements de l’UI dépendent du type d’UI et de l’interprétation souhaitée par le concepteur.

Le **changement des modalités d'interactions** doit préserver les actions de l'UI de départ et l'adapter aux dispositifs d'interactions de l'UI d'arrivée.

3.1.3.2 Équivalences des modalités d'interactions

La migration de la plateforme desktop vers une table interactive peut être considérée comme un processus de changement de modalités d'interactions. En effet dans le but de réutiliser l'UI d'une application de départ avec les dispositifs d'interactions de la plateforme d'arrivée, il doit être possible d'établir des équivalences entre les dispositifs d'interactions et les langages d'interactions des plateformes de départ et d'arrivée.

Pour décrire les interactions d'une UI de la plateforme de départ à l'aide des dispositifs d'interactions de la plateforme d'arrivée (table interactive), l'une des approches à envisager peut être la mise en correspondance des dispositifs d'interactions en établissant des équivalences entre les différentes modalités d'interactions des plateformes source et cible. Ce qui consiste par exemple à décrire des équivalences d'abord entre les modalités d'interactions en sortie M1 et M'1, puis entre les modalités d'interactions en entrée M2, M3 et M'2, M'3 M'4.

L'équivalence entre M1 et M'1 consiste à comparer les deux dispositifs assez proches, des écrans, qui peuvent afficher des UI graphiques mais aussi les langages L1 et L'1 qui permettent la manipulation de composants graphiques appartenant à des bibliothèques graphiques différentes. L'équivalence entre les modalités d'interactions en entrée M2, M3 et M'2, M'3, M'4 n'est possible que si l'on peut comparer les langages L2, L3, L'2, L'3 et L'4.

3.1.3.3 Constat

Le changement de modalités d'interactions est possible en décrivant des équivalences entre les différentes modalités d'interactions d'une UI. Ce qui peut se faire en se basant sur un modèle indépendant des dispositifs d'interactions.

Un modèle d'interactions abstraites permet de décrire des équivalences entre deux modalités d'interactions en jouant un rôle de langage pivot. Dans notre cadre, un langage pivot permet d'établir des équivalences entre les modalités d'interactions des plateformes source et cible.

3.1.4 Modèle d'interactions abstraites pour la migration des UI

Dans cette section nous présentons deux modèles d'interactions abstraites et nous en déduirons quelques caractéristiques d'un modèle d'interactions abstraites pour la migration.

3.1.4.1 Modèle de Vlist

Vlist *et al.* [vdVNHF11] proposent les primitives d'interactions (*interaction primitives*) qui constituent les plus petits éléments d'interactions identifiables ayant une relation significative avec l'interaction elle-même. Selon cette approche, il est possible de décrire les capacités d'interactions des différents dispositifs d'interactions à l'aide des primitives d'interactions, puis de décrire ensuite, à l'aide d'un mapping sémantique, la relation entre les dispositifs et l'UI. Les différentes primitives d'interactions sont associées à chaque dispositif d'interactions. Elles sont choisies, identifiées et associées aux dispositifs par le concepteur de l'UI. Par exemple les primitives d'interactions "Up" et "Down" correspondent aux touches étiquetées '+' et '-', ces primitives d'interactions ont des types de données et des valeurs, elles sont mappées sur des composants graphiques tels que les contrôles de volume pour permettre l'utilisation des composants avec ces touches. L'objectif de ce mapping sémantique est de faciliter son adaptation en fonction des contextes.

Dans le cadre de la migration, la mise en correspondance des dispositifs d’interactions des plateformes de départ et d’arrivée semble être possible en utilisant une description sémantique des capacités des dispositifs d’interactions d’une plateforme par les primitives d’interactions. Pour cela il est nécessaire de trouver une correspondance à l’ensemble des primitives d’interactions de la plateforme de départ. Par exemple, si “Up” et “Down” existent dans la plateforme de départ, il faut les redéfinir pour un table interactive en inventant le geste à décrire pour les activer. “Up” pourrait être représenté par le touché simple et “Down” par le touché double. Mais on peut aussi imaginer l’association toucher et glisser, etc.

3.1.4.2 Modèle de Gellersen

Gellersen [GH95] propose un modèle d’interactions abstraites qui a pour objectif de décrire les interactions en entrée et en sortie indépendamment des modalités d’interactions. Ce modèle est aussi indépendant d’un domaine ou d’une application. Il est présenté à la figure 3.6 et il permet de décrire une hiérarchie d’interactions en entrée et en sortie. Les interactions en entrée sont raffinées en deux catégories, les interactions d’entrée de données telles que *Editor*, *Valuator* (éditer du texte) et *Option* (électionner un élément d’une liste) qui sont des sous-classes de *Entry* d’une part et les interactions de *Command*⁷ et de *Signal*⁸ d’autre part. Les interactions en sortie sont aussi de deux types : les messages (alertes, confirmation, etc.) et la vue qui permet l’affichage des données et des composants de l’UI.

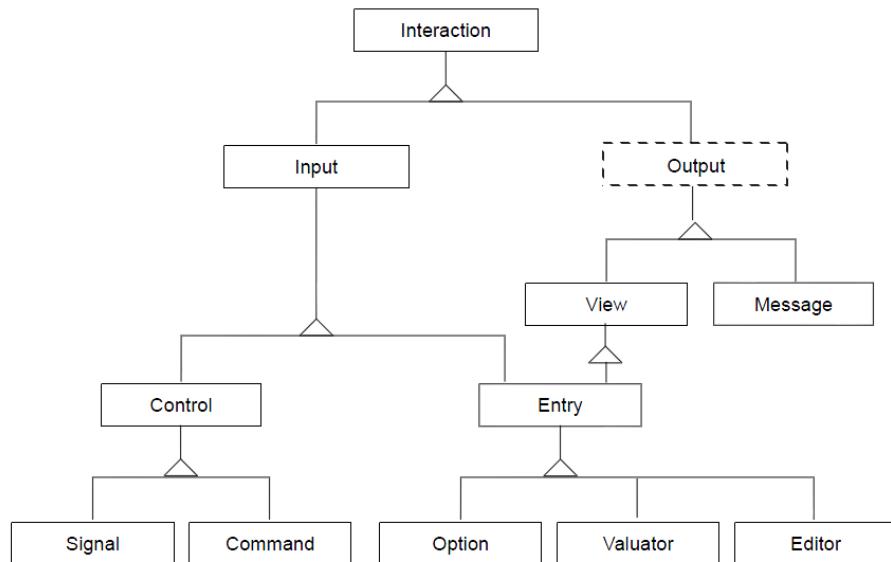


FIGURE 3.6 – Modèle d’interactions abstraites de Gellersen

Dans le cadre du changement de modalité d’interactions entraîné par la migration, ce modèle permet de décrire les interactions des différents composants graphiques de l’UI de manière indépendante des modalités d’interactions. Par exemple une fenêtre d’authentification comportant des champs de

7. Les *Command* sont des interactions en entrée de contrôle avec des paramètres (par exemple “copier texte”, “coller texte”, etc.)

8. Les *Signal* sont des interactions en entrée sans paramètres (exemple valider)

texte et des boutons, les champs de texte seront représentés par les *Editor* qui sont des interactions à la fois en entrée et en sortie et les boutons seront représentés par des *Command* dans un modèle abstrait.

Ce modèle identifie à la fois les interactions en entrée et en sortie de haut niveau indépendamment des modalités d'interactions. Dans le cadre de la migration de l'UI CBA par exemple, les interactions telles que la rotation, le déplacement ou le redimensionnement qui sont liés aux guidelines de la table interactives doivent être interprétés comme des commandes. Cette interprétation n'est pas générique et dépend du concepteur des interactions. En effet le déplacement peut être considéré comme une rotation en modifiant l'angle par exemple. De manière générale, le modèle de Gellersen est un modèle d'interactions abstraites qui nécessite la spécification de certaines interactions indépendantes des modalités d'interactions pendant la migration.

3.1.4.3 Constats

Deux caractéristiques sont essentielles pour les modèles d'interactions abstraites :

- leurs indépendances par rapport à une modalité d'interactions
- et leur capacité à décrire toutes les interactions du langage d'une modalité.

Le changement des modalités d'interactions est une transformation de la dimension dialogue (cf section 2.2.5) d'une UI de départ en établissant des équivalences entre les instruments des plateformes source et cible. L'équivalence des modalités d'interactions permet de garantir l'accessibilité des dialogues de l'UI de départ avec des instruments équivalents (cf section 2.2.2.2). Cette transformation ne prend pas en compte tous les problèmes liés à la transformation des dialogues.

Les problèmes de la sous-dimensions des dialogues multi-utilisateurs et co-localisés par exemple ne peuvent pas être pris en compte par des équivalences simples sans tenir compte du nombre d'utilisateurs et des types d'interactions. Le changement des modalités d'interactions ne prend pas non plus en compte les autres dimensions (Structure et Positionnement, Style). La section 3.2 suivante identifie les principes qui guident la transformation des dimensions.

3.2 Principes de conception des UI pour les tables interactives

La conception ou la transformation des différentes dimensions des UI pour les tables interactives est une activité d'ingénierie qui est guidée par des critères ou des heuristiques afin d'assurer l'utilisabilité du résultat final. Les critères ergonomiques de conception par exemple constituent des propriétés génériques (de haut niveau d'abstraction) applicables à toutes les plateformes. Pour mettre en place un processus de migration réutilisable, il est indispensable d'affiner les critères ergonomiques généralement décrit en langage naturel en règles applicables pour transformer les différentes dimensions d'une UI.

L'affinement des critères abstrait se fait par rapport aux spécificités de l'environnement ciblé. Dans le cadre des tables interactives, nous identifions à la section 3.2.1 les propriétés qui les caractérisent et qui influencent la migration des UI. La section 3.2.2 présente les critères ergonomiques de conception et leur affinement par rapport aux propriétés caractéristiques des tables interactives. La section 3.2.3 présentent l'ensemble des recommandations pour la transformation des différentes dimensions des UI pendant la migration.

3.2.1 Propriétés caractéristiques des tables interactives

Les éléments du modèle d'interactions instrumentales des tables interactives tels que les dispositifs matériels d'interactions en entrée et en sortie, les bibliothèques graphiques ou les modalités

d'interactions nous permettent d'identifier des propriétés, caractéristiques des tables interactives, qui impactent la migration des UI vers des tables interactives.

3.2.1.1 Dispositifs d'interactions en entrée

Ces dispositifs influencent la conception des dialogues. Dans le cadre des tables interactives nous identifions deux propriétés qui correspondent aux moyens d'interactions tangibles et tactiles.

Propriété 1 *Tangibilité des interactions*

Les dispositifs d'interactions en entrée permettent d'associer des objets physiques aux fonctionnalités ou aux composants graphiques. En effet, les objets physiques peuvent aussi être utilisés pour activer des fonctionnalités et pour afficher ou déplacer des objets virtuels d'une UI.

Propriété 2 *Tactilité des interactions*

Les dispositifs d'interactions en entrée des tables interactives permettent de décrire des manipulations directes des UI telles que la sélection, l'édition, le redimensionnement, le déplacement, etc.

3.2.1.2 Dispositifs d'interactions en sortie

Ce sont les surfaces d'affichage des tables interactives, les propriétés caractéristiques liées à la taille et à la disposition des tables interactives qui impactent la conception de la structure et du positionnement des éléments des UI.

Propriété 3 *Taille de la surface d'affichage*

Cette propriété permet d'adapter la taille des composants graphiques par rapport à la taille de l'écran pour faciliter l'utilisation de l'UI.

Propriété 4 *Disposition de la surface d'affichage*

La surface d'affichage peut être disposée de manière horizontale comme une table de travail ou de manière verticale comme un tableau collaboratif. Ces dispositions influencent l'orientation et l'utilisation des composants graphiques d'une UI en particulier si des utilisateurs peuvent être tout autour de la table.

3.2.1.3 Utilisateurs des tables interactives

Le nombre d'utilisateurs influencent la conception des dialogues et de la structure des UI. En effet le nombre permet de savoir si les dialogues de l'UI favorisent la collaboration ou ils sont destinés à un utilisateurs. Le nombre permet aussi de savoir si la structure de l'UI est accessible à plusieurs utilisateurs ou si elle est destinée à un utilisateur.

Propriété 5 <i>Nombre d'utilisateurs</i>
Cette propriété permet la description des dialogues et de la structure des éléments graphiques pour faciliter la collaboration et l'accessibilité de l'UI des tables interactives.
Propriété 6 <i>Répartition des utilisateurs</i>
Cette propriété permet de savoir comment décrire la collaboration entre les utilisateurs autour de la table. La répartition de l'espace de travail de chaque utilisateur peut se faire par une division géographique de la surface ou permettre aux utilisateurs d'accéder à toute la surface.

3.2.1.4 Types d'UI des tables interactives

Les tables interactives présentées à la section 3.1 présentent des UI de types collaboratifs et UI tangibles. Elles sont classées dans le tableau 3.2.

Tables Interactives	Utilisateurs	Instruments d'interactions	Types d'UI
Diamond Touch [DL01]	1 à 4	Écran non capacitif, DiamondSpin	UI Tactile, UI Collaborative (Id Utilisateurs)
metaDesk [UI97]	1 ou plusieurs	Écran, Camera Infrarouge, Objets Tangibles	UI Tangible, UI Collaborative
Microsoft PixelSense [Mic09]	1 ou Plusieurs (<= de points de contact)	Écran capacitif, Tags et Formes Objets, SurfaceSDK	UI Tactile, UI Tangible, UI Collaborative

TABLE 3.2 – Type d'UI pour les tables interactives

3.2.2 Critères ergonomiques de conception des UI

Les propriétés caractéristiques identifiées ci-dessus (*Tangibilité des interactions, Tactilité des interactions, Taille de la surface d'affichage, Disposition de la surface d'affichage, Nombre d'utilisateurs et Répartition des utilisateurs*) nous permettent d'affiner les principaux critères ergonomiques génériques dans le cadre des tables interactives. Dans l'objectif de proposer un ensemble de principes pour la migration des UI, nous présentons dans cette section une grille d'affinement des critères ergonomiques de conception. Nous nous basons sur les huit critères ergonomiques de conception proposés par Scapin [Sca86] :

1. La **compatibilité** est un principe qui repose sur le fait que les transferts d'information seront plus rapides et plus efficaces si le recodage d'informations est réduit [Sca86]. Ce principe préconise d'avoir des dénominations et des commandes compatibles avec le vocabulaire de l'utilisateur.
2. L'**homogénéité** est un principe qui repose sur le fait que la prise de décision, le choix des solutions, le rappel, etc., peuvent se répéter de façon d'autant plus satisfaisante que l'environnement est constant [Sca86]. Ce principe préconise d'avoir des séquences de commandes

identiques pour arriver au même résultat et s'assurer que les labels, prompts et autres catégories d'informations sont toujours localisés au même endroit quelque soit l'écran.

3. La **concision** est un principe qui repose sur l'existence d'une limite en mémoire à court terme de l'opérateur humain. Il convient donc de réduire la charge mnésique de l'utilisateur [Sca86]. L'application de ce principe préconise par exemple de réduire les informations longues et trop nombreuses, de préférer des procédures courtes pour favoriser la tache de mémorisation de l'utilisateur.
4. La **flexibilité** est une exigence liée à l'existence de variations au sein de la population des utilisateurs. Il est préférable que le logiciel comporte différents niveaux et qu'il prenne en considération l'acquisition de l'expérience des utilisateurs [Sca86]. Par exemple l'usage du ctrl-c, ctrl-v et ctrl-x relève du respect de ce principe. Un utilisateur novice utilisera les menus tandis qu'un utilisateur aguerri privilégierra les raccourcis.
5. Le **feedback et le guidage** reposent sur l'influence de la connaissance du résultat sur la qualité de la performance. Les résultats des actions des utilisateurs doivent toujours être répercutés de façon explicite [Sca86]. Il n'est pas par exemple pas possible de déplacer une souris sans feedback.
6. La **charge de travail** est un principe qui préconise que la prise en compte de la charge informationnelle de l'utilisateur est essentielle dans la mesure où la probabilité d'erreur humaine augmente dans les situations à charge élevée [Sca86]. Par exemple il convient de minimiser le nombre d'opérations à effectuer par l'utilisateur ainsi que les temps de traitement.
7. Le **contrôle explicite** signifie que même si c'est le logiciel qui a le contrôle, l'interface doit apparaître comme étant sous le contrôle de l'utilisateur et surtout exécuter, en règle générale, des opérations uniquement à la suite d'actions explicites [Sca86].
8. La **gestion des erreurs** est un principe qui consiste à fournir aux utilisateurs des moyens pour corriger leurs erreurs [Sca86]. La clarté des messages est un élément fondamental.

Pour affiner ces huit critères ergonomiques à l'aide des propriétés caractéristiques des tables interactives, nous utilisons le tableau 3.3 ci-dessous. Ce tableau nous permet de savoir comment une propriété caractéristique des tables influence un critère ergonomique. Les influences possibles sont :

- F : une propriété **favorise** un critère ergonomique de conception,
- RA : une propriété **risque d'altérer** un critère ergonomique de conception,
- C : une propriété **conserve** un critère ergonomique de conception,
- N : une propriété est **neutre** par rapport à un critère ergonomique de conception.

3.2.2.1 Exemple de lecture du tableau d'affinement

La *tangibilité* favorise la *flexibilité* dans le cadre de la migration des UI vers les tables interactives car elle ajoute une nouvelle modalité d'interactions par rapport à l'UI de départ.

Le *nombre d'utilisateurs* risque d'altérer le *feedback* et le *guidage* de l'UI cible après la migration si les affichages des feedbacks sont bloquants pour les autres utilisateurs dans un contexte multi-utilisateurs.

La *tactilité* permet de conserver la *flexibilité* car les interactions tactiles et la manipulation directe (souris) des UI de la source sont équivalentes.

La *taille de la surface d'affichage* est une propriété neutre par rapport à la *compatibilité* car elle n'implique pas de modifications des éléments du vocabulaire⁹ de l'utilisateur final de l'UI.

9. Commandes et les dénominations des éléments de l'UI

	Tangibilité	Tactilité	Taille	Disposition	Nombre	Répartition
Compatibilité	F	RA	N	N	N	N
Homogénéité	RA	RA	RA	N	RA	RA
Concision	F	F	RA	N	RA	F
Flexibilité	F	C	F	F	RA	F
Feedback & Guidage	N	C	F	C	RA	RA
Charge de Travail	F	F	RA	N	N	N
Contrôle Explicite	F	F	F	RA	RA	F
Gestion des erreurs	C	C	C	N	RA	RA

TABLE 3.3 – Liens entre critères ergonomiques de Scapin et les propriétés caractéristiques des tables interactives.

3.2.2.2 Constats

La projection des critères ergonomiques de conception sur les propriétés caractéristiques des tables interactives telles que présenté dans le tableau 3.3 a pour objectif d'émettre des recommandations suivant les critères ergonomiques de conception avec pour objectif de garantir l'utilisabilité de l'UI, les cohérences des dialogues et du style.

Dans le cas où une propriété **favorise** (F) un critère ergonomique, les recommandations doivent être décrites pour transformer l'UI source et elles sont respectées sans conditions. Par exemple pour savoir comment les interactions tangibles favorisent la réduction de la charge de travail d'une UI source, il est indispensable de décrire comment afficher et cacher des éléments des UI à l'aide d'objets tangibles.

Dans le cas où une propriété **risque d'altérer** (RA) un critère ergonomique de conception, il est indispensable que les recommandations associées identifient les contraintes pour éviter les risques. Par exemple pour savoir comment la propriété *Nombre d'utilisateurs* peut altérer le *Feedback* et le *Guidage*, il est indispensable d'identifier les éléments bloquants de l'UI source et proposer une manière de les migrer.

Dans les cas où une propriété **conserve** (C) un critère ergonomique de conception alors les équivalences entre les éléments de l'UI source et cible suffisent pour assurer la conformité à ces critères ergonomiques de conception. Il n'est pas indispensable de décrire des recommandations. Par exemple les équivalences entre les modalités d'interactions permettent de préserver la flexibilité d'une UI source et les équivalences entre les éléments graphiques permettent de préserver la gestion des erreurs des UI sources.

Dans le cas où une propriété est **neutre** (N) par rapport à un critère de conception alors aucune recommandation n'est décrite. Par exemple la taille de la surface d'affichage n'affecte pas la compatibilité car les éléments du vocabulaire ne sont pas modifiés par cette propriété.

Les propriétés qui favorisent ou qui risquent d'altérer des critères ergonomiques de conception nous permettent d'identifier des recommandations de haut niveau pour les différentes dimensions des UI. Nous proposons dans la section suivante de définir et d'identifier les recommandations pour la transformation des différentes dimensions des UI à migrer.

3.2.3 Recommandations pour la migration des UI vers les tables interactives

La section 3.1.3 met en évidence les différences de modalités d’interactions entre un desktop et une table interactive. Ces différences impactent aussi la conception des UI pour ces deux plateformes. Besacier et *al.* montrent que la réutilisation des applications desktop sur les tables interactives en adaptant les éléments de l’UI aux métaphores du papier par exemple facilite l’utilisation des UI [BRNB07].

Par ailleurs, une réutilisation d’une application desktop sur des tables interactives sans prise en compte de ces spécificités pose deux problématiques majeures : la transformation de l’UI de départ en UI collaborative d’une part et la transformation d’une GUI en TUI d’autre part. Ces deux caractéristiques font partie de l’ensemble des **guidelines** qui guident la conception des UI pour les tables interactives.

“A Guideline is a source of inspiration or a set of standards that gives recommendations, tips and directives” [Que01]

Les guidelines pour la conception des UI constituent un ensemble de recommandations pour les concepteurs des UI qui indiquent comment décrire les aspects tels que les interactions instrumentales, le layout (ou le placement des éléments graphiques) et aussi les styles de présentations (couleurs, polices, tailles, etc.).

Les guidelines de haut niveau doivent être traduites en règles formelles utilisables pendant la migration [Van97]. La synthèse présentée par le tableau 3.3 d’affinement des critères ergonomiques de conception dans le cadre des tables interactives nous permet d’avoir un lien entre les critères ergonomiques de conception et les guidelines.

Dans cette section nous caractérisons les guidelines pour la migration des UI vers les tables interactives en deux catégories : les guidelines pour les UI tangibles et les guidelines pour les UI collaboratives et co-localisées.

3.2.3.1 Corpus des recommandations de conception des UI pour Microsoft PixelSense

Les recommandations de conception des UI pour une table interactive Microsoft PixelSense sont décrites dans le document *User Experience Design Guideline* [Mic11]. Elles sont le fruit des expériences des développeurs des premières applications sur tables interactives. L’objectif de ces guidelines est de faciliter la conception des interfaces utilisateurs pour qu’elles soient plus naturelles [Res13] et plus intuitives. Ces guidelines couvrent plusieurs aspects du processus de conception des UI tels que la conception des interactions entre les UI et les utilisateurs finaux, les guides de styles pour une cohérence visuelle, les guides d’utilisation des textes, etc.

3.2.3.2 Guidelines pour UI collaborative

Cette catégorie regroupe les recommandations pour la conception d’une UI collaborative et co-localisée pour une table interactive. Les guidelines liées à la propriété 5 du nombre d’utilisateurs risquent d’altérer l’homogénéité, la concision, la flexibilité, le feedback, le contrôle explicite et la gestion des erreurs (cf tableau 3.3). Par ailleurs les guidelines liées à la propriété 6 de la répartition des utilisateurs risquent d’altérer l’homogénéité, le feedback et la gestion des erreurs et elles favorisent aussi le contrôle explicite (cf tableau 3.3). Les guidelines liées à la propriété 3 de la taille de la surface d’affichage risquent d’altérer l’homogénéité, la concision et la charge de travail, elles favorisent la flexibilité et le feedback (cf tableau 3.3). Enfin la propriété 4 de la disposition de la surface d’affichage risque d’altérer le contrôle explicite et elle favorise la flexibilité (cf tableau 3.3).

Les guidelines, pour favoriser la collaboration des UI migrées, peuvent être déclinées suivant les trois dimensions d’une UI :

- Les guidelines issues de la dimension des *Dialogues* (D1) permettent de décrire les **dialogues entre les utilisateurs**.
- Les guidelines issues de la dimension de la *Structure et du Positionnement* (D2) permettent d'avoir une UI avec un **espace de travail** qui favorise le **partage**.
- Les guidelines issues de la dimension du *Style* (D3) favorisent aussi le partage dans un **espace de travail** grâce à des contraintes sur la taille des éléments graphiques.

Guideline 1 *Couplage des dialogues d'UI avec les utilisateurs*

Cette guideline préconise d'adapter les composants graphiques d'une UI au nombre d'utilisateurs en éliminant toutes les **activités bloquantes** pour les autres utilisateurs de l'UI (boîtes de dialogues ou fenêtres bloquantes).

Guideline 2 *Partage de l'espace de travail*

Cette guideline préconise de prendre en compte le nombre d'utilisateurs de l'UI pendant la migration. Le nombre d'utilisateurs est un facteur important pour le choix du comportement, de la structure, du positionnement et de la taille des éléments d'une UI sur les tables interactives. Pour permettre le partage et la collaboration, l'espace de travail doit :

- avoir des éléments avec des **tailles** variables par les utilisateurs,
- avoir une **structure** (types de données) et un **regroupement** d'éléments des UI adaptées,
- avoir des éléments avec des **comportements** (rotation, déplacement) pour favoriser le partage. Dans ce cas la propriété 360° favorise l'**accessibilité** des éléments graphiques.

Illustration La figure 3.7 illustre un cas d'utilisation des composants graphiques utilisables à 360°. En effet, la figure de droite (barrée d'un trait oblique rouge) montre le cas d'un composant sans la propriété 360°, ce composant oblige des utilisateurs à avoir une position fixe autour de la table. Cette situation n'est pas recommandée par la guideline d'accessibilité des composants graphiques.

La figure de gauche montre des composants graphiques qui implémentent la propriété 360°.

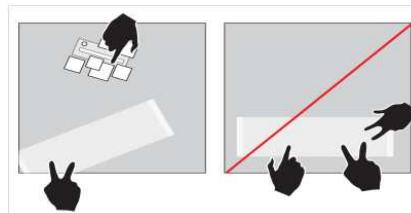


FIGURE 3.7 – Illustration de la propriété 360° des composants graphiques sur une table interactive

Exemples En considérant que l'UI de l'application CBA (cf. figure 2.2) est migrée vers une table Surface pour quatre dessinateurs de BD qui peuvent librement s'installer autour de la table. Les ressources (images, bulles, etc.) utilisées pour la conception des BD doivent être accessibles par les quatre

utilisateurs. La guideline 2 préconisant le partage de l'espace de travail et la guideline 2 préconisant l'utilisation des objets 360° permettent de décrire une UI sans layout avec des groupes d'éléments utilisables à 360°. La guideline 1 par exemple permettra de supprimer les différents composants graphiques bloquants tels que les boîtes de dialogues de l'UI de l'application CBA.

Synthèse Les guidelines qui favorisent la collaboration pendant la migration peuvent être représentées par le diagramme de la figure 3.8. Les rectangles représentent les guidelines à différents niveaux d'abstraction. Les liens entre les rectangles représentent les dimensions et les sous-dimensions des UI.

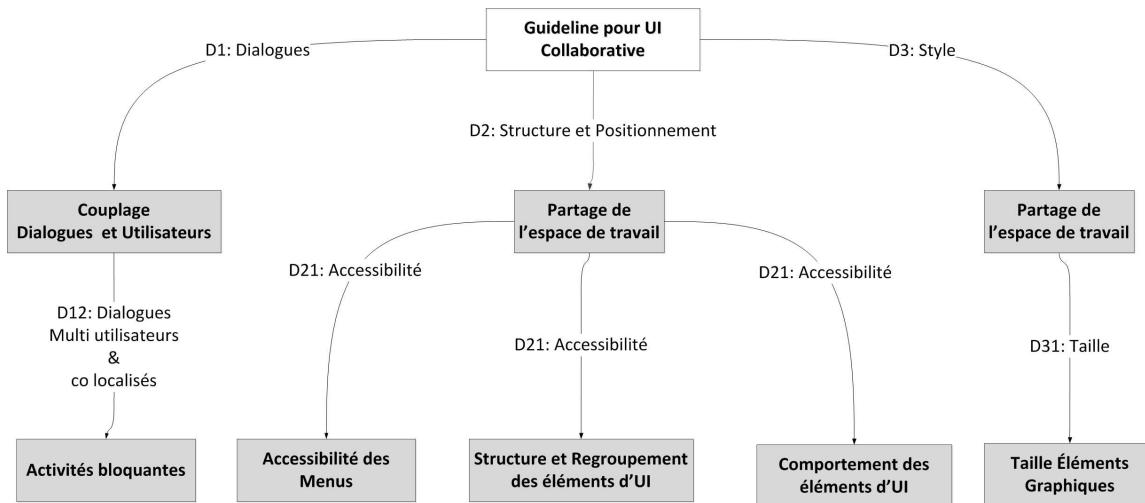


FIGURE 3.8 – Représentation synthétique des guidelines selon les trois dimensions des UI

3.2.3.3 Guidelines pour TUI

Cette catégorie regroupe les recommandations et les contraintes qui permettent de décrire le comportement des éléments de l'UI qui sont des objets virtuels d'une part et l'association entre ces objets virtuels et les objets tangibles d'autre part. L'association peut se faire par des tags qui permettent de marquer les objets physiques ou en se basant sur la forme des objets physiques. Les guidelines de cette catégorie sont inspirées par la propriété de la tangibilité des interactions. Cette propriété risque d'altérer l'homogénéité et elle favorise la concision, la flexibilité, la charge de travail et le contrôle explicite (cf tableau 3.3).

Les guidelines pour TUI se déclinent suivant la dimension des *Dialogues* (D1) et la dimensions de la *Structure et du Positionnement* (D2) des éléments graphiques. Pour ajouter les interactions tangibles sur les UI sources, nous préconisons dans cette section deux guidelines :

- La guideline pour associer un objet tangible à une fonctionnalité est issue de la sous-dimension D11 des *Interactions Tangibles*.
- La guideline pour associer un objet tangible à un élément (ou groupe d'éléments) virtuel est issue de la sous-dimension D11 des *Interactions Tangibles* et la sous-dimension D21 liée à l'*Accessibilité* des éléments graphiques. Les objets tangibles facilitent l'accès aux composants graphiques.

Guideline 3 *Objets tangibles des objets virtuels*

Cette guideline préconise d'associer le comportement des éléments à un objet tangible. Les éléments (ou objets virtuels) de l'UI à migrer doivent être associés à des objets physiques dans le but d'afficher des menus, des formulaires ou des fenêtres et aussi dans le but d'activer ou d'utiliser des fonctionnalités.

Guideline 4 *Objets tangibles et fonctionnalités*

Les objets tangibles peuvent servir à activer une fonctionnalité. Cette guideline préconise quelles fonctionnalités associer à un objet tangible.

- Les fonctionnalités à associer doivent être sélectionnables et activables à partir de l'UI de départ (les éléments d'un menu par exemple).
- Les paramètres des fonctionnalités doivent être connus ou finis (un formulaire ou une liste par exemple).

Exemple En considérant l'UI de l'application CBA à la figure 2.2, le menu principal et le formulaire *Ressources* peuvent être associés à un objet physique pour les afficher facilement sur l'écran. Les règles formelles issues de la guideline 3 permettront d'identifier et de transformer les éléments concrets de l'UI. Les tags permettent par exemple d'utiliser deux objets de la même forme avec des couleurs différentes et marqués des deux différents tags pour afficher un menu ou un formulaire.

Synthèse Les guidelines pour l'utilisation des objets tangibles peuvent être représentées par le diagramme de la figure 3.9. Les rectangles représentent les guidelines à différents niveaux d'abstraction. Les liens entre les rectangles représentent les dimensions et les sous-dimensions des UI.

3.2.3.4 Autres guidelines pour les UI sur tables interactives

Les deux catégories de guidelines présentées ci-dessus qui permettent de décrire des UI tangibles et des UI collaboratives (cf. section 3.2.3.2) ne constituent pas le corpus des guidelines applicables pendant la migration d'une UI vers une table interactive. En effet les aspects de l'UI liés au style des composants graphiques, des textes de l'UI et les aspects liés aux interactions tactiles ne sont pas pris en compte par ces deux catégories de guidelines. Cette troisième catégorie regroupe les guidelines de mise en œuvre des interactions tactiles et des styles de l'UI.

Guidelines pour les interactions tactiles Les tables interactives permettent en général de décrire des interactions tactiles. L'ensemble des actions tactiles de l'utilisateur est interprété par le dispositif d'interactions en entrée. Dans le cadre des UI tactiles, il existe des actions tactiles standards pour des interactions de redimensionnement, de déplacement ou de rotation. Par exemple les guidelines des interactions tactiles pour une table Surface [Mic11] identifient l'ensemble des gestes recommandés aux concepteurs pour la sélection, l'activation, le déplacement, la rotation, le zoom, etc. Cette catégorie de guidelines est liée à la propriété 2 de tactilité des interactions d'une table interactive.

Dans le cadre de la migration de l'UI de l'application CBA vers les tables interactives, il est indispensable de pouvoir décrire des correspondances entre les interactions tactiles et les interactions du clavier et de la souris de l'UI de départ.

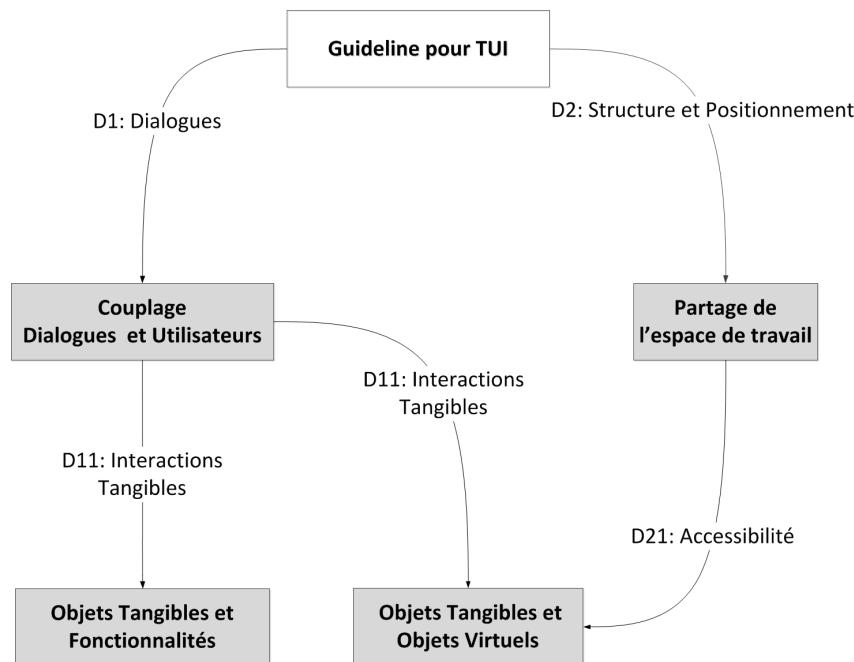


FIGURE 3.9 – Représentation synthétique des guidelines suivant deux dimensions

Formulaire de l'application de départ	Formulaire migré en prenant en compte les guidelines de styles

TABLE 3.4 – Migration de l'aspect visuel d'un formulaire

Guidelines pour le style Cette sous-catégorie regroupe l'ensemble des recommandations pour la personnalisation des aspects visuels d'une UI pour une table interactive. Ces guidelines peuvent être utilisées dans le cadre d'une migration faisant intervenir les concepteurs comme des exemples pour les inspirer du choix de la forme, des couleurs, des icônes et aussi de la disposition des textes d'une UI. Dans le cas du scénario de migration, le formulaire *Ressources* par exemple peut être migré comme l'indique le tableau 3.4.

3.3 Synthèse

Dans ce chapitre, nous avons étudié le modèle d’interactions des tables interactives. Cette étude nous montre que la migration des UI vers des tables interactives provoque dans la majorité des cas un changement de modalité d’interactions. Ce changement peut être effectué en décrivant un modèle d’interactions abstraites qui est indépendant d’une modalité d’interactions. Ce modèle doit aussi être capable de décrire toutes les interactions du langage liés aux différentes modalités. L’objectif d’un modèle d’interactions abstraites est d’établir des équivalences entre les instruments d’interactions des plateformes source et cible.

Nous avons aussi identifié dans ce chapitre différentes règles de bons usage (‘guidelines’) qui vont guider la transformation des UI desktop en une UI tangible et collaborative. Les guidelines sont déduites à partir des propriétés caractéristiques des tables interactives et des critères ergonomiques de conception de haut niveau. Les guidelines sont constituées des recommandations et/ou des contraintes pour transformer les différentes dimensions des UI de la source. Dans le chapitre, nous avons remarqué que les guidelines qui favorisent la collaboration permettent de transformer les trois dimensions (Dialogues, Structure et Positionnement, Style). Par ailleurs, les guidelines pour les TUI concernent la dimension des Dialogues et la dimension de la Structure et du Positionnement.

Les processus de transformation des différentes dimensions sont décrits par des approches de migration des UI. Nous présentons dans le chapitre suivant les différentes approches de migrations dans le but d’identifier les mécanismes de migration des UI adéquats pour nos objectifs.

Approches de migration des UI

Sommaire

4.1	Introduction	45
4.2	Approches spécifiques de migration des UI	46
4.2.1	Migration manuelle	46
4.2.2	Portage des applications existantes sur des tables interactives	48
4.3	Approches de migration basées sur les modèles de l'UI	51
4.3.1	Approches de migration automatiques des UI	52
4.3.2	Approche semi automatique de migration des UI	57
4.4	Synthèse et objectifs	62
4.4.1	Synthèse	62
4.4.2	Objectifs	64

4.1 Introduction

La migration des applications est une activité de déplacement et d’adaptation d’un logiciel d’un environnement source vers un environnement cible. Elle est plus globale que le portage d’applications [Moo95] car elle ne se limite pas qu’au changement de langages de programmation ou au changement des systèmes d’exploitation. La migration englobe les problématiques de ré engineering, de reverse engineering, de forward engineering et de portage d’applications. McClure *et al.* [McC92] définissent le ré engineering comme une amélioration d’un système existant pour le rendre conforme aux standards. Le reverse engineering consiste à analyser un système existant pour décrire la représentation d’origine de manière plus abstraite [McC92]. Le forward engineering est une concrétisation de la représentation abstraite d’un système dans une implémentation. Une approche de migration des UI vers une table interactive implique un ré engineering de l’UI source en impliquant d’abord une phase de reverse engineering de l’UI source suivit d’une phase de forward engineering.

Dans notre contexte, nous avons fait le choix d'imposer aux applications à migrer une décomposition fonctionnelle minimale qui permet de distinguer une UI et un NF. Cette décomposition facilite la migration de l’application en permettant la réutilisation du NF, si les plateformes sont compatibles. Nos travaux se concentrent sur la migration de l’UI pour laquelle les différentes approches possibles se basent sur un ensemble de mécanismes qui visent dans un premier temps à construire des **équivalences** entre les instruments d’interactions des plateformes source et cible. Il est ensuite possible **d’adapter** la structure, le positionnement et le style de l’UI source par rapport à celle cible. Dans un dernier temps, il s’agit de **respecter les guidelines** de la cible. Cet enchainement d’actions permettant la migration des UI peut être manuel, automatisé ou semi automatique.

Nous étudions dans ce chapitre différentes approches de migration des UI dans l’objectif d’évaluer :

- La **flexibilité** et l'**automaticité** de chaque approche afin d’évaluer le degré d’intervention du concepteur pendant le processus de migration.

- La prise en compte des guidelines pour assurer le respect des **critères ergonomiques** de conception des UI.
- La **réutilisabilité** des mécanismes d'équivalences et d'adaptation dans le but de réduire les coûts de la migration.

La section 4.2 présente par conséquent les approches de migration spécifiques à des applications ou à des boîtes à outils. La section 4.3 présente les approches de migration des UI basées sur des modèles d'UI. La section 4.3 fait une synthèse des différentes approches de migration des UI et présente les caractéristiques de la solution que nous avons adoptée.

4.2 Approches spécifiques de migration des UI

Dans cette section nous étudions les processus de migrations spécifiques à une application ou à une boîte à outils. Nous présentons d'abord un processus qui permet de migrer manuellement une application vers une table interactive. Ce processus identifie des guidelines pour les UI collaboratives et s'appuie sur une démarche structurée. Ensuite nous présentons une famille de mécanismes de migration des UI des applications existantes sur les tables interactives sans re-conception de l'UI de départ.

4.2.1 Migration de l'application AgilePlanner vers une table interactive

AgilePlanner est une application de planification et de gestion de projet suivant la méthode Agile ; sur une table interactive, elle est utilisée pour faire du brainstorming dans le cadre de la gestion de plannings de projets. Nous étudions cette approche dans le cadre de la migration de cette application vers une table interactive [WGM08] afin d'identifier les différents mécanismes manuels mis en œuvre. Le processus mis en place repose sur quatre phases :

- La première phase consiste à analyser l'UI de l'application à migrer. Elle permet d'identifier les différentes zones : menu, légendes, zones d'interaction, espace de travail, etc.
- La deuxième phase consiste à évaluer l'UI de l'application à migrer. Le but de cette évaluation est d'identifier les différences majeures entre la plateforme source et cible pour en déduire des recommandations qui seront des guidelines pour le concepteur. Il en est ressorti les 7 guidelines suivantes :
 - Avoir des composants de l'UI de l'application déplaçables et utilisables en 360°.
 - Utiliser la reconnaissance gestuelle pour les interactions utilisateurs et éviter les menus traditionnels.
 - Utiliser l'écriture à main levée au lieu du clavier pour la saisie des textes.
 - Prendre en compte les interactions concurrentes pendant la conception de l'UI.
 - Pouvoir utiliser des composants graphiques de taille assez grande pour faciliter les interactions tactiles.
 - Éviter l'utilisation des boîtes de dialogues pop-up.
 - Permettre à l'UI de l'application de s'adapter aux différentes tailles des tables interactives.
 Ces guidelines sont conformes aux guidelines pour les UI collaboratives et aux guidelines pour les interactions tactiles que nous avons présentées au chapitre précédent (cf. section 3.2.3).
- La troisième phase consiste à appliquer les guidelines de la phase précédente pour concevoir la nouvelle UI de l'application AgilePlanner. Certaines des guidelines telles que la rotation et le déplacement des composants graphiques, l'écriture à main levée, les reconnaissances gestuelle et vocale peuvent être fournies par l'environnement logiciel de la table interactive. Évidemment, cela dépend de la table et des bibliothèques graphiques disponibles.

- La dernière phase du processus consiste à évaluer l’UI produite en demandant à des utilisateurs finaux d’évaluer l’utilisabilité de chaque fonctionnalité de l’application. Les différentes remarques émises permettent de modifier l’UI migrée. Une fois ces différentes modifications effectuées, l’UI a été de nouveau évaluée à nouveau pour mesurer la satisfaction des testeurs. Par exemple, les résultats de cette réévaluation ont montré que l’utilisabilité de l’écriture à main levée sur la table interactive peut être améliorée.

La migration manuelle d’une UI est un processus qui comporte plusieurs phases. Dans cette section nous avons présenté une approche qui nous permet d’identifier quatre phases importantes :

1. Analyser l’UI de départ pour identifier la structure des éléments qui la compose et ses fonctionnalités.
2. Identifier les guidelines qui permettront la migration : cette identification se fait en se basant sur les différences entre la plateforme de départ et celle d’arrivée.
3. Appliquer les guidelines pour migrer l’UI de départ.
4. Évaluer et améliorer le résultat obtenu.

4.2.1.1 Réutilisation du processus pour d’autres applications

Les étapes du processus de migration de l’application AgilePlanner vers une table interactive et la mise en œuvre des guidelines identifiées constituent des “savoir faire” **réutilisables** pour d’autres applications.

Cependant en ce qui concerne l’application AgilePlanner, les mécanismes d’équivalences entre les instruments d’interactions et l’adaptation du layout de la structure ne peuvent pas être réutilisés pour d’autres applications car ils sont spécifiques à cette application. En effet, l’analyse et l’identification des différentes zones de l’UI ne s’appuient pas sur des modèles indépendants de la plateforme.

La mise en œuvre des guidelines identifiées par ce processus est basée sur l’intuition. Pour compléter l’étude, nous avons considéré l’UI de deux autres applications que nous souhaiterions faire migrer. L’UI de l’application CBA et celle d’une application agenda présentée par la figure 4.1. Nous souhaitons effectuer cette migration en appliquant les guidelines identifiées à la section 4.2.1. La migration de l’application agenda ne pose pas trop de problèmes et par exemple, les divers composants graphiques représentant la barre de menu, la barre de recherche, la liste des catégories et le tableau peuvent tous être déplaçables sur la table. On peut aussi considérer la liste des catégories et le tableau de contacts comme tout indissociable. En ce qui concerne l’application CBA, le choix des groupes utilisables à 360° peut se faire comme le propose la figure 2.3. Nous remarquons que l’utilisation des guidelines pour deux applications différentes repose sur l’**interprétation** faite par les personnes en charge de la migration et laisse une certaine latitude car ces guidelines précisent formellement assez peu de choses. Par exemple, plusieurs possibilités sont offertes pour regrouper les composants graphiques à déplacer.

Le processus de migration décrit par cette approche est très **flexible** pour deux raisons, d’abord il est essentiellement manuel et ensuite le processus intègre une phase d’évaluation et d’amélioration du résultat. L’intervention humaine est omniprésente et permet d’améliorer le résultat en continu.

Par contre, le fait que l’approche se base uniquement sur des mécanismes manuels rend plus lourd le travail des personnes en charge de la migration. Les mécanismes d’équivalence des instruments d’interactions doivent être décrit manuellement pour chaque application à migrer. Les mécanismes de prise en compte des guidelines sont basés uniquement sur les connaissances des développeurs. Cette approche n’offre que peu de mécanismes permettant de réduire la charge de travail des développeurs et par conséquent le coût de migration reste presque équivalent à une nouvelle conception de l’UI.

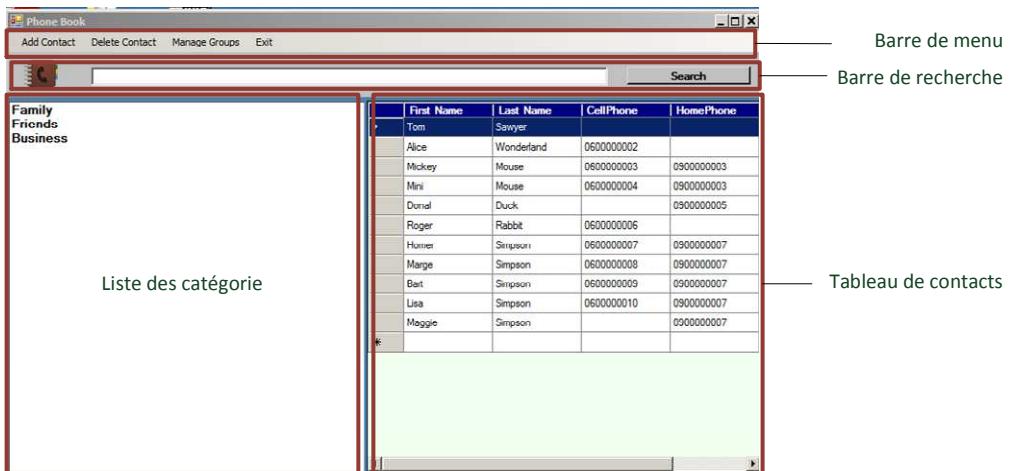


FIGURE 4.1 – Application de consultation des contacts

4.2.1.2 Synthèse des approches manuelles

La figure 4.2 met en évidence que la mise en équivalence des composants graphiques ou des dispositifs d’interactions des plateformes source et cible est manuelle tout comme les mécanismes de prise en compte des guidelines. La qualité de l’UI générée est uniquement basée sur les connaissances et les compétences des personnes en charge de la migration. Ces mécanismes sont donc difficilement **réutilisables** car non formalisés. Par contre, cette approche est relativement **flexible** et permet d’avoir une UI conforme aux **critères ergonomiques** pour peu que les concepteurs utilisent pleinement les différentes possibilités qui leur sont offertes et tirent partie des différentes évaluations.

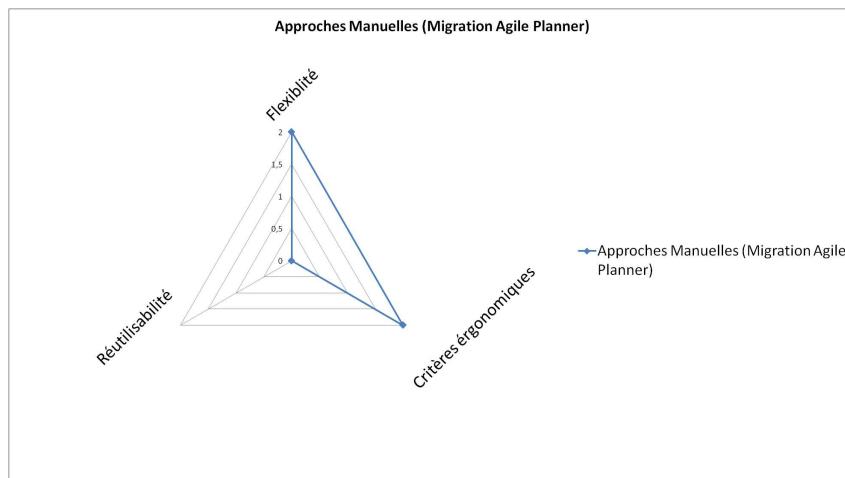


FIGURE 4.2 – Synthèse de la migration manuelle des UI

4.2.2 Portage des applications existantes sur des tables interactives

Le portage d’une UI vers une nouvelle plateforme consiste à adapter les instruments d’interactions de la plateforme cible pour l’UI source ainsi que le positionnement ou le style de l’UI source sans

adapter la structure. Dans cette section nous présentons une approche de portage des UI proposée par Besacier [Bes10] qui a pour but de réutiliser des applications développées pour desktop sur une table interactive sans reconception de l'UI. L'objectif premier d'adapter les différents moyens d'interactions (clavier et souris) des desktops pour les rendre opérants sur une table interactive DiamondTouch selon différentes technologies.

Pour Besacier [Bes10], six technologies permettent la réutilisation souhaitée :

La capture d'écran est une technologie qui enregistre une copie exacte de l'écran de l'UI de départ à intervalle régulier. La copie est stockée dans une zone mémoire où elle peut être modifiée pour être adaptée à l'environnement de la table interactive. Cette approche se situe dans un contexte de portage des applications existantes. Elle ne permet pas de porter des interactions en entrée car l'UI portée est constituée d'images capturées et ces images ne contiennent pas de métadonnées.

La carte graphique virtuelle est une amélioration de la technologie précédente basée sur la capture d'écran. L'approche utilise le serveur Metisse [RCPSC05] qui stocke les fenêtres de l'application de départ sous forme d'images. Le serveur Metisse joue un rôle de carte graphique virtuelle car il redessine les fenêtres sur l'écran de la table interactive à travers un compositeur. Il redirige aussi les événements en provenance de la souris ou du clavier vers l'application. Cette technologie permet de porter les fenêtres des applications desktops en offrant la possibilité aux utilisateurs de les manipuler facilement. Cependant les composants graphiques des fenêtres portées ne sont toujours pas interactifs car ils restent des images de l'UI de départ.

L'émulation du clavier et de la souris est une technologie qui améliore les deux technologies présentées ci-dessus. Elle permet de porter aussi les interactions en entrée. La technique permet de porter plusieurs applications à la fois sur une table interactive et pour chaque application est associée un utilisateur avec son clavier et sa souris virtuels [HCT06]. L'interaction d'un utilisateur verrouille la table aux utilisateurs de la même application. Cette technologie permet d'associer à chaque utilisateur son application mais ne permet pas de rendre collaborative une application. Cette technologie n'est pas spécifique à une application comme celles basées sur la capture d'écran ; elle est réutilisable pour toutes les applications desktops (utilisant un clavier et une souris) que l'on souhaiterait migrer sur une table interactive. Par contre elle reste spécifique aux dispositifs d'interactions (clavier et souris).

Le langage de script est une technologie qui permet de porter les UI d'applications existantes vers une table interactive en résolvant les limites liées aux multi-touches et aux multi-utilisateurs de la technologie précédente. Le langage de script est spécifique à une application et consiste à écrire dans un scénario un ensemble de scripts qui font chacun appel à une fonctionnalité de l'application à porter. A chaque événement généré par la table interactive est associé un script. Par exemple pour l'affichage d'un fichier sur la table est réalisé par l'association d'un script d'ouverture et de lecture de fichiers à l'événement déclenché par la pose d'une feuille de papier, par exemple munie d'un tag, sur la table interactive. Cette technique est difficilement **réutilisable** d'une application à l'autre car chaque script reste très dépendant de l'application pour laquelle il a été conçu. Cette technologie est utilisée par Besacier pour afficher les données d'un tableau Excel sur une table interactive DiamondTouch tout en prenant compte les guidelines liées à la structure d'une UI à travers la métaphore du papier. Cette métaphore stipule que les éléments graphiques d'une UI peuvent être utilisés comme une feuille de papier grâce aux interactions de déplacement, de rotation, de redimensionnement, de changement d'échelle et de pliage [BRNB07]. La technologie du langage de script nécessite, dans le cas d'un tableau Excel, l'implémentation d'un contrôleur et d'une vue pour afficher les données d'un fichier

Excel (ou du modèle). La prise en compte des guidelines dépend fortement des connaissances et de l'intuition du concepteur en charge du portage. Par ailleurs le coût de sa mise en œuvre reste élevé par rapport à sa **réutilisabilité**. En effet pour une UI à porter il faut décrire des scénarios d'utilisation afin de construire les scripts.

Les API d'accessibilité numérique permettent d'obtenir des informations sémantiques à propos des UI graphiques en cours d'exécution. Elles sont par exemple utilisées pour offrir des interfaces alternatives à des groupes d'utilisateurs spécifiques (déficients visuels). Ces informations constituent des instances de modèles de structure d'une UI donnée comprenant les composants graphiques, les données, les positions, etc. Ce modèle permet de décrire une technique de migration qui segmente les images obtenues par capture d'écran de l'UI source. Les parties de l'UI de départ peuvent donc être dupliquées ou juxtaposées en se basant sur le modèle de structure et les images segmentées. Le modèle de structure et les images segmentées sont les éléments de l'UI source qui permettent de générer l'UI cible. Comparée aux technologies précédentes, cette technologie est moins **réutilisable** que la capture d'écran, la carte graphique virtuelle et l'émulation des dispositifs d'interactions. Elle se base sur des technologies spécifiques à l'environnement Windows et toutes les applications ne respectent pas forcément cette approche. Le coût de sa mise en œuvre reste élevé par rapport à sa **réutilisabilité**. En effet, pour une UI à porter il faut écrire les algorithmes de segmentation de chaque image capturée et les mécanismes d'analyse des données sémantiques de l'UI.

La réécriture d'une boîte à outils d'interface homme machine consiste à utiliser la boîte à outils de la table interactive à la place de celle du desktop. Le remplacement se fait en interceptant tous les appels de fonction de l'application vers les objets de la boîte à outils d'IHM de départ et en les redirigeant vers les éléments de la boîte à outils de la table interactive. On construit donc, à la manière d'une machine virtuelle un émulateur de la boîte à outils source à l'aide de la boîte à outils cible. L'avantage de cette approche est d'utiliser des composants graphiques respectant les guidelines de la table interactive sans une nouvelle conception de l'application de départ puisque celle-ci est inchangée. Cette solution peut être implémentée en utilisant les wrappers [TKR10]. La mise place des mécanismes d'équivalences entre les instruments d'interactions de cette technologie nécessite une bonne connaissance des bibliothèques graphiques et des dispositifs d'interactions des plateformes source et cible. L'utilisation de cette technologie nécessite un temps d'apprentissage pour des personnes non expertes des instruments d'interactions des plateformes source et cible. Cette approche ne permet pas le respect de l'ensemble des guidelines car la structure et le layout de l'UI par exemple ne sont pas modifiés pendant le portage. Elle permet de choisir les composants graphiques qui ont des interactions ou des styles conformes à la table interactive.

4.2.2.1 Synthèse des approches de portage des UI

Le tableau 4.1 présente une synthèse de ces différentes approches en prenant en compte les critères d'équivalences entre les éléments de la plateforme, les modèles utilisés et la prise en compte des guidelines. Nous remarquons que les équivalences entre les dispositifs d'interactions sont toujours décrites manuellement pendant la mise en œuvre de la solution. Ces approches utilisent une modélisation des interactions et seulement deux approches modélisent la structure de l'UI. La prise en compte des guidelines dépend de l'utilisation ou non de la boîte à outils cible. En effet le langage de script, les API d'accessibilité et la réécriture d'une boîte à outils permettent d'utiliser les éléments de la boîte à outils cible. Cependant dans le cas de la réécriture d'une boîte à outils, la prise en compte est partielle car la structure et le layout de l'UI de départ ne sont pas modifiés pendant le portage.

Approches	Équivalences	Modélisations	Guidelines
Capture d'écran	Aucune équivalence	Aucune modélisation	Aucune prise en compte
Carte graphique virtuelle	Aucune équivalence	Aucune modélisation	Aucune prise en compte
Émulation du clavier et de la souris	Équivalences entre les dispositifs d'interactions en entrée des plateformes source et cible définies manuellement	Aucune modélisation	Aucune prise en compte des guidelines
Langage de script	Équivalences manuelles des dispositifs d'interactions	Modélisation des données, approche non réutilisable	Prise en compte partielle des guidelines
API d'accessibilité numérique	Manuelles	Modélisation de la structure	Prise en compte partielle des guidelines
Réécriture d'une boîte à outils	Manuelles	Aucune	Prise en compte partielle des guidelines

TABLE 4.1 – Synthèse des technologies de portage des UI sur tables interactives

Les mécanismes d'équivalences des approches de portage des UI (cf. figure 4.3) ne sont pas **flexibles** car les équivalences entre les instruments d'interactions ou avec les wrappers en ce qui concerne les bibliothèques graphiques sont définis avant le portage de l'application. Aucune modification n'est possible pendant le processus de migration.

Le portage des UI ne prend pas en compte toutes les guidelines de la cible, les résultats obtenus ne sont pas conformes aux **critères ergonomiques** des tables interactives. En effet, dans le cas de la migration vers les tables interactives, le portage des UI présente un inconvénient car il ne modifie pas la structure et le positionnement des éléments graphiques de l'UI source. Or la prise en compte des guidelines pour les UI collaboratives impose l'adaptation de la structure et du positionnement de l'UI source.

Contrairement à la migration manuelle, le portage des UI est une approche non spécifique à une application, mais spécifique à des instruments d'interactions. Le portage des UI permet de migrer les applications entre plateformes déterminées. Les mécanismes d'équivalences définis pour une migration sont **réutilisables** pour les autres applications mais pas pour d'autres instruments d'interactions. Le portage des UI réduit indiscutablement le coût de la migration par rapport à une approche manuelle.

4.3 Approches de migration basées sur les modèles de l'UI

Cette section présente une famille d'approches de migration basées sur la définition de modèles permettant d'être indépendants des applications et des plateformes. En effet, même si les solutions de portage des UI présentées à la section 4.2.2 utilisent des modèles de données et de structure de l'UI de départ pour générer l'UI cible. Ces modèles comportent des informations sémantiques qui incluent pour chaque composant son nom, son rôle (bouton menu, case à cocher, etc.), sa position sur l'écran et dans la hiérarchie en terme de contenant et de contenu. Ces informations sont exploitées par des mécanismes non réutilisables et spécifiques aux applications car ces mécanismes sont constitués de scripts spécifiques à une fonctionnalité d'une UI. Ils ne sont donc pas facilement réutilisables. Les

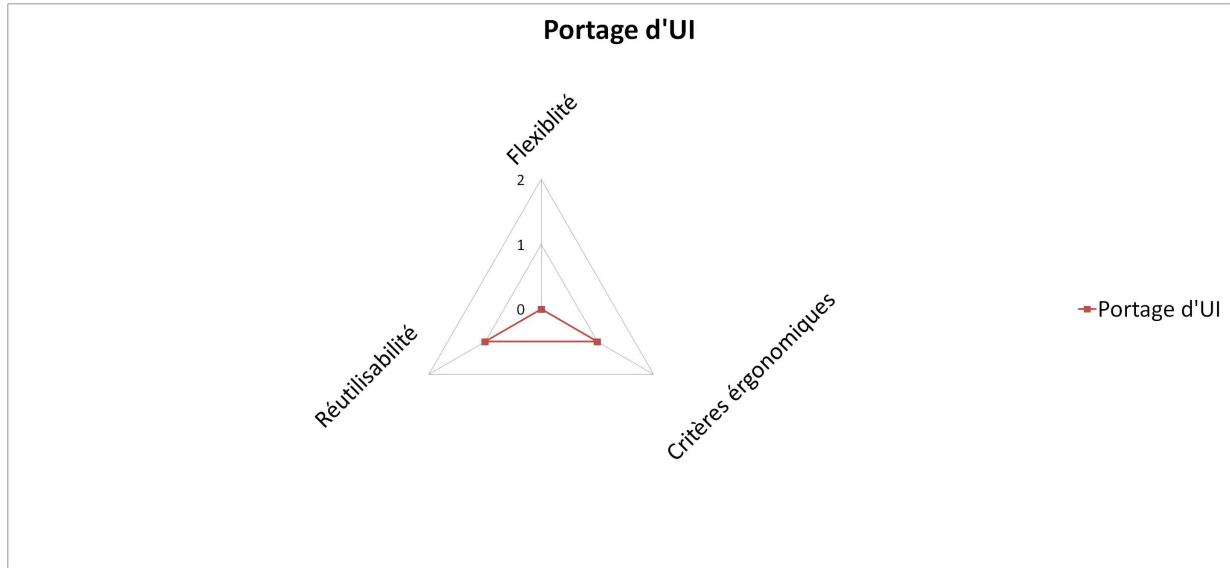


FIGURE 4.3 – Synthèse des approches de portage des UI sur tables interactives

mécanismes de migration des UI dans notre contexte sont constitués des processus de transformation des UI pour la plateforme cible.

Dans cette section nous étudions les approches de migration qui se basent sur des modèles qui décrivent la totalité des caractéristiques de l'UI. Nous décrivons en premier les approches automatiques de migration basées sur des modèles des UI, puis une approche semi automatique, basée sur des modèles de structure et d'interactions permettant la description des mécanismes de migration des UI.

4.3.1 Approches de migration automatiques des UI

Dans cette section nous présentons les approches de migration réutilisables pour différentes plateformes qui s'appuient sur des modèles de l'UI et qui font appel à des mécanismes automatiques.

Ces solutions de migration des UI sont génériques et sont utilisées par des services de migration des UI dans des contextes ubiquitaires comportant plusieurs types de plateformes. Les services de migration des UI sont chargés d'adapter une UI pendant son exécution. C'est une migration à la volée ou entre deux sessions d'une UI d'une application où l'état courant de l'UI est sauvegardé par des modèles abstraits de l'UI et adapté à autre plateforme afin de permettre à un utilisateur de continuer une tâche sans interruption [CCB⁺02]. Nous étudions dans cette section les modèles de l'UI et les mécanismes adoptés par ces approches.

4.3.1.1 Modèles de l'UI

Les modèles abstraits permettent de décrire les UI comportant en général plusieurs niveaux d'abstraction dans l'objectif de décrire les différents aspects des UI. Le framework de référence CAMELEON (CRF) [CCB⁺02] en propose quatre : le niveau des tâches et concepts, le niveau interface abstraite (AUI), le niveau interface concrète (CUI) et le niveau interface finale (FUI).

Modèle de tâches et concepts Ce modèle exprime les tâches et les concepts des UI dans un contexte précis tel que défini par le concepteur. Ce modèle exprime les interactions de l'UI avec les utilisateurs

ou le système, le comportement et les différents états des composants graphiques. Dans le cadre de la migration des UI à la volée, le modèle de tâches permet de conserver l'état d'une UI en cours d'exécution par exemple. Le modèle de tâches permet d'exprimer les interactions entre les utilisateurs et l'UI indépendamment des modalités d'interactions.

Modèle AUI Ce modèle permet de représenter les éléments d'une UI indépendamment des modalités d'interactions en exprimant leurs fonctionnalités essentielles. Ce modèle concrétise les éléments du domaine utilisé par le modèle de tâches, par exemple une tâche de saisie de données correspond à un élément de type Input dans le modèle AUI (cf. figure 4.4). Les éléments du modèle AUI permettent en général d'exprimer les différents types d'interactions tels que l'entrée de données (Input), l'affichage d'informations (Output) et l'activation d'une commande (Command). Ce modèle exprime aussi la structure d'une UI à travers les regroupements (Container) ou les types des données.

Le modèle d'AUI exprime les interactions de haut niveau entre UI et NF et indépendamment des dispositifs d'interactions. Ce modèle n'exprime pas les interactions sur les propriétés visuelles des composants graphiques (redimensionnement, déplacement, rotation, etc.) car il est indépendant de toute modalité, il peut être concrétisé en UI graphique, vocale ou multimodale, etc.

Modèle CUI Ce modèle permet de décrire une représentation concrète de l'UI suivant une modalité. Par exemple, dans le cadre des UI graphiques qui nous concernent, des composants graphiques sont choisis afin de raffiner les éléments du modèle AUI. Ce modèle exprime la structure, le positionnement (layout) et même le style des éléments graphiques. Les composants graphiques de ce modèle sont exprimés dans un langage indépendant des bibliothèques graphiques pour garantir leur réutilisabilité.

Dans le cas d'une migration vers une table interactive, le modèle de CUI exprime des UI de modalité graphique, interactive et tangible. On peut considérer la migration des UI desktop vers une table interactive comme un changement de bibliothèque graphique qui nécessite une adaptation des interactions, de la structure, du layout et du style de l'UI de départ.

Plusieurs implémentations du framework CAMELEON ont été mises en œuvre. Le langage XML MARIA (cf. Paternò *et al.* [PSS09]) et le langage USIXML (User Interface eXtensible Markup Language) [VLM⁺04] décrivent les UI aux niveaux des modèles AUI et CUI. Le modèle de tâches et concepts du CRF est implémenté par CTT (Concur Task Trees) [FCLD12]. Il permet d'exprimer de manière structurée et hiérarchisée les tâches d'une UI. La structure hiérarchique de CTT permet aussi d'exprimer les relations entre les sous tâches grâce à des opérateurs temporels.

Les différents modèles du CRF permettent de concevoir des UI multi plateformes, dans une approche de conception top down qui consiste à partir du modèle le plus abstrait (modèle de tâches et concepts), le concepteur génère les UI par raffinements successifs des modèles du CRF. Dans le cadre de la migration des UI, ces modèles sont utilisés soit par des mécanismes de génération des UI finales pour la nouvelle plateforme à partir des modèles abstraits [TS99], soit par reverse engineering de l'UI existante [PZ10].

4.3.1.2 Mécanismes de migration des UI top down

Les mécanismes de migration des UI top down sont basés sur le raffinement des modèles abstraits du CRF. Les concepteurs génèrent dans un premier temps le modèle CUI spécifique à chaque plateforme à partir des modèles AUI et des tâches. Cette approche permet de migrer les états des UI à l'exécution, MigriXML [MVL06] par exemple définit grâce à ces mécanismes un environnement virtuel comprenant différentes plateformes (desktops avec différentes tailles d'écran ou smartphones). MigriXML permet par exemple la migration d'une UI à l'exécution d'un desktop vers un smartphone

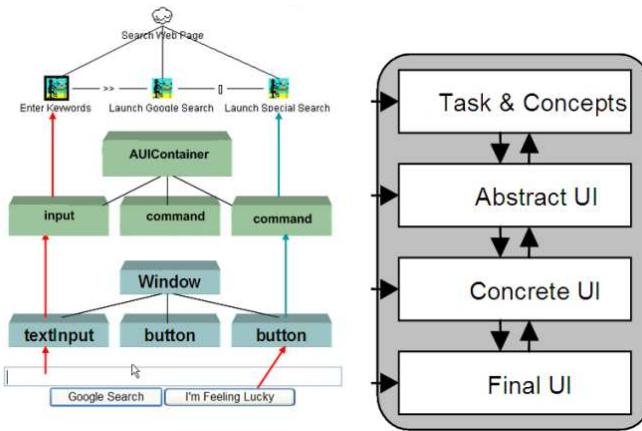


FIGURE 4.4 – Exemple de transformation USIXML

si certaines conditions sont remplies. En effet l’UI de l’application doit être décrit à l’aide d’USIXML et les modèles de CUI pour les desktops et les smartphones aussi doivent être décrits avant la migration de l’UI. MigriXML permet alors de migrer un contexte d’exécution d’une UI du desktop vers un smartphone et vice versa.

La génération des modèles les moins abstraits dans cette approche doit être effectuée dès la phase de conception. L’approche permet d’étendre une application à d’autres plateformes en réutilisant les modèles de tâches et d’AUI dans le cas d’une nouvelle plateforme ayant des modalités différentes de la plateforme de départ. Pour la migration d’une application existante, il est indispensable de construire les différents modèles du CRF associés. Cette contrainte exclue toutes les UI des applications qui ne sont pas conçues suivant le CRF sauf à les construire par reverse engineering.

4.3.1.3 Mécanismes de migration des UI bottom up

Ces mécanismes permettent la migration des UI par abstraction des UI existantes pour obtenir les modèles du CRF. Paternò *et al.* [PZ10] proposent par exemple, un service d’adaptation d’une page web à un téléphone portable (cf. figure 4.5) basé sur le reverse engineering et l’adaptation d’une UI existante. Ce service abstrait les pages HTML dans les modèles CUI et AUI du langage MARIA XML et transforme les modèles obtenus en fonction des principes suivants :

- adapter les tableaux pour qu’ils tiennent sur la surface d’écran disponible en découplant les tableaux trop larges ou en remplaçant une donnée d’un cellule trop large par un lien hypertexte,
- transformer tous les textes trop longs en liens hypertexte comme pour les données des cellules d’un tableau,
- transformer les images en réduisant leur taille,
- convertir les listes, par exemple en menu drop down,
- adapter la taille et la disposition des composants graphiques, par exemple en les redimensionnant et en adoptant un layout vertical.

Le concepteur définit des mécanismes de transformations [LVMB05] pour appliquer ces principes sur les modèles de CUI et d’AUI. La migration ne change pas de bibliothèque graphique car l’UI reste toujours écrite en HTML. Cependant le layout, la taille des composants graphiques, les données et le type de certains composants graphiques doivent être transformés et remplacés pendant la phase de migration. Les transformations des instances des modèles AUI et CUI de l’UI de départ sont horizontales

car elle génèrent d'autres instances de ces modèles pour la plateforme d'arrivée.

L'avantage de cette approche est que seuls les modèles indispensables à la migration sont abstraits à partir de l'UI de départ. Par exemple, si l'adaptation concerne le layout, le modèle de tâche n'est pas indispensable. Dans le cadre de la migration des UI vers les tables interactives qui implique un changement de dispositifs d'interactions, il est indispensable d'exprimer les interactions de manière abstraite dans les différents les modèles Tâches et AUI du CRF. Cependant, comme les modèles de tâches et AUI du CRF n'expriment pas toutes les interactions nécessaire à la migration (cf. section 4.3.1.1), il est donc nécessaire que les applications respectent la séparation entre UI et NF pour que toutes les interactions puissent être découvertes. MVC ou ARCH sont des architectures respectant la séparation souhaitée.

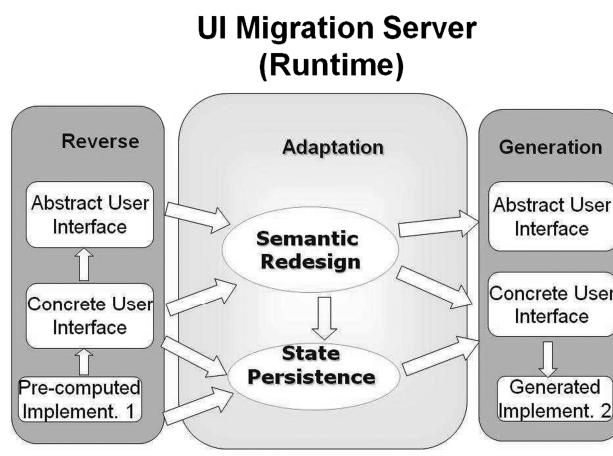


FIGURE 4.5 – Service de migration des UI

4.3.1.4 Migration des UI desktop vers les tables interactives par une approche automatique

Dans cette section nous étudions les adaptations nécessaires dans le cas d'une réutilisation du service de migration des UI (proposée par Paternò à la figure 4.5) pour la migration des UI vers les tables interactives. Nous souhaitons montrer par cette étude les points forts et les difficultés liés à l'adaptation de cette solution pour notre objectif. En considérant toujours notre exemple fil rouge : l'application desktop de départ (cf. section 2.2) respecte une architecture MVC et que la vue décrite à l'aide de la bibliothèque graphique Java Swing. La table interactive est la plateforme d'arrivée et elle permet de décrire des UI en utilisant XAML. Une adaptation du service de migration des UI nécessite :

- Une table d'équivalences (cf. tableau 4.2) entre les éléments du langage MARIA XML, XAML et l'API JavaSwing pour abstraire et concrétiser l'UI de départ. La table d'équivalences permet de décrire les correspondances entre les bibliothèques graphiques des plateformes sources et cibles. Ces équivalences sont symétriques car elles peuvent aussi être utilisées pour la migration des tables interactives vers les desktops.
- Une formalisation des guidelines des tables interactives en règles de migration pour décrire des UI qui respectent les critères ergonomiques de la cible. Nous remarquons que les guidelines formalisées en règles ne sont pas symétriques comme les tables d'équivalences car la formalisation des guidelines est spécifique à la plateforme cible.

MARIA XML	Java Swing	XAML Surface
Activator	JButton	Button
Single choice	JCombox	ListBox
Text Edit	JTextField	TextBox
Object	Image	Image
Grouping	JPanel	ScatterViewItem, Grid

TABLE 4.2 – Table d'équivalences

Les correspondances sont statiques et doivent être établies pour l'ensemble des composants graphiques d'une bibliothèque graphique. Évidemment, si une application n'utilise pas une partie de la bibliothèque source, il est inutile de rechercher les correspondances. La construction de cette table est donc évolutive et permet de capitaliser sur les migrations déjà effectuées. Les **équivalences statiques** sont établies en se basant sur les types des composants graphiques. Cependant le processus de migration se basant sur les instances des composants graphiques, chaque instance peut implémenter ou non les interactions décrites par le type. Par exemple une liste d'éléments peut implémenter ou non le glisser déposer, une instance qui ne l'implémente pas peut dans certains cas être transformée en un menu.

Par ailleurs, Silva *et al.* [SC12] proposent plusieurs critères pour la correspondance entre bibliothèques graphiques basées sur les langages XML. Le premier critère est le comportement des composants graphiques car il caractérise les actions utilisateurs indépendamment de la représentation du composant graphique. Les autres critères utilisables dans le cadre de la migration sont le style des UI et les balises des éléments graphiques. Par exemple, en considérant la guideline de partage de l'espace de travail (*Guideline 2 Partage de l'espace de travail*), elle peut être traduite dans les termes du langage MARIA XML, pour obtenir par exemple, à la règle suivante : tous les *Groupings* sont transformés en *ScatterViewItem* pour être conformes à la guideline d'utilisation 360° de l'UI.

4.3.1.5 Synthèse des approches automatiques de migration des UI

Cette section présente des mécanismes de migration des UI basés sur les modèles de CRF. Le modèle de CUI graphique permet de décrire la structure hiérarchique, les données et le positionnement d'une UI. Dans un mécanisme de migration des UI vers la table interactive, le modèle de CUI est transformé pour rendre l'UI de départ conforme aux guidelines des UI cibles qui sont tangibles et collaboratives.

Le modèle d'AUI décrit à la fois le regroupement des éléments abstraits de l'UI et les interactions (en entrée ou en sortie) entre l'utilisateur et le système (NF). Les modèles de tâches et de concepts décrivent les activités de l'UI et les comportements de l'UI de façon globale dans un langage de haut niveau.

Les interactions décrites par les modèles de tâches et AUI expriment les interactions sur une UI indépendamment des dispositifs d'interactions, mais elles n'expriment pas l'ensemble des comportements des composants graphiques nécessaires pour établir des équivalences.

Les approches automatiques de migrations sont plus **réutilisables** que le portage des UI car elles s'appuient sur des modèles qui décrivent tous les aspects d'une UI. Sur la figure 4.6, nous évaluons la réutilisabilité par la valeur maximale car les mécanismes d'équivalences et d'adaptations décrits sur les modèles de l'UI sont indépendantes des plateformes et des applications. Cependant les mécanismes d'équivalences des instruments d'interactions et les mécanismes d'adaptation (du layout

par exemple [PZ10]) sont spécifiques aux plateformes source et cible, leur réutilisation pour d'autres plateformes implique une charge de travail pour la mise en place de nouveaux mécanismes.

Les services de migration des UI se basent sur un processus automatique, ce qui réduit leur **flexibilité**. En effet les concepteurs n'interviennent pas pendant le processus pour l'ajuster ou le modifier par exemple. Sur la figure 4.6, nous évaluons la flexibilité de cette approche par une valeur non nulle mais inférieure à celle de l'approche manuelle. Le peu de **flexibilité** des mécanismes d'adaptation des services de migration des UI implique une charge de travail supplémentaire pendant la migration. En effet si la transformation de la structure ou du layout d'une UI n'est pas conforme aux attentes des utilisateurs par exemple, le concepteur doit modifier les règles de transformation des modèles abstraits. Cette modification se fait avant une migration des UI car le processus est automatique.

Les UI migrées par les services de migration des UI respectent les **critères ergonomiques** de la plateforme cible plus que celles obtenues par portage des UI par exemple. Cependant le peu de flexibilité de cette approche par rapport aux approches manuelles fait que la prise en compte des guidelines dépend des règles de transformation des modèles définis avant la migration. Sur la figure 4.6, nous évaluons le respect de critères ergonomiques par une valeur inférieure à la valeur maximale car toutes les guidelines ne sont pas toujours formalisées.

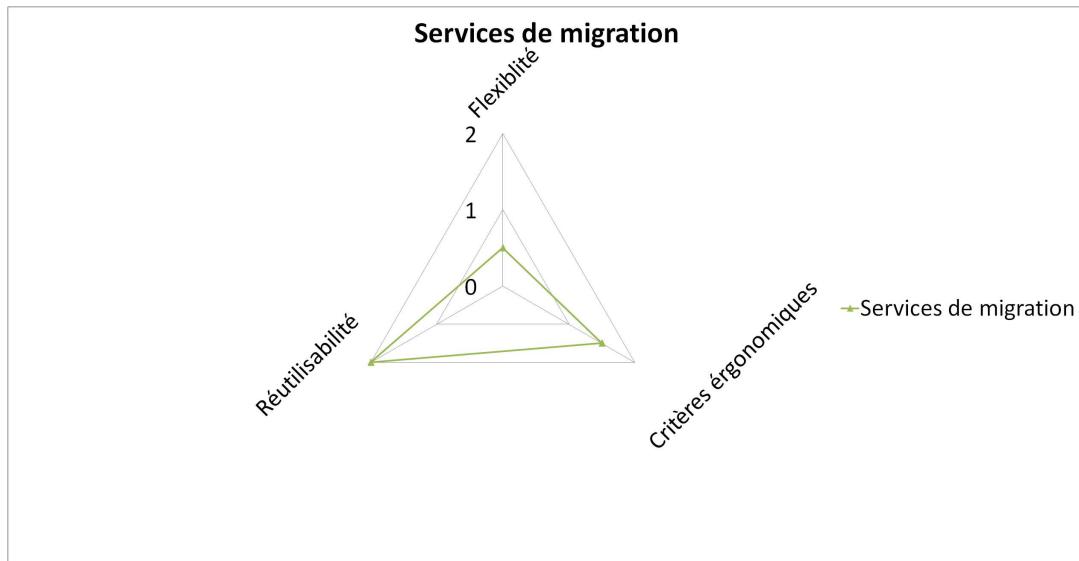


FIGURE 4.6 – Synthèse des approches automatiques de migration des UI

4.3.2 Approche semi automatique de migration des UI

Les processus de migration automatiques des UI sont limités en terme de flexibilité pour les personnes en charge de la migration. Dans cette section nous étudions un processus de migration semi automatique. MORPH [MR97] est par exemple une solution de migration d'une UI textuelle vers une UI graphique en se basant sur des modèles abstraits de l'UI et un support pour les transformations vers de nouvelles implantations graphiques. Le processus de migration avec MORPH implique une reconception de l'UI textuelle en UI graphique car les UI graphiques de type WIMP [vD97] supportent les dispositifs d'interactions de manipulations directes comme une souris. Cette nouvelle conception est aussi un **changement de modalités d'interactions** et l'utilisation d'une boîte à outils graphique. Nous étudions cette approche de migration car comme la migration des UI vers des tables interactives,

les modalités d’interactions des plateformes d’arrivée sont différentes des plateformes de départ avec de nouveaux dispositifs d’interactions.

4.3.2.1 Processus de migration semi automatique

Il est représenté par l’ensemble des mécanismes qui permettent l’extraction du modèle de l’UI, sa transformation et enfin sa génération pour la plateforme cible. Les modèles abstraits sont utilisés pour la représentation de l’UI. Par exemple, MORPH décrit un processus de migration en trois étapes : la détection, la représentation et la transformation (cf. figure 4.7).

1. La **détection** est une activité de reverse engineering qui consiste à analyser le code source de l’application à migrer dans le but d’identifier les modèles de structure et d’interactions.
2. La **génération** est l’opération inverse de la détection qui consiste à produire le code source de l’application migrée à partir de modèles abstraits transformés.
3. La **représentation** de l’UI de départ est un ensemble de modèles abstraits issu de la phase de détection. La **transformation** consiste à modifier et enrichir les aspects visuels de l’UI à travers les modèles, à ajouter et compléter les composants graphiques ou les fonctionnalités de l’UI de départ, à modifier et restructurer les types de données ou la structure hiérarchique de l’UI des modèles abstraits de l’UI source pour être utilisables dans l’environnement cible.

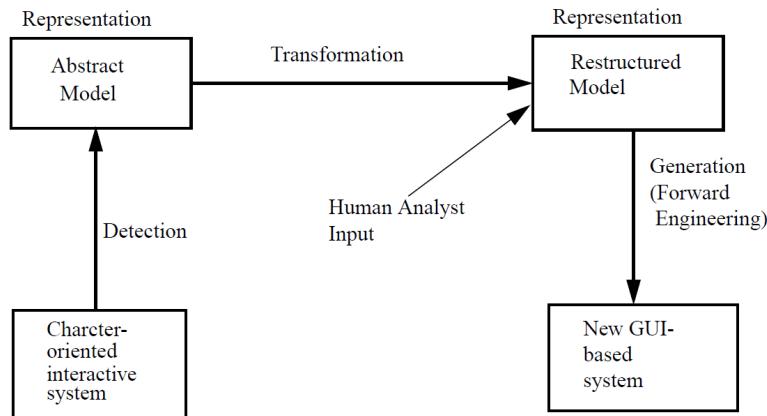


FIGURE 4.7 – Processus de migration avec MORPH

4.3.2.2 Les modèles abstraits des UI

Ce sont des éléments clés du processus MORPH car ils représentent les différents aspects (layout, activités, etc.) de l’UI à migrer. Les modèles sont décrits suivant deux niveaux d’abstractions :

- Le niveau des **tâches d’interactions** qui regroupe quatre interactions de base d’une UI [FvDFH90] qui sont :
 - la sélection dans une liste,
 - la quantification ou la saisie d’une donnée numérique,
 - l’indication de la position¹⁰ d’un élément sur l’écran

10. Sur une UI textuelle la position est exprimée en terme de ligne et de colonne pour positionner le curseur du clavier par exemple

- et la saisie d'une donnée textuelle.

Ces tâches d'interactions sont identifiées pour la migration des UI textuelle vers une UI graphique. Ce niveau décrit les activités utilisateurs sur une UI textuelle dans le but de l'adapter à une UI graphique. Il correspond à une partie du modèle des tâches du CRF (cf. section 4.3.1.1).

- Le niveau d'**objet d'interactions abstraites** qui représente les éléments abstraits indépendants d'une bibliothèque graphique (tel que Button, List, Menu, etc.), ce modèle est spécifique aux UI graphiques. Ce niveau permet de décrire la structure de l'UI indépendamment de la bibliothèque graphique, il correspond au niveau CUI du CRF (cf. section 4.3.1.1).

Dans le cadre de la migration, les tâches d'interactions permettent de décrire les interactions des objets d'interactions abstraits indépendamment du dispositif d'interactions de départ (clavier).

Les tâches d'interactions constituent un **modèle d'interactions abstraites** pour la migration d'une UI textuelle vers une UI graphique.

Les objets d'interactions abstraits sont raffinés à partir des attributs des différentes tâches d'interactions et en se basant sur le langage de représentation des connaissances CLASSIC [RBB⁺95].

Extraction des modèles abstraits Les tâches d'interactions et les objets d'interactions abstraites sont identifiés à partir des UI textuelles en se basant sur l'architecture de l'application à migrer et sur des **règles d'identifications** [Moo96, RFJ08]. Pour identifier les tâches d'interactions, Moore *et al.* [Moo96] décrivent les règles d'identifications sous la forme :

Si Condition est vraie; Alors identifier une Tâche d'interactions

où **Condition** correspond à une situation d'une instance des UI qui représente une **Tâche d'interactions** du modèle abstrait. Cette forme de règle nécessite que l'ensemble des situations soit identifié par un mapping entre la boîte à outils de l'UI source et les interactions abstraites.

Il existe d'autres approches pour extraire un modèle à partir d'une UI existante, par exemple Ratiu *et al.* [RFJ08] proposent une ontologie pour analyser et comprendre des API spécifiques à un domaine comme les bibliothèques graphiques. Le principe de l'approche consiste d'abord à construire une ontologie capable de représenter les concepts d'une bibliothèque graphique que nous souhaitons utiliser pour la migration ; dans notre cas ce sont la structure et les interactions abstraites d'une UI. Toujours dans notre cas l'ontologie décrirait les liens de contenance entre les composants graphiques, les données et les tâches d'interactions en entrée (sélection, quantification, position, édition).

Ensuite, à partir d'une UI décrite à l'aide des instances des éléments d'une bibliothèque graphique, nous pouvons extraire les objets d'interactions abstraites grâce à l'algorithme d'extraction décrit dans [RFJ08] se basant sur l'ontologie proposée. Les objets d'interactions abstraites de l'UI source sont utilisés comme pivot pour décrire l'UI cible.

Cette approche d'extraction présente un avantage par rapport aux règles d'identification de [Moo96] car elle ne s'appuie pas sur des situations décrivant des cas possibles dans une UI mais elle se base sur les types des éléments utilisés pour décrire l'UI et elle est plus générique. Cependant le choix de l'ontologie doit être exhaustive pour une API et pour les concepts présents.

Mécanismes de transformations et de génération de l'UI Une fois l'UI source décrite par les tâches d'interactions et ensuite raffinée en objets d'interactions abstraites, la représentation abstraite de l'UI de départ est transformée en modifiant manuellement le modèle de l'UI.

La **transformation** consiste d'une part à remplacer des objets d'interactions de l'UI source. Par exemple, une tâche de sélections dans une liste qui est raffinée en une liste à choix unique peut être remplacée par un menu si le nombre d'éléments est inférieur à N (N étant une constante à définir selon la longueur de menu souhaité).

D'autre part cette phase fait intervenir un utilisateur humain pour définir la position, la taille ou pour modifier l'UI. Le concepteur intervient manuellement sur les modèles de l'UI après la transformation dans le but de placer les éléments de l'UI. En effet, l'UI textuelle de départ n'est pas structurée comme une UI graphique et les modèles de tâches d'interactions et d'objets abstraits d'interactions ne permettent pas, par exemple, de décrire le layout.

La sélection des éléments de la bibliothèque graphique cible se fait en recherchant dans une bibliothèque graphique décrite par une ontologie des éléments qui correspondent le plus à un objet abstrait du modèle restructuré.

Le mapping entre les objets abstraits et les éléments d'une bibliothèque graphique est fait **dynamiquement en se basant sur les ontologies**. En effet les correspondances entre les objets abstraits et les éléments de la bibliothèque graphique ne sont pas définies manuellement et de manière exhaustive à la conception de la solution, mais elles sont établies en se basant sur les attributs de chaque élément. Le mécanisme d'équivalences entre les bibliothèques graphiques se base sur les propriétés de chaque composant graphique décrit par une ontologie. Cependant ce mécanisme ne prend pas en compte les propriétés de l'instance des composants mais celles décrites par le type.

Les mécanismes de transformations et générations utilisés par MORPH sont basés aussi sur des ontologies. Dans le cadre de la transformation de l'UI de départ par exemple, il est possible d'introduire des guidelines dans la base de connaissances, en préférant remplacer une liste en menu si elle contient moins de 10 éléments par exemple.

La transformation utilisée par MORPH permet d'inclure les principes de conception des UI pour la plateforme cible dans la base de connaissances [MR97].

4.3.2.3 Comment adapter MORPH pour la migration des UI vers les tables interactives ?

Dans l'objectif d'étudier les points forts et les limites d'une approche de migration semi automatique comme MORPH, nous l'adaptons dans notre cadre.

La réponse à la question posée passe par l'ajout de la bibliothèque graphique de la table interactive à la base de connaissances pour permettre l'extraction et la génération de l'UI finale. Ensuite, il faut décrire les modèles abstraits. Par exemple, nous utilisons les modèles abstraits de l'approche MORPH pour représenter les interactions des éléments graphiques d'un artefact de l'UI CBA (cf. figure 3.4) :

- le menu principal a une tâche d'interactions de type SELECTION-OBJET avec les rôles (action = *Procedural-Action*, number-of-states = (2..10), variability = *fixed*, grouping= *not-grouped*)
- la liste déroulante (*ComboBox*) a une tâche d'interactions de type SELECTION-OBJET avec les rôles (action= *Procedural-Action*, number-of-states = (2..10), variability = *fixed*, grouping = *not-grouped*)
- la liste d'images a une tâche d'interactions de type SELECTION-OBJET avec les rôles (action = *Procedural-Action*, number-of-states = (2..10), variability = *fixed*, grouping = *not-grouped*)

Enfin, il faut mettre en place des règles de transformations des modèles abstraits en prenant en compte les guidelines. Par exemple, la prise en compte de la guideline 2 de partage de l'espace de travail qui préconise de remplacer les composants graphiques qui en contiennent d'autres avec ceux qui sont déplaçables. Cependant, nous remarquons que le modèle de tâches d'interactions présenté ci-dessus ne permet pas de décrire les comportements des composants graphiques d'une bibliothèque mais les activités possibles des utilisateurs. Par exemple ce modèle permet de caractériser les fonctionnalités d'un menu, mais ne permet pas de préciser si un menu est déplaçable. Dans le cadre de la migration vers les tables interactives, les comportements des éléments graphiques sont indispensables pour prendre en compte les guidelines pour les UI collaboratives. Les modèles abstraits utilisés dans ce cadre doivent être capables de décrire et de sélectionner les composants graphiques conformes à une guideline.

4.3.2.4 Synthèse du processus semi automatique de migration des UI

Cette approche propose une solution de migration réutilisable qui prend en compte une phase de reverse engineering (**détection**), une phase de re engineering (**transformation**) et une phase de forward engineering (**génération**). Le tableau 4.3 présente l'approche MORPH suivant les caractéristiques d'équivalences des éléments des plateformes, de modélisation et de prise en compte des guidelines.

	Équivalences	Modélisations	Prise en compte des guidelines
MORPH	Dynamique des bibliothèques graphiques basées sur des modèles de connaissances	Modélisation des interactions abstraites, Modélisation de la structure	Prise en compte par les règles de transformations des modèles abstraits

TABLE 4.3 – Récapitulatif de MORPH

L'approche est basée sur des modèles abstraits qui décrivent la structure et les interactions (et les comportements) des UI à migrer. Le layout et le style des UI ne sont pas modélisés par cette approche et leur migration est effectuée manuellement par un utilisateur humain. Cependant les modèles abstraits proposés par MORPH ne sont pas adaptés pour être utilisés sur une plateforme comme une table interactive. Le modèle de tâches d'interactions par exemple ne prend pas en compte les mouvements (rotation, déplacement, etc) des composants graphiques des tables interactives. Le modèle de tâches d'interactions permet d'établir des équivalences entre dispositifs d'interactions en se basant sur des tâches de haut niveau d'abstraction et indépendantes des plateformes. Cependant ce modèle n'est pas adapté à la migration des UI vers les tables interactives.

Les équivalences entre les bibliothèques graphiques sont basées sur des modèles de connaissances et peuvent être établies dynamiquement. Ces équivalences sont uniquement basées sur les types des composants graphiques.

Les **équivalences dynamiques** entre les bibliothèques graphiques permettent de sélectionner les composants graphiques appropriés pour chaque cas en fonction de leurs caractéristiques et de leur utilisation.

Cette approche permet aussi la prise en compte des guidelines à travers les transformations (remplacement des objets abstraits ou sélection des composants graphiques spécifiques). Cependant l'identification des guidelines et leur prise en compte pendant la transformation doivent être décrit par le concepteur de la solution de migration sur une plateforme comme les tables interactives.

L'évaluation de cette approche suivant l'axe de la **réutilisabilité** des différents mécanismes révèle qu'elle est moins réutilisable que les approches basées sur des processus automatiques car tous les aspects de l'UI ne sont pas décrits par des modèles abstraits. Cette approche présente un coût de mise en œuvre important. Dans le cadre de la migration des UI vers les tables interactives, il est d'abord indispensable de décrire un modèle d'interactions abstraites capables de prendre en compte toutes les interactions. Ensuite il est nécessaire de décrire les transformations automatiques de la structure. Sur la figure 4.8, nous évaluons la réutilisabilité de cette approche par une valeur inférieure à celle de l'approche automatique et supérieure au portage des UI.

La **flexibilité** de la migration semi automatique est supérieure à celle des approches automatiques car le processus fait intervenir un humain pour adapter les aspects qui ne sont pas pris en compte par les modèles abstraits. Sur la figure 4.8, nous évaluons la flexibilité de cette approche par une valeur supérieure à celle de l'approche automatique et inférieure à l'approche manuelle.

La prise en compte des guidelines dans cette approche est faite par les processus automatiques et en se basant sur les connaissances des personnes en charge de la migration pour les adaptations manuelles. L'UI produite permet un respect des **critères ergonomiques** égal à l'approche manuelle.

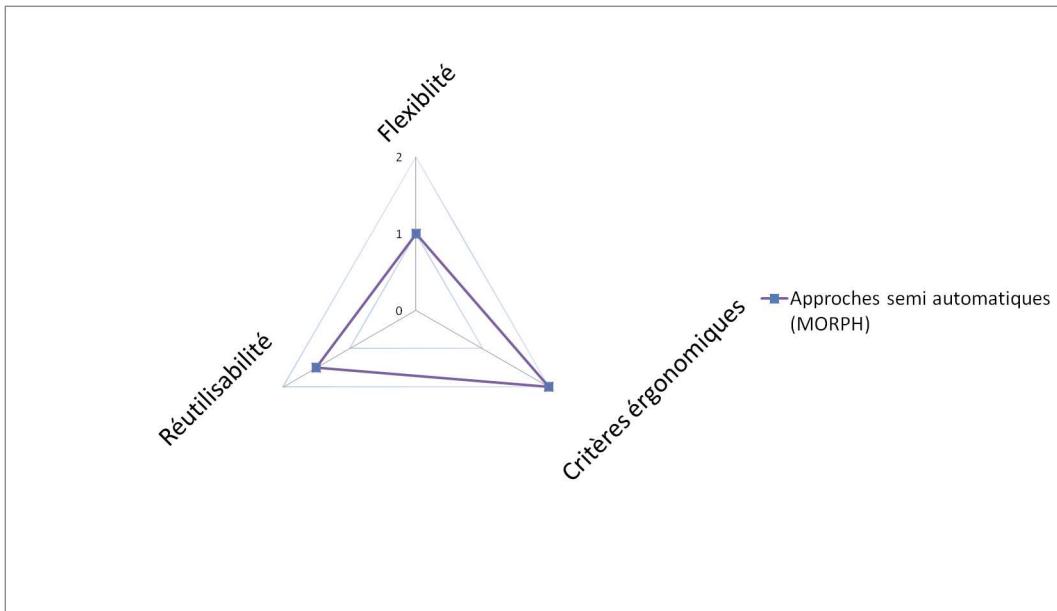


FIGURE 4.8 – Synthèse de l'approche semi automatique

4.4 Synthèse et objectifs

Dans cette section nous faisons une synthèse des approches de migration des UI décrites dans ce chapitre et nous spécifions nos objectifs en nous basant sur les résultats de cette synthèse.

4.4.1 Synthèse

Nous avons présenté à la figure 4.9 le récapitulatif des approches de migration des UI étudiées suivant les critères d'évaluation décrits à la section 4.1. Ces approches de migration des UI nous montrent que les solutions de migration peuvent être spécifiques à une application ou à une bibliothèque graphique. Ces solutions peuvent aussi être réutilisables en se basant sur une modélisation des différents aspects d'une UI.

Cette étude nous a permis de raffiner les mécanismes d'équivalences. Nous avons identifié deux types d'équivalences entre les éléments des plateformes :

- les **équivalences statiques** entre les bibliothèques graphiques qui sont définies par les concepteurs à l'aide d'une table d'équivalences par exemple,
- et les **équivalences dynamiques** qui sont établies en se basant sur les caractéristiques des éléments à comparer en utilisant les inférences d'un modèle de connaissances par exemple.

En ce qui concerne les mécanismes de prise en compte des guidelines, nous avons constaté que le concepteur peut se baser sur ses **connaissances** dans une approche manuelle pour décrire les mécanismes de transformations et d'équivalences. Pour des approches qui modélisent l'UI à migrer, les guidelines à considérer sont **traduites en règles** de transformation des différents aspects. Les guidelines permettent aussi d'établir des équivalences entre les bibliothèques graphiques.

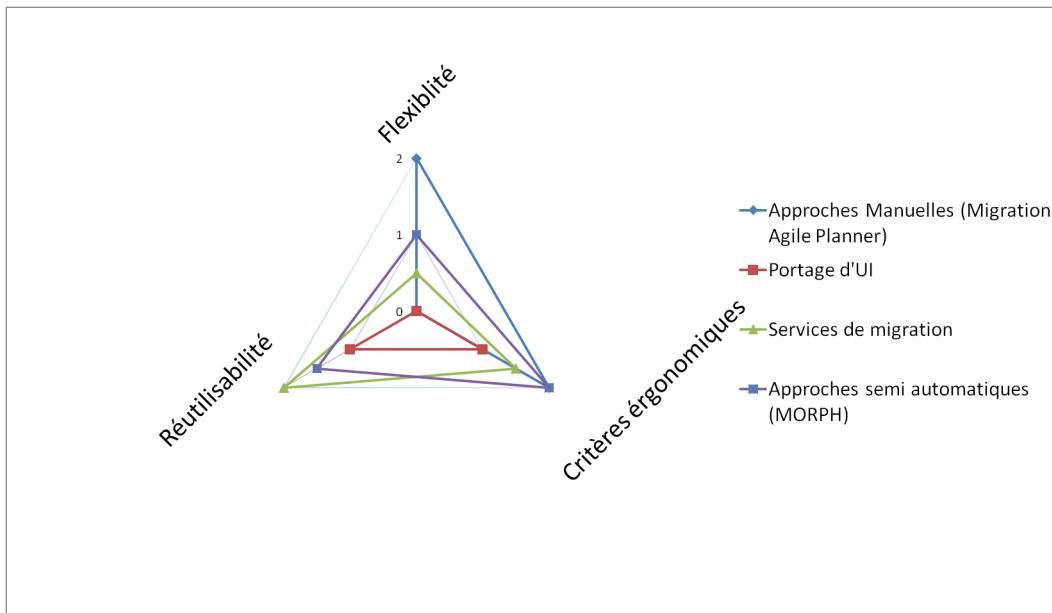


FIGURE 4.9 – Synthèse des approches de migration des UI

Les mécanismes d’adaptation basés sur des modèles de l’UI utilisés par les différentes approches présentées nous permettent d’affirmer que :

- les **interactions** des UI peuvent être modélisées partiellement par les modèles de tâches et AUI [PMM97] ainsi que les tâches d’interactions [FvDFH90, KSM99a].
- la **structure** des UI comprend à la fois les données de l’UI et les relations hiérarchiques entre les différents composants de l’UI. Cet aspect de l’UI peut être modélisé par des **méta données** [BRNB07] ou des modèles de l’UI [VLM⁺04, PSS09] pour décrire les composants graphiques et les données de l’UI.
- le **positionnement** des éléments de l’UI graphiques peut aussi être décrit dans un modèle indépendant d’une plateforme. Les langages de description des UI (UIDL) tels que USIXML, MARIA, UIML permettent de décrire le layout.
- le **style** des UI peut être modélisé au travers d’UIDL comme UIML¹¹.

4.4.1.1 Les points forts des approches présentées

La figure 4.9 présente la synthèse des approches étudiées suivant les axes de la réutilisabilité, la flexibilité des mécanismes et le respect des critères ergonomiques à travers la prise en compte des guidelines. Nous identifions les points forts suivants :

- Les solutions de migration des UI [WB03, MR97] **flexibles** et qui font intervenir les utilisateurs humains permettent d’avoir des UI qui respectent des critères ergonomiques et qui sont proche des attentes des utilisateurs finaux. En effet ces approches permettent une reconception manuelle pendant la migration et les prises en compte des guidelines sont plus fines pour les utilisateurs ayant une bonne connaissance des plateformes cibles. Parmi les approches réutilisables, les processus semi automatiques sont les plus flexibles.
- Les solutions de migration basées sur les modèles permettent de décrire des mécanismes de transformations et d’équivalences réutilisables pour des applications respectant une architecture

11. UIML : An Appliance-Independent XML User Interface Language

définie. L'utilisation des modèles de l'UI permet d'accroître la réutilisabilité d'une approche et permet aussi la réduction du temps de la migration des applications une fois que les modèles et les mécanismes de transformation sont mis en place.

4.4.1.2 Les limites

La réutilisation des approches basées sur des modèles de l'UI dans le cadre des tables interactives nous permet d'identifier les limites ci-dessous :

- Le portage des UI n'est pas une approche de migration des UI adaptées pour la migration vers les tables interactives car elle ne permet pas la prise en compte des guidelines des UI collaboratives. Cette approche peut par contre être adoptée pour la migration entre plateformes proches, par exemple pour faire migrer des UI Microsoft PixelSense 1.0 vers la version 2.0 grâce aux techniques de types *wrappers*.
- Les approches de migration manuelles présentent des coûts de mise en œuvre comparables à ceux d'une nouvelle conception. L'automatisation des mécanismes d'équivalences et d'adaptations de l'UI source permet une réduction de ces coûts.
- Les services de migration des UI sont des solutions avec des processus fermés et très peu flexibles. La flexibilité d'une approche permet d'avoir des résultats conformes aux attentes des utilisateurs finaux.
- L'approche semi automatique est flexible mais les mécanismes d'adaptation ou de prise en compte des guidelines n'aident pas les concepteurs ou développeurs pendant la migration. En effet les mécanismes d'équivalences des composants graphiques peuvent proposer les équivalences en précisant ceux qui sont conformes aux guidelines de la cible par exemple.

4.4.2 Objectifs

Les principaux objectifs de notre travail de thèse est de **réduire le coût de la migration** et de **prendre en compte les guidelines de plateforme d'arrivée** lors de la migration des UI existantes vers les tables interactives. Nous faisons l'hypothèse que les applications à migrer sépare le NF et l'UI. Pour atteindre ces objectifs, nous choisissons d'adapter l'UI source par un processus de migration semi automatique en trois étapes : d'abord extraire l'UI source par reverse engineering en l'abstrayant dans des modèles abstraits. Ensuite, transformer les modèles abstraits tout en faisant intervenir le concepteur pour personnaliser les aspects non modélisés. Et pour terminer, nous générerons l'UI finale à partir des modèles abstraits transformés.

En considérant cet objectif principal, l'étude des différentes approches de migration des UI de façon globale, mais aussi plus particulièrement ceux traitant de la migration vers les tables interactives, nous montre des modèles d'interactions qui ne facilitent pas les équivalences entre les plateformes. Pour atteindre notre objectif, nos contributions sont :

- de proposer un modèle d'interactions qui permet d'accroître la **flexibilité** des mécanismes de changement de modalité d'interactions et la préservation des interactions de l'UI de départ tout en prenant en compte celles de la plateforme d'arrivée,
- de décrire des mécanismes d'adaptations réutilisables et flexibles en prenant en compte les guidelines de la plateforme cible.

Dans la partie suivante, nous présentons le cœur de notre approche. En nous servant de cette étude, nous proposons une approche basée sur un modèle d'interactions abstraites qui permet d'établir des équivalences dynamiques entre les plateformes en prenant en compte les interactions des instances. Nous souhaitons inclure aussi la prise en compte des guidelines pour les tables interactives.

Troisième partie

Méthodes

Modèles pour la migration de UI vers les tables interactives

Sommaire

5.1	Introduction	67
5.2	Primitives d'interactions	68
5.2.1	Les primitives d'interactions en entrée	68
5.2.2	Les primitives d'interactions en sortie	72
5.3	Modèles de composants graphiques	73
5.3.1	Un modèle de types de composants graphiques	75
5.3.2	Un modèle d'instance d'une UI	81
5.3.3	Synthèse des modèles abstraits	90
5.4	Opérateurs d'équivalences	91
5.4.1	Opérateurs d'équivalences	91
5.4.2	Opérateurs d'équivalences & types de données	95
5.5	Synthèse	98

5.1 Introduction

Les approches de migration des UI génériques et réutilisables que nous avons étudiées dans le chapitre précédent se basent sur des modèles abstraits qui décrivent différents aspects¹² des UI indépendamment des plateformes et des applications.

La migration des UI desktops vers les tables interactives implique nécessairement une prise en compte des nouveaux dispositifs d'interactions disponibles dans cet environnement. Afin d'effectuer cette migration de manière cohérente il nous semble essentiel de mettre en place un mécanisme d'équivalences entre les instruments d'interactions disponibles sur les plateformes source et cible. Ce mécanisme d'équivalences prend en compte les principes de conception nécessaires à la mise en œuvre des UI collaboratives et tangibles. Pour ce faire, nous nous basons sur l'idée de décrire les correspondances entre les instruments d'interactions de la source et de la cible à l'aide d'un modèle d'interactions abstraites (cf section 3.1.4). Ce modèle d'interactions abstraites permet de caractériser les instruments d'interactions indépendamment des plateformes et des langages.

Ce chapitre propose donc à la section 5.2 un modèle d'interactions abstraites qui décrit les interactions atomiques possibles sur les composants graphiques indépendamment des instruments d'interactions et celles qui sont nécessaires pour la migration des UI vers les tables interactives. Nous utiliserons ultérieurement ce modèle pour décrire des équivalences entre composants graphiques. La section 5.3 présente les caractéristiques des composants graphiques en fonction du modèle d'interactions abstraites proposé. La section 5.4 propose un ensemble d'opérateurs d'équivalences entre les composants graphiques en se basant sur le modèle d'interactions abstraites proposé.

12. Interactions [GH95, KSM99b], Structure, Positionnement [PSS09, VLM⁺04] et Style

5.2 Un modèle d’interactions abstraites

Les modèles de systèmes interactifs [Nig94, RSP11, Weg97] distinguent deux types d’interactions entre les utilisateurs et une UI [W3C03] : les interactions en entrée et les interactions en sortie. Les interactions en entrée permettent de fournir des données ou d’invoquer des fonctionnalités de l’application grâce à une UI et à des dispositifs d’entrée (souris, clavier, microphone, camera de reconnaissance gestuelle, écran tactile, accéléromètre, etc.). Les interactions en sortie permettent d’effectuer des rendus des données de l’application à travers des dispositifs de sortie (tel un écran, des haut-parleurs, etc.). Les interactions (en entrée ou en sortie) se déclinent en plusieurs modalités d’interactions en fonction des langages et des dispositifs d’entrée ou des dispositifs de sortie utilisés. Chaque modalité d’interactions est utilisée comme un canal de communication entre un utilisateur et une application pour transmettre ou acquérir des informations [Nig94].

Les composants graphiques sont caractérisés par les données qu’ils contiennent, leurs propriétés graphiques et leurs comportements. Ils appartiennent à des bibliothèques graphiques qui permettent de décrire des UI graphiques en 2D, des images de synthèse en 3D, des jeux vidéo, des graphiques de données [BCW⁺06, Lon10, SWND03], etc. Ainsi, les interactions en entrée modifient l’état d’un composant graphique soit à travers ses propriétés (données contenues, taille, position et représentation), soit par son comportement (tels les appels des fonctionnalités du NF et les interactions sur d’autres composants graphiques d’une UI, etc.). Les interactions en sortie quant à elles permettent d’afficher des données de l’application. Les données des composants graphiques sont soit des **données graphiques** si elles décrivent le composant en question, soit des **données de l’application** si elles proviennent du NF ou des utilisateurs.

Une interaction en entrée ou en sortie sur les composants graphiques d’une UI peut être décomposée en une succession d’actions élémentaires que nous appelons **primitives d’interactions**. Il est possible d’obtenir une décomposition atomique. Cette section présente une caractérisation des primitives d’interactions en fonction des deux types d’interactions (entrée et sortie) ainsi qu’une modélisation des primitives d’interactions qui sont utiles pour décrire des équivalences entre les plateformes source et cible.

5.2.1 Les primitives d’interactions en entrée

Elles décrivent de manière abstraite toutes les actions atomiques des utilisateurs qui permettent de sélectionner des composants graphiques et d’en modifier une propriété. Les primitives d’interactions en entrée permettent de caractériser aussi les interactions qui font appel aux fonctionnalités du NF. Par exemple, l’action de cliquer à l’aide d’une souris sur un bouton qui fait appel à une fonctionnalité du NF se décompose en plusieurs primitives d’interactions en entrée.

Nous avons identifié comme des interactions en entrée sur les contenus : l’édition (**Data Edition**), la sélection (**Data Selection**) et le déplacement (**Data Move In** et **Data Move Out**). Ces primitives sont indépendantes des dispositifs d’interactions en entrée et peuvent être effectuées soit avec des dispositifs de manipulation directe (souris, écran tactile, reconnaissance gestuelle, etc.), soit avec des dispositifs de manipulation indirecte (clavier physique ou virtuel, reconnaissance vocale, etc.) ou soit par la combinaison de plusieurs dispositifs (clavier et souris).

Primitive d’interactions 1 : Data Edition

Elle consiste à modifier les données des composants graphiques qui ont des **données de l’application**.

La modification du contenu d'un composant graphique peut se faire avec un clavier, une reconnaissance vocale, gestuelle ou de forme en fonction des plateformes. Dans le cadre des tables interactives, l'édition se fait avec un clavier virtuel ou à main levée sur un écran tactile. En ce qui concerne les desktops, l'édition d'un contenu se fait avec un clavier, une synthèse vocale ou avec un clavier virtuel et une souris.

Primitive d'interactions 2 : Data Selection

Elle permet d'accéder aux **données de l'application** d'un composant graphique.

Les composants graphiques peuvent contenir plusieurs données de même type ; dans le cas des listes par exemple, cette primitive d'interactions permet la sélection d'un contenu précis. Elle se fait à l'aide d'un clavier, d'une souris, d'un écran tactile, d'un dispositif de reconnaissance gestuelle, etc. Dans le cadre d'une table interactive, la sélection de contenus se fait de manière directe sur l'écran tactile ou aussi avec un objet tangible. Par exemple, la sélection dans une liste d'images avec un objet tangible consiste à poser l'objet sur l'élément en question.

Primitive d'interactions 3 : Data Move Out

Elle permet d'exporter la (les) donnée(s) de l'application d'un composant graphique vers un autre composant graphique comportant des données de même type.

Les composants graphiques peuvent s'échanger des contenus. Le drag et le drop permettent de déplacer des contenus à l'aide d'une souris ou d'un geste tactile. Cette primitive d'interactions correspond à la première action d'un drag & drop. L'interaction peut être réalisée sur une table interactive en utilisant l'écran tactile ou être émulée avec un objet tangible. Avec un objet tangible comme un stylo par exemple, cette primitive d'interactions représente l'initiation de l'action drag & drop après la primitive de **Data Selection** du contenu à déplacer. La primitive d'interactions **Data Move Out** exprime la possibilité d'exporter la(les) donnée(s) d'un composant graphique vers un autre.

Primitive d'interactions 4 : Data Move In

Elle permet de recevoir la (les) donnée(s) de l'application d'un autre composant graphique de type compatible.

Cette primitive d'interactions correspond à la seconde étape du drag & drop, l'interaction peut être réalisée sur une table interactive en utilisant l'écran tactile ou être émulée avec un objet tangible. Avec un objet tangible comme un stylo par exemple, cette primitive d'interactions représente la fin de l'action drag & drop après la primitive de **Data Move Out**. La primitive d'interactions **Data Move In** exprime la possibilité d'importer des données vers un composant graphique.

Les quatre primitives d'interactions en entrée sur le(s) contenu(s) des composants graphiques décrivent de manière exhaustive l'ensemble des actions possibles qu'un utilisateur peut faire sur un élément graphique contenant des données de l'application. En effet, ces primitives d'interactions permettent :

- la création (ou “*Create*”, en éditant les données de l’application par la primitive d’interactions **Data Edition**),
- l’accès en lecture (ou “*Read*”, en sélectionnant le(s) contenu(s) par la primitive d’interactions **Data Selection** ou **Data Move Out**),
- la modification (ou “*Update*”, en éditant le(s) contenu(s) par les primitives d’interactions **Data Edition** ou **Data Move In**) et
- la suppression des données (ou “*Delete*”, des contenus avec les primitives d’interactions **Data Move Out** et **Data Edition**).

Ces primitives d’interactions couvrent les opérations *Create*, *Read*, *Update* et *Delete* (CRUD) définies dans [D. 06] sur les données des composants graphiques. Les primitives d’interactions en entrée sur les contenus constituent un ensemble complet d’interactions atomiques et permet de décrire toutes les actions utilisateurs sur les données d’applications des composants graphiques.

Par ailleurs, les actions utilisateurs sur une UI peuvent aussi modifier les propriétés graphiques lors d’un déplacement (**Widget Move**), ou par une rotation sans changement d’emplacement (**Widget Rotation**), ou un redimensionnement (**Widget Resize**). Un composant graphique peut être sélectionné de manière directe (**Widget Selection**) ou indirecte en naviguant à travers un container (**Navigation**). Les primitives d’interactions sur les propriétés graphiques sont indépendantes des dispositifs d’interactions en entrée. Elles peuvent être effectuées avec des dispositifs de manipulation directe (souris, écran tactile, reconnaissance gestuelle, etc.), avec des dispositifs de manipulation indirecte (claviers physiques ou virtuels, reconnaissance vocale, etc.) ou en combinant l’utilisation de plusieurs dispositifs (clavier et souris).

Primitive d’interactions 5 : Widget Move

Elle permet d’exprimer le changement de la position d’un composant graphique.

La position des composants graphiques peut être changée par un utilisateur à l’aide d’un dispositif de manipulation directe ou des raccourcis d’un clavier. Cette primitive d’interactions est définie pour les composants graphiques déplaçables à l’aide d’un dispositif d’interactions.

Primitive d’interactions 6 : Widget Rotation

Elle permet d’exprimer le changement d’orientation d’un composant graphique.

L’orientation des composants graphiques est une caractéristique importante dans le cadre des UI collaboratives et co localisées. En effet l’orientation des composants graphiques permet l’utilisation d’une UI à partir de n’importe quelle position autour de la table. Cette opportunité favorise l’utilisation d’une UI par plusieurs personnes autour de la table. La rotation de composants graphiques n’est pas une interaction en entrée indispensable pour les composants graphiques de la plateforme desktop car l’UI est destiné à une utilisation dans une position fixée. Dans le cadre de la migration vers des plateformes multi-utilisateurs cette primitive d’interactions est indispensable.

Primitive d’interactions 7 : Widget Resize

Elle permet de modifier les dimensions d’un composant graphique.

Les dimensions d’un composant graphique peuvent être redéfinies par un utilisateur d’une UI. Cette action peut être effectuée avec un clavier ou une souris sur un desktop et à l’aide d’un geste tactile

sur une table interactive. Le redimensionnement des composants graphiques en contenant d'autres implique de redimensionner leurs composants fils tout en préservant le critère ergonomique de guidage (cf 3.2.2). La préservation du guidage implique d'avoir les mêmes composants graphiques après le redimensionnement. Les tailles des composants graphiques redimensionnables peuvent être encadrées pour préserver le guidage.

Primitive d'interactions 8 : Widget Selection

Elle permet d'exprimer la sélection immédiate de composants graphiques avec un dispositif de manipulation directe.

Cette primitive d'interactions décrit une sélection directe d'un composant graphique par un utilisateur à l'aide d'un dispositif de manipulation directe. Cette action peut être effectuée avec un clavier ou une souris sur un desktop et à l'aide d'un geste tactile sur une table interactive. C'est l'une des deux interactions atomiques préalables à toute action utilisateur sur les données d'application ou les propriétés graphiques d'un composant graphique.

Primitive d'interactions 9 : Navigation

Elle permet d'exprimer la sélection d'un composant graphique de manière séquentielle.

Cette primitive d'interactions décrit une sélection séquentielle des composants graphiques d'une UI par un utilisateur à l'aide de dispositifs de manipulation non directe. C'est l'une des deux interactions atomiques préalables à toutes les actions des utilisateurs sur les données d'application ou les propriétés graphiques d'un composant graphique.

Les primitives d'interactions ci-dessus¹³ caractérisent les actions utilisateurs sur les propriétés graphiques (position, orientation, taille et représentation) d'un composant graphique. Elles permettent de décrire l'ensemble des actions utilisateurs sur l'orientation, la taille et la position des composants graphiques.

Primitive d'interactions 10 : Activation

Elle permet de décrire les interactions des composants graphiques avec les autres composants graphiques ou le NF d'une application.

L'activation représente le lien entre la structure d'une UI et les fonctionnalités. Concrètement, dans le cadre d'une architecture MVC, cette primitive permet de représenter un appel de méthode du contrôleur par un élément de la vue.

5.2.1.1 Exemple

Les primitives d'interactions en entrée caractérisent les actions possibles des utilisateurs sur le(s) contenu(s), la taille, la position, l'orientation des composants graphiques d'une UI. Elles sont des interactions abstraites indépendantes des instruments d'interactions, elles sont dérivées en interactions concrètes sur les composants graphiques. En considérant l'artefact d'une UI de l'application CBA (cf chapitre 3), la liste d'images permet aux dessinateurs de BD de sélectionner des images prédéfinies et de les ajouter à la BD en cours de réalisation. Les éléments de cette liste peuvent être ajoutés au canevas de dessin par une interaction de glisser-déposer (drag & drop). Les primitives d'interactions en entrée de la liste d'images sont :

13. Widget Move, Widget Rotation, Widget Resize, Widget Selection et Navigation

- **Navigation et Widget Selection** pour accéder à chaque élément de la liste
- **Data Display** pour afficher chaque image de la liste
- **Data Selection, Data Move Out et Activation** pour sélectionner et pour effectuer le drag d'une image de la liste.

Les primitives d'interactions en entrée du canevas sont :

- **Navigation et Widget Selection** pour accéder à chaque objet du canevas
- **Data Selection, Data Move In et Activation** pour effectuer le drop d'un objet sur le canevas.
- **Data Selection, Data Edition et Activation** pour sélectionner et modifier les propriétés d'un objet du canevas. Par exemple pour le redimensionnement d'une image, il faut sélectionner l'objet représentant l'image, modifier la taille et faire appel au NF pour valider la modification.

Le tableau 5.1 présente les primitives d'interactions en entrée de la liste d'images, de la liste déroulante et du canevas.

Nous remarquons que les primitives d'interactions en entrée permettent de caractériser les actions des utilisateurs sur un contenant (liste ou canevas) et sur les éléments contenus (images, objets des canevas). La structure des composants graphiques au niveau abstrait doit tenir compte à la fois des contenants et des contenus.

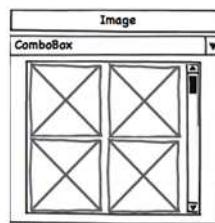


FIGURE 5.1 – Un artéfact d'une UI

Composants graphiques	Primitives d'interactions en entrée
Liste d'images	Widget Selection, Navigation, Data Display, Data Selection, Data Move Out, Activation
Canevas	Widget Selection, Navigation, Data Selection, Data Edition, Data Move In, Activation

TABLE 5.1 – Exemples de primitives d'interactions en entrée

5.2.2 Les primitives d'interactions en sortie

Le modèle d'interactions instrumentales présenté à la section 3.1 caractérise les interactions en sortie en deux types d'interactions atomiques. D'une part, les réponses correspondent aux interactions du NF sur les données d'applications ou les propriétés graphiques des composants graphiques.

D'autre part, les feedbacks et les réactions des éléments d'une UI correspondent aux comportements des composants graphiques. Les feedbacks et les réactions sont les effets visibles d'une interaction, par exemple l'apparition d'un pop up.

Dans cette section nous caractérisons les réponses sur le(s) contenu(s) et les propriétés graphiques d'un composant graphique. Nous avons identifié deux primitives d'interactions en sortie : la primitive d'interactions **Data Display** et la primitive d'interactions **Widget Display**.

Primitive d'interactions 11 : Data Display

Elle permet d'exprimer l'affichage des données d'application par un composant graphique.

Les données d'application sont modifiées par les actions utilisateurs ou par le NF. Cette primitive exprime la modification et l'affichage des données d'application par une réponse du NF.

Primitive d'interactions 12 : Widget Display

Elle exprime les réponses du NF ou d'autres composants graphiques sur l'aspect visuel d'un composant graphique.

Cette primitive d'interactions exprime les réponses du NF ou d'autres composants graphiques qui modifient les propriétés graphiques d'un composant graphique. Par exemple l'affichage d'une boîte de dialogue ou d'une autre fenêtre après une action utilisateur est une réponse, la boîte de dialogue ou la fenêtre concernée par cette réponse doit décrire la primitive d'interactions **Widget Display**.

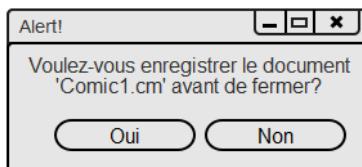


FIGURE 5.2 – Boîte de dialogue

5.2.2.1 Exemple

Les primitives d'interactions en sortie caractérisent les réponses du NF ou d'autres composants graphiques.

Considérons par exemple la boîte de dialogue de la figure 5.2 de l'application CBA : elle est affichée après une demande de fermeture d'un document. La boîte de dialogue aura la primitive d'interactions **Widget Display** telle que décrite dans le tableau 5.2. Le message est affiché dans un label qui obtient le nom du fichier à partir du NF, ce label aura donc la primitive d'interactions **Data Display**.

5.3 Modèles de composants graphiques

Cette section a pour objectif d'expliquer comment les primitives d'interactions (PI) sont utilisées pour caractériser les composants graphiques dans le but de décrire des mécanismes d'équivalences et d'adaptation pour la migration des UI. Pour que ces mécanismes soient **réutilisables** et **flexibles**

Composants graphiques	Primitives d'interactions en sortie
Label Message	Data Display
Boîte de dialogue	Widget Display

TABLE 5.2 – Exemple de primitives d'interactions en sortie

nous allons caractériser aussi les composants graphiques indépendamment des plateformes à l'aide de modèles abstraits.

Les composants graphiques appartiennent à des bibliothèques graphiques (ou boîtes à outils) et sont utilisés sous la forme d'instances pour décrire des UI. Les **composants graphiques d'instance** n'implémentent pas systématiquement toutes les interactions possibles de leur type. Nous considérons la figure 5.3 qui illustre le lien entre des instances et des types de composants graphiques. Par exemple un composant graphique de type *ListBox* définit les PI **Data Move Out** et **Data Move In** car il supporte l'interaction glisser-déposer (drag & drop) de manière intrinsèque. Cependant de manière effective, il est possible de l'instancier sans implémenter ces PI dans le cas d'une liste qui ne permet pas d'effectuer un glisser-déposer de son contenu.

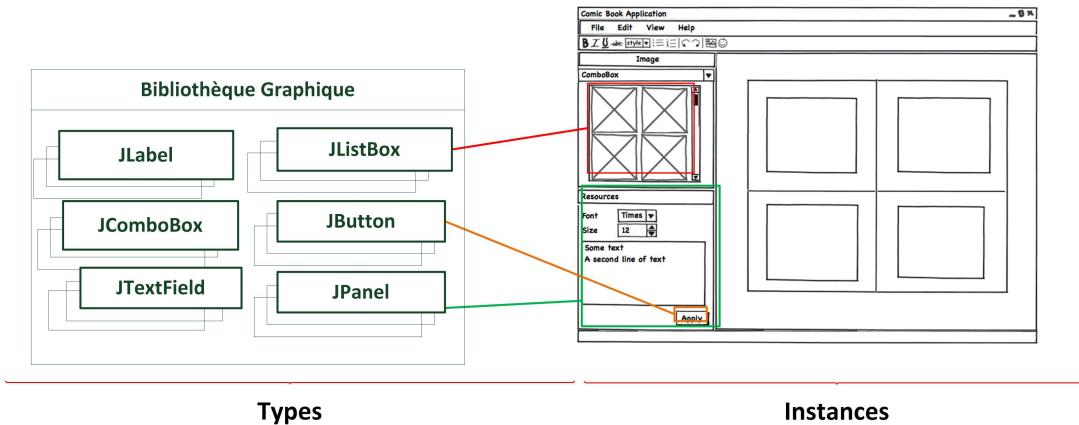


FIGURE 5.3 – Types et instances de composants graphiques

Les PI présentées ci-dessus (cf. section 5.2) sont utilisées pour décrire l'ensemble des interactions possibles sur les composants graphiques. Elles font partie du modèle abstrait des UI. Les aspects structurels de l'UI tels que les types et les cardinalités de données d'application d'une part, les liens de contenance et les groupements des éléments graphiques d'autre part, sont décrits dans un modèle de structure. Nous caractérisons dans cette section deux catégories de PI en fonction des types et des instances des composants graphiques. Une UI finale est décrite par les instances des composants graphiques et les types de composants graphiques sont contenus dans des bibliothèques graphiques. Chaque type de composants graphiques définit des PI intrinsèques. Par ailleurs, les éléments d'une UI finale sont modélisés à l'aide d'un modèle de structure qui décrit les types de données, leur cardinalité et le regroupement des éléments graphiques dont chaque instance implémente des PI effectives.

Dans le cadre de la migration des UI, cette différentiation entre **PI intrinsèques** et **PI effectives**

permet de ne conserver pendant la migration que les interactions d'un composant graphique réellement utilisées par l'application.

Les PI effectives peuvent être identifiées pour chaque bibliothèque graphique suivant deux approches. La première approche consiste à décrire manuellement les PI effectives de chaque composant graphique en se basant sur les définitions des PI. Cette approche implique une identification exhaustive des PI effectives de chaque composant graphique d'une bibliothèque graphique. La deuxième approche consiste d'abord à étiqueter les événements et les propriétés¹⁴ des composants graphiques, puis à décrire des règles basées sur les étiquettes de chaque composant graphique.

Nous avons opté pour une identification des PI effectives basées sur l'étiquetage en identifiant le comportement de chaque événement par rapport aux contenus ou aux propriétés graphiques des composants graphiques. En effet, cette seconde approche peut être reproduite en décrivant les correspondances de chaque étiquette dans une bibliothèque graphique et en appliquant les règles d'identifications. Le nombre d'étiquettes caractérisant les événements et les propriétés est inférieur aux nombres de composants graphiques car il existe des événements communs à plusieurs composants graphiques. Par exemple l'événement *TextChanged* est défini pour tout composant ayant une propriété *Text* en XAML. Nous présentons à la section 5.3.1.3 les étiquettes permettant de caractériser les événements des composants graphiques. L'étiquetage est une tâche manuelle qui nécessite une connaissance de la syntaxe des événements des composants graphiques des différentes bibliothèques graphiques. Elle est effectuée de manière unique pour les bibliothèques graphiques des plateformes source et cible.

Les PI effectives sont ensuite identifiées à partir des UI à l'aide des règles d'identifications spécifiques à chaque bibliothèque graphique en fonction des événements effectivement implémentés. Nous proposons à la section 5.3.2.6 les règles d'identification des PI effectives qui sont utilisées par notre solution de migration des UI vers les tables interactives. Ces règles sont décrites et validées pour la bibliothèque graphique XAML.

La suite de cette section présente d'abord un modèle de types de composants graphiques ainsi que les règles qui permet d'identifier les **PI intrinsèques** à chaque composant graphique. Nous présentons ensuite un modèle de composants graphiques d'instances pour exprimer la structure d'une UI ainsi que les règles d'identifications des **PI effectives**.

5.3.1 Un modèle de types de composants graphiques

Les composants graphiques appartiennent à des bibliothèques graphiques spécifiques (JavaSwing, DiamondSpin, etc). Ce sont des éléments qui définissent des interactions, des comportements et des données spécifiques [Cre01]. Nous appelons dans cette thèse "*Widget*", les types de composants graphiques au niveau modèle qui représentent les éléments d'une UI d'une bibliothèque graphique.

De manière transversale et indépendamment des bibliothèques graphiques, les *Widgets* permettent de caractériser par les différents aspects d'une UI : les interactions entre les utilisateurs et l'application, la structure (type et cardinalité des données), le positionnement dans une UI et le style (couleur, police, taille, etc.). Nous proposons un modèle de types de composants graphiques qui décrit uniquement la **structure** des éléments d'une bibliothèque graphique et les **comportements** possibles des composants graphiques à la suite d'interactions (des utilisateurs, du NF ou d'autres composants graphiques). En effet, dans le cadre d'un processus semi automatique (cf. section 4.3.2) de migration des UI, il n'est pas indispensable de décrire au niveau du modèle tous les aspects de l'UI à migrer car les autres aspects sont ajoutés manuellement.

L'objectif du modèle de types de composants graphiques décrit à la figure 5.4 est de représenter l'ensemble des éléments d'une bibliothèque graphique avec ses caractéristiques structurelles (type

14. Un événement décrit un comportement d'un composant graphique après une interaction

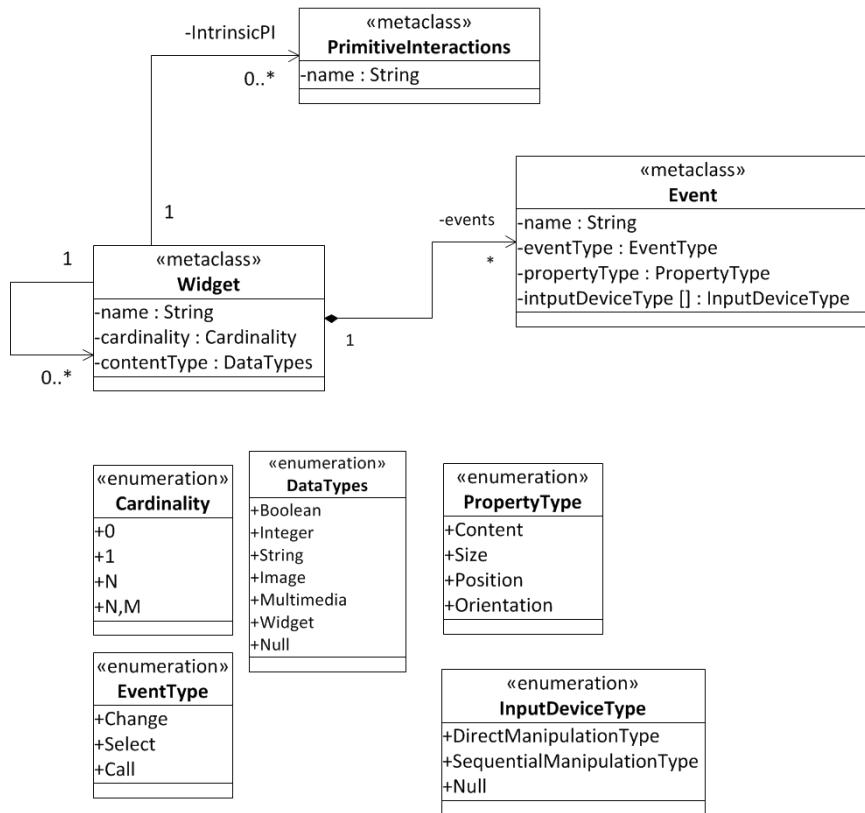


FIGURE 5.4 – Modèle de types de composants graphiques

et cardinalité de données) et ses primitives d’interactions intrinsèques pour décrire un mécanisme d’équivalences réutilisables. La réutilisabilité des équivalences est assurée par un modèle de structure et les PI indépendantes des dispositifs d’interactions et des bibliothèques graphiques.

Le modèle de la figure 5.4 ci-dessus décrit les classes *Widget*, *PrimitiveInteractions* et *Event*.

5.3.1.1 Widget

La classe *Widget* est identifiée par son nom (*name*) et le nombre d’éléments qu’elle peut contenir (*cardinality*). Le champ *name* permet de l’identifier de manière unique dans une bibliothèque graphique. La cardinalité permet quant à elle de préciser le nombre de données ou de *Widget* maximal qu’elle peut contenir.

L’attribut *contentType* précise le type de données d’un *Widget*. Les données d’un *Widget* sont des données de l’application et/ou des données graphiques. Par exemple : en XAML, les données d’un *TextBox* sont des données de l’application pouvant être modifiées par l’utilisateur tandis que les données d’un *label* sont des données graphiques car elles ne sont pas destinées à être modifiées par l’utilisateur.

Une bibliothèque graphique est constituée d’un ensemble d’instances de la classe *Widget* qui représente les types de composants graphiques.

5.3.1.2 *PrimitiveInteractions*

La classe *PrimitiveInteractions* du modèle de types de composants graphiques décrit les PI intrinsèques d'un Widget à travers le rôle *IntrinsicPI*. Elle est caractérisée par l'attribut *name* qui exprime le nom d'une PI.

Les PI intrinsèques d'un Widget sont exprimées par des instances de la classe *PrimitiveInteractions* avec des noms différents.

5.3.1.3 *Event*

La classe *Event* caractérise le comportement des propriétés structurelles des Widgets par rapport aux interactions sur les propriétés graphiques et sur les contenus. Ces comportements peuvent être des appels de méthodes, des modifications ou sélections des propriétés graphiques des composants ou des données d'application. L'attribut *name* précise le nom de l'événement du composant graphique dans sa bibliothèque graphique.

Étiquettes des Events L'attribut *inputDeviceType* caractérise les types des dispositifs d'interactions en entrée qui peuvent provoquer un comportement d'un Widget. Le type *DirectManipulationType* correspond aux dispositifs de manipulations directes tels que les souris, les écrans tactiles, les objets tangibles, etc. Le type *SequentialManipulationType* correspond aux dispositifs de manipulations séquentielles tels que les claviers physique et virtuel. Un comportement peut être déclenché par plusieurs dispositifs d'interactions en entrée. Dans le cas où le comportement est déclenché par une interaction en sortie, le type de dispositif d'interactions en entrée est *Null*.

L'attribut *eventType* précise le type de comportement d'un événement par une étiquette. Ces étiquettes représentent les différents comportements des Widgets :

- La modification des propriétés si *eventType* = **Change**.
- La sélection de données d'application ou d'un composant graphique si *eventType* = **Select**.
- L'appel de méthodes si *eventType* = **Call**.

Dans le but spécifier si une étiquette concerne le contenu ou les propriétés graphiques, l'attribut *propertyType* précise sur quel type de propriété porte un comportement de l'événement concerné de manière suivante :

- si *propertyType* = **Content** alors le comportement porte sur les données d'application ou les données graphiques,
- si *propertyType* = **Size** alors le comportement porte sur une propriété précisant la taille de l'élément graphique,
- si *propertyType* = **Position** alors le comportement porte sur une propriété précisant la position de l'élément graphique,
- si *propertyType* = **Orientation** alors le comportement porte sur une propriété précisant l'orientation de l'élément graphique.

Un événement peut avoir plusieurs types. Par exemple une liste déroulante permet de sélectionner un item de la liste et aussi de déclencher l'appel d'une méthode. Cette liste aura un attribut *Event* avec à la fois *eventType* = **Call** et *eventType* = **Select**.

Les comportements des Widgets sont implémentés de différentes manières par les bibliothèques graphiques. Par exemple :

- En JavaSwing, les comportements de type **Change** sont définis pour un composant graphique si la propriété *isEditable* est définie pour un composant graphique. Cette propriété permet de dire qu'un composant graphique définit de manière intrinsèque un événement pour changer son contenu.

- En XAML, les comportements de type **Select** sont définis si la méthode *SelectedItem* est définie pour un composant graphique. Cette méthode permet la sélection d'un contenu.
- En javaSwing, les comportements de type **Call** sont définis si la méthode *addActionListener* est définie pour un composant graphique. Elle permet de dire qu'un composant graphique peut décrire un handler.

5.3.1.4 Remarques sur le modèle de types de composants graphiques

Pour chaque bibliothèque graphique, nous proposons de construire une table qui décrit les correspondances entre chaque type d'événements et leur représentation spécifique. Cette table permet d'identifier les instances des *Widgets*. Des exemples de cette table sont décrits de manière exhaustive pour les bibliothèques graphiques XAML et XAML Surface dans l'annexe A.1.

Dans le modèle que nous proposons, nous avons identifié un ensemble de types de données d'application ou données graphiques des composants graphiques. Les données prises en compte par ce modèle de composant graphique sont de type booléen (*Boolean*), entier (*Integer*), chaîne de caractères (*String*), image (*Image*), son ou vidéo (*MediaElement*), classes indépendantes de la bibliothèque graphique (*Object*) ou composants graphiques (*Widget*). L'identification des types est faite à l'aide d'une seconde table de correspondances entre les types spécifiques aux bibliothèques graphiques et les types décrits dans notre modèle.

Les mécanismes qui permettent de décrire les instances du modèle de types de composants graphiques sont décrits à l'annexe A.1 pour les bibliothèques graphiques XAML et XAML Surface. Il existe autant d'instances de ce modèle que de bibliothèques graphiques.

5.3.1.5 Identification des primitives d'interactions intrinsèques

En nous basant sur le modèle de types de composants graphiques proposé à la figure 5.4, nous caractérisons les différentes PI en nous basant aussi sur leur définition et sur les types de comportements des *Widgets* après une interaction en entrée (action) ou en sortie (réponse).

Dans cette section, nous présentons les règles d'identification des PI intrinsèques en fonction du modèle de types de composant présenté à la figure 5.4. Les règles précisent les caractéristiques correspondant à chaque PI intrinsèque.

Règle 1 : Widget Selection & Navigation

Les PI Widget Selection et Navigation sont définies de manière intrinsèque pour tous les Widgets qui sont accessibles à l'aide d'un dispositif d'interactions directes (Widget Selection) ou indirectes (Navigation).

$$\begin{aligned}
 \forall w \in \{Widget\}, \exists pi \in w.IntrinsicPI, \\
 \wedge \\
 \exists event \in w.events, \\
 DirectManipulationType \in event.inputDeviceType \Rightarrow pi.name = 'WidgetSelection' \\
 \wedge \\
 SequentialManipulationType \in event.inputDeviceType \Rightarrow pi.name = 'Navigation'
 \end{aligned}$$

Règle 2 : Widget Resize

La PI Widget Resize est définie de manière intrinsèque pour tous les Widgets qui ont un comportement permettant de changer sa taille.

$$\forall w \in \{Widget\}, \exists pi \in w.IntrinsicPI,$$

$$\left(\begin{array}{l} \exists event \in Widget.events, \\ event.eventType = Change \\ \wedge \\ event.propertyType = Size \end{array} \right) \Rightarrow pi.name = 'WidgetResize'$$

Règle 3 : Widget Move

La PI Widget Move est définie de manière intrinsèque pour tous les Widgets qui ont un comportement permettant de changer sa position.

$$\forall w \in \{Widget\}, \exists pi \in w.IntrinsicPI,$$

$$\left(\begin{array}{l} \exists event \in Widget.events, \\ event.eventType = Change \\ \wedge \\ event.propertyType = Position \end{array} \right) \Rightarrow pi.name = 'WidgetMove'$$

Règle 4 : Widget Rotation

La PI Widget Rotation est définie de manière intrinsèque pour tous les Widgets qui ont un comportement permettant de changer son orientation.

$$\forall w \in \{Widget\}, \exists pi \in w.IntrinsicPI,$$

$$\left(\begin{array}{l} \exists event \in Widget.events, \\ event.eventType = Change \\ \wedge \\ event.propertyType = Orientation \end{array} \right) \Rightarrow pi.name = 'WidgetRotation'$$

Règle 5 : Widget Display

La PI Widget Display est définie de manière intrinsèque pour tous les Widgets, car tout composant graphique peut être visible sauf indication contraire du concepteur pour une instance.

$$\forall w \in \{Widget\}, \exists pi \in w.IntrinsicPI,$$

$$pi.name \in \{'WidgetDisplay'\}$$

Règle 6 : Data Edition

La PI Data Edition est définie de manière intrinsèque pour tous les Widgets qui ont un comportement

ment qui permet de changer des propriétés de contenu de type entier ou chaîne de caractères.

$$\forall w \in \{Widget\}, \exists pi \in w.IntrinsicPI,$$

$$\left(\begin{array}{l} w.contentType \in \{Integer, String\} \\ \exists event \in Widget.events, \\ \quad \wedge \\ \quad event.eventType = Change \\ \quad \wedge \\ \quad event.propertyType = Content \end{array} \right) \Rightarrow pi.name = 'DataEdition'$$

Règle 7 : Data Selection

La PI Data Selection est définie de manière intrinsèque pour tous les Widgets qui ont un comportement qui permet la sélection de son contenu.

$$\forall w \in \{Widget\}, \exists pi \in w.IntrinsicPI,$$

$$\left(\begin{array}{l} w.contentType \notin \{Null, Widget\} \\ \exists event \in Widget.events, \\ \quad \wedge \\ \quad event.eventType = Select \\ \quad \wedge \\ \quad event.propertyType = Content \end{array} \right) \Rightarrow pi.name = 'DataSelection'$$

Règle 8 : Data Move In et Data Move Out

Les PI Data Move In et Data Move Out sont définies de manière intrinsèque pour tous les Widgets qui ont un comportement qui permet la sélection et le changement des contenus.

$$\forall w \in \{Widget\}, \exists pi \in w.IntrinsicPI,$$

$$\left(\begin{array}{l} w.contentType \notin \{Null\} \\ \exists event \in Widget.events, \\ \quad \wedge \\ \quad event.eventType = Change \\ \quad \wedge \\ \quad event.eventType = Select \wedge \\ \quad event.propertyType = Content \end{array} \right) \Rightarrow pi.name \in \{'DataMoveIn', 'DataMoveOut'\}$$

Règle 9 : Data Display

La PI Data Display est définie de manière intrinsèque pour tous les Widgets qui n'ont pas de contenu nul.

$$\forall w \in \{Widget\}, \exists pi \in w.IntrinsicPI,$$

$$w.contentProperty \notin \{Null\} \Rightarrow pi.name \in \{'DataDisplay'\}$$

La PI est définie de manière intrinsèque pour tous les Widgets qui ont des événements de type *Call*.

Règle 10 : Activation

$$\begin{aligned} \forall w \in \{\text{Widget}\}, \exists pi \in w.\text{IntrinsicPI}, \\ \left(\begin{array}{l} event \in w.\text{events} \\ event.eventType = \text{Call} \end{array} \right) \Rightarrow pi.name \in \{'\text{Activation}'\} \end{aligned}$$

Remarque Nous avons validé les règles d'identification des PI proposées dans cette section pour les bibliothèques graphiques XAML et XAML Surface (cf. l'annexe A.3).

5.3.1.6 Synthèse

Le modèle de types de composants graphiques décrit à la section 5.3.1 permet de représenter les composants graphiques d'une bibliothèque graphique et les primitives d'interactions intrinsèques associées. La représentation des éléments d'une bibliothèque graphique dans ce modèle se fait à l'aide de deux tables de correspondances. La première décrit les correspondances entre les types de propriétés et leur représentation dans les bibliothèques graphiques (cf annexe 8.2.2). La seconde décrit les correspondances entre les types d'événements et leur représentation dans les bibliothèques graphiques.

Les primitives d'interactions intrinsèques sont identifiées à l'aide des règles se basant sur le modèle des composants graphiques types. Par ailleurs, l'identification des PI intrinsèques des *Widgets* d'une bibliothèque de manière exhaustive à l'aide d'une table de correspondances est possible. Cependant cette approche manuelle dépend de l'interprétation des PI par les personnes en charge de la mise en place et elle nécessite le parcours de tous les *Widgets* d'une bibliothèque graphique. L'approche que nous proposons, basée sur les règles d'identification, utilise aussi des tables de correspondances mais elle est facilement réutilisable car les tailles des tables de correspondances sont inférieures au nombre de *Widgets* d'une bibliothèque graphique. Dans le cadre de notre approche de migration des UI vers les tables interactives, nous considérons que ces tables de correspondances sont décrites pour chaque bibliothèque graphique pendant la mise en œuvre de la solution et fournies avec notre processus de migration des UI.

5.3.2 Un modèle d'instance d'une UI

La structure des instances des UI finales à migrer peut être décrite par différents formats (XML, Archives Java, etc.). Cependant tous ces formats peuvent être représentés par un arbre dont la racine est une fenêtre d'une UI et les feuilles, les différents composants graphiques élémentaires.

Comme pour le modèle de types de composants (cf. section 5.3.1), nous proposons un modèle d'instance d'une UI qui décrit sa structure et ses interactions effectives. En effet nous avons pour objectif de décrire un processus semi automatique de migration des UI qui prend en compte l'adaptation automatique des aspects structurels et les équivalences des instruments d'interactions. Dans cette optique, nous considérons qu'un modèle de CUI capable de décrire les liens de contenance, la cardinalité et les types de données d'une UI peut être utilisé. Le but d'un modèle de structure pour le processus de migration est de préserver les **données graphiques** et d'adapter la **structure** de l'UI de départ.

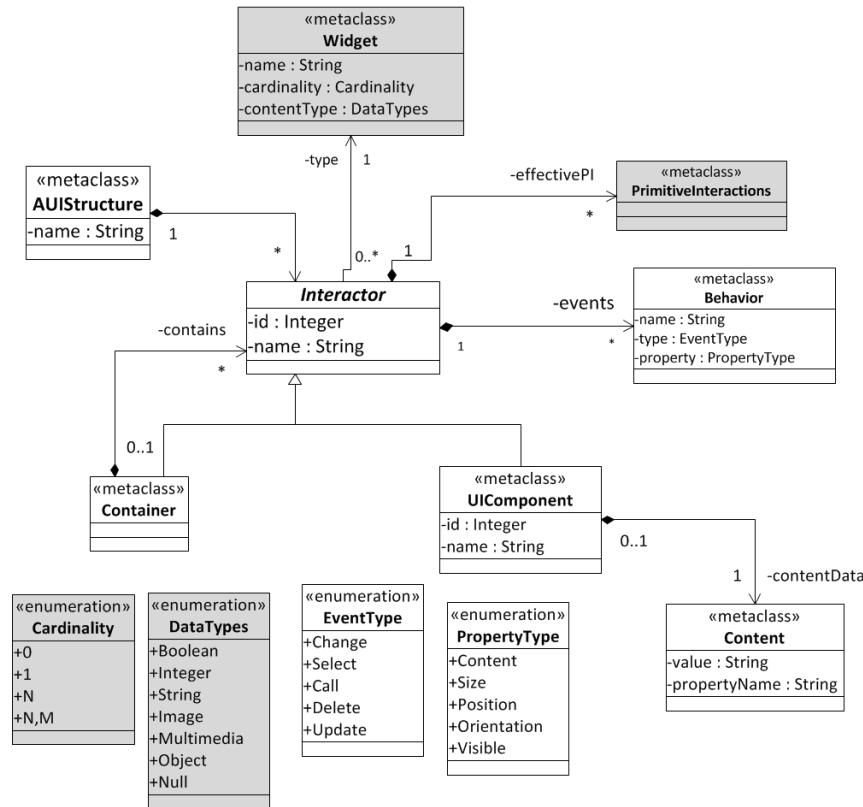


FIGURE 5.5 – Modèle de structure d’instance d’une UI

Le modèle de structure d’une UI de la figure 5.5 que nous utilisons dans notre contexte comporte des *Interactors*. Dans cette thèse nous appelons "*Interactor*" les instances abstraites de composants graphiques, c'est une représentation au niveau modèle des instances de *Widget*.

5.3.2.1 *Interactor*

Un *Interactor* est caractérisé par un identifiant unique pour une UI et un nom qui correspond à celui du composant graphique de l’UI à migrer. Un *Interactor* préserve les valeurs des données structurelles de l’UI à migrer par la classe *Content*, par exemple les étiquettes des labels, des boutons, des images, etc. Un interacteur est relié à la classe *Widget* par une relation de type. Nous distinguons deux types d’instances de *Widget* :

- **Container** représentant l’ensemble des composants graphiques pouvant contenir d’autres instances de *Widget*, les *Containers* sont des nœuds de la structure arborescente.
- **UIComponent** représentant l’ensemble des composants graphiques ne pouvant pas en contenir d’autres. Ils sont identifiés à partir des *Widgets* dont les contenus ne sont pas d’autres *Widgets*. Les *UIComponents* sont les feuilles de la structure arborescente.

5.3.2.2 *AUIStructure*

La classe caractérise l’ensemble des nœuds racines de la structure arborescente de l’UI. La structure des UI à migrer est abstraite à l’aide des éléments décrits par le modèle proposé à la figure 5.5.

5.3.2.3 *ImplementedEvent*

La classe caractérise les comportements réellement implémentés par les *Interactors* au niveau modèle. Elle est caractérisée par les attributs *name*, *type* et *property*.

- L’attribut *name* représente le nom d’un événement implémenté.
- L’attribut *type* représente le comportement effectivement implémenté.
- L’attribut *property* représente la propriété affectée par le comportement implémenté. Les valeurs de cette attribut sont *Size*, *Content*, *Position*, *Orientation*, *WidgetStructure* et *Visible*¹⁵.
- L’attribut *inputDeviceType* correspond à l’attribut de la classe *Event* du modèle des *Widgets* (cf figure 5.4).

Étiquettes des événements implémentés Les comportements effectivement implémentés des *Interactors* sont identifiés à partir de l’UI finale des applications à migrer.

Les comportements de type **Change** sont implémentés pour une instance de la bibliothèque JavaSwing par exemple si la propriété *isEditable* = *True*, ce qui permet de dire qu’un composant graphique définit de manière effective un comportement pour changer son contenu.

Les comportements de type **Select** sont implémentés pour une instance de XAML par exemple si la méthode *SelectedItem* est définie et la propriété *isEnabled* = *True*.

Les comportements de type **Call** sont implémentés pour une instance de la bibliothèque JavaSwing par exemple si un *ActionListener* est défini pour cette instance.

Les comportements de type **Delete** sont implémentés pour une instance ou pour les données d’application d’une instance. Dans une bibliothèque graphique comme XAML par exemple ce comportement est définie de manière effective pour un composant qui implémente un Handler à l’événement *DragEnter*.

Les comportements de type **Update** sont implémentés pour une instance. Ils ont des données d’application modifiées par le NF ou par d’autres composants graphiques. Dans la bibliothèque JavaSwing par exemple si une propriété de contenu est modifiée dans le code d’un listener alors ce comportement est implémenté par l’instance du composant graphique.

Pour chaque bibliothèque graphique une table qui décrit les correspondances entre les types d’événements et leurs représentations spécifiques, permet d’instancier les *Interactors* et leurs types d’événements. Nous avons décrit de manière exhaustive cette table pour les bibliothèques graphiques XAML et XAML Surface à l’annexe A.2.

5.3.2.4 *Content*

La classe *Content* du modèle de structure présenté à la figure 5.5 permet de préserver les données graphiques et les données d’application. Le processus de migration dans notre cas permet un changement de la modalité d’interactions mais les données d’une UI doivent être préservées pour conserver la sémantique de l’UI source. Pour ce faire, les types de données (*dataType*) d’une UI source, leur cardinalité (*cardinality*) et leur(s) valeur(s) (*value*) sont décrits dans notre modèle pour permettre leur préservation et éventuellement leur adaptation à la nouvelle plateforme.

5.3.2.5 Types de Container

Les containers peuvent être catégorisés en fonction des interacteurs qu’ils contiennent. Nous caractérisons les types de container : **Homogène**, **Hétérogène**, **Récursif** et **Racine**. Le tableau 5.3 décrit

15. Cette propriété permet d’implémenter le comportement de changement de visibilité d’une instance de Widget

les types de container en fonction des types de composants graphiques qu'ils contiennent. En parcourant la structure arborescente du modèle d'instance d'une UI, les types de container sont identifiés de manière récursive en identifiant d'abord les types des interacteurs fils. Un container peut être de plusieurs types à la fois.

Ces quatre types de containers caractérisent les différents regroupements de composants graphiques d'une UI à migrer. Nous avons remarqué à la section 3.2.3 qui identifie les guidelines pour la migration des UI vers les tables interactives que la plupart des transformations des UI de départ en UI collaboratives et tactiles porte sur les regroupements des composants graphiques. Les catégories identifiées dans cette section constituent des éléments basiques pour appliquer les guidelines d'une UI collaboratives et tangibles pendant l'adaptation de la structure des UI sources.

Container Racine C'est l'élément racine d'une structure d'une UI. Il peut contenir des containers et des interacteurs simples. En considérant la figure 5.6, le container bleu est une illustration d'un *Container* de type *Racine*.

Container Récursif Il ne contient que des *Containers*. Ce type permet d'identifier un regroupement de plusieurs containers sur lequel on pourra appliquer des transformations. Un container de type *Récursif* ne peut donc pas contenir des interacteurs de types *UIComponent*. En considérant la figure 5.6, le container vert est une illustration d'un *Container* de type *Récursif*.

Container Homogène Il contient uniquement des *UIComponents* dont les données sont de même type et de même cardinalité. Un container de ce type permet de définir un groupe de composants graphiques de même type (tel qu'une liste, un tableau, un menu, etc.). Les containers jaune et orange de la figure 5.6 sont une illustration d'un *Container* de type *Homogène*.

Container Hétérogène C'est un container pouvant contenir à la fois des *UIComponents* et des *Containers*. Ce type permet d'identifier les groupes de composants graphiques contenant des données de types différents. Les containers de types *Hétérogène* peuvent représenter des formulaires qui seront adaptés aux tables interactives pour les rendre facilement accessibles ou pour les afficher par un objet tangible. En considérant la figure 5.6, le container rouge est une illustration d'un *Container* de type *Hétérogène*.

Contenu	Types de container
{ <i>UIComponent</i> }	Homogène : données du même type Hétérogène : données de type différent
{ <i>UIComponent</i> } \cup { <i>Container</i> }	Hétérogène : si a un parent
{ <i>Container</i> }	Récursif : si contient des containers

TABLE 5.3 – Types de container

5.3.2.6 Identification des primitives d'interactions effectives

Les primitives d'interactions réellement utilisées par chaque interacteur sont identifiées à partir de la structure de l'UI de départ. Dans le but de préserver l'assemblage des composants graphiques de

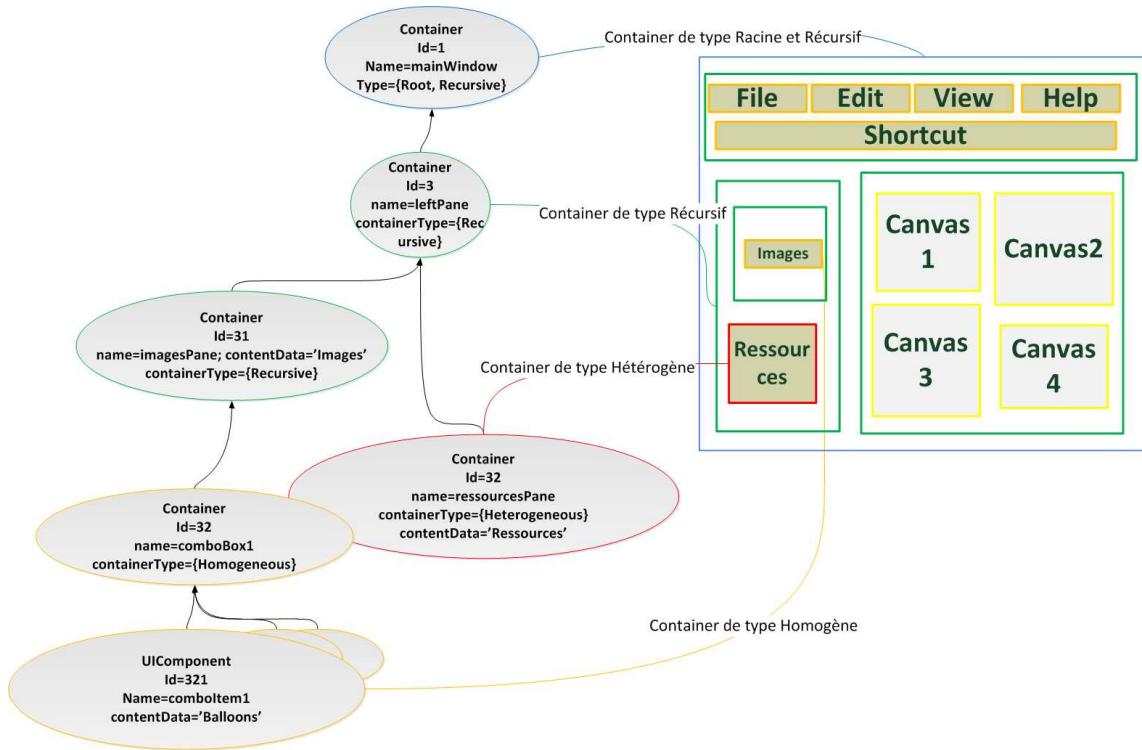


FIGURE 5.6 – Illustration des types de container

l’UI à migrer, nous extrayons au moyen d’interacteurs cette structure arborescente à l’aide du modèle de la figure 5.5. Ce modèle décrit la structure de l’UI comme un ensemble d’arbres dont les racines sont des fenêtres, les nœuds des interacteurs et les arcs la relation de contenance entre les interacteurs.

Dans cette section, nous présentons les règles d’identification des PI effectives en fonction du modèle de structure d’une UI présenté à la figure 5.5. Les règles précisent les caractéristiques correspondant à chaque PI effective.

Règle 11 : Widget Selection & Navigation

Les PI Widget Selection et Navigation sont définies de manière effective pour tous les interacteurs qui implémentent les comportements de sélection de leur structure.

$$\begin{aligned}
 & \forall interactor \in AUIStructure, \exists pi \in interactor.EffectivePI, \\
 & \quad \wedge \\
 & \quad \exists iEvt \in interactor.events, iEvt.type = Select \\
 & \quad \wedge \\
 & \quad iEvt.property = WidgetStructure \\
 & \quad \wedge \\
 & \quad pi.name = 'WidgetSelection' \Rightarrow \left(\begin{array}{c} \exists p \in interactor.type.IntrinsicPI, \\ p.name = 'WidgetSelection' \\ \wedge \\ DirectManipulationType \in iEvt.inputDevice \end{array} \right) \\
 & \quad \wedge \\
 & \quad pi.name = 'Navigation' \Rightarrow \left(\begin{array}{c} \exists p \in interactor.type.IntrinsicPI, \\ p.name = 'Navigation' \\ \wedge \\ SequentialManipulationType \in iEvt.inputDevice \end{array} \right)
 \end{aligned}$$

Règle 12 : Widget Resize

La PI Widget Resize est définie de manière effective pour les interacteurs qui implémentent les comportements de changement de taille.

$$\begin{aligned}
 & \forall interactor \in AUIStructure, \exists pi \in interactor.EffectivePI, \\
 & \quad \wedge \\
 & \quad \exists iEvt \in interactor.events, iEvt.type = Change \\
 & \quad \wedge \\
 & \quad iEvt.property = Size \\
 & \quad \wedge \\
 & \quad \left(\exists p \in interactor.type.IntrinsicPI, p.name = 'WidgetResize' \right) \Rightarrow pi.name = 'WidgetResize'
 \end{aligned}$$

Règle 13 : Widget Move

La PI Widget Move est définie de manière effective pour les interacteurs qui implémentent les

comportements de changement de position.

$$\begin{aligned}
 & \forall interactor \in AUIStructure, \\
 & \exists pi \in interactor.EffectivePI, \\
 & \quad \wedge \\
 & \exists iEvt \in interactor.events, \\
 & \quad iEvt.type = Change \\
 & \quad \wedge \\
 & \quad iEvt.property = Position \\
 & \quad \wedge \\
 & \left(\begin{array}{l} \exists p \in interactor.type.IntrinsicPI, \\ p.name = 'WidgetMove' \end{array} \right) \Rightarrow pi.name = 'WidgetMove'
 \end{aligned}$$

Règle 14 : Widget Rotation

La PI Widget Rotation est définie de manière effective pour les interacteurs qui implémentent les comportements de changement d'orientation.

$$\begin{aligned}
 & \forall interactor \in AUIStructure, \\
 & \exists pi \in interactor.EffectivePI, \\
 & \quad \wedge \\
 & \exists iEvt \in interactor.events, \\
 & \quad iEvt.type = Change \\
 & \quad \wedge \\
 & \quad iEvt.property = Orientation \\
 & \quad \wedge \\
 & \left(\begin{array}{l} \exists p \in interactor.type.IntrinsicPI, \\ p.name = 'WidgetRotation' \end{array} \right) \Rightarrow pi.name = 'WidgetRotation'
 \end{aligned}$$

Règle 15 : Widget Display

La PI Widget Display est définie de manière effective pour tous les interacteurs qui implémentent

les PI effectives **Widget Selection ou Navigation** et pour ceux qui sont visibles.

$$\begin{aligned}
 & \forall interactor \in AUIStructure, \exists pi \in interactor.EffectivePI, \\
 & \quad \wedge \\
 & \quad \exists iEvt \in interactor.events, iEvt.type = Select \\
 & \quad \wedge \\
 & \quad iEvt.property = WidgetStructure \\
 & \quad \vee \\
 & \quad iEvt.type = Change \wedge iEvt.property = Visible \\
 & \quad \wedge \\
 & \left(\exists p \in interactor.type.IntrinsicPI, \begin{array}{l} p.name = 'WidgetDisplay' \end{array} \right) \Rightarrow pi.name = 'WidgetDisplay'
 \end{aligned}$$

Règle 16 : Data Edition

La PI Data Edition est définie de manière effective pour tous les interacteurs qui ont des contenus et qui décrivent un comportement de changement de contenus de manière effective.

$$\left(\begin{array}{l} \exists iEvt \in interactor.events, iEvt.type = Change \\ \wedge \\ iEvt.property = Content \\ \wedge \\ interactor.ContentData! = Null \end{array} \right) \Rightarrow pi.name = 'DataEdition'$$

Règle 17 : Data Selection

La PI Data Selection est définie de manière effective pour tous les interacteurs qui ont un contenu et qui implémentent un comportement de sélection de ce contenu de manière effective.

$$\left(\begin{array}{l} \exists iEvt \in interactor.events, iEvt.type = Select \\ \wedge \\ iEvt.property = Content \\ \wedge \\ interactor.ContentData! = Null \end{array} \right) \Rightarrow pi.name = 'DataSelection'$$

Règle 18 : Data Move In

La PI Data Move In est définie de manière effective pour tous les interacteurs qui implémentent des comportements de sélection et de modification des contenus.

$$\forall interactor \in AUIStructure, \exists pi \in interactor.EffectivePI,$$

$$\left(\begin{array}{l} \exists iEvt \in interactor.events, \\ \quad iEvt.type = Select \\ \quad \wedge \\ \exists iEvt \in interactor.events, \\ \quad iEvt.type = Change \\ \quad \wedge \\ \quad iEvt.property = Content \\ \quad \wedge \\ \quad interactor.ContentData! = Null \end{array} \right) \Rightarrow pi.name = 'DataMoveIn'$$

Règle 19 : Data Move Out

Les PI Data Move Out est définie de manière effective pour tous les interacteurs qui implémentent des comportements de sélection et de suppression des contenus.

$$\forall interactor \in AUIStructure, \exists pi \in interactor.EffectivePI,$$

$$\left(\begin{array}{l} \exists iEvt \in interactor.events, \\ \quad iEvt.type = Select \\ \quad \wedge \\ \exists iEvt \in interactor.events, \\ \quad iEvt.type = Delete \\ \quad \wedge \\ \quad iEvt.property = Content \\ \quad \wedge \\ \quad interactor.ContentData! = Null \end{array} \right) \Rightarrow pi.name = 'DataMoveOut'$$

Règle 20 : Data Display

La PI Data Display est définie de manière effective pour tous les interacteurs qui ont un contenu ou dont la propriété de contenu est modifiée par une méthode.

$$\forall interactor \in AUIStructure, \exists pi \in interactor.EffectivePI,$$

$$\left(\begin{array}{l} \exists iEvt \in interactor.events, \\ \quad iEvt.type = Update \end{array} \right) \Rightarrow pi.name = 'DataDisplay'$$

Règle 21 : Activation

La PI Activation est définie de manière effective pour tous les interacteurs qui font appel à une méthode du contrôleur (dans une architecture MVC).

$$\forall interactor \in AUIStructure, \exists pi \in interactor.EffectivePI,$$

$$\left(\begin{array}{l} \exists iEvt \in interactor.events, \\ \quad iEvt.type = Call \end{array} \right) \Rightarrow pi.name = 'Activation'$$

Remarque Nous avons validé les règles d'identification proposées dans cette section pour les UI décrites à l'aide des bibliothèques graphiques XAML et XAML Surface (cf. l'annexe A.3.). Pour la validation, nous avons décrit le modèle d'instance et le modèle de types comme des méta modèles avec EMF [Fou13] et nous avons implémenté les différentes règles en Java.

5.3.2.7 Résumé

Une UI est exprimée au niveau abstrait par un modèle décrivant les primitives d'interactions effectives, les liens de contenances et les données des UI à migrer.

Le modèle de structure décrit dans cette section permet d'exprimer, sous la forme d'un arbre, l'instance de l'UI indépendamment des bibliothèques graphiques et des dispositifs d'interactions de départ pour la transformer par rapport aux guidelines des tables interactives.

5.3.3 Synthèse des modèles abstraits

Dans cette section nous avons présenté un modèle de composants graphiques et un modèle de structure pour une UI. Le modèle de composants graphiques décrit de manière abstraite les primitives d'interactions intrinsèques et la structure (les données d'application et les données graphiques et la cardinalité) des Widgets indépendants des bibliothèques graphiques. En effet, le modèle de la figure 5.4 décrit le contenu, la cardinalité, les comportements sur les propriétés (données, taille, position, orientation, etc.) et les interactions d'un composant graphique indépendamment de sa représentation par une boîte à outils donnée.

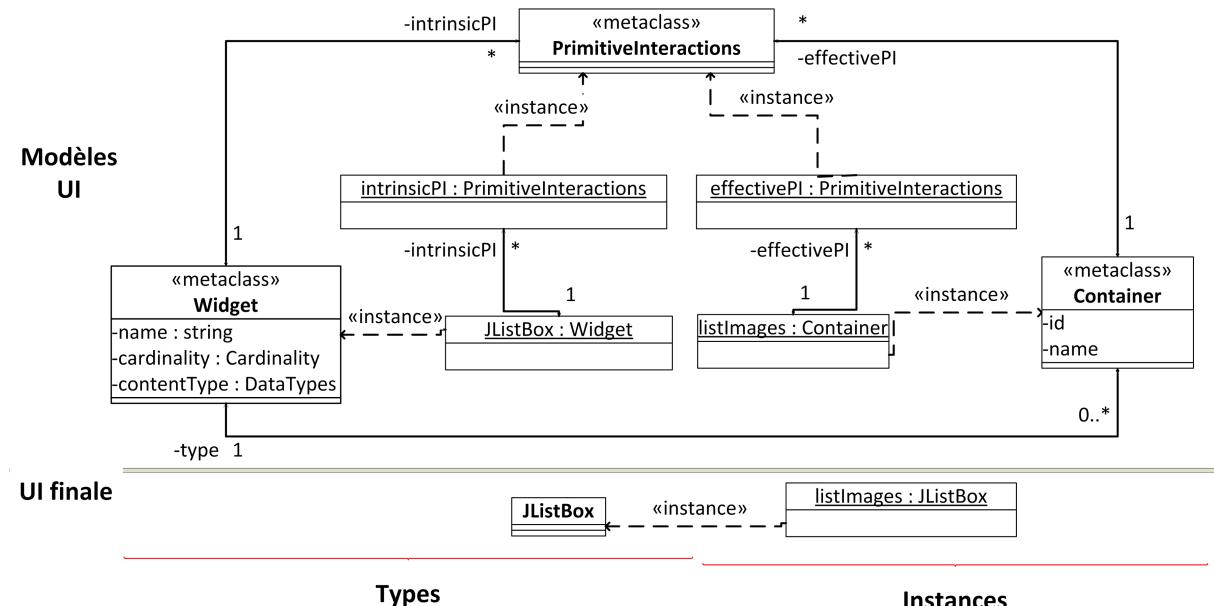


FIGURE 5.7 – Modèle abstrait et UI finale

Par ailleurs le modèle de structure (cf. figure 5.5) représente les liens de contenance et les données d'une UI à migrer. Ce modèle est certes composé d'instances de composants graphiques mais il ne décrit que les aspects structurels et les interactions d'une UI à migrer.

Les modèles de structure et d'interactions proposés dans ce chapitre permettent de décrire les types et les instances des UI. La figure 5.7 par exemple représente le type et l'instance d'une ListBox.

Dans le cadre de la migration, les PI effectives des instances permettent d'établir des équivalences avec les éléments de la plateforme cible.

Les sections 5.2 et 5.3 décrivent les aspects (d'interactions et de structures) des composants graphiques que nous prenons en compte dans notre approche de migration des UI semi automatique. Ces modèles permettent de décrire des mécanismes d'équivalences des instruments d'interactions qui passent par l'équivalence des composants graphiques basée sur les PI. Dans la section suivante nous présentons les opérateurs d'équivalences des composants graphiques basés sur les primitives d'interactions dans le but de décrire un mécanisme d'équivalences des instruments d'interactions.

5.4 Opérateurs d'équivalences des composants graphiques basés sur des PI

L'objectif des primitives d'interactions identifiées à la section 5.2 est de caractériser les composants graphiques indépendamment des dispositifs physiques et des bibliothèques graphiques. À la section 5.3, nous avons proposé une modélisation des composants graphiques (types et instances). Ces modèles nous permettent de caractériser les composants graphiques en nous basant sur leur contenu (type de données et cardinalité) et leurs primitives d'interactions (effectives et intrinsèques). L'utilisation des opérateurs par les mécanismes d'équivalences a pour objectif d'établir des correspondances à la volée entre les composants graphiques de la source et de la cible. Ce type d'équivalences accroît la flexibilité du processus de migration des UI car les équivalences ne sont pas définies de manière statique dans une table d'équivalences.

Les modèles de type et d'instance des composants graphiques décrits à la section 5.3 caractérisent la structure et les interactions. Les opérateurs que nous proposons dans cette section se basent sur les PI (effectives et intrinsèques), les types et les cardinalités des données.

La section 5.4.1 présente les opérateurs d'équivalences basés uniquement sur les PI et la section 5.4.2 présente ceux qui sont basés sur PI et les types de données.

5.4.1 Opérateurs d'équivalences basés sur des PI

Le premier enjeu pendant la migration des UI est d'avoir les interactions nécessaires pour l'utilisation de l'UI source sur la cible. Pour évaluer la sélection d'un composant graphique de la cible pendant le processus de migration, nous avons identifié trois opérateurs d'équivalences qui se basent sur les primitives d'interactions des composants graphiques à migrer.

Nous considérons que deux PI sont égales si elles ont le même nom.

$$\forall pi_1, pi_2 \in \{PrimitiveInteractions\},$$

$$pi_1 = pi_2 \Rightarrow pi_1.name = pi_2.name$$

5.4.1.1 Équivalence stricte : \equiv

Cet opérateur permet de retrouver sur la cible les composants graphiques qui ont les mêmes interactions que la source. Cet opérateur s'assure aussi que les opérandes ont les mêmes types de données et les mêmes cardinalités.

$$\forall int \in instanceOf(Interactor), \forall wid \in instanceOf(Widget),$$

$$int \equiv wid \Rightarrow \left\{ \begin{array}{l} int.contentData.cardinality = wid.cardinality \\ ((int.contentData! = Null \wedge \\ int.contentData.dataType = wid.ContentType) \\ \vee (int.contentData = Null \wedge wid.contentType = Null)) \\ \forall pi1 \in int.effectivePI, \\ \exists pi2 \in wid.IntrinsicPI, \\ (pi1 = pi2) \\ \wedge \\ card(int.effectivePI) = card(wid.IntrinsicPI) \end{array} \right.$$

Exemple d'équivalence stricte Pour une UI source décrite en JavaSwing qui instancie un JButton que l'on souhaite migrer vers une table Microsoft PixelSense, button1 est une instance de JButton et sbutton est une instance de SurfaceButton. Nous supposons que :

- *int* est une instance de *Interactor* qui représente *button1* au niveau modèle
- *wid* est une instance de *Widget* qui représente *sbutton* au niveau modèle

$$\begin{aligned} int \Leftrightarrow & \left\{ \begin{array}{l} id = 1 \\ name = button1 \\ intrinsicPI = \{Navigation, WidgetSelection, Activation\} \\ contentData = \{dataType = String, cardinality = 1, \\ value = 'Apply', propertyName = 'Text'\} \end{array} \right. \\ wid \Leftrightarrow & \left\{ \begin{array}{l} name = sbutton \\ effectivePI = \{Navigation, WidgetSelection, Activation\} \\ contentType = String, cardinality = 1 \end{array} \right. \\ \Rightarrow int \equiv wid \end{aligned}$$

5.4.1.2 Équivalence large : $\leq_{AdditionalPI}$

Cet opérateur permet de retrouver les composants graphiques de la cible qui ont toutes les primitives d'interactions du composant graphique de la source et qui proposent d'autres interactions additionnelles. Les primitives d'interactions supplémentaires¹⁶ sont précisées dans l'ensemble *AdditionalPI*. Cet opérateur permet par exemple de choisir des composants graphiques qui permettent de mieux respecter des guidelines (que l'équivalence stricte) sur la cible tout en conservant l'ensemble des interactions de la source.

$$\forall int \in instanceOf(Interactor), \forall wid \in instanceOf(Widget),$$

16. Ces PI sont préconisées par les guidelines de migration des UI

$$int \leqq_{AdditionalPI} wid \Rightarrow \left\{ \begin{array}{l} int.contentData.cardinality = wid.cardinality \\ ((int.contentData! = Null) \\ \quad \wedge \\ int.contentData.dataType = wid.ContentType) \\ \quad \vee \\ (int.contentData = Null \wedge wid.contentType = Null)) \\ \quad \forall pi1 \in int.effectivePI, \\ \quad \exists pi2 \in wid.IntrinsicPI, \\ \quad pi1 = pi2 \\ \quad \wedge \\ card(int.effectivePI) < card(wid.IntrinsicPI) \\ \quad \wedge \\ \exists pi \in wid.IntrinsicPI \setminus int.effectivePI \wedge pi \in AdditionalPI \end{array} \right.$$

Exemple d'équivalence large Pour une UI source décrite en JavaSwing, les instances des JFrame sont migrées vers une table DiamondTouch en y ajoutant l'interaction de rotation pour être conforme à la guideline du partage de l'espace de travail, frame1 est une instance de JFrame et dsFrame est une instance de DSFrame. Nous supposons que :

- *int* est une instance de *Interactor* qui représente *frame1* au niveau modèle
- *wid* est une instance de *Widget* qui représente *dsFrame* au niveau modèle
- *AdditionalPI* = {*WidgetRotation*, *WidgetResize*}

$$\begin{aligned} int \Leftrightarrow & \left\{ \begin{array}{l} id = 2 \\ name = frame1 \\ intrinsicPI = \{Navigation, WidgetSelection, \\ WidgetDisplay, WidgetMove, \\ WidgetResize, Activation\} \\ contentData = Null \end{array} \right. \\ wid \Leftrightarrow & \left\{ \begin{array}{l} name = dsFrame \\ effectivePI = \{Navigation, WidgetSelection, \\ WidgetDisplay, WidgetMove, \\ WidgetResize, Activation, \\ \textcolor{red}{WidgetRotation}\} \end{array} \right. \\ \Rightarrow int \leqq_{AdditionalPI} wid \end{aligned}$$

5.4.1.3 Équivalence faible : $\geqq_{EssentialPI}$

Cet opérateur d'équivalence recherche les composants graphiques de la cible qui ont un minimum d'interactions essentielles pour permettre l'utilisation de l'UI sur la cible. Cet opérateur permet une migration en mode "dégradé" dans le cas où la cible ne peut pas fournir certaines interactions. Les primitives d'interactions essentielles à conserver sont identifiées dans l'ensemble *EssentialPI*. Cet ensemble de PI est défini en fonction de la plateforme d'arrivée par les personnes en charge de la migration. Dans le cadre des tables interactives, nous avons identifié l'ensemble des PI essentielles en prenant en compte les guidelines qui préservent l'homogénéité des interactions sources et cibles.

$$\forall int \in instanceOf(Interactor), \forall wid \in instanceOf(Widget),$$

$$int \leq_{EssentialPI} wid \Rightarrow \left\{ \begin{array}{l} int.contentData.cardinality = wid.cardinality \\ (int.contentData! = Null) \\ \quad \wedge \\ int.contentData.dataType = wid.ContentType \\ \quad \vee \\ int.contentData = Null \wedge wid.contentType = Null \\ \quad \forall pi1 \in int.effectivePI, \\ \quad \exists pi2 \in wid.IntrinsicPI, \\ \quad (pi1 = pi2) \\ \quad \wedge \\ card(int.effectivePI) > card(wid.IntrinsicPI) \\ \quad \wedge \\ \exists pi \in int.effectivePI \cap wid.IntrinsicPI \wedge pi \in EssentialPI \end{array} \right.$$

Exemple d'équivalence faible Nous illustrons cet opérateur par un exemple de remplacement d'une liste éditable par une liste non éditable. Si une UI source décrite en XAML instancie un ComboBox éditable, nous pouvons la remplacer par une SurfaceListBox non éditable sur une table Microsoft PixelSense. Dans cet exemple, nous pensons que les PI Activation, Data Selection sont essentielles car elles préservent la fonction de sélection d'une liste même si elle n'est pas éditable après la migration.

La liste éditable list1 est une instance de ComboBox et surfacelist est une instance de SurfaceListBox. Nous supposons que :

- *int* est une instance de *Interactor* qui représente *list1* au niveau modèle
- *wid* est une instance de *Widget* qui représente *surfacelist* au niveau modèle
- *EssentialPI* = {Activation, DataSelection}

$$\begin{aligned} int \Leftrightarrow & \left\{ \begin{array}{l} id = 4 \\ name = list1 \\ intrinsicPI = \{Navigation, WidgetSelection, \\ WidgetDisplay, DataSelection, \\ DataMoveIn/Out, \\ \textcolor{red}{DataEdition}, Activation\} \\ contentData = \{dataType = String, cardinality = N, \\ value = ', propertyName = 'Content'\} \end{array} \right. \\ wid \Leftrightarrow & \left\{ \begin{array}{l} name = dsFrame \\ effectivePI = \{Navigation, WidgetSelection, \\ WidgetDisplay, DataSelection, \\ DataMoveIn/Out, \\ Activation\} \\ contentType = String, \\ cardinality = N \end{array} \right. \\ \Rightarrow int \geq_{EssentialPI} wid \end{aligned}$$

5.4.1.4 Remarques

Les opérateurs d'équivalences décrits par cette section 5.4.1 permettent d'établir des correspondances à la volée entre les composants graphiques des plateformes source et cible en s'appuyant sur les PI, les types de données et les cardinalités.

L'opérateur d'équivalence stricte s'applique dans le cas où l'on souhaite avoir des composants graphiques cibles identiques à la source. Cependant dans le cadre des UI pour les tables interactives,

il est indispensable de prendre en compte des guidelines qui préconisent des composants graphiques accessibles à plusieurs personnes par exemple.

L'opérateur d'équivalence large permet d'établir des correspondances entre les composants graphiques des plateformes source et cible mais en intégrant les interactions préconisées par les guidelines de la cible. Par exemple pour un menu d'une UI desktop, cet opérateur permet de retrouver un composant graphique équivalent et avec les interactions de déplacement pour rendre le menu accessible.

L'opérateur d'équivalence faible est l'inverse de l'équivalence large et l'ensemble *EssentialPI* est constitué

- des PI liées à la sélection ou à la navigation (**Widget Select** ou **Navigation**)
- des PI liées à la modification de contenus (**Data Edition** ou **Data Selection**)
- de la **PI Activation**.
- et des PI d'affichage de contenus **Display Data**

Ces PI sont fondamentales pour des interactions en entrée et en sortie. Concernant une UI en mode "dégradée" par exemple, les PI liées à la taille, la position ou l'orientation ne sont pas essentielles. Même si les guidelines de la plateforme cible ne sont pas respectées, l'UI produite conserve l'accessibilité aux fonctionnalités de l'application à migrer.

Nous remarquons que ces trois opérateurs ne prennent pas en compte les composants graphiques équivalents du point de vue des interactions mais avec des types de données différents. Pour la migration des UI vers les tables interactives, un menu contenant des sous menus textuels par exemple, les données graphiques textuelles peuvent être remplacées par des icônes. Les opérateurs décrits à section 5.4.2 répondent à cette limite.

5.4.2 Opérateurs d'équivalences basés sur des PI et des données

Dans l'objectif de combler les limites des opérateurs identifiés à la section 5.4.1 nous définissons dans cette section trois autres opérateurs qui prennent en compte la différence du type de données entre la source et la cible. Les trois opérateurs d'équivalences basés uniquement sur les PI sont modifiés ici pour tenir compte de la variation des types de données.

5.4.2.1 Équivalence stricte des PI avec types de données différents : $\cong_{TargetType}$

Cet opérateur permet de sélectionner les composants graphiques qui ont les mêmes interactions que celles de la source mais avec des types de données différents. L'ensemble *TargetType* exprime les types de données souhaités pour un composant graphique sur la cible.

$$\forall int \in \text{instanceOf}(\text{Interactor}), \forall wid \in \text{instanceOf}(\text{Widget}), \\ int \cong_{TargetType} wid \Rightarrow \left\{ \begin{array}{l} int.\text{contentData}.cardinality = wid.\text{cardinality} \\ (int.\text{contentData}! = \text{Null} \wedge \\ wid.\text{ContentType} \in \text{TargetType}) \\ \forall pi1 \in int.\text{effectivePI}, \\ \exists pi2 \in wid.\text{IntrinsicPI}, \\ pi1 = pi2 \\ \wedge \\ \text{card}(int.\text{effectivePI}) = \text{card}(wid.\text{IntrinsicPI}) \end{array} \right.$$

Exemple Dans cet exemple nous montrons que *ListBox* $\cong_{TargetType}$ *LibraryBar* avec :

- int est une instance de *Interactor* qui représente $list2$ au niveau modèle
- wid est une instance de *Widget* qui représente $library$ au niveau modèle
- $TargetType = \{Image, Object, MediaElement\}$

$$\begin{aligned}
 int \Leftrightarrow & \left\{ \begin{array}{l} id = 5 \\ name = list2 \\ intrinsicPI = \{Navigation, WidgetSelection, \\ WidgetDisplay, DataSelection, DataMoveIn/Out, \\ DataDisplay, Activation\} \\ contentData = \{dataType = String, cardinality = N, \\ value = 'Content', propertyName = 'Content'\} \end{array} \right. \\
 wid \Leftrightarrow & \left\{ \begin{array}{l} name = library \\ effectivePI = \{Navigation, WidgetSelection, \\ WidgetDisplay, DataSelection, DataMoveIn/Out, \\ DataDisplay, Activation\} \\ contentType = Object \in TargetType, \\ cardinality = 1 \end{array} \right. \\
 \Rightarrow int \cong_{TargetType} wid
 \end{aligned}$$

5.4.2.2 Équivalence large des PI avec types de données différents : $\lesssim_{AdditionalPI, TargetType}$

Cet opérateur permet de retrouver les composants graphiques offrant des interactions supplémentaires et comprenant un type de données différent de la source. C'est une amélioration de l'opérateur $\leq_{AdditionalPI}$ en prenant en compte la différence entre les données. L'ensemble *TargetType* exprime les types de données souhaités pour un composant graphique sur la cible.

$$\forall int \in instanceOf(Interactor), \forall wid \in instanceOf(Widget),$$

$$int \lesssim_{AdditionalPI, TargetType} wid \Rightarrow \left\{ \begin{array}{l} int.contentData.cardinality = wid.cardinality \\ (int.contentData! = Null \wedge wid.ContentType \in TargetType) \\ \forall pi1 \in int.effectivePI, \exists pi2 \in wid.IntrinsicPI, pi1 = pi2 \\ card(int.effectivePI) < card(wid.IntrinsicPI) \\ \exists pi \in wid.IntrinsicPI \setminus int.effectivePI \wedge pi \in AdditionalPI \end{array} \right.$$

Exemple Dans cet exemple, nous montrons que *Button* $\lesssim_{AdditionalPI, TargetType} Image$ avec :

- int est une instance de *Interactor* qui représente $button2$ au niveau modèle
- wid est une instance de *Widget* qui représente $image$ au niveau modèle
- $TargetType = \{Image, Object, MediaElement\}$
- $AdditionalPI = \{WidgetRotation, WidgetResize\}$

$$\begin{aligned}
 int &\Leftrightarrow \left\{ \begin{array}{l} id = 6 \\ name = button2 \\ intrinsicPI = \{Navigation, WidgetSelection, \\ WidgetDisplay, Activation\} \\ contentData = \{dataType = String, cardinality = N, \\ value = 'Content'\} \end{array} \right. \\
 wid &\Leftrightarrow \left\{ \begin{array}{l} name = dsFrame \\ effectivePI = \{Navigation, WidgetSelection, \\ WidgetDisplay, DataDisplay, \\ Activation\} \\ contentType = Image \in TargetType, \\ cardinality = 1 \end{array} \right. \\
 \Rightarrow int &\lesssim_{AdditionalPI, TargetType} wid
 \end{aligned}$$

5.4.2.3 Équivalence faible des PI avec types de données différents : $\gtrsim_{EssentialPI, TargetType}$

Cet opérateur permet d'établir des UI équivalentes en mode "dégradé" et en prenant la différence de type de données. C'est une amélioration de l'opérateur $\geq_{EssentialPI}$ en prenant compte de la différence entre les données. L'ensemble *TargetType* exprime les types de données souhaités pour un composant graphique sur la cible.

$$\forall int \in instanceOf(Interactor), \forall wid \in instanceOf(Widget),$$

$$int \gtrsim_{EssentialPI, TargetType} wid \Rightarrow \left\{ \begin{array}{l} int.contentData.cardinality = wid.cardinality \\ (int.contentData! = Null \\ \wedge \\ wid.ContentType \in TargetType) \\ \forall pi1 \in int.effectivePI, \\ \exists pi2 \in wid.IntrinsicPI, \\ pi1 = pi2 \\ \wedge \\ card(int.effectivePI) > card(wid.IntrinsicPI) \\ \wedge \\ \exists pi \in int.effectivePI \cap wid.IntrinsicPI \wedge pi \in EssentialPI \end{array} \right.$$

Exemple Nous montrons que $Image \gtrsim_{EssentialPI, SourceType} SurfaceButton$ avec :

- *int* est une instance de *Interactor* qui représente *image2* au niveau modèle
- *wid* est une instance de *Widget* qui représente *button3* au niveau modèle
- *SourceType* = {Image, Object, MediaElement}
- *EssentialPI* = {Activation, WidgetResize}

$$\begin{aligned}
 int &\Leftrightarrow \left\{ \begin{array}{l} id = 3 \\ name = image2 \\ intrinsicPI = \{Navigation, WidgetSelection, \\ WidgetDisplay, DataDisplay, \\ Activation\} \\ contentData = \{dataType = Image \in SourceType, cardinality = N, \\ value = 'Content'\} \end{array} \right.
 \end{aligned}$$

$$\begin{aligned}
 & \left\{ \begin{array}{l} name = button3 \\ effectivePI = \{Navigation, WidgetSelection, \\ WidgetDisplay, Activation\} \\ contentType = String, \\ cardinality = 1 \end{array} \right. \\
 wid \Leftrightarrow & \Rightarrow int \gtrsim_{EssentialPI, TargetType} wid
 \end{aligned}$$

Remarque Les opérateurs d'équivalences basés sur les PI et prenant en compte des types de données différents entre la source et la cible, permettent de sélectionner les composants graphiques équivalents du point de vue des PI. Les changements des types de données impliquent des conversions des données sources vers la cible. Les conversions peuvent être automatisées s'il existe des relations entre les types de données sources et cibles (par exemple les entiers peuvent être transformés en chaîne de caractères). Dans d'autres cas, les changements de types nécessitent une intervention humaine pour établir un mapping entre les données de la source avec celles de la cible (par exemple la conversion de chaîne de caractères en image nécessite le choix des images représentant les différentes chaînes de caractères).

5.5 Synthèse

Les PI présentées constituent un modèle d'interactions abstraites. L'objectif de ce modèle est d'établir les équivalences entre instruments d'interactions en prenant en compte les spécificités de la cible¹⁷. Les PI intrinsèques des *Widgets*, les PI effectives des *Interactors* et les éléments structurels offrent des éléments de comparaison entre les composants graphiques de manière générique.

Nous avons proposé dans ce chapitre une représentation abstraite des UI à migrer en prenant en compte les interactions (à travers les PI) et la structure (à travers les modèles de type et d'instance d'une UI). En faisant ces choix, nous avions les objectifs suivant :

- Identifier dans un modèle abstrait les interactions en entrée et en sortie des UI graphiques en prenant en compte les différences entre les types et les instances des composants graphiques qui sont importantes dans un processus de migration.
- Montrer qu'il est possible d'identifier les éléments concrets (menu, fenêtre, tableau, boutons, etc.) d'une UI en se basant sur un modèle de structure minimal et les PI.

Nous avons montré aussi qu'en se basant sur les primitives d'interactions et la structure d'un composant graphique, il est possible de décrire des opérateurs d'équivalences. Cependant nous remarquons que ces opérateurs peuvent produire plus d'un composant graphique équivalent. La question du choix du meilleur se pose pour les concepteurs en charge de la partie manuelle de la migration . Leurs choix en général sont guidés par le soucis du respect des critères ergonomiques et la baisse du coût de la mise en œuvre. Le chapitre 6 suivant présente les mécanismes de classement des composants graphiques équivalents en prenant en compte les guidelines de la cible.

Au-delà de ces questions de sélection et de respect des critères ergonomiques, le modèle d'une UI que nous avons présenté dans ce chapitre a pour objectif de décrire des mécanismes d'adaptations des interactions et de la structure de l'UI qui prennent en compte les guidelines de la cible. Le chapitre répond aux questions soulevées dans cette synthèse. Nous proposons aussi un processus de migration qui se base sur ce modèle d'interactions abstraites et sur un modèle de structure minimal (contenu, type de données, cardinalité de données et lien de contenance) et qui prend en compte les guidelines.

17. Dans notre cas ces spécificités sont caractérisées par les guidelines pour la migration vers les tables interactives

Mécanismes de migration des UI vers les tables interactives

Sommaire

6.1	Introduction	99
6.2	Prise en compte des guidelines	100
6.2.1	Conception assistée des UI	101
6.2.2	Interprétation des guidelines pour la migration	102
6.2.3	Utilisation effective des guidelines	104
6.3	Transformations du modèle de l'UI source	108
6.3.1	Transformation des groupes d'éléments graphiques	108
6.3.2	Transformation d'un élément	119
6.4	Classement des éléments équivalents	124
6.4.1	Conformité des composants graphiques aux guidelines	124
6.4.2	Charge de travail	127
6.4.3	Algorithme de classement	129
6.5	Synthèse	133

6.1 Introduction

Le modèle de l'UI présenté au chapitre 5 permet de décrire les UI à migrer indépendamment des bibliothèques graphiques et des dispositifs d'interactions des plateformes source et cible. Ce modèle décrit particulièrement la structure et les interactions abstraites des UI à migrer. Dans ce chapitre nous présentons le processus de migration des UI vers les tables interactives en décrivant les mécanismes de prise en compte des guidelines et les transformations de l'UI source en UI collaborative et tangible. Ces mécanismes sont guidés par un ensemble de guidelines pour la migration des UI vers les tables interactives identifiées à la section 3.2.3. Les guidelines sont des recommandations décrites en langage naturel qui doivent être interprétées en règles de migrations des éléments concrets (tels que fenêtres, pop-ups, menus, tableaux, formulaires, boutons, champs de texte, etc.) d'une UI.

Nous montrons dans ce chapitre qu'en utilisant notre modèle d'UI, il est possible de transformer la structure et les interactions des UI desktops pour les tables interactives et en respectant les guidelines des UI collaboratives et tangibles.

Pour atteindre cet objectif, nous présentons à la section 6.2 notre processus de migration des UI ainsi que les interprétations des guidelines identifiées au chapitre 3 pour décrire les mécanismes de transformation d'un modèle d'UI source. La section 6.3 présente les mécanismes de transformations de la structure et les interactions des UI à migrer. La section 6.4 présente le mécanisme de classement des éléments graphiques équivalents en prenant en compte les guidelines pour la migration. Le classement des éléments équivalents en fonction des guidelines facilite le choix des composants graphiques à utiliser pour d'écrire l'UI cible. Le chapitre se termine par une synthèse qui présente les points forts

et les limites de notre approche de transformation d'une UI source en UI collaborative et tangible pour les tables interactives.

6.2 Prise en compte des guidelines dans le processus de migration des UI vers les tables interactives

Nous proposons un processus semi automatique de migration des UI vers les tables interactives. Nous optons pour une approche semi automatique à la section 4.3.2 afin de bénéficier de la flexibilité, de la réutilisabilité et du respect des critères ergonomiques qu'elle offre. Notre processus se décompose en trois étapes (cf. figure 6.1).

- La première étape consiste à **abstraire** de manière automatique la structure et les interactions de l'UI source dans les modèles présentés au chapitre 5. Nous considérons que chaque application de départ respecte les contraintes d'architecture présentées à la section 2.3.
- La deuxième étape consiste à **restructurer** le modèle abstrait de l'UI en s'appuyant sur les guidelines et sur les opérateurs d'équivalences. La restructuration est automatique dans un premier temps puis manuelle pour permettre à l'utilisateur de personnaliser l'UI cible après avoir **concrétiser** le modèle restructuré. La restructuration automatique consiste à retrouver les interacteurs équivalents pour la cible et à substituer les interacteurs de la source par ceux de la cible dans le but de proposer une UI concrète. La restructuration manuelle permet à la personne en charge de la migration de modifier l'UI proposée automatiquement et de réintroduire le positionnement et le style dans l'UI migrée.
- La troisième étape consiste à **générer** automatiquement l'application pour la table interactive en considérant l'UI transformée et le NF de l'application source. Cette étape consiste d'abord à générer l'UI finale pour la plateforme cible qui décrit les interactions, la structure, le positionnement et le style. Ensuite cette étape consiste à générer le lien entre l'UI et le NF en tenant compte du modèle d'architecture de l'application départ.

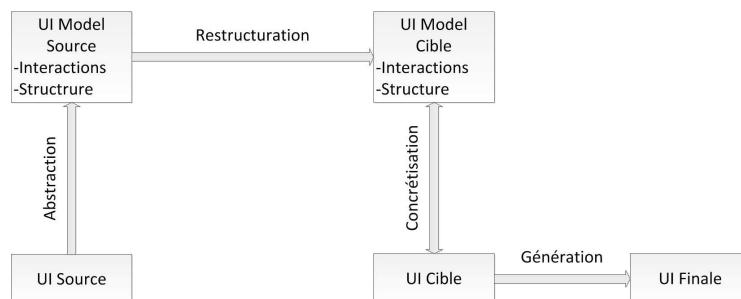


FIGURE 6.1 – Processus de migration assistée

La prise en compte des guidelines par les mécanismes automatiques et manuels de la restructuration a pour but de respecter les critères ergonomiques de conception et les spécificités des tables interactives. Pendant la restructuration, les guidelines doivent être traduites en recommandations ou en contraintes de migration des UI dans le but de favoriser la collaboration et l'utilisation des objets tangibles.

Une approche de prise en compte des guidelines dans un processus de conception des UI est proposée par Vanderdonckt [Van97] dans le but de minimiser les coûts de développement des UI. Nous étudions à la section 6.2.1 cette approche afin de décrire les guidelines de migration des UI utilisables pendant la restructuration et la concrétilsation. La section 6.2.2 est une interprétation des

guidelines à partir des critères ergonomiques de conception et la section 6.2.3 présente les mécanismes d'utilisation effective des guidelines interprétées.

6.2.1 Conception assistée des UI

La conception assistée comporte quatre étapes pour transformer des tâches interactives en UI finale tout en prenant en compte les guidelines de la plateforme cible.

1. La première étape consiste à déduire des facteurs d'utilité et d'utilisabilité que l'UI à produire doit faire, à partir d'une description des tâches interactives, des utilisateurs et de la plateforme. L'utilité concerne l'adéquation qui doit exister entre les fonctions sémantiques présentées par une IHM et les actions nécessaires pour l'utilisateur en vue d'accomplir sa tâche [Van97]. L'utilisabilité concerne l'adéquation entre la manière dont une tâche interactive est accomplie par un utilisateur particulier dans un contexte donné et le profil cognitif de cet utilisateur [FPV95]. Les facteurs d'"utilité" et d'"utilisabilité" concrétisent les facteurs critiques de succès suivant les deux perspectives d'utilité et d'utilisabilité. Dans le cadre de la migration des UI pour les tables interactives, nous avons identifié à la section 3.2 les propriétés qui favorisent la mise en place des UI collaboratives et tangibles. Nous considérons que les propriétés de *tangibilité et tactilité des interactions*, de la *taille et la disposition de la surface d'affichage* et celles du *nombre et la répartition des utilisateurs* constituent des facteurs critiques qui influencent la migration des UI vers les tables interactives.
2. La deuxième étape consiste à faire ressortir à partir des facteurs identifiés précédemment une pondération des critères ergonomiques de conception¹⁸ pour les tâches interactives [Van97]. Dans le cadre de la migration des UI, cette étape consiste à considérer le tableau 3.3 de raffinement des critères ergonomiques de conception par rapport aux propriétés caractéristiques (qui sont aussi considérées comme facteur critiques dans notre contexte). Ce tableau nous a permis à la section 3.2.3 d'identifier les guidelines pour la migration des UI vers les tables interactives.
3. La troisième étape consiste à déduire l'ensemble des guidelines non conflictuelles. Dans le cadre de la conception des UI, les guidelines constituent des recommandations pour décrire la structure, les interactions, le layout et le style de l'UI finale. Dans le cadre de la migration des UI, les guidelines permettent de transformer automatiquement les interactions et la structure de l'UI source. Nous avons identifié à la section 3.2.3 les guidelines pour favoriser les UI collaboratives et les UI tangibles.
4. La quatrième étape consiste à générer automatiquement les objets interactifs concrets de l'UI en se basant sur un modèle des UI et en s'appuyant aussi sur les guidelines identifiées. Dans le cadre de notre processus de migration des UI, cette phase consiste à transformer selon les guidelines de migration le modèle de l'UI source préalablement extrait.

Considérons l'hypothèse de la section 2.3 qui préconise la non modification du NF de l'application de l'UI source. Et considérons que notre modèle des UI décrit les interactions et la structure de l'UI source (cf chapitre 5). Alors les deux premières étapes du processus de Vanderdonckt qui consistent à adapter les tâches par rapport à la plateforme cible ne peuvent pas être considérées dans le cadre de la migration assistée. En effet, la transformation des activités d'une UI mono-utilisateur pour prendre en compte des tâches collaboratives nécessite une modification du NF.

Cependant les interactions et la structure de l'UI source peuvent être adaptées pour favoriser la collaboration et permettre l'utilisation des objets tangibles si les recommandations des guidelines sont décrites de manière non ambiguë.

18. Nous considérons les huit critères ergonomiques de conception proposés par Scapin [Sca86] (cf section 3.2.2).

6.2.2 Interprétation des guidelines pour la migration

Les guidelines peuvent être comprises et interprétées de différentes manières par les concepteurs des UI car elles sont décrites par un langage naturel. Dans le cadre de la migration des UI vers les tables interactives et en considérant les deux groupes de guidelines identifiés à la section 3.2.3, nous interprétons les guidelines par des actions sur une représentation abstraite des UI à migrer. Les interprétations proposées dans cette section ont pour objectifs de mettre en place des règles de transformations réutilisables pour différents types d'UI.

6.2.2.1 Guidelines pour favoriser la collaboration

Dans le but de décrire les guidelines identifiées à la section 3.2.3 à partir des propriétés caractéristiques et des critères ergonomiques de conception, nous présentons dans cette section une interprétation des guidelines qui favorise la collaboration. Les propriétés caractéristiques de taille, de disposition de la surface d'affichage, du nombre et de la répartition des utilisateurs permettent d'affiner les guidelines qui favorisent la collaboration. Le tableau 3.3 d'affinement des critères ergonomiques de conception nous permet d'identifier deux types de guidelines.

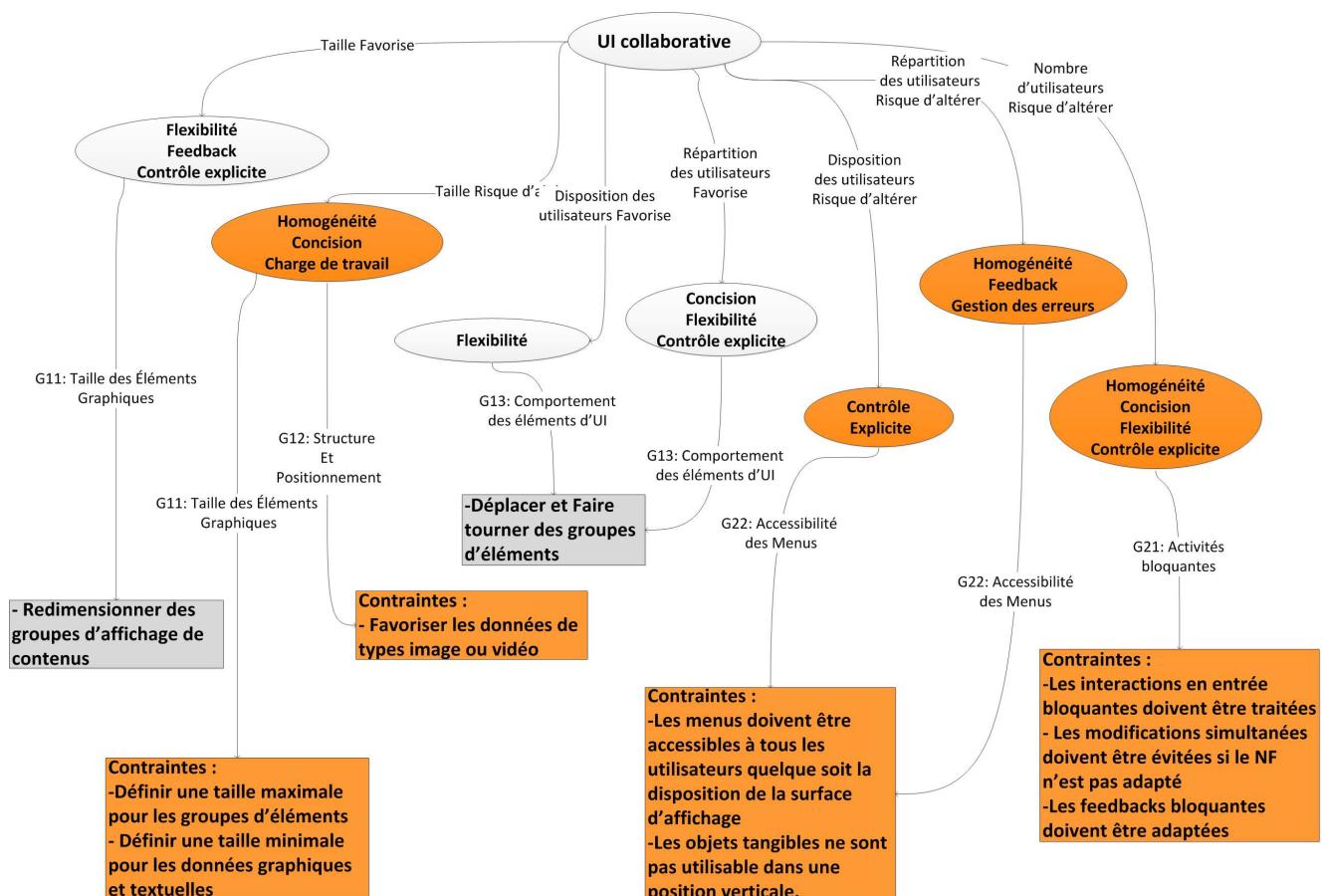


FIGURE 6.2 – Critères ergonomiques de conception et guidelines pour favoriser la collaboration

- Les guidelines déduites des propriétés qui **favorisent** des critères ergonomiques ; ces guidelines constituent des recommandations à appliquer sur les PI et la structure de l'UI.

- Les guidelines déduites à partir des propriétés qui **risquent d'altérer** des critères ergonomiques ; ces guidelines comportent des contraintes à respecter pendant la migration.

Le diagramme de la figure 6.2 est un arbre qui regroupe les critères ergonomiques de conception et les guidelines. Les nœuds oranges sont les critères ergonomiques qui risquent d'être altérés si les contraintes (représentées par les feuilles rectangulaires) ne sont pas respectées pendant la migration. Les nœuds gris sont des critères ergonomiques de conception favorisés si les recommandations (représentées par les feuilles rectangulaires) sont respectées.

6.2.2.2 Guidelines pour des UI tangibles

La propriété caractéristique de tangibilité permet d'affiner les guidelines pour l'utilisation des objets tangibles. La figure 6.3 présente les recommandations et les contraintes pour chaque critère ergonomique.

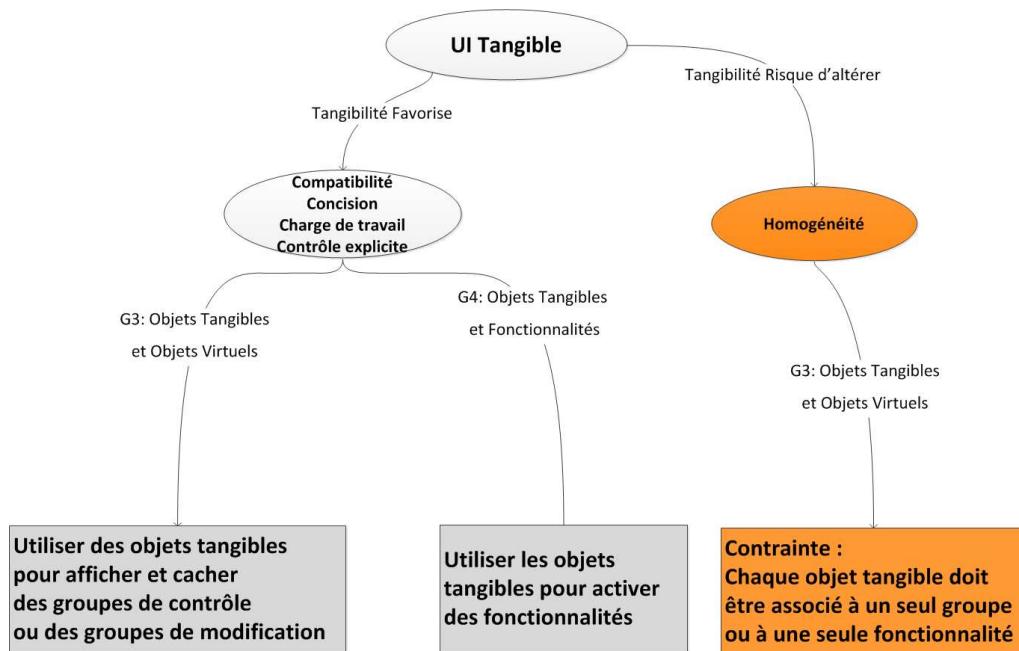


FIGURE 6.3 – Critères ergonomiques de conception et guidelines pour des UI tangibles

6.2.2.3 Synthèse

Les recommandations et les contraintes permettront de transformer la structure et les interactions du modèle de l'UI source.

Les guidelines sont aussi à affiner sous la forme des contraintes sur les types de données, la taille des éléments graphiques, le comportement des groupes d'éléments graphiques, etc.

Les diagrammes des figures 6.2 et 6.3 sont des interprétations des guidelines à l'aide des critères ergonomiques de conception qui constitueront des règles de transformation de l'UI source.

L'interprétation des guidelines est effectuée avant l'implémentation de notre processus de migration des UI. Elle nécessite un affinement des critères ergonomiques de conception à l'aide des spécificités de la plateforme cible. L'interprétation dans notre cadre est une prise de position sur la définition et l'effet des guidelines dans le but de produire des règles de transformation de la source.

Dans la section suivante nous proposons un modèle de règles pour les transformations du modèle des UI pour permettre une utilisation effective des guidelines par le processus de migration des UI vers les tables interactives.

6.2.3 Utilisation effective des guidelines

Les guidelines sont utilisées par les mécanismes de restructuration pendant les transformations des modèles abstraits [MVG06]. Notre processus de migration des UI vers les tables interactives implémente deux types de transformation des modèles.

- Les transformations exogènes et verticales [MVG06] qui comprennent le mécanisme d’abstraction de l’UI finale dans le modèle abstrait et le mécanisme de **concrétisation** du modèle abstrait en UI finale à l’aide des éléments de la plateforme cible.
- Les transformations endogènes et horizontales [MVG06] qui consistent à **restructurer** le modèle abstrait de l’UI. Dans notre cas la restructuration de l’UI de départ s’appuie sur les guidelines de la plateforme d’arrivée afin d’avoir une UI conforme aux critères ergonomiques de la cible.

Pour utiliser les guidelines, nous proposons de les traduire sous la forme des règles utilisables par les mécanismes de restructuration et de concrétisation (cf figure 6.4). Le modèle d’UI à restructurer se présente sous la forme d’un arbre dont les nœuds et les feuilles correspondent à des *Interactors*. .

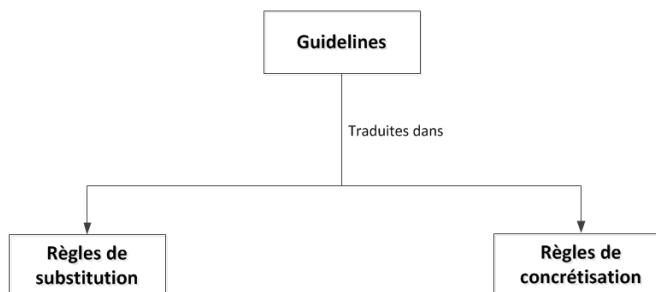


FIGURE 6.4 – Utilisations effectives des guidelines par les mécanismes de migration des UI

Nous avons opté pour une stratégie de restructuration basée sur la substitution des *Interactors* de la source pour déduire la cible. La substitution des *Interactors* conserve la structure arborescente de l’UI source tout en utilisant des *Interactors* de la cible qui ont des interactions et une structure conforme aux guidelines. Une des recommandations du diagramme de la figure 6.2 préconise de “déplacer et faire tourner des groupes d’éléments graphiques”. En considérant que les groupes d’éléments correspondent aux *Containers*, nous pourront décrire une règle de substitution qui remplace les *Containers* de la source par ceux qui implémentent les PI de déplacement et de rotation.

Par ailleurs, les règles de concrétisation ont pour objectifs de générer une UI concrète qui sera personnalisée par la personne en charge de la migration. Les guidelines sont traduites dans les **règles de concrétisation**. Les recommandations et les contraintes issues de la propriété de tangibilité sont prises en compte pendant la concrétisation. Les contraintes sur la taille des éléments graphiques sont appliquées par les règles de concrétisation.

Dans l’objectif d’assister les personnes en charge de la migration pendant la phase de personnalisation en précisant la guideline et les critères ergonomiques de conception associés à chaque substitution ou concrétisation. Nous présentons d’abord un modèle des règles de substitution (cf section 6.2.3.1),

ensuite un modèle des règles de concrétisation (cf section 6.2.3.2). Ces modèles permettent de décrire des règles formelles issues des guidelines et des critères ergonomiques de conception.

6.2.3.1 Modèle de règles de substitution

Les règles de substitution ont pour objectif d'indiquer pour un ou plusieurs interacteur(s) de l'UI source les caractéristiques des interacteurs équivalents pour le modèle cible. Ces règles de substitution sont constituées de deux membres : le membre de gauche (de la figure 6.5) correspond aux caractéristiques des interacteurs de l'UI source et le membre de droite (de la figure 6.5) correspond aux caractéristiques souhaitées pour les éléments équivalents du modèle cible. Les composants graphiques correspondant au membre de droite de la règle de substitution sont identifiés à l'aide des opérateurs d'équivalences.

Chaque règle de substitution est conforme à une recommandation ou une contrainte des guidelines des figures 6.2 et 6.3 ci-dessus.

Le modèle de règles de substitution sert à implémenter un feedback du respect des critères ergonomiques de conception pendant les substitutions manuelles des interacteurs par les personnes en charge de la migration. En effet ce modèle permet de savoir si la substitution d'un *Interactor* est conforme à une guideline.

Nous spécifions formellement les règles de substitution à l'aide du modèle de la figure 6.5.

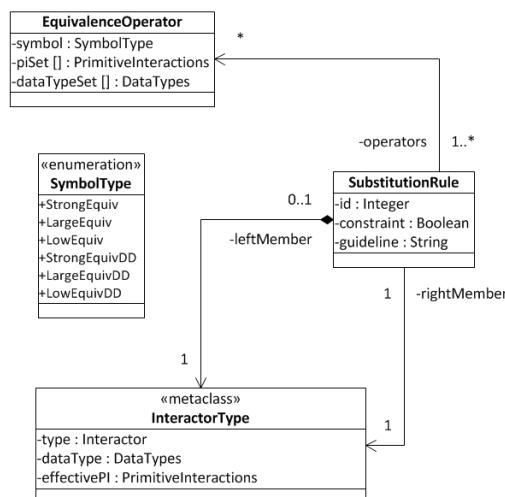


FIGURE 6.5 – Modèle de règles de substitution

La classe SubstitutionRule caractérise une règle de substitution, elle est constituée d'un identifiant unique pour chaque règle et des rôles :

- *operators* correspond aux opérateurs d'équivalences à appliquer pour déterminer les *Widgets* équivalents de la plateforme cible
- *leftMember* correspond à la caractéristique des interacteurs du modèle source sur lesquels s'applique ces règles
- *rightMember* correspond à la caractéristique souhaitée des interacteurs de la cible

L'attribut *guideline* précise la guideline qui a permis de déduire la règle de substitution conformément aux diagrammes de raffinement des figures 6.2 et 6.3 ci-dessus.

L'attribut *constraint* précise si une règle de substitution est issue d'une guideline qui comporte des contraintes.

La classe InteractorType caractérise les membres d'une règle de substitution. Elle est une métaclass de l'interface *Interactor* et correspond à l'ensemble des interacteurs qui se définit par les attributs suivants :

- *type* est le type d'interacteur de l'ensemble (UIComponent ou Container)
- *dataType* est le type de données des interacteurs de l'ensemble
- *effectivePI* correspond aux PI effectives des interacteurs de l'ensemble

La classe EquivalenceOperator caractérise les opérateurs d'équivalences utilisés, elle se définit par les attributs suivants :

- *symbol* est l'un des symboles des six opérateurs décrits à la section 5.4. L'énumération **SymboleType** présente les six opérateurs d'équivalences présentés à la section 5.4.
- *piSet* correspond à l'ensemble des PI utilisé comme paramètre par les opérateurs d'équivalences
- *dataTypeSet* correspond aux types de données de l'ensemble *TargetType* des opérateurs d'équivalences

6.2.3.2 Modèle de règle de concrétisation

En ce qui concerne la concrétisation du modèle obtenu après l'application des règles de substitution, nous proposons aussi un modèle qui permet de décrire de manière formelle les règles de concrétisation. La concrétisation permet de proposer une instance du modèle de l'UI en UI finale en utilisant les composants graphiques et les dispositifs d'interactions de la plateforme cible. Les guidelines permettent de décrire l'utilisation de ces instruments d'interactions. Le modèle de la figure 6.6 permet de décrire les règles de concrétisation.

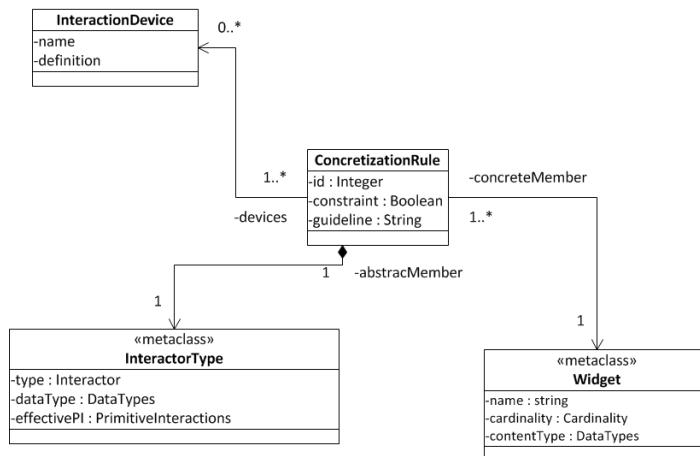


FIGURE 6.6 – Modèle de règles de concrétisation

La classe ConcretizationRule caractérise les règles de concrétisation des interacteurs d'un modèle de l'UI. Chaque règle a un identifiant (*id*) unique et elle se définit aussi par les rôles :

- *devices* qui correspond aux dispositifs d'interactions utilisables pendant la concrétisation

- *abstractMember* qui correspond au type d’interacteur (*InteractorType*) sur lequel s’applique cette règle de concrétisation ; à chaque type d’interacteur correspond une seule règle de concrétisation

- *concreteMember* qui correspond aux *Widgets* équivalents de la plateforme cible

L’attribut *guideline* précise la guideline qui a permis de déduire la règle de concrétisation conformément aux diagrammes d’affinement des figures 6.2 et 6.3 ci-dessus.

L’attribut *constraint* précise si une règle de concrétisation est issue d’une guideline qui comporte des contraintes.

La classe InteractionDevice caractérise les dispositifs physiques d’interactions qui permettent de spécifier les interactions utilisateurs dans le cadre d’une guideline. Elle est caractérisée par l’attribut *name* du dispositif et l’attribut *definition* qui est une description textuelle du dispositif.

La classe InteractorType correspond à celle décrite par le modèle de règles de substitution à la figure 6.5.

La classe Widget correspond à celle décrite par le modèle des types de composants graphiques à la figure 5.4

6.2.3.3 Remarques

Si une règle de substitution issue d’une contrainte n’est pas respectée pendant la personnalisation d’une UI, il est indispensable d’alerter la personne en charge de la migration en précisant la guideline et les critères ergonomiques de conception associés.

Il en est de même si une règle de concrétisation issue d’une contrainte est modifiée.

Les guidelines de migration des UI dans notre contexte ont pour objectif de permettre la description des caractéristiques des membres de droite au sein règles de substitution et concrétisation. Ces caractéristiques dépendent de l’interprétation de chaque guideline en fonction des éléments de notre modèle de l’UI (PI, Container, UIComponent, Type de données, etc.).

Dans la section suivante nous décrivons les différentes règles de substitution et concrétisation.

6.3 Transformations du modèle de l'UI source

Notre processus semi automatique propose aux personnes en charge de la migration, des UI destinées aux tables interactives sans le positionnement des éléments graphiques et sans le style. L'UI proposée est ensuite personnalisée en respectant les guidelines pour les UI collaboratives et pour les UI tangibles. La proposition et la personnalisation des UI pour la cible sont des transformations du modèle de l'UI. La restructuration du modèle source est une forme de transformation par la substitution et la concrétisation des interacteurs équivalents.

Dans cette section nous décrivons les règles de substitution et de concrétisation pour chaque type d'interacteurs caractérisé par la classe *InteractorType*. Les transformations concernent des ensembles de composants graphiques caractérisés par les PI, les types de données et les types d'interacteurs. Par exemple, les composants graphiques de contrôle tels que les menus ou les groupes de boutons sont caractérisés par la PI Activation. Les composants graphiques d'affichage de contenus (tels que les tableaux, les listes, les carrousels, etc.) sont caractérisés par la PI Data Display. Les composants graphiques de modifications de contenus tels que les champs de texte par exemple peuvent être caractérisés par la PI Data Edition.

Nous avons opté pour deux stratégies de substitution et de concrétisation du modèle source qui prend en compte les deux types d'interacteurs. La première consiste à considérer les groupes d'éléments (section 6.3.1) et la seconde considère les éléments graphiques de manière individuelle (section 6.3.2) pendant la restructuration. En effet, le modèle d'instance que nous avons présenté à la section 5.3.2 décrit deux types d'interacteurs : *Container* et *UIComponent*, ils représentent respectivement les groupes et les éléments individuels. La substitution et la concrétisation de ces deux types d'éléments graphiques n'ont pas les mêmes objectifs. Dans notre cadre, la substitution et la concrétisation des *Containers* permettent de les déplacer, de les roter ou de les utiliser avec des objets tangibles tout en respectant les contraintes des guidelines. Par ailleurs, la substitution et la concrétisation des *UIComponent* ont pour but de conserver les PI de départ et d'utiliser les types de données adaptés pour les tables interactives.

6.3.1 Transformation des groupes d'éléments graphiques

La transformation des groupes d'éléments graphiques a pour objectifs de favoriser les activités collaboratives en rendant accessible chaque groupe à tous les utilisateurs. Elle permet aussi d'associer les groupes d'éléments graphiques à des objets tangibles. Les règles de substitution et de concrétisation des groupes de composants graphiques ont pour objectif de favoriser la flexibilité¹⁹, le contrôle explicite²⁰, la charge de travail²¹ sans altérer l'homogénéité²² et le contrôle explicite²³ (cf. figures 6.2 et 6.3).

Pour les UI graphiques de manière générale, les groupes d'éléments graphiques ont une sémantique pour l'utilisateur. En effet les panels, les formulaires, les menus ou les tableaux représentent des groupes d'éléments dans une UI graphique qui permettent d'afficher des données ou d'accéder à des fonctionnalités. Dans notre modèle des instances d'une UI présenté à la section 5.3.2 nous proposons de décrire les groupes d'éléments d'une UI graphique en nous basant sur la nature du container. Nous

19. Le redimensionnement, le déplacement et la rotation de certains groupes facilitent l'accessibilité

20. L'affichage des groupes d'éléments à l'aide des objets tangibles par exemple est un contrôle explicite

21. Afficher les groupes cachés par des objets tangibles réduit la charge de travail car l'utilisateur aura moins d'éléments graphiques à l'écran

22. Les containers sans taille limite ne permettent pas d'avoir une UI homogène par exemple

23. Le non respect de l'accessibilité des menus ne permet pas une utilisation aisée de l'UI par exemple

avons identifié les containers de types **Racine**²⁴, **Récursif**²⁵, **Homogène**²⁶ et **Hétérogène**²⁷.

La transformation des groupes d’éléments de l’UI source passe par la substitution et la concrétisation de ces quatre types de containers en fonction des guidelines interprétées à la section 6.2.2. L’avantage de ces quatre types de containers est qu’ils permettent de représenter l’UI source en se basant sur les interactions et les données des instances de composants graphiques et non sur les types des composants graphiques. En effet les différences entre les types et les instances des composants graphiques sont importantes pour les processus de migration car le choix des éléments graphiques équivalents en dépend. Une instance de composant graphique peut implémenter moins de PI effectives que son type, le processus de migration ne doit prendre en compte que les PI effectives pour établir les équivalences avec la cible. Cette contrainte a pour but de ne préserver que les dialogues des UI de départ pendant la migration. Dans ce cas deux instances de composant d’un même type peuvent avoir des équivalences différentes, si elles ont des PI effectives différentes.

Pour transformer la structure des UI de départ, nous considérons quatre groupes d’éléments graphiques en fonction de leurs utilisations dans une UI graphique. Les PI, les containers et les types de données nous permettent de les caractériser. Le but de cette considération est de permettre une spécialisation des groupes en fonction des interactions.

1. Les groupes de contrôle (*ControlGroup*) représentent un ensemble d’éléments graphiques qui permettent d’activer des fonctionnalités d’une UI graphique (cf section 6.3.1.1).
2. Les groupes d’affichage de contenus (*DisplayGroup*) représentent un ensemble de composants graphiques qui permettent de visualiser des données (cf section 6.3.1.2).
3. Les groupes de modification de contenus (*UpdateGroup*) représentent un ensemble de composants graphiques qui permettent des modifications des données d’application (cf section 6.3.1.3).
4. Les groupes mixtes (*MixedGroup*) représentent un ensemble de composants graphiques qui contient des groupes de contrôle, des groupes d’affichage et/ou des groupes de modification de contenus (cf section 6.3.1.4).

La figure 6.7 illustre un exemple de ces différents groupes en s’appuyant sur l’application Comic Book décrite à la section 2.2. Dans la suite de cette section nous présentons les règles de substitution de concrétisation des différents groupe d’éléments.

6.3.1.1 Transformation des groupes de contrôle

Les groupes de contrôle représentent les composants graphiques tels que les menus, les groupes de boutons, les listes de sélections, etc. Ces groupes sont caractérisés par un container de type **Homogène** dont chaque élément implémente les PI *Activation* et *Navigation* et/ou *Widget Selection* (éventuellement *Display Data* pour les boutons munis des icônes).

Ce groupe est caractérisé par la classe *InteractorType* décrit par le tableau 6.1

Les *ControlGroups* sont transformés pour les tables interactives suivant la guideline “G22 : Accessibilité des Menus” qui a pour objectif de rendre les groupes de contrôle accessibles. Cette accessibilité se traduit par l’ajout des comportements de déplacement et de rotation. Si le concepteur le souhaite il peut associer un groupe de contrôle à un objet tangible en appliquant la guideline “G3 : Objets Tangibles et Objets virtuels”.

24. Ce sont les containers qui contiennent tous les éléments d’une UI

25. Ce sont les containers qui ne contiennent que les éléments de type container

26. Ce sont les containers qui contiennent les composants graphiques avec des données du même type

27. Ces containers contiennent à la fois les *Containers* et des *UIComponents* avec des types de données différents

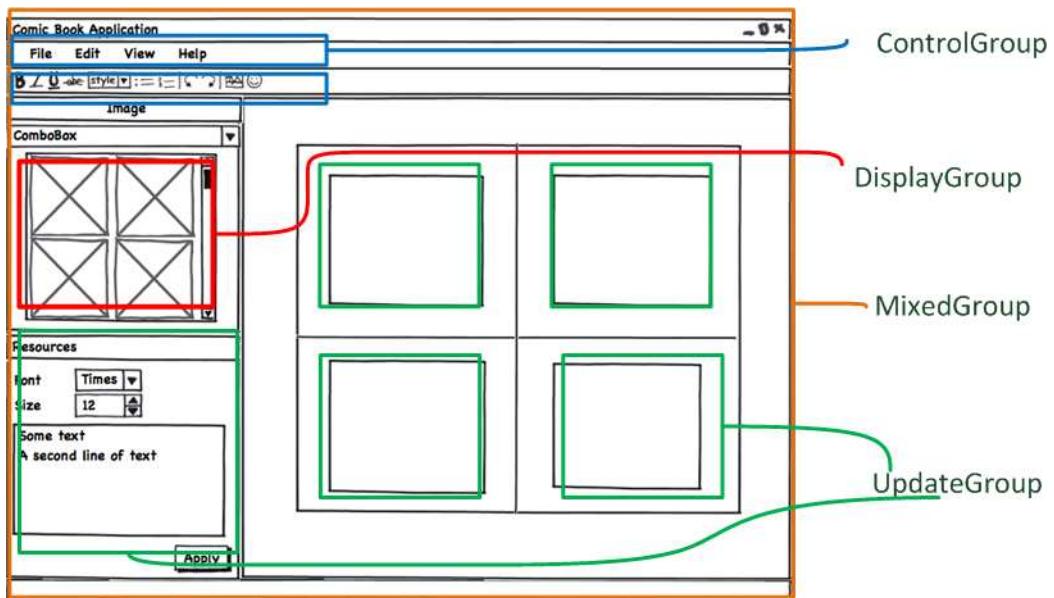


FIGURE 6.7 – Exemple des groupes d’éléments graphiques à transformer

InteractorType	
type	$type \in \{Container\}$
	$type.containerType() = \{Homogeneous\}$
	$\forall i \in type.contains, i.effectivePI \subseteq \{ Activation, WidgetSelection, Navigation, DisplayData \}$
	$i.type.contentType \in \{String, Image\}$
dataType	$type.type.contentType \in \{String, Image\}$
effectivePI	$type.effectivePI \subseteq \{WidgetSelection, Navigation\}$

TABLE 6.1 – Caractéristiques des *ControlGroups*

Règle de substitution Cette règle est issue de la contrainte de la guideline “G22 : Accessibilité des Menus”. Le tableau 6.2 caractérise la règle de substitution des groupes de contrôle. Cette règle préconise l’application de l’opérateur d’équivalence large $\leq_{AdditionalPI}$ en privilégiant les PI *WidgetMove* et *WidgetRotation* pour le container. L’utilisation de l’opérateur d’équivalence stricte avec différence de données $\cong_{TargetType}$ doit privilégier les données de type *Image* pour les éléments des menus.

Règle de concrétisation Cette règle est issue de la guideline “G3 : Objets Tangibles et Objets virtuels”. Le tableau 6.3 caractérise la concrétisation d’un *ControlGroup*. L’utilisation des objets tangibles a pour but d’afficher le *ControlGroup* qui sera caché à tout moment. Chaque utilisateur qui souhaite accéder à un groupe de contrôle doit avoir un objet capable de l’afficher.

Remarques

- Nous avons noté que l’affichage multiple d’un *ControlGroup* entraîne des incohérences si deux utilisateurs font appel à une fonctionnalité qui modifie une donnée. En effet le NF de l’application n’étant pas modifié, les appels de fonctionnalités se font de manière séquentielle et les

SubstitutionRule	
id	<i>ControlGroupSRule</i>
guideline	G22
constraint	True
leftMember	InteractorType du tableau 6.1
rightMember	InteractorType du tableau 6.1
operators	$\leq_{AdditionalPI}$ et $\cong_{TargetType}$
	avec $AdditionalPI = \{WidgetMove, WidgetRotation\}$
	et $TargetType = \{Image\}$

TABLE 6.2 – Règle de substitution des groupes de contrôle

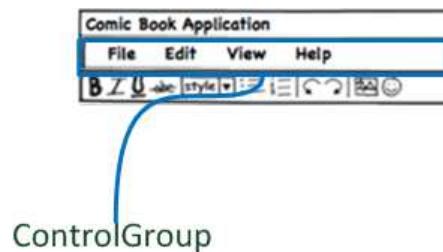
ConcretizationRule	
id	<i>ControlGroupCRule</i>
guideline	G3
constraint	True
abstractMember	InteractorType du tableau 6.11
concreteMember	$w \in \{Widget\}$
devices	Écran Tactile, Objet Tangible unique pour chaque <i>ControlGroup</i>

TABLE 6.3 – Règle de concréétisation des groupes de contrôle

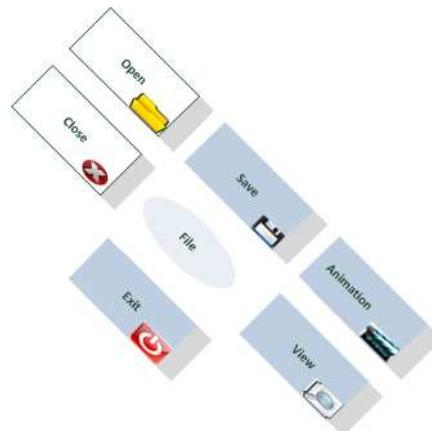
modifications (changement de couleur ou de taille par exemple) se feront sur la même donnée pour les deux utilisateurs. Cependant les appels vers le NF qui permettent d'afficher des composants graphiques (une nouvelle fenêtre par exemple), créent des instances différentes de ces composants graphiques pour chaque utilisateur.

- Ces observations nous ont poussés à opter pour l'affichage unique d'un *ControlGroup* dans le but d'éviter les incohérences. Dans le cas où deux utilisateurs ont des objets tangibles pour le même *ControlGroup*, un seul *ControlGroup* sera affiché s'ils les posent sur la surface d'affichage au même moment. Dans les autres cas, le dernier objet posé sera considéré.
- La transformation des *ControlGroups* que nous proposons permet de résoudre les problèmes d'accessibilité des éléments graphiques et des menus en particulier. Par exemple les menus *File* et *Edit* seront accessibles par deux utilisateurs autour de la table au même moment sans qu'ils se gênent.
- Cependant cette transformation ne permet pas à deux utilisateurs d'accéder à un menu au même moment. Par exemple, le menu *File* ne peut être affiché plusieurs fois au même moment. Cette restriction permet de garder des dialogues cohérents et qui ne perturbent pas les utilisateurs. Par exemple si deux utilisateurs souhaitent ouvrir deux fichiers (représentant des bandes dessinées) différents, cette transformation ne le permet pas car l'application ne peut afficher qu'un seul document au même moment.

Exemple de *ControlGroup* Nous considérons la figure 6.8 qui est un artefact de l'application CBA présentée au chapitre 3, représentant les menus de cette application. En effet, les menus *File*, *Edit*, etc. de la figure 6.8 représentent des groupes de contrôle conformes au type d'interacteurs décrit par le tableau 6.1. Chaque sous-menu implémente les PI *Activation*, *Navigation* et *Widget Selection* car ils sont accessibles par un clavier ou une souris et activent des fonctionnalités. Les données de ces menus et sous-menus sont des chaînes de caractères.

FIGURE 6.8 – Exemple de *ControlGroup*

L'exemple présenté est migré sur une table interactive en appliquant d'abord la règle de substitution décrite par le tableau 6.2 ensuite la règle de concrétisation décrite par le tableau 6.3. Les menus *File*, *Edit*, *View* et *Help* auront les PI *Widget Move* et *Widget Rotation* et ils seront affichés chacun par des Tags associés à des objets dans le cadre d'une table Micorsoft PixelSense. La figure 6.9 présente une illustration du menu *File* pour table interactive.

FIGURE 6.9 – Exemple de *ControlGroup* sur table interactive Micorsoft PixelSense

6.3.1.2 Transformation des groupes d'affichage de contenus

Les groupes d'affichage de contenus représentent les composants graphiques tels que les listes, les tableaux, les groupes de données, etc. Ces groupes sont caractérisés par les containers de type **Homogène** ou **Hétérogène**. Chaque élément de *DisplayGroup* est un *UIComponent* implémentant les PI *Data Display*, *Widget Selection* et *Navigation*. Les éléments *DisplayGroup* n'implémentent pas les PI *Activation*, *Data Edition*, *Data Move In/Out* et *Data Selection*. Ce groupe est caractérisé par la classe *InteractorType* décrit par le tableau 6.4.

La transformation des groupes d'affichage de contenus pour les tables interactives se fait en fonction des types de données et dans le but de partager l'espace de travail en permettant à plusieurs personnes de consulter les contenus affichés.

- Si les contenus des éléments de la cible sont des Images, *MediaElements* ou *Objects*
- La guideline “*G11 : Taille des éléments graphiques*” permet d'ajouter la PI *WidgetResize* à chaque élément de *DisplayGroup*.
- La guideline “*G12 : Structure et Positionnement des éléments graphiques*” permet le changement de type de données des contenus si possible en favorisant les types Images par exemple.

InteractorType	
type	$type \in \{Container\}$ $type.containerType() \in \{Homogeneous, Heterogeneous\}$ $\forall i \in type.contains, i \in \{UIComponent\}$ $\left\{ \begin{array}{l} Activation, DataEdition, \\ DataMoveIn/Out, DataSelection \end{array} \right\} \not\subseteq i.effectivePI$ $DataDisplay \in i.effectivePI$ $i.type.contentType \in \{Image, MediaElement, Object\}$ \vee $i.type.contentType \in \{Integer, String, Boolean\}$
dataType	$type.type.contentType \in DataType$
effectivePI	$type.effectivePI \neq \emptyset$

TABLE 6.4 – Caractéristiques des groupes d'affichage de contenus

Cependant les recommandations liées aux positionnements des éléments par la guideline G12²⁸ ne peuvent pas être appliquées dans ce cas car les éléments de ce groupe sont déplaçables conformément à la guideline G13²⁹.

- La guideline “G13 : Comportement des éléments graphiques” permet d'ajouter les PI WidgetResize et WidgetMove à chaque éléments de *DisplayGroup*
- Si les contenus des éléments de la cible sont de type Boolean, Integer ou String
 - La guideline “G11 : Taille des éléments graphiques” permet d'ajouter la PI WidgetResize au container et les éléments sont migrés sans changement. Dans le cas d'un tableau contenant des données numériques, il est préférable de conserver la structure du tableau pour faciliter sa lecture et sa compréhension.
 - La guideline “G13 : Comportement des éléments graphiques” permet d'ajouter les PI WidgetResize, WidgetMove au container et les éléments sont migrés sans changement.

Règle de substitution des *DisplayGroups* La substitution de ce groupe dépend des types de données de ses éléments. Si tous les éléments sont des chaînes de caractères ou des données numériques provenant du NF, il est difficile de transformer ces données en images par exemple. En effet les données provenant du NF peuvent constituer un ensemble infini et le concepteur ne peut pas prévoir les images représentant chaque donnée du NF. Le tableau 6.5 caractérise la règle de substitution des groupes d'affichage des contenus dans le cas où les données sont de type Boolean, Integer, String et proviennent du NF. Cette règle préconise l'application de l'opérateur d'équivalence large $\leq_{AdditionalPI}$ pour le container en ajoutant les PI WidgetMove, WidgetRotation et WidgetResize. L'opérateur d'équivalences strictes pour les éléments du groupe container.

Dans le cas où les données graphiques peuvent être traduites vers les types Image, MediaElement ou Object, la règle décrite par le tableau 6.5 est utilisée en appliquant l'opérateur d'équivalence large $\leq_{AdditionalPI}$ pour les éléments du groupe en ajoutant les PI WidgetMove, WidgetRotation et WidgetResize. L'opérateur d'équivalence stricte est appliqué pour le container du groupe.

La règle de substitution des *DisplayGroups* est issue des guidelines “G11 : Taille des éléments graphiques”, “G13 : Comportement des éléments graphiques” et de la contrainte de la guideline “G12 : Structure et Positionnement des éléments graphiques”.

28. Structure et Positionnement des éléments graphiques

29. Comportement des éléments graphiques d'une UI

SubstitutionRule	
id	DisplayGroupSRule1
guideline	G11, G12, G13
constraint	True
leftMember	InteractorType du tableau 6.4)
rightMember	InteractorType du tableau 6.4)
operators	$\leq_{AdditionalPI}$ et \equiv avec AdditionalPI = { WidgetMove, WidgetRotation, WidgetResize }

TABLE 6.5 – Règle de substitution des groupes *DisplayGroups*

Règle de concrétisation des *DisplayGroup*

- Les groupes contenant des données numériques ou textuelles sont concrétisés en tableaux avec les comportements WidgetMove, WidgetResize et WidgetRotation.
- Les groupes contenant des données de type Image, MediaElement ou Object sont concrétisés en implémentant pour chaque élément les comportements WidgetMove, WidgetResize et WidgetRotation.

Remarques

- Nous préconisons dans le cas où une UI migrée contient plusieurs groupes d'affichage de délimiter pour chaque groupe une zone ou de cacher certains groupes (par les groupes contenant les données numériques) et l'afficher avec des objets tangibles en appliquant la guideline *Objets Tangibles et Objets virtuels* (G3). Dans ce cas plusieurs instances d'un même DisplayGroup peuvent être affichées pour consultation.
- La transformation des *DisplayGroups* concerne la dimension Structure et Positionnement des éléments d'une UI. Elle traite les questions d'accessibilité des éléments graphiques pour présenter des données pour tous les utilisateurs. La contrainte liée à la taille des éléments graphiques résout l'un des problèmes liés à la dimension Style de l'espace des problèmes.

Exemple d'un *DisplayGroup* Nous considérons la figure 6.10 qui représente une liste d'images. Chaque élément de la liste d'images implémente les PI *Navigation*, *Widget Selection* et *Data Display* car ils sont accessibles par un clavier ou une souris et activent des fonctionnalités.

L'exemple présenté est migré sur une table interactive en appliquant d'abord la règle de substitution décrite par le tableau 6.5 ensuite chaque élément de la liste sera un ScatterViewItem pour être accessible à tous les utilisateurs dans le cadre d'une table Microsoft PixelSense. La figure 6.11 illustre un exemple de *DisplayGroup* pour une table Microsoft PixelSense

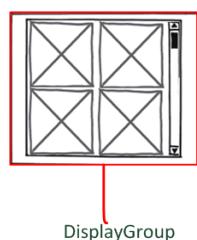
FIGURE 6.10 – Exemple de *DisplayGroup*

FIGURE 6.11 – Exemple de *DisplayGroup* migré sur une table interactive

6.3.1.3 Transformation des groupes de modification de contenus

Les groupes de modification de contenus représentent les composants graphiques tels que les formulaires, des canevas, des tableaux éditables, etc. Ces groupes sont caractérisés par des containers de type **Hétérogène** ou **Homogène**. Un *UpdateGroup* contient des interacteurs implémentant les PI *Activation*, *Data Edition*, *Data Selection* ou *Data Move In/Out*. Nous caractérisons ces groupes par la classe *InteractorType* décrit par le tableau 6.6.

InteractorType	
type	$type \in \{Container\}$
	$type.containerType() = \{Heterogeneous, Homogeneous\}$
	$\exists i \in type.contains, i \in \{UIComponent\}$
	$\left\{ \begin{array}{l} Activation, DataEdition, \\ DataMoveIn/Out, DataSelection \end{array} \right\} \subseteq i.effectivePI$
	$i.type.contentType \in DataType$
dataType	$type.type.contentType \in DataType$
effectivePI	$type.effectivePI \neq \emptyset$

TABLE 6.6 – Caractéristiques des groupes de modification de contenus

La transformation des *UpdateGroups* pour les tables interactives se fait selon la guideline “G21 : *Activités Bloquantes*” qui recommande d’avoir des activités de modification de contenus qui n’empêchent pas les différents utilisateurs d’accéder à d’autres fonctionnalités de l’application. Dans le cadre d’une UI desktop la saisie de deux champs de texte distincts n’est pas possible, les tables interactives offrent néanmoins la possibilité de le faire. Nous considérons que la guideline “G21 : *Activités Bloquantes*” permet d’avoir plusieurs *UpdateGroups* distincts. Cependant il n’est pas possible d’avoir plusieurs instances d’un *UpdateGroup* dans modifier le NF pour prendre en compte la modification d’une donnée du NF par deux utilisateurs distincts.

La guideline “G13 : *Comportement des éléments graphiques*” permet d’ajouter les PI *WidgetResize* et *WidgetMove* au container représentant le groupe.

La guideline “G12 : *Structure et Positionnement des éléments graphiques*” est appliquée pour les éléments du container pour permettre le changement de type de données des contenus si possible en favorisant le type *Images* par exemple. Les recommandations liées aux positionnements des éléments par la guideline G12³⁰ sont appliquées dans ce cas pour les éléments.

Règle de substitution d’un *UpdateGroup* Les éléments équivalents à ce groupe sont obtenus en appliquant l’opérateur d’équivalence large $\leq_{AdditionalPI}$ pour le container en y ajoutant les PI *Widget-*

30. Structure et Positionnement des éléments graphiques

Move, WidgetRotation. Le tableau 6.7 caractérise la règle de substitution. Les éléments du container sont substitués en appliquant l'opérateur d'équivalence stricte avec une différence dans les données afin de privilégier des données de type Image ou Object.

SubstitutionRule	
id	DisplayGroupSRule1
guideline	G11, G12, G13
constraint	True
leftMember	InteractorType du tableau 6.4)
rightMember	InteractorType du tableau 6.4)
operators	$\leq_{AdditionalPI}$ et $\cong_{TargetType}$ avec $AdditionalPI = \{ WidgetMove, WidgetRotation, WidgetResize \}$ et $TargetType = \{ Image \}$

TABLE 6.7 – Règle de substitution des *UpdateGroups*

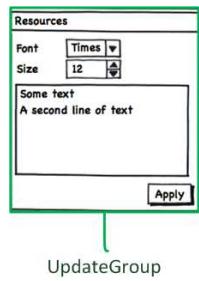
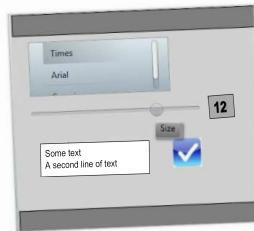
Règle de concrétisation des *UpdateGroups* La concrétisation des groupes de modification de contenu se fait comme les groupes de contrôle. En effet les objets tangibles peuvent être associés à des *UpdateGroups* pour réduire le nombre de groupes visibles dans l'espace de travail.

Nous remarquons que la duplication d'un groupe de modifications de contenus dans le but de permettre à plusieurs personnes de modifier des contenus différents est possible si le NF est adapté. Cependant nous préconisons de cacher les *UpdateGroups* si une UI en compte plusieurs et de les afficher à l'aide d'un objet tangible.

Remarques

- La transformation des *UpdateGroups* a pour objectif de rendre accessibles ses groupes à tous les utilisateurs. Ces groupes peuvent être affichés par les utilisateurs à l'aide des objets tangibles dans le but modifier des données. Le type et la forme des objets tangibles sont choisis par les personnes en charge de la migration. Le processus de migration garantit que chaque objet est associé à une fonctionnalité ou à un groupe d'éléments graphiques.
- Pour éviter les problèmes liés à la cohérence de la modification des données, un *UpdateGroup* ne peut être affiché plus d'une fois au même moment.
- Les *UpdateGroups* contenant des interacteurs avec des données de types *Boolean*, *Integer* ou *String* représentent des panels de configurations par exemple. Ils sont migrés sur les tables interactives pour un dialogue mono-utilisateur.

Exemple d'un *UpdateGroup* Nous considérons la figure 6.12 qui représente un formulaire de changement de taille, de police et de contenu d'une chaîne de caractères. Le container représentant le groupe est concrétisé en ScaterViewItem pour être accessible à tous les utilisateurs dans le cadre d'une table Microsoft PixelSense. Ce formulaire est associé à un Tag pour l'afficher à l'aide d'un objet dans le cadre d'une table Microsoft PixelSense. La figure 6.13 est un *UpdateGroup* migré.

FIGURE 6.12 – Illustration d'un *UpdateGroup*FIGURE 6.13 – Exemple d'un *UpdateGroup* sur une table interactive

6.3.1.4 Transformation des groupes mixtes

Ils regroupent les *ControlGroups*, des *DisplayGroups* ou des *UpdateGroups*. Ces groupes représentent des fenêtres ou des containers d'une UI graphique. Les groupes mixtes sont caractérisés par des containers de type **Récursif** ou **Racine**. Chaque élément d'un groupe mixte est un container de type **Homogène**, **Hétérogène** ou **Récursif**. Cette catégorie de regroupement a pour objectif de considérer une UI graphique dans sa globalité pour la prise en compte des guidelines liées au partage de l'espace de travail. Nous caractérisons ce groupe par le tableau 6.8.

InteractorType	
type	$type \in \{Container\}$ $\{Recursive, Root\} \in type.containerType()$ $\exists i \in type.contains, i \in \{Container\}$ $\left\{ \begin{array}{l} Homogeneous, \\ Heterogeneous, \\ Recursive \end{array} \right\} \subseteq i.containerType()$
dataType	$type.type.contentType \in DataType$
effectivePI	$type.effectivePI \neq \emptyset$

TABLE 6.8 – Caractéristiques des groupes mixtes

Règle de substitution de *MixedGroup* La transformation des groupes mixtes pour les tables interactives dépend du type d'éléments qu'ils contiennent.

Si la fenêtre principale d'une application est un *MixedGroup*, alors dans l'objectif de rendre accessibles les différents sous-groupes nous recommandons d'appliquer les règles de transformations des différents groupes identifiées ci-dessus.

Si un des sous-groupe d'un *MixedGroup* est aussi mixte alors le container de type **Recurcif** correspondant à ce sous-groupe doit avoir une position fixée pour garder dans un cadre les groupes qu'il contient.

Exemple d'un *MixedGroup* En considérant l'illustration des différents groupes à la figure 6.7, la fenêtre principale est un groupe mixte et le container des canevas est aussi un groupe mixte.

La transformation de la fenêtre principale permettra d'avoir des groupes de contrôle déplaçables et utilisables avec des objets tangibles. Les *DisplayGroups* et *UpdateGroups* de l'application seront aussi transformés en suivant leurs règles substitution et de concrétisation décrites ci-dessus. Le groupe mixte contenant des canevas ne sera pas déplaçable ou redimensionnable pour rassembler les canevas dans un même groupe visible à l'écran. Le cadre vert de la figure 6.14 ci-dessous illustre le groupe mixte contenant les canevas. Cependant, il est possible que les dimensions d'un groupe mixte soient équivalentes à la surface d'affichage pour que les éléments contenus soient accessibles partout sur l'écran.

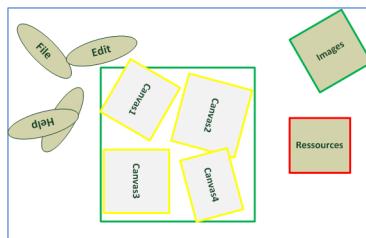


FIGURE 6.14 – Représentation de l'UI CBA pour tables interactives

Remarque

- La transformation des *MixedGroups* a pour objectif d'accroître l'accessibilité des éléments de l'UI de départ. Cette transformation permet aussi d'obtenir des éléments graphiques de tailles variables. Elle concerne la migration de la dimension Structure et positionnement des UI de départ.

6.3.1.5 Mécanismes de substitution et de concrétisation des groupes

Les règles de substitution des groupes³¹ favorisent une utilisation collaborative de l'UI de départ en permettant le déplacement des groupes d'éléments graphiques. L'identification des groupes est effectuée en se basant sur la structure et le type de PI de leurs éléments.

L'application de ces règles de substitution est faite de manière automatique ou interactive. Cependant les règles de concrétisation sont appliquées par un mécanisme automatique.

La substitution automatique des groupes permet de proposer une première version de l'UI pour la personnaliser. Le mécanisme en charge parcourt tous les interacteurs du modèle de l'UI source et applique les règles de substitution à chaque interacteur de ce modèle afin de déduire le modèle cible. Nous remarquons que plusieurs règles peuvent être appliquées sur les containers comportant plusieurs types (par exemple Racine et Récursive, Racine et Homogène, etc.)

31. *ControlGroup*, *DisplayGroup*, *UpdateGroup*, *MixedGroup*

La substitution interactive des containers a pour but de permettre à la personne en charge de la migration des UI de les personnaliser en se basant sur l'ensemble d'équivalences de chaque container. En effet les opérateurs d'équivalences utilisés par les règles de substitution permettent d'avoir plusieurs équivalences pour un interacteur du modèle source. Ces interacteurs équivalents sont utilisés pour la personnalisation de l'UI source. Pour chaque interacteur sélectionné par l'utilisateur, le mécanisme lui propose la liste des interacteurs équivalents. Par ailleurs, pour chaque règle de substitution à appliquer, le mécanisme est en mesure d'identifier les guidelines correspondantes, ces guidelines permettront aux utilisateurs moins expérimentés d'appréhender facilement la migration des UI sur les tables interactives.

La concrétisation des groupes permet d'avoir un rendu graphique des UI après les substitutions. Ce mécanisme crée des instances de chaque *Widget* des différentes règles de concrétisation en appliquant les contraintes associées (par exemple les contraintes liées à la taille des containers)

6.3.1.6 Synthèse des transformations des groupes

- L'interprétation des guidelines et leurs utilisations par les mécanismes de transformation se font à travers les interacteurs et les caractéristiques des composants graphiques (PI et type de données).
- Le mécanisme de substitution opère de manière automatique ou de manière interactive. La substitution interactive a pour objectif d'assister les personnes en charge de la migration des UI. Concernant le choix des Widgets équivalents, un mécanisme de classement de ces derniers en prenant compte des guidelines est indispensable.
- L'ordre de transformation des groupes du modèle de l'UI dépend de la stratégie du parcours de l'arbre représentant sa structure. Nous avons remarqué que pour considérer les plus petits groupes d'éléments contenus dans l'arbre, il faut un parcours en profondeur pour identifier les types de groupes et appliquer les règles associées. Un parcours en largeur de l'arbre représentant le modèle de structure de l'UI à transformer n'est pas possible car pour identifier le type d'un groupe il faut parcourir tous ses éléments fils. La section 6.3.2 présente les règles de substitution et de concrétisation des éléments simples d'un groupe.
- Le tableau 6.9 ci-dessous présente les sous problèmes traités par les transformations des groupes décrites dans cette section.

6.3.2 Transformation d'un élément

Les mécanismes de transformation de la section 6.3.1 ne précisent pas comment substituer les éléments fils qui ne sont pas des groupes. En effet une règle de substitution d'un *UpdateGroup* ne précise pas comment substituer les composants graphiques simples (tels que les labels, boutons, champs de texte) d'un container. Cette section décrit les règles de substitution et de concrétisation d'un composant graphique simple sans altérer l'homogénéité³², la concision³³ et favoriser le contrôle explicite³⁴ (cf figures 6.2 et 6.3).

Les interacteurs de type *UIComponent* constituent l'ensemble des éléments concernés par cette substitution. Nous catégorisons ces interacteurs en fonction de leurs PI effectives.

32. Les contraintes issues de ce critère ergonomique de conception préconisent d'avoir une taille minimale pour les données graphiques et cette taille doit être respectée pour toutes les données de l'UI

33. L'utilisation des données de type Image ou vidéo favorise la compréhension des interactions et réduit la charge de travail

34. L'activation d'une fonctionnalité à l'aide d'un objet tangible par exemple est une action explicite

Transformation des groupes	Dialogues	Structure et Positionnement	Style
ControlGroup	Mono-utilisateur, Interactions Tangibles (D11)	Regroupement (D22) et Accessibilité des Menus	
DisplayGroup	Multi-utilisateurs (D13)	Regroupement (D22) et Positionnement (D23)	Taille variable (D31)
UpdateGroup	Mono-utilisateur, Interactions Tangibles (D11)	Regroupement (D22) et Positionnement(D23)	
MixedGroup		Regroupement (D22) et Positionnement (D23)	

TABLE 6.9 – Synthèse des problèmes traités par les transformations des groupes

- Les interacteurs qui n’implémentent que les PI en sortie³⁵ et les PI en entrée³⁶ peuvent être classés dans la catégorie des **interacteurs des données en sortie** car ils sont transformés par les contraintes sur les types de données³⁷ et sur la taille³⁸.
- Les interacteurs qui implémentent les PI en entrée sur les données³⁹ appartiennent à la catégorie des **interacteurs des données en entrée** car ils représentent des éléments éditables ou modifiables par les utilisateurs. Ils sont transformés suivant les contraintes issues de la guideline “G21 : Activités Bloquantes”.
- Les **interacteurs d’activation** implémentent la PI Activation en plus des PI sur les Widgets⁴⁰ car ces interacteurs représentent les éléments d’activation de fonctionnalité, ils sont transformés en suivant la guideline “G12 : Structure et Positionnement des éléments graphiques” pour les transformer en image interactive. La guideline “G4 : Objets tangibles et fonctionnalités” permet d’associer ces interacteurs à un objet tangible dans le cas où la guideline G12 n’est pas appliquée. Le choix des interacteurs de cette transformation est laissé à la personne en charge de la migration, car il est le seul capable de juger de l’utilité de la transformation pour un interacteur d’activation. Cependant le processus l’assiste en lui proposant les interacteurs d’activation d’une UI.

6.3.2.1 Transformation des interacteurs de données en sortie

Ce sont des interacteurs de type *UIComponent* appartenant à une UI et qui permettent d'afficher des données d'application. Ils n'appartiennent pas aux containers de type Homogène, car ce cas est traité par la transformation des containers de ce type. Ils sont caractérisés par l'InteractorType du tableau 6.10.

Ces interacteurs peuvent contenir des données graphiques ou des données d'application. Dans le cas où ils contiennent des données graphiques, la transformation consiste d'abord à lui substituer un

35. Widget Display et Data Display

36. Widget Selection et Navigation

37. “G12 : Structure et Positionnement des éléments graphiques”

38. “G11 : Taille des éléments graphiques”

39. Data Edition, Data Selection, Data Move In/Out

40. cf. note³⁶

InteractorType	
type	$type \in \{UIComponent\}$
dataType	$type.type.contentType! = null$
effectivePI	$type.effectivePI \subset \left\{ \begin{array}{l} WidgetDisplay, \\ DataDisplay, \\ WidgetSelection, \\ Navigation \end{array} \right\}$

TABLE 6.10 – Caractéristiques des interacteurs de données en sortie

interacteur pouvant contenir des données graphiques de type *Images* en utilisant l’opérateur d’équivalence stricte avec différences de données. Ensuite la concrétisation et la personnalisation permettront de le positionner manuellement et de préciser une taille.

Dans le cas où ces interacteurs contiennent des données d’application, ils sont substitués par des éléments strictement équivalents en appliquant l’opérateur d’équivalence stricte.

La règle de substitution des interacteurs des données en sortie est décrite en considérant la contrainte issue de la guideline “*G12 : Structure et Positionnement des éléments graphiques*” qui préconise de favoriser les données de type image ou vidéo. Nous appliquons cette contrainte pour les données graphiques. En ce qui concerne les données de l’application, il est indispensable soit de modifier le NF soit d’identifier de manière exhaustive l’ensemble des valeurs d’une donnée et d’établir des correspondances avec des images par exemple.

La Règle de concrétisation des données en sortie est décrite en considérant la contrainte issue de la guideline “*G11 : Taille des éléments graphiques*”⁴¹.

Remarques

- La substitution de ces interacteurs se fait d’abord par un mécanisme automatique ensuite manuellement si les concepteurs souhaitent modifier les transformations proposées.
- La concrétisation ne permet pas le placement de ces composants graphiques. Les concepteurs doivent positionner l’ensemble des interacteurs de ce type après la concrétisation. Cependant, il est possible de proposer un positionnement par défaut, mais il n’est pas possible de garantir une cohérence des positions proposées car notre modèle ne conserve pas les positions ou les liens de proximité entre les composants graphiques de l’UI source.
- Cette transformation concerne les problèmes liés aux sous-dimensions *Taille* et *Regroupement et de Positionnement* des éléments graphiques.

6.3.2.2 Transformation des interacteurs des données en entrée

Ce sont des interacteurs de type *UIComponent* qui permettent de modifier ou d’éditer des données d’une UI graphique. Ils appartiennent à tous les types de containers. Ce type d’interacteur est caractérisé par l’InteractorType du tableau 6.11

Ces interacteurs sont aussi caractérisés par les PI de modification des données Data Edition, Data Selection et Activation, Data Move Out et Activation et Data Move In.

41. Cette contrainte préconise une taille maximale et une taille minimale pour les données et les composants graphiques (cf figure 6.2)

InteractorType	
type	$type \in \{UIComponent\}$
dataType	$type.type.contentType! = null$
effectivePI	$\left\{ \begin{array}{l} DataEdition, \\ DataSelection, Activation, \\ DataMoveOut, \\ DataMoveIn \end{array} \right\} \subset type.effectivePI$

TABLE 6.11 – Caractéristiques des interacteurs de données en entrée et en sortie

La règle de substitution des interacteurs des données en entrée est décrite en considérant les contraintes issues de la guideline “G21 : Activité Bloquantes”. Les interactions en entrées des UI desktops sont des activités bloquantes. Sur une table interactive elles doivent permettre aux autres utilisateurs d'accéder à d'autres fonctionnalités.

Ces interacteurs sont substitués par les mêmes types d'interacteurs avec des types de données équivalents en appliquant les opérateurs d'équivalence stricte.

La règle de concrétisation des interacteurs de données en entrée est issue de la contrainte de la guideline “G11 : Taille des éléments graphiques”. La taille des Widgets doit être définie en prenant compte de la taille de son container. Son positionnement est effectué manuellement par la personne en charge de la migration.

Remarques

- Les containers des interacteurs des données en entrée ne sont pas accessibles par plusieurs utilisateurs au même moment. Cette restriction évite les problèmes liés à la cohérence des données modifiées.
- La transformation de ces interacteurs a pour objectif de conserver les dialogues (interactions en entrée) de l'UI de départ. Cependant cette transformation ne permet pas les dialogues multi-utilisateurs.

6.3.2.3 Transformation des interacteurs d'activation

Ce sont des interacteurs de type *UIComponent* qui permettent de déclencher des fonctionnalités. Ils n'appartiennent qu'aux containers de type Homogène car ce cas est traité par la règle de substitution des menus. Ce type d'interacteur est caractérisé par l'InteractorType du tableau 6.12.

InteractorType	
type	$type \in \{UIComponent\}$
dataType	$type.type.contentType! = null$
effectivePI	$type.effectivePI \subseteq \{Navigation, WidgetSelection, Activation\}$

TABLE 6.12 – Caractéristiques des interacteurs d'activation

La règle de substitution des interacteurs d'activation est issue de la contrainte qui favorise les données de type image ou vidéo⁴². Ils sont substitués par les interacteurs de même type avec une dif-

42. “G12 : Structure et Positionnement des éléments graphiques”

férence de type de données. La règle de substitution des éléments correspondant au type d'interacteur $\cong_{TargetType}$, avec $TargetType = \{Image\}$ est appliquée.

La règle de concrétisation des interacteurs d'activation est issue de la contrainte qui favorise l'image et la vidéo et de la recommandation qui préconise l'utilisation des objets tangibles pour activer les fonctionnalités. Le concepteur précise les interacteurs d'activation à associer aux objets tangibles.

Remarques

- La transformation des interacteurs d'activation a pour objectif l'accessibilité des fonctionnalités par des dialogues homogènes et par un contrôle explicite à l'aide d'objets physiques.
- Ces interacteurs ne permettent pas des dialogues multi-utilisateurs car un seul utilisateur a accès à un interacteur au même moment.

6.3.2.4 Synthèse des transformations des interacteurs

- Les règles de substitution des interacteurs de type *UIComponent* préservent les PI et adaptent les types de données de la source.
- Le positionnement des éléments substitués se fait au niveau de l'UI finale par la personne en charge de la migration.
- L'association d'un objet tangible à une fonctionnalité se fait au niveau de l'UI finale par la personne en charge de la migration. Le processus de migration des UI semi automatique propose une liste d'interacteurs appartenant au type décrit par le tableau 6.12.
- Les transformations des interacteurs proposées dans cette section traitent les problèmes d'accessibilité des interacteurs et des fonctionnalités par des transformations automatiques et manuelles. Le tableau 6.13 présente de manière synthétique les problèmes abordés selon les dimensions des UI.

Transformation des groupes	Dialogues	Structure et Positionnement	Style
Interacteur des données en entrée	Mono-utilisateur	Positionnement Manuel (D23)	Taille définie Manuellement (D31)
Interacteur des données en sortie	Mono-utilisateur	Accessibilité, Positionnement Manuel (D23)	Taille variable, Taille définie Manuellement (D31)
Interacteur d'activation	Mono-utilisateur, Interactions Tangibles	Accessibilité des fonctionnalités, Positionnement Manuel (D23)	Taille définie Manuellement (D31)

TABLE 6.13 – Synthèse des problèmes traités par les transformations des interacteurs

6.4 Classement des éléments équivalents

Nous proposons dans ce manuscrit une approche de migration des UI semi automatique qui assiste les concepteurs en proposant des options de migration des UI conformément aux guidelines de la cible. Cependant le choix de certaines options peut impliquer des tâches supplémentaires non automatisables⁴³ pour les concepteurs.

Les stratégies de transformation de la structure de l'UI source telles que présentées à la section 6.3 utilisent les mécanismes d'équivalences des composants graphiques. Nos opérateurs d'équivalences permettent de retrouver plusieurs éléments équivalents pour un composant graphique de l'UI source. Dans cette section, nous proposons de classer les composants graphiques équivalents dans le but de faciliter leur choix pour le concepteur en prenant en compte les guidelines de la cible. Par ailleurs le choix d'un composant graphique équivalent est guidé par l'objectif de réduire le coût de sa mise en œuvre pour le concepteur.

Nous présentons dans cette section deux critères pour classer les éléments équivalents : la **conformité** des composants graphiques aux guidelines et la **charge de travail** engendrée par le choix d'un élément équivalent. Le but de ces critères est d'accroître le respect des guidelines tout en réduisant les coûts des interventions humaines pendant le processus de migration.

6.4.1 Conformité des composants graphiques aux guidelines

Nous caractérisons les composants graphiques à la section 5.3 par les PI (intrinsèques et effectives) et les éléments structurels (types et cardinalités des données). Ces caractéristiques nous permettent d'évaluer la conformité de chaque composant graphique aux guidelines de la cible.

La section 6.4.1.1 présente une méthode d'évaluation de la conformité des composants graphiques aux guidelines qui favorisent la collaboration à l'aide des PI. La section 6.4.1.2 présente une évaluation de la conformité aux guidelines qui favorisent la collaboration à l'aide des caractéristiques structurelles des composants graphiques. La section 6.4.1.3 présente la méthode d'évaluation de la conformité d'un composant graphique aux guidelines en prenant en compte les PI et les caractéristiques structurelles.

6.4.1.1 Évaluation de la conformité aux guidelines en fonction des PI

Les PI décrivent les interactions atomiques des composants graphiques indépendamment des instruments d'interactions. Nous identifions trois catégories de PI ; celles qui sont conformes à des guidelines, celles qui ne favorisent pas la conformité à des guidelines et celles qui sont neutres par rapport aux guidelines.

PI favorisant le respect des guidelines : Les PI **Widget Move**, **Widget Rotation** et **Widget Resize** favorisent la mise en œuvre d'un espace de travail partagé si elles sont appliquées à des groupes d'éléments graphiques (cf section 6.3).

La guideline “*G11 : Taille des éléments graphique*” est interprétée à la figure 6.2, elle recommande des tailles variables pour les groupes d'affichage de contenus. Dans ce cas, les composants graphiques implémentant la PI **Widget Resize** sont conformes à G11 car ils peuvent être redimensionnés.

43. La transformation des données textuelles en icônes ou images nécessite d'associer à chaque texte une représentation graphique ayant la même sémantique

La guideline “*G13 : Comportement des éléments graphiques*” recommande des éléments graphiques déplaçables pour favoriser leur accessibilité. Les groupes des éléments graphiques implémentant les PI **Widget Move** et **Widget Rotation** sont conformes à la G13.

PI ne favorisant pas le respect des guidelines : Les interactions d'édition de contenu (**Data Edition**) limitent une utilisation collaborative car elles décrivent des activités bloquantes pour d'autres utilisateurs. La guideline “*G21 : Activités Bloquantes*” recommande l'utilisation des interactions non bloquantes pour les autres utilisateurs.

PI neutres par rapport aux guidelines : Les PI **Widget Selection**, **Navigation**, **Data Selection**, **Data Move In/Out**, **Data Display** et **Activation** sont neutres par rapport aux guidelines des UI collaboratives et tangibles présentées à la section 6.2.

Comment évaluer la conformité aux guidelines à l'aide des PI ? Pour évaluer la conformité des composants ayant des PI aux guidelines des UI collaboratives, nous avons pondéré toutes les PI suivant les trois catégories évoquées précédemment.

- Les PI favorisant les guidelines ont une pondération supérieure à zéro
- Les PI neutres ont une pondération nulle
- Les PI ne favorisant pas les guidelines ont une pondération inférieure à zéro

Le poids de chaque PI détermine son importance pour une migration. Ceci dans le but de rendre flexible le mécanisme d'équivalences pour les personnes en charge de la migration. Les poids des PI sont déterminés par la fonction

$$\text{InteractionsWeight} : \text{Widget} \rightarrow \{\text{Interger}\}$$

et en utilisant le tableau 6.14 (ci-dessous) des poids des PI par rapport aux guidelines. Dans ce tableau, les poids $P_f > 0$ expriment les PI qui favorisent des guidelines. Les poids $P_n = 0$ expriment les PI qui sont neutres et les poids $P_a < 0$ ne favorisent pas les guidelines. La fonction *InteractionsWeight* calcule la somme des poids des PI d'un *Widget*.

$$\forall w \in \{\text{Widget}\} \wedge \forall pi \in w.\text{effectivePI}$$

$$\text{InteractionsWeight}(w) = \sum P_{pi}, P_{pi} \in \{P_a, P_j, P_n\}$$

6.4.1.2 Évaluation de la conformité aux guidelines en fonction des types de données

Les types de données des composants graphiques permettent comme les PI d'évaluer leur conformité aux guidelines. Nous constatons que les contenus de certains types permettent de décrire des UI faciles à apprécier et qui favorisent la collaboration dans le cadre des tables interactives.

Types de données conseillés pour les tables interactives Les données graphiques de type **Image**, **MediaElement** ou **Object** sont adaptées pour les tables interactives. En effet la guideline “*G12 : Structure et Positionnement des éléments graphiques*” conseille aussi l'usage de ces types de données car ils illustrent mieux les interactions ou les informations d'une UI.

Primitives d'interactions	Contraintes sur le poids	Guidelines
Widget Resize, Widget Move, Widget Rotation	$P_f > 0$	G11, G13
Widget Selection, Navigation, Data Selection, Data Move In/Out, Data Display, Activation	$P_n = 0$	/
Data Edition	$P_a < 0$	G21

TABLE 6.14 – Poids des PI par rapport aux guidelines

Types de données non conseillés pour les tables interactives Les contenus de type **String**, **Boolean** ou **Integer** ne sont pas adaptés en tant que données graphiques pour les tables interactives. En effet ces types de données n’illustrent pas les interactions mais obligent les utilisateurs à lire les données d’un menu par exemple.

Comment évaluer la conformité à l'aide des types de données ? Pour évaluer la conformité des composants à la guideline préconisant les types de données, nous avons pondéré tous les types de données suivant les deux catégories citées précédemment.

- Les types de données conseillés ont une pondération supérieure à zéro
- Les types de données non conseillés ont une pondération inférieure à zéro.

Les poids des types des données d'un composant graphique sont déterminés par la fonction

$$\begin{aligned}
 \text{DataTypeWeight} : \text{Widget} &\rightarrow \{\text{Integer}\} \\
 \forall w \in \{\text{Widget}\}, \\
 \text{DataTypeWeight}(w) &= P_i, P_i \in \{P_{bl}, P_{in}, P_{st}, P_{im}, P_{me}, P_{ob}, P_{at}\}
 \end{aligned}$$

et en utilisant le tableau 6.15. Dans ce tableau, les données de types **Image**, **MediaElement** et **Object** ont des poids positifs car elles sont adaptées aux les tables interactives. Les données de types **Boolean** et **Integer** ont aussi des pondérations positives car elles sont décrites pour composants graphiques qui permettent de définir des valeurs discrètes et continues des contenus qui peuvent être sélectionnés. La sélection est une interaction qui favorise la collaboration contrairement à l'édition de contenus.

6.4.1.3 Évaluation de la conformité aux guidelines

Elle est calculée à l'aide des deux critères présentés précédemment. La fonction

$$\text{GuidelinesRate} : \text{Widget} \rightarrow \text{Integer}$$

Data Type	Contraintes sur le poids	Guidelines
Image	$P_{im} > 0$	G12
MediaElement	$P_{me} > 0$	G12
Object	$P_{ob} > 0$	G12
Boolean	$P_{bl} < 0$	G12
Integer	$P_{in} < 0$	G12
String	$P_{st} < 0$	G12

TABLE 6.15 – Poids des types de données par rapport aux guidelines

permet de faire la somme des fonctions représentant les évaluations selon les PI et les types de données pour prendre en compte la conformité aux guidelines :

$$\forall w \in \{Widget\}, \\ GuidelinesRate(w) = InteractionsWeight(w) + DataTypeWeight(w)$$

6.4.1.4 Remarques

La fonction *GuidelinesRate* permet d'évaluer la conformité des composants graphiques aux guidelines “G11 : Taille des éléments graphiques”, “G12 : Structure et Positionnement des éléments graphiques”, “G13 : Comportement des éléments graphiques” et “G21 : Activités Bloquantes”. Cependant la guideline “G22 : Accessibilité des Menus” ne peut pas être vérifiée car elle est spécifique à un type de composant graphique (menus) dont les comportements sont décrits par la guideline G13.

Les fonctions *InteractionsWeight* et *DataTypeWeight* se basent sur le tableau 6.14 et le tableau 6.15 dont les valeurs des poids de chaque PI ou de chaque type de données peuvent être précisées par la personne en charge de la migration.

La conformité de certains types de données entraîne une charge de travail pour les personnes en charge de la migration car elle nécessite la mise en place de nouvelles ressources. Dans la section suivante nous évaluons cette charge de travail pour le classement des composants graphiques équivalents.

6.4.2 Charge de travail

Le choix d'un composant graphique conforme aux principes des guidelines peut entraîner un ajout d'un connecteur pour l'adaptation de type de données, un ajout de nouvelles données pour l'interface utilisateur, un ajout de codes supplémentaires pour la prise en compte des nouvelles primitives d'interactions ou encore l'utilisation des objets tangibles.

L'ajout d'un connecteur [MMP00] consiste à générer un code qui permet de faire un changement de type entre celui de la source et de la cible, cette tâche est automatisable pour les types de données **Boolean**, **Integer** et **String** et n'entraîne pas une charge de travail pour le programmeur. Les connecteurs sont fournis dans ce cas.

Cependant, si les données de la cible sont de type **Image**, **MediaElement** ou **Object** alors un ajout de données supplémentaires est indispensable. En effet dans ce cas la substitution n'est pas automatisable car l'utilisateur doit trouver ou créer ces données et ensuite faire le ‘mapping’ avec les types de données de départ. Par exemple le remplacement d'un menu classique dont les étiquettes

sont des chaînes de caractères avec un menu dont les étiquettes sont des icônes (Images) nécessite la recherche ou la création de ces icônes.

Par ailleurs, dans le cas où les composants graphiques équivalents ont des PI supplémentaires par rapport à la source, l'implémentation de ces PI est une tâche automatisable pour une bibliothèque graphique car les codes à ajouter correspondent à des PI intrinsèques à implémenter. Par exemple l'ajout de la PI Widget Rotation implique par exemple d'avoir la propriété *canRotate=True* dans le cas de la bibliothèque graphique Microsoft Surface.

L'association d'un objet virtuel ou d'une fonctionnalité avec un objet tangible est une tâche automatisable pour un ensemble de types de composants graphiques. Cependant, il est nécessaire de limiter le nombre d'objets tangibles à utiliser dans une UI pour faciliter la prise en main de l'UI. Le choix des fonctionnalités ou des objets virtuels doit être fait par le concepteur pendant la personnalisation. Le processus propose l'ensemble des objets virtuels et des fonctionnalités utilisables avec des objets tangibles conformément aux règles de concrétisation présentées à la section 6.3 . Cette tâche implique un coût pour les personnes en charge de la migration.

6.4.2.1 Évaluation de la charge de travail

Les critères permettant d'évaluer la charge de travail engendrée par le choix d'un Widget sont :

- la différence entre le type de données de l'instance de la cible et celui de la source⁴⁴ et
- le choix des fonctionnalités et des objets virtuels.

À chacun de ces critères, le programmeur associe un coût en se basant sur ses compétences pour réaliser une tâche. L'estimation de ce coût peut se faire en se basant sur la technique *Pomodoro* [GV08] qui est utilisée en *Extreme Programming* [Bec00] pour permettre aux programmeurs d'optimiser leur temps de programmation en évaluant au mieux le temps des différentes tâches à effectuer.

Au niveau du modèle, les composants graphiques de la source sont des instances de type *Interactor* et les composants graphiques équivalents sont des *Widgets*. La fonction

$$\text{WorkLoad} : \text{Interactor} \times \text{Widget} \rightarrow \{\text{Integer}\}$$

calcule la charge de travail engendrée par le choix d'un *Widget* équivalent à l'*Interactor* de la source. C'est une somme des coûts de chaque critère vérifié par le *Widget* à choisir. Les critères sont :

1. La nécessité des données supplémentaires à la suite d'un changement de type. Ce critère est vérifié si l'*Interactor* et le *Widget* à choisir n'ont pas le même type de données et si le type de données du *Widget* est **Image**, **MediaElement** ou **Object**. Le coût de ce critère est donné par la fonction

$$\text{DataTypeCost} : \text{Interactor} \times \text{Widget} \rightarrow \text{Integer}$$

et à l'aide du tableau ci-dessous qui permet aussi au programmeur de préciser avant la migration le coût de chaque opération.

2. La concrétisation des primitives d'interactions intrinsèques. Ce critère est calculé par la fonction

$$\text{InteractionCost} : \text{Interactor} \times \text{Widget} \rightarrow \text{Integer}$$

en utilisant le tableau 6.16.

Dans le tableau 6.16, le coût de chaque tâche manuelle est estimé par le programmeur avant de commencer une migration. De nouvelles opérations manuelles peuvent être définies, cette liste n'est pas exhaustive.

44. Dans le cas des types Image, MediaElement ou Object

Opérations Manuelles	Coût estimé par le programmeur
Nouvelles ressources de type Image	$C_1 > 0$
Nouvelles ressources de type Media Element	$C_1 > 0$
Nouvelles ressources de type Object	$C_2 > 0$
Sélection des fonctionnalités	$C_3 > 0$
Sélection des objets virtuels	$C_4 > 0$

TABLE 6.16 – Coût des interventions manuelles

La charge de travail est calculée par la fonction :

$$\begin{aligned} WorkLoad(interactor, widget) &= \sum C_i, \\ C_i &\in DataTypeCost(interactor, widget) \cup InteractionCost(interactor, widget) \end{aligned}$$

6.4.3 Algorithme de classement

L'algorithme de classement détermine le rang de chaque *Widget* de la classe d'équivalence d'un *Interactor* en faisant la différence entre la conformité aux guidelines et la charge de travail. Cette différence détermine la valeur de l'objectif d'un *Widget* appartenant à une classe d'équivalence. On suppose que l'ensemble $EquivalentWidget_{interactor}$ contient les *Widgets* équivalents à un *Interactor*.

$$\begin{aligned} \forall i \in EquivalentGroup \cup EquivalentElement, \\ \forall w \in EquivalentWidget_i, \end{aligned}$$

Le ‘meilleur’ *Widget* est obtenu en classant par ordre décroissant la différence $GuidelinesRate(w) - WorkLoad(i, w)$ pour les Widgets équivalents

$$BestEquivalentWidget(i, EquivalentWidget_i) \quad (6.1)$$

$$= \max \left(\bigcup_{w \in EquivalentWidget_i} (GuidelinesRate(w) - WorkLoad(i, w)) \right)$$

6.4.3.1 Exemple de classement

La figure 6.15 représente le modèle de structure de l’UI CBA. Considérons quelques interacteurs de l'exemple décrit à la figure 6.7 ; le modèle d'interacteurs de la figure 6.15 et les *Widgets* de leurs classes d'équivalences sont présentés par le tableau 6.17.

Considérons que le programmeur souhaite utiliser les *Widgets* les plus conformes aux principes des guidelines et pour ce faire fixe la valeur du poids des primitives d'interactions à $P_f = 2$ et les coûts à $C_i = 1$, avec $0 < i < 6$

Le *Widget SurfaceListBox* de la classe d'équivalence de l'*Interactor id=32* a pour type de données *String*.

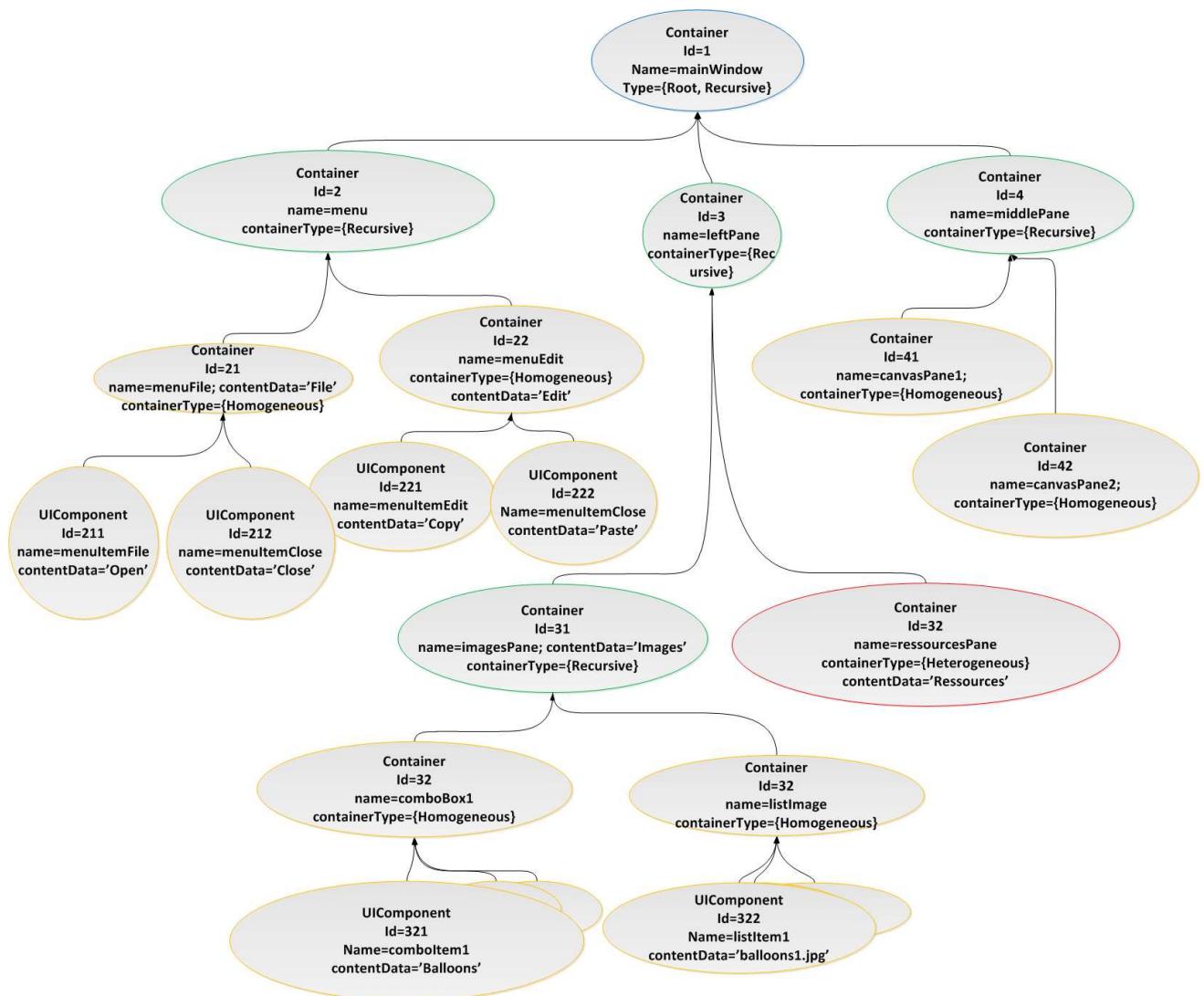


FIGURE 6.15 – Modèle UI CBA

$$\begin{aligned}
 GuidelineRate(SurfaceListBox) &= \left(\begin{array}{l} InteractionWeight(SurfaceListBox) = 0 \\ \quad + \\ DataTypeWeight(SurfaceListBox) = 0 \end{array} \right) cf6.1 \\
 &= \left(\begin{array}{l} 0 \\ \quad + \\ 0 \end{array} \right) \\
 &= 0
 \end{aligned}$$

$$Workload(listBox, SurfaceListBox) = 0 + 0 + 0 + 0 + 0,$$

id, name	$\equiv / \cong_{\text{TargetType}}$	$\leq_{\text{AdditionalPI}} / \lesssim_{\text{AdditionalPI, TargetType}}$	$\geq_{\text{EssentialPI}} / \gtrsim_{\text{EssentialPI, TargetType}}$
id=2, name=menu	- / -	$\text{ScatterView} / -$	- / -
$\text{id=21, name=menuFile}$	- / -	$\text{ElementMenu, SurfaceMenu} / -$	- / -
$\text{id=32, name=listImage}$	- / -	$\text{SurfaceListBox} / \text{LibraryContainer, LibraryBar}$	- / -

TABLE 6.17 – Widgets équivalents

avec $C_i = 0$, $0 < i < 6$ car la source et la cible ont les mêmes types de données.

$$\begin{aligned} \text{GuidelinesRate}(\text{SurfaceListBox}) - \text{Workload}(\text{listBox}, \text{SurfaceListBox}) &= 0 - 0 \\ &= 0 \end{aligned}$$

Les Widgets *LibraryContainer* et *LibraryBar* ont pour type de données *Object*.

$$\begin{aligned} \text{GuidelineRate}(\text{LibraryContainer}) &= \left(\begin{array}{c} \text{InteractionWeight}(\text{LibraryContainer}) \\ + \\ \text{DataTypeWeight}(\text{LibraryContainer}) \end{array} \right) \\ &= \left\{ \begin{array}{l} 0 \\ + \\ 2 \end{array} \right. \\ &= 2 \end{aligned} \tag{6.2}$$

$$\text{Workload}(\text{listImage}, \text{LibraryContainer}) = 0 + 0 + 1 + 0 + 0,$$

avec $C_2 = 1$, et $C_i = 0$, $0 < i < 6$ et $i \neq 2$ car la source est de type *String* et la cible de type *Object* et le coût d'une nouvelle ressource est fixé à 1.

$$\begin{aligned} \text{GuidelinesRate}(\text{LibraryContainer}) - \text{Workload}(\text{listImage}, \text{LibraryContainer}) &= 2 - 1 \\ &= 1 \end{aligned}$$

Le Widget *ScatterView* a en plus les PI **Widget Rotation** et **Widget Move**.

$$\begin{aligned}
 GuidelineRate(\text{ScatterView}) &= \left(\begin{array}{l} \text{InteractionWeight}(\text{ScatterView}) \\ + \\ \text{DataTypeWeight}(\text{ScatterView}) \end{array} \right) \\
 &= \left(\begin{array}{l} 4, \text{ car } \text{InteractionWeight}(\text{ScatterView}) = P_f + P_f = 2 + 2 \\ + \\ 0, \text{ car pas de type de données} \end{array} \right) \\
 &= 4
 \end{aligned} \tag{6.3}$$

$$Workload(\text{mainMenu}, \text{ScatterView}) = 0 + 0 + 0 + 0 + 0,$$

avec $C_i = 0$, $0 < i < 6$ car aucune intervention manuelle n'est nécessaire pour instancier ce container.

$$\begin{aligned}
 GuidelinesRate(\text{ScatterView}) - Workload(\text{menu}, \text{ScatterView}) &= 4 - 2 \\
 &= 2
 \end{aligned}$$

Le Widget *Grid* n'a pas de PI en plus ou des types de données différents

$$\begin{aligned}
 GuidelineRate(\text{Grid}) &= \left(\begin{array}{l} \text{InteractionWeight}(\text{Grid}) \\ + \\ \text{DataTypeWeight}(\text{Grid}) \end{array} \right) \\
 &= \left(\begin{array}{l} 0 \\ + \\ 0, \text{ car pas de type de données} \end{array} \right) \\
 &= 0
 \end{aligned} \tag{6.4}$$

$$\begin{aligned}
 GuidelinesRate(\text{Grid}) - Workload(\text{menu}, \text{Grid}) &= 0 - 0 \\
 &= 0
 \end{aligned}$$

id, name	Widget Equivalent	GuidelinesRate(w)-WorkLoad(i,w)	Rang
id=32, name=listImage	SurfaceListBox	0	3
	LibraryContainer	1	1
	LibraryBar	1	1
id=2 Name=menu	Grid	0	0
	ScatterView	2	1

Remarques

- L'algorithme de classement propose le(s) *meilleur(s)* *Widget(s)* en tenant compte de la conformité aux guidelines et de la charge de travail.
- Les éléments proposés garantissent des comportements, des interactions et une structure qui facilitent la collaboration entre les utilisateurs.

- Dans le cas où l'algorithme de classement propose plusieurs éléments équivalents pour un élément de la source, le processus sélectionne d'abord au hasard un élément équivalent car nous sommes sûrs qu'ils respectent tous les critères fixés. Ensuite l'utilisateur peut modifier l'élément sélectionné automatiquement en se basant sur le style du composant graphique.

6.5 Synthèse

Dans ce chapitre nous avons proposé une interprétation des guidelines en règles de substitution et de concrétisation pour restructurer le modèle de l'UI de départ. Ces interprétations illustrées par les diagrammes 6.2 et 6.3 permettent de décrire de manière formelle les guidelines qui favorisent la collaboration et l'utilisation des objets tangibles.

Nous avons aussi présenté les mécanismes de restructuration du modèle de l'UI source en se basant sur une stratégie de substitution d'éléments graphiques guidée par les guidelines. Le choix des 'meilleurs' composants graphiques est fait grâce à l'algorithme de classement qui prend en compte les guidelines et la charge de travail.

Les transformations des groupes ou des interacteurs simples ont pour but d'accroître l'accessibilité des groupes d'éléments et des fonctionnalités. Les dialogues transformés pendant la migration restent mono-utilisateur pour les interactions en entrée et les activations. Toutefois les interactions en sortie sont des dialogues multi-utilisateurs après la migration. Les transformations des styles sont effectuées manuellement en proposant des dimensions des éléments graphiques.

Le modèle résultant après les substitutions est concrétisé en utilisant les éléments de bibliothèque graphique cible. La personnalisation de l'UI proposée permet de définir d'abord un layout pour les éléments des groupes de modification de contenu (*UpdateGroup*) et pour les groupes mixtes (*Mixed-Group*). Ensuite la personnalisation permet l'association des objets tangibles aux fonctionnalités et aux objets virtuels. Enfin cette phase permet aussi de modifier le style de l'UI en fonction des besoins de l'utilisateur.

Les mécanismes de transformations proposés dans ce chapitre ont pour objectifs de décrire un processus de migration semi automatique. L'une des étapes manuelles de ce processus concerne le positionnement d'éléments graphiques. En effet les personnes en charge de la migration doivent positionner les éléments fils des containers.

Le processus de migration des UI vers les tables interactives que nous proposons se décompose en trois étapes :

1. L'abstraction de la structure et des interactions de l'UI source, cette étape est automatisable pour la plateforme source.
2. La restructuration et la concrétisation du modèle abstrait suivant les règles présentées dans ce chapitre : cette étape comporte d'abord une phase entièrement automatique qui propose une version de l'UI aux concepteurs, ensuite une phase de personnalisation qui consiste à positionner les éléments graphiques, à faire le mapping avec les nouvelles ressources, à associer les objets tangibles aux objets virtuels ou aux fonctionnalités, etc.
3. La génération l'UI cible finale avec le NF source : l'application résultante est exécutable sur la table interactive et cette étape est automatisable pour la plateforme cible.

Dans le chapitre suivant nous validons les mécanismes présentés dans ce chapitre en présentant notre prototype de migration d'une UI desktop vers la table Microsoft PixelSense.

Quatrième partie

Expérimentations

Validations du processus de migration assistée des UI vers les tables interactives

Sommaire

7.1	Introduction	137
7.2	Implémentation du processus de migration	138
7.2.1	Abstraction de l'UI source	139
7.2.2	Proposition d'une version des UI migrées	143
7.2.3	Personnalisation des UI proposées	145
7.2.4	Génération de l'UI finale	148
7.2.5	Remarques	149
7.3	Évaluation	149
7.3.1	Critères d'évaluation de l'approche proposée	150
7.3.2	Résultats	151
7.3.3	Interprétation des résultats	154

7.1 Introduction

Nous proposons un processus semi automatique de migration assistée des UI vers les tables interactives qui est basé sur les PI⁴⁵ et un modèle de structure de l'UI. L'implémentation de notre solution de migration permet de décrire un processus comportant plusieurs étapes.

La première étape consiste à **abstraire le modèle d'une UI**⁴⁶ à partir d'une représentation concrète des applications à migrer. Dans notre cas les codes sources des UI des applications à migrer constituent des éléments concrets de départ et les applications sont décrites selon un modèle d'architecture qui permet une séparation entre l'UI et le NF. La deuxième étape consiste à **proposer une version de l'UI** cible à l'utilisateur. L'UI proposée est constituée des interacteurs équivalents sélectionnés automatiquement par les règles de substitution décrites à la section 6.3. La troisième étape est une **personnalisation de l'UI** proposée dans le but de décrire manuellement les aspects qui ne sont pas pris en compte par le processus automatique. En effet, le positionnement, la taille, la police et la couleur des éléments concrétisés sont à décrire manuellement par le concepteur pendant la personnalisation. Cette phase offre la possibilité aux concepteurs de modifier les interacteurs de l'UI proposée en les remplaçant par les règles de substitution. Enfin la **génération de l'UI finale** est une étape qui consiste à générer le code source exécutable de l'UI finale en faisant le lien avec le NF de départ.

Pour valider ce processus de migration, nous proposons un atelier de migration assistée des UI vers les tables interactives qui est composé d'un mécanisme d'abstraction des UI de départ, un éditeur des UI proposées et un générateur des applications pour la cible.

45. Elles constituent un modèle pour exprimer les dialogues entre les utilisateurs et les systèmes indépendamment des modalités d'interactions

46. Le modèle d'une UI que nous utilisons décrit la structure et les interactions (cf chapitre 5)

Dans ce chapitre nous présentons à la section 7.2 une implémentation du processus de migration des UI vers les tables interactives pour valider notre approche. La section 7.3 est une évaluation des règles de transformation selon différents types d'applications. L'évaluation de l'approche a pour but d'évaluer les interprétations des guidelines dans les règles de transformation. Nous nous basons sur le regroupement, l'accessibilité des éléments graphiques et l'utilisation des objets tangibles.

7.2 Implémentation du processus de migration

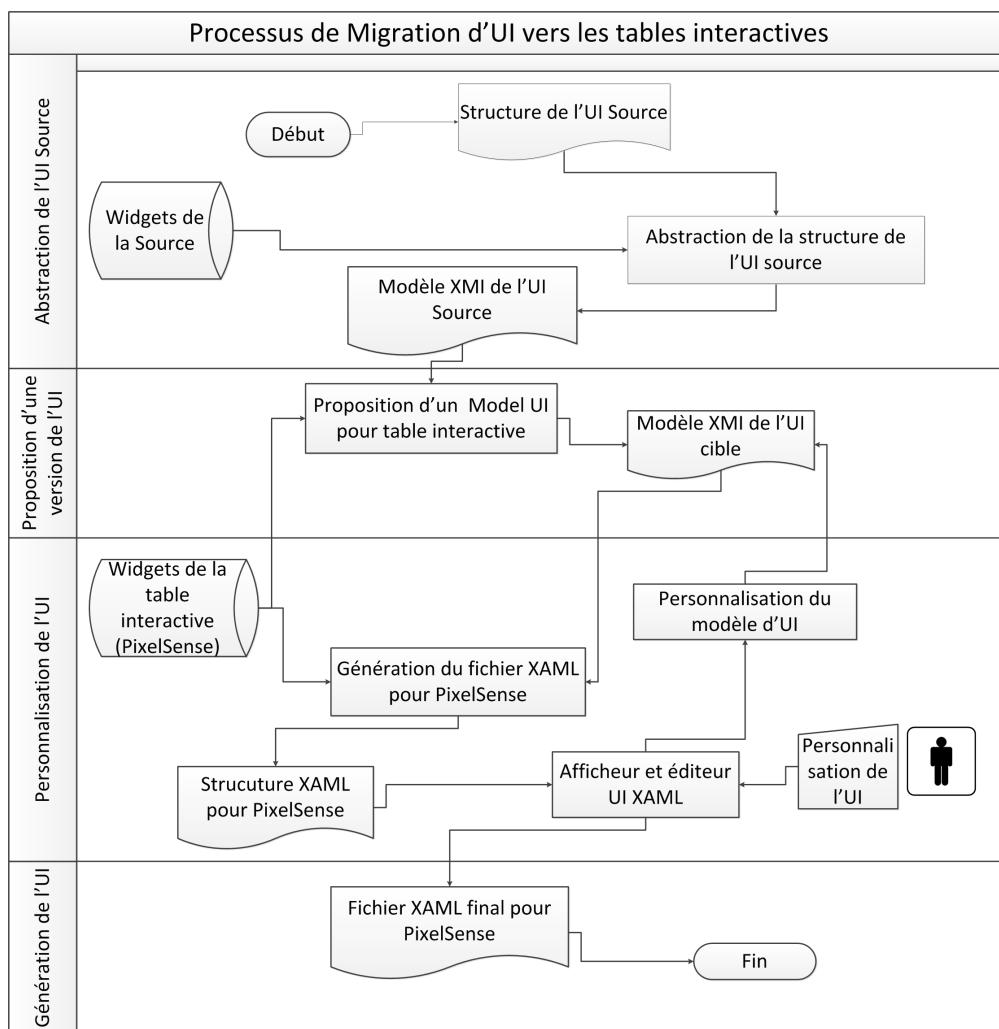


FIGURE 7.1 – Processus semi automatique de migration d'une UI vers les tables interactives

Nous illustrons les différents mécanismes de notre processus de migration des UI par le diagramme de flux (cf figure 7.1).

La phase d'abstraction des UI est décrite à la section 7.2.1. La phase de proposition d'une UI est décrite à la section 7.2.2, la phase de personnalisation des UI proposées est décrite à la section 7.2.3 et la section 7.2.4 décrit la phase de génération de l'UI finale.

7.2.1 Abstraction de l'UI source

Cette phase a pour objectif de décrire l'UI de l'application de départ avec notre modèle de l'UI. Avant de présenter l'algorithme d'abstraction, nous décrivons à la sous section 7.2.1.1 l'architecture des applications de la source. La sous section 7.2.1.2 présente les algorithmes d'abstractions et nous présentons un exemple de modèle de l'UI à la sous section 7.2.1.3

7.2.1.1 Architecture des applications à migrer

Les applications à migrer sont décrites selon un modèle d'architecture MVC [KP⁺88]. Nous choisissons ce modèle car il permet une séparation entre l'UI et le NF d'une application et par rapport à ARCH [Dev92] ou à PAC-Amodeus [Nig94], l'architecture MVC est implémentée à l'aide de plusieurs technologies largement répandues (ASP .Net, JSF, Struts, Spring MVC, etc.).

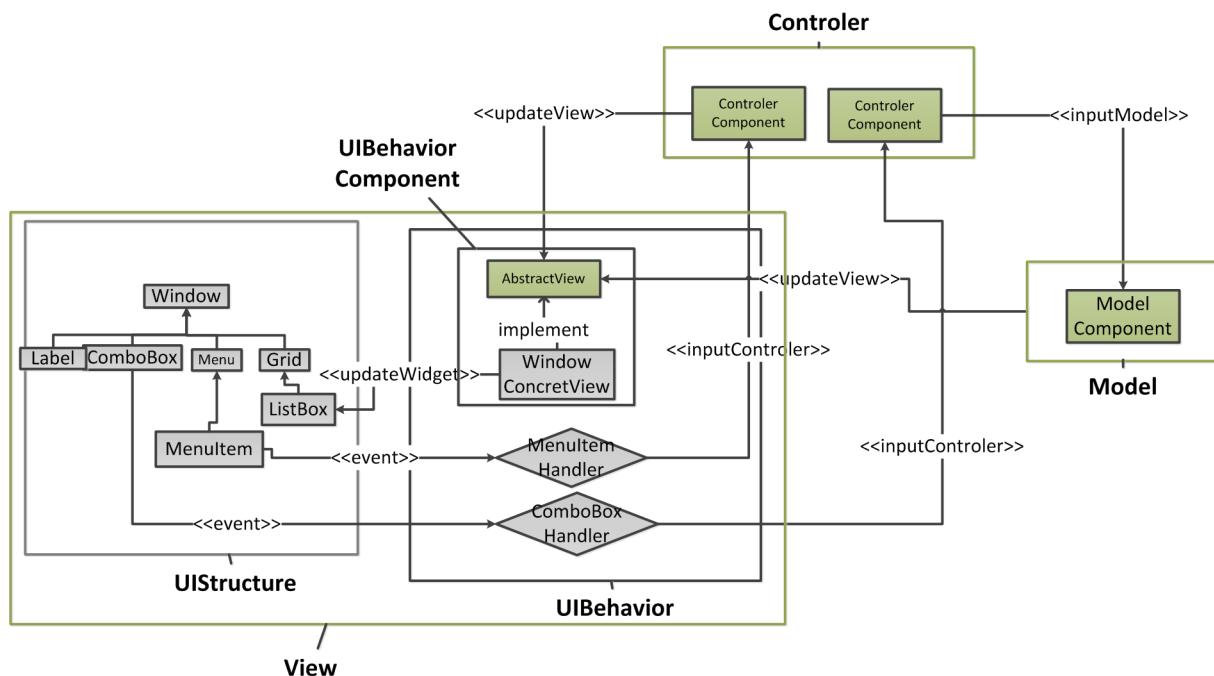


FIGURE 7.2 – Architecture des applications migrées vers les tables interactives

L'architecture MVC que nous décrivons à la figure 7.2 est un choix de l'implémentation de notre prototype qui permet de décrire cadre pour les applications à migrer. L'objectif de cette architecture est de faciliter la migration de la vue sans modifier le contrôleur et le modèle de l'UI de départ.

Le modèle est représenté sur la figure 7.2 par le cadre vert intitulé *Model* et comporte des composants logiciels qui peuvent faire appel à la vue pour le mettre à jour. Les flèches étiquetées «*updateView*» constituent des interactions en sortie.

Le contrôleur est illustré par le cadre vert intitulé *Controller*. Le contrôleur fait appel au modèle par des interactions en entrée illustrées par les flèches étiquetées «*inputModel*». Par ailleurs, les contrôleurs font appel aussi à la vue grâce aux flèches étiquetées «*updateView*» pour mettre à jour des données ou pour afficher des éléments graphiques par exemple.

La vue est représentée par un cadre intitulé *View*, elle est constituée de deux parties : la partie décrivant la structure, le positionnement et le style (*UIStructure*) et la partie décrivant les comportements (*UIBehavior*).

ments (*UIBehavior*) représentant les interactions en entrée et en sortie. La partie *UIBehavior* de la vue dispose d'une interface (*AbstractView*) qui décrit les méthodes de mise à jour de la vue appelées par le modèle ou le contrôleur. Cette interface est implémentée par des classes de type *ConcretView* qui utilisent les éléments de la bibliothèque graphique de départ. La migration de l'UI dans ces conditions consiste à remplacer les classes concrètes, les événements et la structure de l'UI de départ par les éléments de la bibliothèque d'arrivée.

Les rectangles en vert représentent les éléments des applications qui ne sont pas modifiés pendant la migration. Les éléments du modèle et du contrôleur des applications de départ sont conservés et réutilisés pour les applications cibles.

UIStructure Cette partie de la vue est constituée des instances des éléments de la bibliothèque graphique de la plateforme de départ. Notre implémentation considère les applications WPF, dans ce cas la structure, le positionnement et le style sont représentés par des éléments décrits en XAML.

Les flèches étiquetées «*event*» de notre architecture (cf la figure 7.2) représentent les interactions en entrée sur chaque composant graphique. Les «*event*» représentent les événements implementés (*ImplementedEvents*) d'un modèle de la structure d'une instance des UI (cf figure 5.5). Les PI effectives sont identifiées à partir des événements implementés selon les règles décrites à la section 5.3.2.6.

UIBehaviour Cette partie de la vue est constituée des handlers des événements déclenchés par les éléments de la structure et par les méthodes de mise à jour des données d'applications présentées par la structure. Les interactions d'activation ou d'entrée vers le NF sont représentées par les flèches «*inputController*», ces interactions sont représentées par la PI **Activation** dans notre modèle. En effet les appels de méthodes appartenant aux contrôleurs constituent des interactions en entrée. La section A.2.6 de l'annexe A décrit le comportement de l'UI de l'application CBA.

Les interactions en sortie du NF sont représentées par les flèches «*updateView*» (de l'architecture décrite à la figure 7.2), ces interactions sont abstraites en PI **Data Display** et **Widget Display** dans notre modèle. La notification de la vue peut être faite par le modèle ou par le contrôleur ; dans les deux cas, il existe une vue abstraite (*AbstractView*) qui décrit les méthodes de notification de la vue indépendamment des composants graphiques.

L'interface *AbstractView* fait partie des éléments de la vue qui ne sont pas modifiés pendant la migration de l'UI. Elle décrit les méthodes appelées par le modèle ou le contrôleur. Elle permet donc de faire lien entre l'UI et le NF. Le listing A.3 de l'annexe A présente un exemple de vue abstraite.

Les flèches étiquetées «*updateWidget*» de l'architecture décrite à la figure 7.2 représentent les implémentations des méthodes de l'interface *AbstractView*. Ces interactions représentent les modifications des données d'application ou les changements des propriétés graphiques d'une UI à partir du contrôleur ou du modèle. La section A.2.7 de l'annexe A présente un exemple d'*AbstractView*.

Remarques

- L'architecture MVC que nous avons présentée nous permet de distinguer les éléments à modifier pendant la migration des éléments à conserver des UI de la source.
- Les algorithmes d'abstraction parcourrent les éléments à modifier et particulièrement les éléments de type *UIStructure*, *UIBehavior* et les implémentations des vues abstraites pour extraire la structure et les PI effectives.
- On considère qu'une *UIStructure* est une structure analysable qui est constituée d'un ensemble

d'arbres (*Tree*) qui décrit les différentes fenêtres de l'UI à migrer.

$$UIStructure = \bigcup_{i=0}^N Tree_i, \quad Tree_i = \langle root, Node \rangle$$

7.2.1.2 Mécanismes d'abstraction

L'algorithme 2 parcourt toutes les structures représentées par des arbres et applique un algorithme *DFSAbstraction* pour parcourir l'arbre en profondeur.

Algorithme 1 : DFSAbstraction

Données : *WidgetName* : ensemble des noms des widgets d'une bibliothèque graphique,
TabImpEvent : Tableaux de correspondances des ImplementedEvent, *EffPIIdRule* :
Règles d'identification des PI effectives

Entrées : *Tree, root*

Sorties : *interactor : Interactor*

si *root.child! = Null* $\wedge \exists c_i \in root.child \wedge c_i.name \in WidgetName **alors**$

- pour** *c_i ∈ root.child faire*
 - si** *NotMarked(c_i) alors*
 - DFSAbstraction(Tree, c_i)*
 - fin**
- fin**

//La racine de l'arbre est un container si les fils sont des Widgets //Initialiser *interactor* avec un *Container* à partir de *root* et des tableaux de correspondances de la bibliothèque cible;
interactor $\leftarrow InitializeContainer(root, TabImpEvent);$

sinon

- //La racine est un interacteur simple sans fils, initialiser *interactor* avec un //UIComponent à partir de *root* et des tableaux de correspondances de la bibliothèque cible;
interactor $\leftarrow InitializeUIComponent(root, TabImpEvent);$

fin

si *interactor != Null* **alors**

- //Appliquer les règles d'identification des PI effectives (cf section 5.3.2.6);
interactor.EffectivePI.add(FoundEffectivePI(interactor, EffPIIdRule))

fin

Mark(root);

L'algorithme *DFSAbstraction* parcourt en profondeur la structure de l'arbre représentant l'UI de départ. Cet algorithme marque en premier lieu tous les fils d'un container dans le but de déterminer son type (Homogène, Hétérogène, Récursif ou Racine).

Algorithme 2 : Abstraction

Données : *AUIsource* : AUIStructure (Modèle de structure de l'instance de l'UI source)

Entrées : *UIstructure* : Structure de l'UI de départ

Sorties : *ModelSource* : Modèle de l'UI source (fichier XMI décrivant le modèle)

pour *Tree_i* ∈ *UIstructure faire*

//Parcourir chaque nœud racine de la structure de l'UI source

pour *root_i* ∈ *Tree_i* **faire**

| *AUIsource.contains* ← *DFSAbstraction(Tree_i, root_i)*;

fin

fin

ModelSource ← *SerializeXMI(AUIsource)*

7.2.1.3 Applications

Nous avons implémenté les modèles des *Widgets* et des *Interactors* et les PI à l'aide du framework de modélisation EMF [Fou13]. EMF offre la possibilité de générer les codes de manipulation de ces modèles. Dans notre cas, les manipulations du modèle de *Widgets* consistent à :

- vérifier si un élément du code source appartient à une bibliothèque graphique,
- instancier un composant graphique dans une UI finale à partir d'un Widget équivalent.

Cette dernière manipulation est utilisée pour concrétiser un composant graphique à partir de son type.

Les manipulations concernant les *Interactors* consistent à :

- instancier un *Container* ou un *UIComponent* à partir d'une UI finale, en précisant les valeurs des attributs.
- remplacer un interacteur par un autre dans la structure de l'UI en appliquant les règles de substitution.
- comparer un Interacteur avec un Widget en prenant en compte les PI et les types de données pendant les recherches d'équivalences.

Nous avons appliqué l'Algorithme 2 Abstraction sur l'UI de l'application CBA. Le modèle d'une UI résultant est décrit par le listing 7.1 représentant le modèle au format XMI.

Listing 7.1 – Modèle d'une UI

```

1  <?xml version="1.0" encoding="ASCII"?>
<AUIStructure name="SelecteurContenuWPF">
<contains name="MainWindow">
  <Container id="1" name="Window0" containerType="Window" >
    <Container id="11" name="MenuPane">
      <EffectivePI>
        <PrimitiveInteractions name="WidgetNavigation"/>
        <PrimitiveInteractions name="WidgetSelection"/>
      </EffectivePI>
    <Container id="111" name="menuFile">
      <Content propertyName="Header" value="File" />
      <UIComponent id="1111" name="Open" >
        <PrimitiveInteractions name="WidgetNavigation "/>
        <PrimitiveInteractions name="WidgetSelection "/>
        <PrimitiveInteractions name="Activation "/>
        <events>
          <ImplementedEvent name="Open_Click" type="Call" >
            <inputDeviceType="DirectManipulationType" >
            <inputDeviceType="SequentialManipulationType" >
          </ImplementedEvent>
        </events>
      </UIComponent>
    <... />
  </Container>
</AUIStructure>

```

11

21

```

31           </Container>
</Container>
<... />
<Container id="4" name="WorkspacePane" >
    <Container id="44" name="Canvas1">
        <PrimitiveInteractions name="WidgetNavigation" />
        <PrimitiveInteractions name="WidgetSelection" />
        <PrimitiveInteractions name="DataMoveIn" />
        <PrimitiveInteractions name="Activation" />
        <events>
            <ImplementedEvent name="Canvas_Drop" type="Change" property=
                Content" >
                <inputDeviceType='DirectManipulationType' >
            </ImplementedEvent>
        </events>
    </Container>
    <... />
41    </Container>
</Container>
</contains>
</AUIStructure>

```

Le modèle d'une UI obtenu après abstraction est transformé pour proposer une première version de l'UI pour la cible. Dans la section suivante nous présentons les différents modules de notre processus de migration.

7.2.2 Proposition d'une version des UI migrées

Le mécanisme de proposition d'une première version de l'UI se base sur un algorithme de génération de modèle d'une UI qui utilise les règles de substitution et l'algorithme de classement des Widgets (cf section 6.4). Ce mécanisme est automatique et prend en entrée un modèle d'une UI.

L'algorithme 3 *DFSProposition* parcourt une structure arborescente de l'UI de départ afin de générer une structure de l'UI cible en utilisant les règles de substitution. Cet algorithme est utilisé par l'algorithme 4 de proposition de modèle pour parcourir toutes les structures d'un modèle source.

Algorithme 3 : DFSProposition

Données : *EquivWidget* : Dictionnaire pour contenir les Widgets équivalents, *TargetLibrary* : Ensemble des Widgets de la cible, *SetofSRule* : Ensemble des règles de substitution

Résultat : *EquivWidget*

Entrées : *AUISource* : *AUIStructure*, *intSource* : *Interactor*, *nodeTarget* : *Container*, *subRule* : *SubstitutionRule*

Sorties : *intTarget* : *Interactor*

```

si nodeTarget = Null alors
|   nodeTarget ← intTarget
fin
si intSource.contains ≠ Null ∧ ∃ci ∈ intSource.contains alors
|   //intSource est un Container
|   pour ci ∈ intSource.contains faire
|   |   si NotMarked(ci) alors
|   |   |   targetParent.contains ← DFSProposition(AUISource, ci, targetParent)
|   |   fin
|   fin
fin
fin
//initialisation de intTarget à partir des widgets équivalent à intSource
si intSource ∉ EquivWidget.keySet() alors
|   EquivWidget.put(intSource, Null);
|   ∀widgeti ∈ TargetLibrary;
|   intFromWidgeti ← InitializeFromWidget(widgeti);
|   si (∃sr ∈ SetofSRule, intSource ∈ sr.leftMember, intFromWidgeti ∈ sr.rightMember) ∧
|   (∃op ∈ sr.operators, intSourceop intFromWidgeti) alors
|   |   EquivWidget.get(intSource).add(widgeti);
|   fin
fin
intTarget ← InitializeFromWidget(BestEquivalentWidget(EquivWidget(intSource))) ;
Mark(intSource);

```

Algorithme 4 : Proposition du Modèle UI

Entrées : *ModelSource* (fichier au format XMI)

Sorties : *ModelTarget* (fichier au format XMI)

```

AUISource ← DeserializeXMI(ModelSource) ;
AUITarget ← InitilizeModel() ;
pour Treei ∈ AUISource.contains faire
|   //Parcourir chaque nœud racine de la structure de l'UI source
|   pour rooti ∈ Treei faire
|   |   AUITarget.contains.add(DF SProposition(AUISource, rooti, Null)) ;
ModelTarget ← SerializeXMI(AUITarget) ;

```

7.2.3 Personnalisation des UI proposées

La phase de personnalisation des UI proposées est constituée des algorithmes et des processus suivants :

- L'afficheur et l'éditeur de l'UI proposée permettent de modifier les substitutions effectuées automatiquement et de définir le positionnement et le style des UI proposées. Les modifications qui concernent le style et le positionnement sont appliquées uniquement sur la structure XAML de l'UI au niveau de l'afficheur, elles n'impactent pas le modèle de l'UI.
- Le mécanisme de modification du modèle d'une UI cible prend en compte les actions de l'éditeur liées à la structure ou aux PI.
- Les algorithmes de génération de la structure de l'UI cible utilisent sa bibliothèque graphique. Ils sont constitués de l'algorithme 5 et de l'algorithme 6

Algorithme 5 : ConcrétisationXAML

Entrées : *ModelTarget* : FichierXMI, *subRule* : SubstitutionRule

Sorties : *UITarget* : $\bigcup_{i=0}^N Tree_i$

AUITarget \leftarrow Deserialize(*ModelTaget*);

pour $\forall window \in AUITarget.contains$ faire

pour $\forall interactor_i \in window.contains$ faire

Tree_i \leftarrow DFSConcretisation(*interactor_i*)

Algorithme 6 : DFSConcretisation

Données : *EquivWidget* : Dictionnaire pour contenir les Widgets équivalents,

SetofCRule : Ensemble des règles de concrétisation, *nodeXaml* : représente un composant graphique de l'UI finale

Entrées : *window* : Container,

Sorties : *nodeXaml* : structure des éléments en XAML représentant le container fourni en entrée

si *window.contains* \neq Null **alors**

//intSource est un Container

pour *interactor_i* \in *window.contains* faire

si NotMarked(*interactor_i*) **alors**

nodeXaml.child.add(DFSConcretisation(interactor_i))

fin

fin

fin

//initialisation de *nodeXAML*

si $\exists rule \in SetofCRule, window \in rule.leftMember$ **alors**

nodeXaml \leftarrow InitializeFromInteractor(*window, rule*)

fin

Mark(window);

7.2.3.1 Afficheur et éditeur des UI proposées

- Ce mécanisme affiche la structure XAML de l'UI proposée en instanciant les composants graphiques avec des tailles prédéfinies. Le positionnement des éléments de l'UI proposée est défini par la personne en charge de la migration.
- L'éditeur des UI proposées permet d'effectuer les actions suivantes :
 - Sélectionner et positionner des éléments graphiques
 - Définir les tailles, les couleurs et les polices des éléments graphiques
 - Substituer les éléments graphiques en utilisant des Widgets équivalents
 - Supprimer des éléments graphiques
 - Associer les éléments graphiques aux objets tangibles en utilisant des Tags
 - Concrétiser l'UI personnalisée.

Implémentation de la sélection et du déplacement des éléments graphiques Nous avons implémenté cette fonctionnalité en décrivant un cadre pour chaque composant graphique affiché par notre éditeur. Les cadres sont aussi des composants graphiques qui implémentent des événements pour sélectionner un composant graphique de l'UI à migrer. Ces cadres permettent aussi de modifier la position d'un élément graphique en changeant la valeur des propriétés de positionnement des éléments qu'elles contiennent. La figure 7.3 présente une illustration d'un cadre contenant un bouton sélectionné.

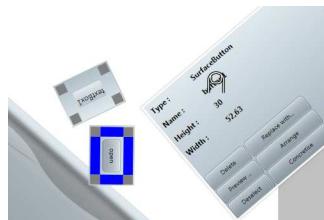


FIGURE 7.3 – Exemple de cadre de sélection des éléments graphique dans l'éditeur du prototype

Implémentation de la modification du style Cette fonctionnalité permet à la personne en charge de la migration de modifier la taille (largeur et hauteur) d'un élément graphique après l'avoir sélectionné. Les cadres qui contiennent les éléments graphiques à éditer permettent à la personne en charge de la migration de redimensionner les composants graphiques par un geste de déplacement d'un coin gris du cadre sélectionné. Cette fonctionnalité permet de définir une couleur pour les éléments graphiques sélectionnés.

Implémentation de la substitution d'un élément graphique Cette fonctionnalité permet de remplacer un composant graphique par les éléments de sa classe d'équivalence. La substitution se fait aux niveau des instances (structure XAML de l'UI) et elle est répercutée au niveau du modèle de l'UI. Nous illustrons à la figure 7.4 un exemple de substitution d'un *LibraryBar* avec un composant graphique *SurfaceListBox*.

Implémentation de la suppression d'un élément graphique Cette fonctionnalité a pour objectif de permettre aux concepteurs de modifier l'UI d'arriver en supprimant les composants graphiques qu'ils

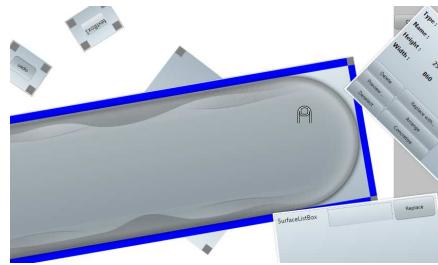


FIGURE 7.4 – Exemple de substitution de *LibraryBar* par *SurfaceListBox*

jugent non indispensables sur les tables interactives. Par exemple une barre de menus qui comportent des raccourcis peut être supprimer si le menu principal est plus accessible après la migration.

Implémentation de l'ajout d'un objet tangible Cette fonctionnalité permet à la personne en charge d'associer un tag pour afficher/cacher un groupe d'éléments graphiques (menu, formulaire, etc). L'éditeur permet d'abord aux concepteurs de sélectionner le type de tags à associer à un groupe d'éléments graphiques. Ensuite il précise le numéro de série et le système se charge de vérifier si le tag associé est déjà utilisé dans l'UI en cours de migration. Cette vérification se fait en parcourant la structure XAML présente dans l'éditeur. En effet la bibliothèque graphique de la table PixelSense permet grâce au composant graphique *TagVisualizer* d'associer un numéro de tag à un groupe d'éléments graphiques. La figure 7.5 présente le menu d'association d'un tag avec un container par exemple. Cette fonctionnalité ajoute un *TagVisualizer* dans la structure XAML affichée par l'éditeur.

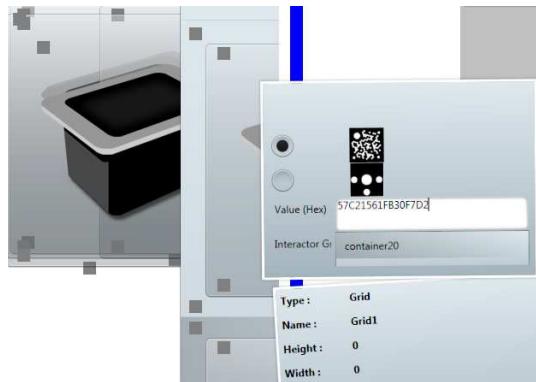


FIGURE 7.5 – Menu d'association d'un tag avec un container

Implémentation de la concrétisation de la structure de l'UI finale Cette fonctionnalité sérialise la structure XAML de l'UI personnalisée à l'aide de l'éditeur. Le mécanisme en charge de la sérialisation enlève les cadres des composants graphiques et génère un fichier XAML qui comporte la position et le style de l'UI cible. Le lien avec les contrôleurs et le modèle est effectué pendant la phase de génération de l'UI finale.

Éditeur graphique du prototype La figure 7.6 ci-dessous présente une capture d'écran de l'éditeur graphique pour personnaliser l'UI proposée. Cette figure présente les composants graphiques de l'UI

CBA migrée pour les tables interactives avant l'introduction du layout et du style.

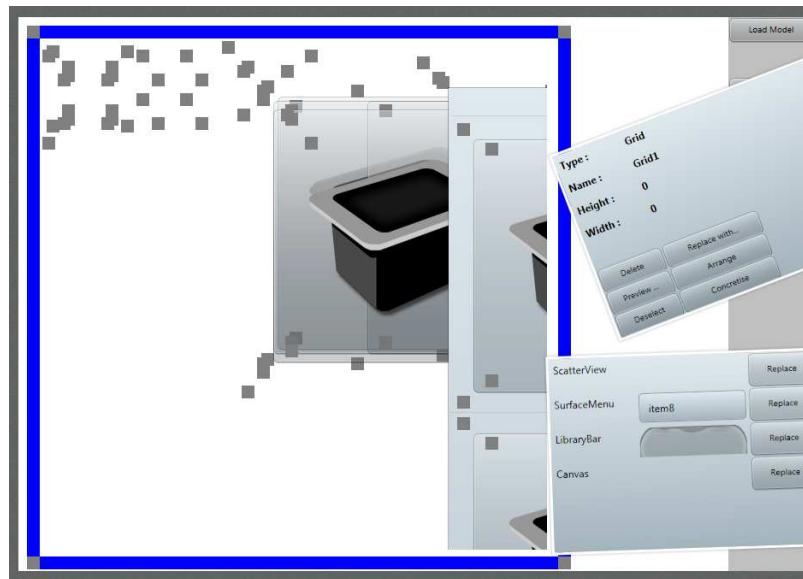


FIGURE 7.6 – Éditeur graphique du prototype

7.2.3.2 Mécanisme de modification du modèle d'une UI

- Ce mécanisme est représenté par le processus “*Personnalisation du Modèle d'UI*” du diagramme de flux de la figure 7.1. Il a pour objectif de garder une cohérence entre le modèle d'une UI et sa représentation graphique dans l'afficheur et l'éditeur.
- Les opérations de substitution ou de suppression des éléments graphiques modifient la structure et les interactions du modèle d'une UI.
- Nous avons implémenté ce mécanisme à l'aide du pattern Observer-Observable. Les actions des utilisateurs dans l'éditeur d'une UI sont toutes observables. Le mécanisme de modification du modèle d'une UI dispose d'un Observer abonné aux actions de l'éditeur.

7.2.4 Génération de l'UI finale

La génération de l'UI finale consiste à sérialiser la structure XAML⁴⁷ de l'UI affichée par l'éditeur et ensuite à générer les fichiers de la partie *UIBehavior* représentés par les handlers et les *ConcreteViews*.

7.2.4.1 Génération de la *ConcreteView*

Cette génération a pour objectif d'obtenir à partir de l'*UIStructure* modifiée pendant la phase de personnalisation l'UI exécutable de la table interactive ciblée.

Les classes représentant les *ConcreteViews* de l'UI de départ implémentent les interfaces de type *AbstractView*. Les méthodes étiquetées «*updateView*» sont implémentées en utilisant les composants

47. Ce mécanisme est décrit à la section 7.2.3

graphiques de la table interactive. Ces méthodes représentent les PI **Data Display** et **Widget Display** aux niveaux de l'UI finale.

Exemple Si un *Interactor* du modèle de l'UI implémente la PI effective **Data Display** ou **Widget Display**, alors il existe une méthode étiquetée «*updateView*» dans la *ConcretView* de l'UI de départ qui doit être modifiée en remplaçant les instances du *Widget* de départ par celles de la table interactive.

Si un *Interactor* représentant un composant graphique *ListBox* est substitué par un *LibraryBar* et si l'*Interactor* implémente une PI **Data Display** et la *ConcretView* de l'UI de départ implémente une méthode nommée *updateList* (qui est étiquetée «*updateView*»), alors la méthode *updateList* sera modifiée pour remplacer toutes les instances du *ListBox* par celles du *LibraryBar* pour générer les *ConcretViews* de la cible. Cette substitution implique un changement de type de données. Il est indispensable de décrire un adaptateur qui permet de transformer les données de type *String* en *Object*.

Le prototype que nous avons mis en place ne génère pas automatiquement les *ConcretViews*. Mais notre modèle de l'UI permet d'identifier les classes et méthodes de l'UI de départ qui doivent être modifiées manuellement pour obtenir l'UI finale.

7.2.4.2 Génération des handlers

Les handlers constituent une implémentation des PI en entrée. Ils permettent de faire le lien entre l'UIStructure et les contrôleurs.

La classe contenant les handlers est générée à partir du modèle de l'UI et en considérant les *ImplementedEvents* de chaque *Interactor* du modèle de l'UI. En effet, les *eventTypes* Call, Change et Select des *ImplementedEvents* du modèle de l'UI sont traduits en handlers dans un fichier C#.

Exemple Si un *Interactor* représentant un *SurfaceListBox* a un *ImplementedEvent* Call et Select , alors un événement *SelectionChanged* peut être créé et sont contenu correspond à l'événement du Widget de l'UI de départ. Les événements font appel à des méthodes du contrôleur.

7.2.5 Remarques

L'architecture détaillée des applications à migrer que nous avons présentée dans cette section permet d'identifier les éléments de l'UI à abstraire et à transformer. Notre modèle d'une UI permet une abstraction de la partie UIStructure de la vue. La partie UIBehavior permet d'identifier les PI effectives en se basant sur une implémentation les règles d'identification décrites à la section 5.3.2.6.

Nous avons implémenté les différentes règles de transformation présentées à la section 6.3. Les algorithmes de proposition d'une version de l'UI et l'éditeur de l'UI proposée implémentent les règles de substitution et les règles de concrétisation.

La génération automatique des *ConcretViews* et des handlers constituent des perspectives d'implémentations à moyen terme pour améliorer notre prototype de migration.

7.3 Évaluation

L'évaluation présentée dans ce manuscrit a pour objectif d'étudier la pertinence des règles de transformations présentées au chapitre 6. Pour ce faire, nous avons considéré quatre applications desktop avec des structures d'une UI différentes. Le choix de ces applications est motivé par le but d'évaluer le regroupement, l'accessibilité des éléments migrés et aussi d'évaluer l'utilisation des objets tangibles.

- Une application d’agenda de consultation des contacts ; nous considérons cette application car elle comporte des formulaires et des tableaux. Les règles de transformations utilisées par notre processus peuvent identifier et migrer les tableaux et les formulaires de cette application tout en garantissant leur accessibilité.
- Une application d’album photo qui permet de consulter d’un ensemble d’images. Nous considérons cette application car elle décrit des *DisplayGroups* qui contiennent des photos. L’UI produite permet des dialogues multi-utilisateurs.
- Une application de dessin qui offre la possibilité de dessiner à l’aide d’une souris en choisissant la couleur du pinceau. La migration de cette application permet de vérifier que les dialogues de l’application source restent cohérents dans un contexte multi-utilisateurs.
- Une application calculatrice ; la migration de cette application permet de vérifier si les regroupements des éléments graphiques en se basant sur les PI et la structure est pertinents.

Nous caractérisons à la section 7.3.1 les critères de validations. La section 7.3.2 présente les résultats de la validation des quatre applications migrées avec notre approche. La section 7.3.3 est une interprétation de ces résultats.

7.3.1 Critères d’évaluation de l’approche proposée

Les critères sont le regroupement, l’accessibilité des éléments graphiques et l’utilisation des objets tangibles. Ces critères permettent d’évaluer uniquement le comportement de l’approche proposée par rapport à des applications différentes règles de transformations.

Regroupement des éléments graphiques Notre processus se base sur des containers et sur les interactions des éléments des containers pour décrire un regroupement des éléments pendant la migration. Ce critère permet de tester la pertinence de notre regroupement pour des cas différents. Nous pensons que les différents types de regroupement peuvent être qualifiés de trois manières :

- *Pertinent* : si **tous** les regroupements proposés et les substitutions possibles sont conformes aux attentes des concepteurs.
- *Acceptable* : si **au moins la moitié** des regroupements proposés et les substitutions sont conformes aux attentes des concepteurs.
- *Inexploitable* : si **aucune** des propositions ou des substitutions est conforme aux attentes des concepteurs.

Accessibilité des groupes Nous pensons que l’accessibilité des différents types de groupes⁴⁸ peut être qualifiée de trois manières :

- *Totale* : si **tous** les groupes identifiés ont un mouvement de rotation et de déplacement.
- *Partielle* : si **tous** les *ControlGroups* au moins sont accessibles à tous les utilisateurs.
- *Aucune* : si **aucun** des groupes migrés sont accessibles.

Objets Tangibles Ce critère permet d’évaluer le mécanisme d’association des objets tangibles aux fonctionnalités et aux menus. Nous qualifions ce critère selon la pertinence des associations proposées.

- *Pertinent* : si **toutes** les associations proposées peuvent être réalisées sans altérer l’utilisabilité de l’UI.
- *Appréciable* : s’il **existe** des associations proposées qui peuvent être réalisées à la discréption du concepteur.
- *Inutile* : Si la réalisation des associations proposées altère l’utilisabilité de l’UI finale.

48. *ControlGroup*, *DisplayGroup*, *UpdateGroup* cf section 6.3

Nous avons évalué la migration de ces quatre applications en utilisant la fiche d'évaluation décrite à la figure 7.7. Nous décrivons les résultats de cette étude à la section 7.3.2.

Fiche d'évaluation			
Application à migrer :			
	ControlGroup	UpdateGroup	DisplayGroup
Regroupement des composants graphiques	<input type="radio"/> Pertinent <input type="radio"/> Acceptable <input type="radio"/> Inexploitable	<input type="radio"/> Pertinent <input type="radio"/> Acceptable <input type="radio"/> Inexploitable	<input type="radio"/> Pertinent <input type="radio"/> Acceptable <input type="radio"/> Inexploitable
Accessibilité des groupes de composants graphiques	<input type="radio"/> Totale <input type="radio"/> Partielle <input type="radio"/> Aucune	<input type="radio"/> Totale <input type="radio"/> Partielle <input type="radio"/> Aucune	<input type="radio"/> Totale <input type="radio"/> Partielle <input type="radio"/> Aucune
Objets Tangibles	<input type="radio"/> ControlGroup <input type="radio"/> Pertinent <input type="radio"/> Appréciable <input type="radio"/> Inutile	<input type="radio"/> UpdateGroup <input type="radio"/> Pertinent <input type="radio"/> Appréciable <input type="radio"/> Inutile	<input type="radio"/> DisplayGroup <input type="radio"/> Pertinent <input type="radio"/> Appréciable <input type="radio"/> Inutile

FIGURE 7.7 – Fiche d'évaluation de la migration des applications

7.3.2 Résultats

Dans cette section nous présentons les résultats de l'évaluation de la migration de quatre applications.

7.3.2.1 Application Agenda

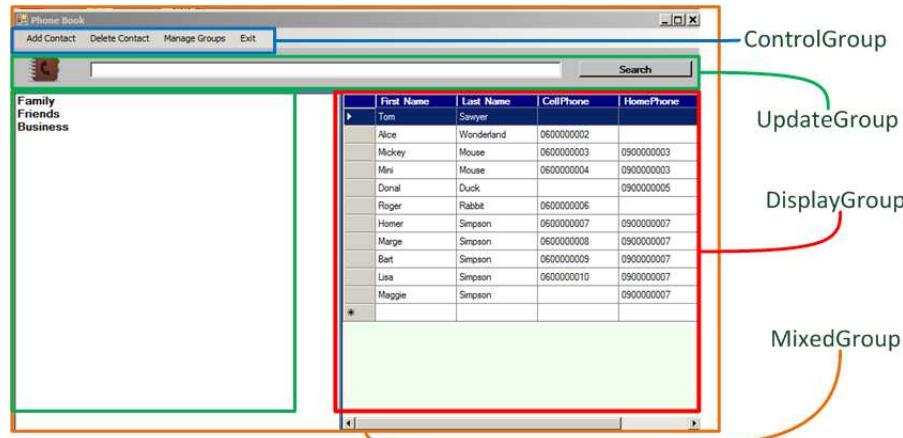


FIGURE 7.8 – Application agenda

Pour la migration de cette UI (cf figure 7.8), le processus de migration propose quatre groupes d'éléments graphiques pour cette UI.

- Le menu principal constitue un *ControlGroup*. Les règles de transformation permettent de choisir un *ElementMenu* par exemple qui sera affiché par un tag lié à un objet tangible. Ce regroupement est *pertinent* car il favorise l'accessibilité des menus.

- Le tableau présentant les contacts constitue un *DisplayGroup*. Les règles de transformation permettent d'utiliser un *LibraryBar* ou un *LibraryContainer* si l'on souhaite garder une présentation structurée des contacts mais avec des données de type Image. Cependant, ce groupe permet aussi de représenter tous les contacts dans un *ScatterView* de manière dispersée ; dans ce cas le panel de recherche et la liste des catégories peuvent être supprimés. Ce regroupement est pertinent car il permet dans les deux cas présentés d'avoir une présentation plus claire des contacts.
- Le panel de recherche des contacts et la liste des catégories constituent des *UpdateGroups* car ils disposent des interactions en entrée. Le panel de recherche est utile dans une présentation structurée des contacts, il facilite l'accès aux données mais dans le cas d'une utilisation à plusieurs il constitue un handicap pour les autres utilisateurs qui doivent attendre. Les regroupements des *UpdateGroups* proposés dans ce cas sont *pertinents* car ils permettent de les identifier afin de les rendre plus accessibles ou pour les supprimer dans certains cas.
- Le *MixedGroup* représente la fenêtre principale de l'UI. Son identification est pertinente car elle permet de savoir les principaux éléments à afficher sur la table interactive.

7.3.2.2 Application Album Photo

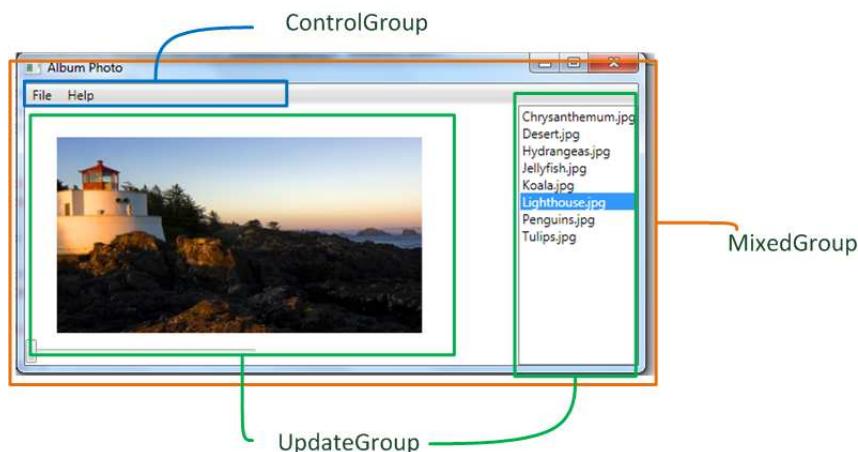


FIGURE 7.9 – Regroupement des éléments de l'application Album Photo

Pour la migration de cette UI (cf figure 7.9), le processus identifie deux *UpdateGroups*, un *ControlGroup* et un *MixedGroup* d'éléments graphiques en se basant sur la structure et les PI.

Le *ControlGroup* représente le menu principal de l'application. Ce regroupement est pertinent car il favorise l'accessibilité des menus.

Le premier *UpdateGroup* est constitué de la liste des images à afficher. Cette liste est chargée en sélectionnant un répertoire qui contient des images. La sélection d'un élément de cette liste permet d'afficher une image. L'identification et les éléments équivalents pour sa substitution qui sont proposés par le processus sont *pertinents*. En effet il est possible de substituer la liste d'images par un *LibraryBar* pour afficher les images dans un carrousel. Un *ScatterView* par exemple permet d'afficher toutes les images de manière dispersée. L'accessibilité des Images est accrue dans le cas d'un *ScatterView* car plusieurs personnes peuvent consulter les images. Le groupe d'affichage de l'image peut être supprimé si l'on choisit un *ScatterView* par exemple.

Le deuxième *UpdateGroup* est constitué de l'image affichée et d'un curseur pour modifier sa taille. L'identification de ce groupe est pertinente car elle permet de le supprimer si nous utilisons un ScatterView par exemple. Cependant les options substitution présentées sont *acceptables* car elles permettent d'avoir un groupe déplaçable mais ne peuvent pas afficher plusieurs images au même moment.

Le *MixedGroup* représente la fenêtre principale de l'UI. Son identification est pertinente car elle permet de savoir les principaux éléments à afficher sur la table interactive.

7.3.2.3 Application de dessin

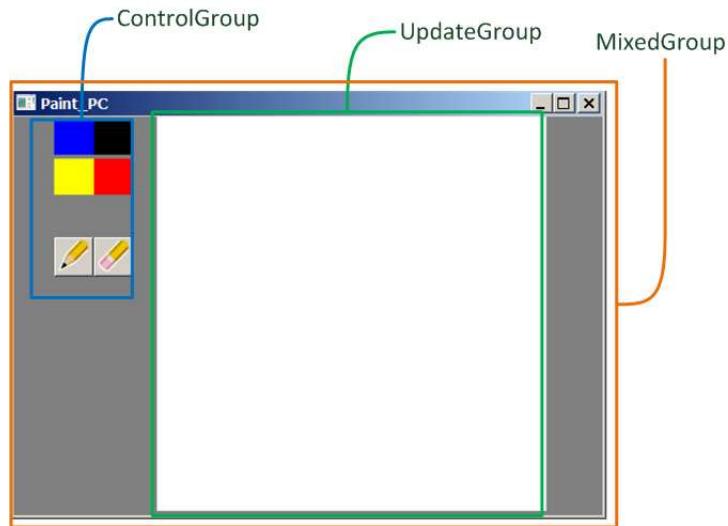


FIGURE 7.10 – Regroupement des éléments de l'application de dessin

Le processus de migration de l'UI de l'application de dessin sur la table Microsoft PixelSense (cf figure 7.10) identifie trois groupes d'éléments graphiques.

Un *ControlGroup* qui représente un panel de couleurs, un crayon et une gomme. Ce regroupement et sa migration sur la table PixelSense en ajoutant les PI de rotation et de déplaçant sont *pertinents* pour cette application. L'utilisation d'un objet tangible par un tag pour afficher le panel est *pertinent* dans cette migration.

Un *MixedGroup* qui représente la fenêtre principale de l'UI, son identification est pertinente car elle permet de savoir les principaux éléments à afficher sur la table interactive.

Un *UpdateGroup* qui représente la zone de dessin de l'application. Les options de substitution proposées pour ce groupe sont acceptables. En effet il est possible de le substituer avec un Grid en le plaçant au centre de l'écran sans possibilité de déplacer cette zone de dessin. Par ailleurs il est possible de placer la zone de dessin dans un ScatterView pour permettre son déplacement. Cependant, l'application dessin que nous avons migrée sur la table PixelSense présente un problème d'utilisabilité dans le cadre multi-utilisateurs. En effet, la zone de dessin permet à plusieurs personnes d'écrire, mais avec un crayon de la même couleur. Un changement de couleur par un utilisateur implique aussi un changement pour les autres utilisateurs.

Cette limite peut être comblée si la table permet d'identifier les utilisateurs comme la table DiamondTouch [DL01] (cf section 3.1.1.1). En effet, l'identification permet d'associer à chaque utilis-

teur un *ControlGroup* constitué d'une palette de couleurs, d'un crayon et d'une gomme. Les changements de couleurs d'un utilisateur n'impacteront pas les autres utilisateurs dans ce cas. Par ailleurs, cette limite s'explique car notre processus de migration ne modifie pas le NF de l'application de départ.

7.3.2.4 Application Calculatrice

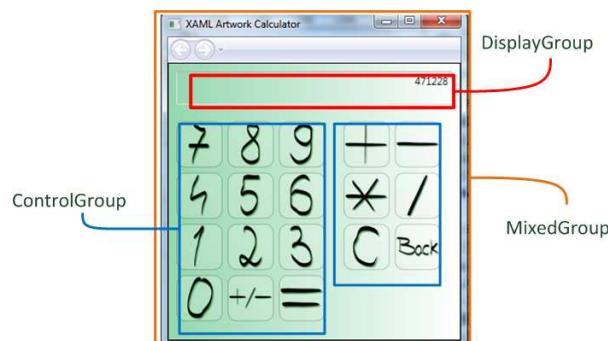


FIGURE 7.11 – Regroupement des éléments de l'application de calculatrice

Le processus de migration de l'UI de cette application de calculatrice sur la table Microsoft PixelSense (cf figure 7.11) identifie quatre groupes d'éléments graphiques.

- Un *MixedGroup* représentant la fenêtre principale de l'UI, son identification est pertinente car elle permet de savoir les principaux éléments à afficher sur la table interactive.
- Un *DisplayGroup* représentant l'écran d'affichage des résultats.
- Deux *ControlGroups* représentant respectivement des chiffres numériques et les opérations de la calculatrice.

Les différents groupes identifiés pour cette application permettent d'avoir une calculatrice comportant trois groupes. Les regroupements proposés dans ce cas ne sont pas pertinents, mais les substitutions permettent une reconstitution d'un groupe. L'accessibilité des éléments est totale en permettant une rotation et un déplacement de la calculatrice. L'utilisation des objets tangibles est *inutile* dans cette migration car le nombre de *ControlGroups* n'accroît pas la charge de travail et les *ControlGroups* sont utilisés à chaque opération de calcul.

7.3.3 Interprétation des résultats

Les validations que nous avons effectuées de notre processus de migration assistée des UI vers les tables interactives, nous permettent d'affirmer que les regroupements et les options de substitution proposées ont pour objectif d'accroître l'accessibilité des éléments graphiques pour favoriser un travail collaboratif.

En effet la figure 7.12 ci-dessous présente les évaluations de la migration des quatre applications présentées à la section 7.3.2. Nous remarquons que les regroupements proposés pour la migration des applications Agenda, Album Photo et Dessin sont pertinents et ils favorisent l'accessibilité des éléments graphiques. Cependant dans le cas de l'application calculatrice les propositions de groupes de départ ne sont pas pertinentes mais les équivalences proposées permettent d'avoir une UI non dispersée et facilement utilisable.

Nous pensons que cette différence par rapport à l'application calculatrice est due à sa simplicité. En effet une calculatrice peut être utilisée par une personne sur une table interactive sans disperser les différents groupes. La flexibilité de notre solution permet à la personne en charge de la migration de disperser ou non les groupes.

L'association des objets tangibles avec des groupes d'éléments graphiques est acceptable car elle a pour objectif d'accroître l'accessibilité des éléments (comme les menus) et aussi de réduire la charge de travail dans le cas d'une UI comportant plusieurs groupes d'éléments graphiques de types différents.

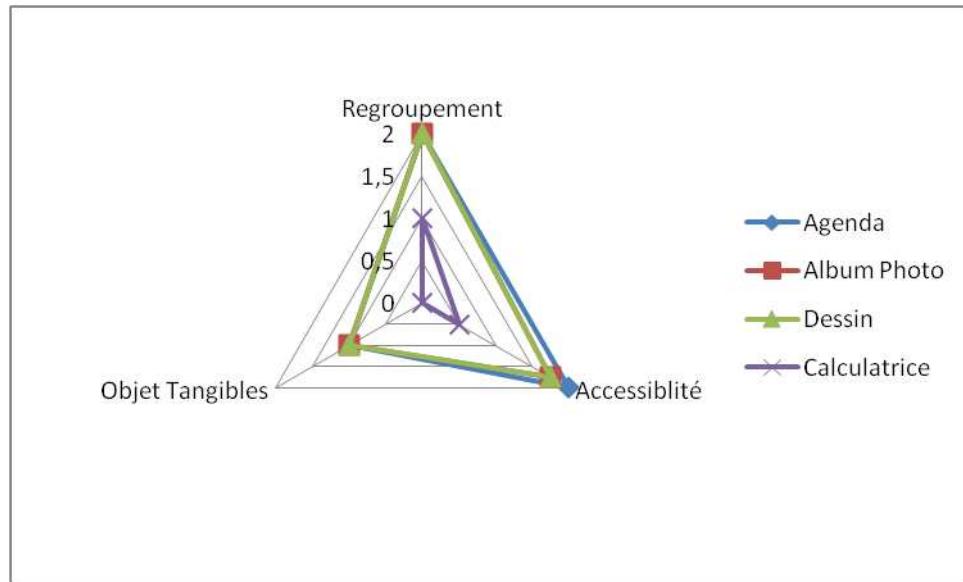


FIGURE 7.12 – Synthèse de l'étude de la migration des quatre applications

Cinquième partie

Conclusions et Perspectives

Conclusions et perspectives

Sommaire

8.1	Synthèse	159
8.2	Perspectives	161
8.2.1	A moyen terme	161
8.2.2	A long terme	161

DANS ce chapitre nous faisons une synthèse des travaux présentés dans cette thèse puis nous identifions quelques perspectives de poursuite de ces travaux.

8.1 Synthèse

Nous avons observé, en étudiant les approches de migration des UI, au chapitre 4 qu'une approche semi automatique est à la fois réutilisable, flexible et permet le respect des critères de conception. Une approche semi automatique de migration d'UI est **réutilisable** si elle dispose des mécanismes de transformations non spécifiques à une application et à une plateforme. La **flexibilité** est le degré d'intervention de l'humain pendant la migration des UI. L'avantage d'une approche flexible de migration des UI est la possibilité pour les concepteurs de personnaliser les UI produites. Cependant les approches très flexibles impliquent des charges de travail importantes pour les concepteurs. Par ailleurs le respect des **critères de conception** permet de produire des UI conformes aux spécificités de la plateforme visée.

Dans cette thèse proposons un atelier de migration assistée des UI desktop vers les tables interactives. Notre solution est basée sur une approche semi automatique de migration des UI utilisant un modèle d'UI. Cette approche a pour but d'adapter les UI de départ conçues pour une personne en UI favorisant la collaboration et l'utilisation des objets tangibles. Dans cette optique, nous avons supposé que les NF des applications de départ seront réutilisés sans modification sur la cible et que l'UI est migrée en considérant la dimension des dialogues, la dimension de la structure et du positionnement et la dimension du style. Ce sont aussi trois dimensions de problèmes liés à la migration des UI vers les tables interactives.

Concernant la dimension qui adresse les problèmes liés à la migration des **dialogues** de l'UI source vers la cible, nos travaux ont pour but de décrire des équivalences entre les dialogues de l'UI de départ et de la cible. En effet les tables interactives proposent des modalités d'interactions différentes de celles d'un desktop par exemple.

Ensuite nous avons identifié la dimension adressant les problèmes liés à la migration de la **structure et du positionnement** des éléments graphiques d'une UI source vers une table interactive. La disposition des éléments graphiques sur les tables interactives doit favoriser une utilisation à plusieurs par exemple. Concernant cette dimension, nos travaux ont pour but de décrire une structure équivalente à celle de départ mais qui prenne en compte les guidelines des tables interactives.

Enfin nous avons décrit la dimension exprimant les problèmes liés à la migration du **style** de l'UI de départ. En effet l'aspect visuel et la taille des éléments graphiques sont importants pour décrire

des interactions pour plusieurs personnes. Concernant cette dimension, les réponses aux problèmes identifiés constituent une perspective importante de nos travaux.

Nous concluons cette synthèse en positionnant de quelle manière les travaux réalisés dans cette thèse permettent de répondre aux questions soulevées par l'espace des problèmes liés à la migration des UI décrit à la section 2.2.

En ce qui concerne la dimension des problèmes liés aux dialogues, les interrogations étaient :

- *Comment utiliser les objets tangibles comme moyen d'interactions ?*

Nous préconisons d'associer les objets tangibles à des groupes d'éléments graphiques ou à des fonctionnalités. Les groupes représentent des menus, des formulaires ou des panels de modification des contenus. Nous identifions ces groupes (ou les fonctionnalités) à l'aide des PI qui permettent de déterminer les interactions effectives de chaque élément d'un groupe (ou d'une fonctionnalité). L'utilisation des objets tangibles pour afficher (ou cacher) des groupes d'éléments graphiques ou pour activer des fonctionnalités permet de décrire des dialogues plus accessibles. Dans notre prototype, les objets tangibles sont utilisés à l'aide des tags. L'association avec des objets virtuels se fait manuellement pendant la phase de personnalisation des UI grâce à un éditeur graphique.

- *Comment transformer les dialogues pour une plateforme multi-utilisateurs et co-localisée ?*

La transformation des dialogues pendant la migration des UI desktop vers les tables interactives impactent l'UI et le NF. Notre contribution ne concerne que la transformation de la partie UI. Nous proposons d'abord les PI qui constituent un modèle pour décrire les dialogues des utilisateurs de manière atomique et indépendamment des modalités d'interactions. Les PI nous permettent de décrire des équivalences entre les modalités d'interactions des différentes plateformes. Elles permettent d'assurer que les dialogues de l'UI de départ sont préservés par notre processus de migration grâce aux opérateurs d'équivalences.

Ensuite pour favoriser la collaboration, nous avons proposé des règles de transformation de l'UI qui permettent de supprimer les activités bloquantes (cf section 6.3). Par exemple l'affichage des boîtes de dialogues des UI desktop ne facilite pas la collaboration sur une table interactive.

- *Comment assurer la cohérence des dialogues après la migration ?*

Pour assurer la cohérence des dialogues après la migration, nous proposons de conserver les dialogues mono-utilisateur pour les groupes qui décrivent des interactions en entrée. En effet la modification d'une donnée ou l'appel d'une fonctionnalité (grâce à un menu) sont des interactions en entrée dont la migration nécessite une modification du NF.

En ce qui concerne la structure des UI par transformation les questions étaient :

- *Comment favoriser l'accessibilité des éléments graphiques ?*

Nous proposons au chapitre 6 des règles de substitution et de concrétisation qui permettent de remplacer les groupes d'éléments par ceux qui ont la capacité d'être déplaçables et utilisable à 360 degré. L'utilisation des objets tangibles comme des moyens d'interactions facilitent l'accès aux composants graphiques des UI migrées sur les tables interactives.

- *Comment garantir l'utilisabilité des UI migrées ?*

Pour aborder ce problème, nous avons considéré dans cette thèse que les règles de transformation de l'UI de départ doivent être basées sur les guidelines pour la migration des UI vers les tables interactives. Nous avons pour cela affiné les guidelines à partir des critères ergonomiques de conception de Scapin [Sca86] et des spécificités des tables interactives.

De manière concrète, les guidelines permettent de transformer la structure des éléments graphiques en garantissant l'**accessibilité** des groupes d'éléments graphiques pertinents. Dans ce cadre les PI et le modèle de structure qui sont proposés, permettent d'établir des équivalences entre les éléments de la plateforme de départ et ceux des tables interactives.

Le modèle de l'UI que nous avons proposé dans ce manuscrit décrit les PI et la structure des UI. En ce qui concerne le positionnement et le style des éléments graphiques, leur migration est manuelle et se fait par le biais d'un éditeur graphique pour les UI destinées aux tables interactives.

En ce qui concerne le style des UI par transformation les questions étaient :

- *Comment prendre en compte la taille de la surface d'affichage d'une table interactive ?*
L'affinement des critères de conception en considérant les propriétés liées au nombre d'utilisateurs et à la taille de l'écran des tables interactives a permis d'identifier des contraintes sur la taille des éléments graphiques. En effet il est indispensable de décrire une borne de valeurs pour les composants graphiques redimensionnables.
- *Comment assurer la cohérence globale des styles migrés ?*
Notre solution permet une introduction manuelle du style des UI de la cible. Les questions liées à la cohérence globale font partie des perspectives.

8.2 Perspectives

Ce travail engendre de nombreuses perspectives à moyen et à long terme.

8.2.1 A moyen terme

Nos travaux bénéficieraient des extensions suivantes à moyen terme :

- **Migration semi automatique du positionnement** : Les questions liées à la transformation de la position des éléments graphiques sont traitées par les personnes en charge de la migration avec l'aide d'un éditeur graphique. Nous pensons qu'il est possible d'identifier des guidelines pour introduire un layout pour les UI des tables interactives pendant les phases de proposition d'UI et de personnalisation. Cette perspective fait partie des extensions pour améliorer l'éditeur de notre prototype de migration des UI vers les tables interactives.
- **Migration semi automatique du style** : elle permet d'améliorer la flexibilité de notre processus de migration des UI de départ. Nous pensons que l'utilisation d'un modèle pour décrire la dimension de style permet de décrire des UI homogènes et réduira davantage le travail des personnes en charge de la migration. L'enjeu majeur de cette extension de notre processus est la manière d'interpréter les guidelines liées aux styles dans un modèle de style pour permettre aux concepteurs de décrire des modèles de styles conformes aux critères ergonomiques des tables interactives.
- **Moteur d'apprentissage des transformations** : il permettra de capitaliser les choix de substitutions et de concrétisations des personnes en charge de la migration en fonction des éléments graphiques et des UI dans le but d'améliorer les UI proposées et de réduire les interventions humaines. L'enjeu de cette amélioration est de sélectionner les meilleurs parmi les éléments équivalents, en tenant compte des choix effectués dans les migrations antérieures et aussi en tenant compte de la configuration de l'UI d'arrivée. Cependant, il est possible que les substitutions et les concrétisations proposées ne soient pas adéquates pour une application, dans ce cas il faut s'assurer que les dialogues de départ et les guidelines soient respectées. Par ailleurs le moteur d'apprentissage ne doit pas exclure la phase de personnalisation des UI mais la réduire.

8.2.2 A long terme

Nos travaux pourraient bénéficier des améliorations suivantes à long terme :

- **Migration des dialogues :** notre solution actuelle ne transforme pas les dialogues mono-utilisateur en dialogues multi-utilisateurs. Cette transformation implique une modification de l'UI et du NF. Pour transformer le NF, il est indispensable de considérer et de décrire l'impact des dialogues sur le NF. En effet la migration des dialogues d'un formulaire ou d'un menu d'une UI desktop vers une table interactive implique de permettre à plusieurs personnes de les utiliser au même moment. Pour ce faire, il est d'abord indispensable d'identifier les éléments de l'UI et du NF qui sont concernés et ensuite d'écrire des règles de transformation qui préservent la cohérence des dialogues. L'enjeu de la migration des dialogues dans ce cadre consiste à modifier aussi le NF de l'UI de départ.
- **Autres plateformes de départ :** notre solution peut être étendue pour prendre en compte d'autres plateformes de départ comme une tablette ou un téléphone portable. Nous avons étudié dans notre solution l'accessibilité des éléments graphiques et des interactions dans le cadre de la migration des UI desktop vers les tables interactives. En considérant d'autres plateformes de départ, comme une tablette par exemple, des enjeux concernant la réutilisabilité des guidelines, des modèles et des règles de transformation de notre solution actuelle peuvent être dégagés. Concernant les guidelines pour la migration des UI, elles sont identifiées en considérant les UI desktops. Dans le cadre des UI pour les tablettes ou les smartphones, il sera indispensable de vérifier si les guidelines actuelles sont réutilisables. Une autre plateforme de départ permettra d'évaluer et de compléter les guidelines que nous avons décrites dans cette thèse.
Par ailleurs, il est intéressant d'évaluer les règles de transformations et le modèle que nous avons proposé dans cette thèse pour d'autres plateformes de départ. Notre modèle décrit la structure et les PI, ce choix ignore le style et le positionnement car nous avons considéré que leurs différences par rapport à celle des tables interactives ne facilite pas une transformation. L'extension de notre solution à d'autres plateformes de départ confirmera cette hypothèse et enrichira notre modèle d'UI.
- **Une autre plateforme d'arrivée :** dans le but de décrire un processus de migration générique et totalement indépendant d'une plateforme de départ et d'arrivée, nous pensons que le remplacement des tables interactives par d'autres plateformes permet de décrire et tester les processus d'affinement et d'interprétation des critères ergonomiques en guidelines puis en règles de transformation.

Comme nous pouvons le constater, de nombreuses pistes, à plus ou moins long terme, peuvent venir compléter les travaux réalisés dans cette thèse.

Bibliographie

- [App95] Apple Computer Inc. *Macintosh Human Interface Guidelines*. Addison-wesley Publishing, 1995.
- [Bar88] Marie F Barthet. Logiciels interactifs et ergonomie. *Paris : Dunod Informatique*, 1988.
- [BCL⁺08] Markus Bischof, Bettina Conradi, Peter Lachenmaier, Kai Linde, Max Meier, Philipp Pötzl, and Elisabeth André. Xenakis : combining tangible interaction with probability-based musical composition. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, TEI '08, pages 121–124, New York, NY, USA, 2008. ACM.
- [BCW⁺06] Doug A. Bowman, Jian Chen, Chadwick A. Wingrave, John F. Lucas, Andrew Ray, Nicholas F. Polys, Qing Li, Yonca Hacihmetoglu, Ji-Sun Kim, Seonho Kim, Robert Boehringer, and Tao Ni. New directions in 3d user interfaces. *IJVR*, 5(2) :3–14, 2006.
- [Bec00] Kent Beck. *Extreme Programming Explained : Embrace Change*. Addison-Wesley Professional, 2000.
- [Ber03] F. Berard. The Magic Table : Computer Vision Based Augmentation of a Whiteboard for Creative Meetings, 2003.
- [Bes10] Guillaume Besacier. *Interactions post-WIMP et applications existantes sur une table interactive*. PhD thesis, UNIVERSITÉ PARIS-SUD 11, 2010.
- [BRNB07] Guillaume Besacier, Gaétan Rey, Marianne Najm, and Stéphanie Buisine. Paper Metaphor for Tabletop Interaction Design. In *HCII'07 Human Computer Interaction International*, pages 758–767, 2007.
- [BS08] Renata Bandelloni and Carmen Santoro. Reverse Engineering Cross-Modal User Interfaces for Ubiquitous Environments. *Work*, 2008.
- [BV02] Laurent Bouillon and Jean Vanderdonckt. Retargeting Web pages to other computing platforms with VAQUITA. *Ninth Working Conference on Reverse Engineering, 2002. Proceedings.*, pages 339–348, 2002.
- [CCB⁺02] Gaëlle Calvary, Joëlle Coutaz, Laurent Bouillon, Murielle Florins, Quentin Limbourg, L. Marucci, Fabio Paternò, Carmen Santoro, N. Souchon, David Thevenin, and Jean Vanderdonckt. CAMELEON Project. Technical report, CAMELEON Project, 2002.
- [Cre01] M. Crease. *A Toolkit of Resource-sensitive, Multimodal Widgets*. University of Glasgow, 2001.
- [D. 06] D. Heinemeier Hansson. World of Resource, 2006.
- [Dev92] UIMS Tool Developers. A metamodel for the runtime architecture of an interactive system : the uims tool developers workshop. *SIGCHI Bull.*, 24(1) :32–37, January 1992.
- [DL01] Paul Dietz and Darren Leigh. Diamondtouch : a multi-user touch technology. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, UIST '01, pages 219–226, New York, NY, USA, 2001. ACM.
- [FCLD12] Paternò Fabio, Santoro Carmen, and Spano Lucio Davide. Concur Task Trees (CTT), 2012.
- [Fou13] The Eclipse Foundation. Eclipse modeling framework project (emf), 2013.

- [FPV95] Christelle Farenc, Philippe Palanque, and Jean Vanderdonckt. User interface evaluation : is it ever usable ? *Advances in Human Factors/Ergonomics*, 20 :329–334, 1995.
- [FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer graphics : principles and practice (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.
- [GH95] Hans-w Gellersen and Hans-W. Gellersen. Modality Abstraction : Capturing Logical Interaction Design as Abstraction from "User Interfaces for All". In *1st ERCIM Workshop on "User Inter- faces for All"*. ERCIM, 1995.
- [GMAD05] Derek Glover, David Miller, Doug Averis, and Victoria Door. The interactive whiteboard : a literature survey. *Technology, Pedagogy and Education*, 14(2) :155–170, 2005.
- [GPF10] André MP Grilo, Ana CR Paiva, and João Pascoal Faria. Reverse engineering of gui models for testing. In *Information Systems and Technologies (CISTI), 2010 5th Iberian Conference on*, pages 1–6. IEEE, 2010.
- [GV08] Federico Gobbo and Matteo Vaccari. The pomodoro technique for sustainable pace in extreme programming teams. In *Agile Processes in Software Engineering and Extreme Programming*, pages 180–184. Springer, 2008.
- [HCT06] P. Hutterer, B.S. Close, and B.H. Thomas. Supporting Mixed Presence Groupware in Tabletop Applications. In *First IEEE International Workshop on Horizontal Interactive Human-Computer Systems (TABLETOP '06)*, pages 63–70. IEEE, January 2006.
- [IU97] Hiroshi Ishii and Brygg Ullmer. Tangible bits. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '97*, pages 234–241, New York, New York, USA, March 1997. ACM Press.
- [JOF11] Cédric JOFFROY. *Composition d'applications et de leurs interfaces homme-machine dirigée par la composition fonctionnelle*. PhD thesis, UNIVERSITÉ DE NICE SOPHIA ANTIPOLES UFR Sciences, 2011.
- [KKM03] M. Kassoff, D. Kato, and W. Mohsin. Creating GUIs for web services. *IEEE Internet Computing*, 7(5) :66–73, September 2003.
- [KLL⁺09] Sébastien Kubicki, Sophie Lepreux, Yoann Lebrun, Philippe Santos, Christophe Kolski, and Jean Caelen. New human-computer interactions using tangible objects : Application on a digital tabletop with rfid technology. In JulieA. Jacko, editor, *Human-Computer Interaction. Ambient, Ubiquitous and Intelligent Interaction*, volume 5612 of *Lecture Notes in Computer Science*, pages 446–455. Springer Berlin Heidelberg, 2009.
- [KODPR12] Andre Kalawa, Occello, Anne-Marie Dery-Pinna, and Michel Riveill. Reusing user interface across devices with different design guidelines. *Knowledge and Systems Engineering, International Conference on*, 0 :211–216, 2012.
- [KP⁺88] Glenn E Krasner, Stephen T Pope, et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3) :26–49, 1988.
- [KSM99a] L Kong, E Stroulia, and B Matichuk. Legacy interface migration : A task-centered approach. . . . of the 8th International Conference on . . . , 1999.
- [KSM99b] Lanyan Kong, Eleni Stroulia, and Bruce Matichuk. Legacy interface migration : A task-centered approach. In *Proc. of 8 th int. conf. on Human-Computer Interaction HCI Internationalâ€TM99 (Munich*, pages 1167–1171, 1999.

- [LL09] Kee Yong Lim and John B Long. *The MUSE method for usability engineering*, volume 8. Cambridge University Press, 2009.
- [Lon10] Nguyen Hoang Long. *Web Visualization of Trajectory Data using Web Open Source Visualization Libraries Web Visualization of Trajectory Data using Web Open Source Visualization Library*. PhD thesis, INTERNATIONAL INSTITUTE FOR GEO- INFORMATION SCIENCE AND EARTH OBSERVATION ENSCHEDE, THE NETHERLANDS, 2010.
- [LVMB05] Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, and Laurent Bouillon. USIXML : A Language Supporting Multi-path Development of User Interfaces. In *Ifip International Federation For Information Processing*, pages 200–220, 2005.
- [McC92] Carma McClure. *The three Rs of software automation : re-engineering, repository, reusability*. Prentice-Hall, Inc., 1992.
- [Mic09] Microsoft. Microsoft Surface User Experience Guidelines, 2009.
- [Mic11] Microsoft. Microsoft Surface 2 Design and Interaction Guide. Technical Report July, Microsoft, 2011.
- [Mic12a] Microsoft. MSDN XAML, 2012.
- [Mic12b] Microsoft WPF. WPF, 2012.
- [Mic12c] Microsoft XNA. XNA, 2012.
- [Mit] Mitsubishi Electric Research Laboratoriesb. MERL .
- [MMP00] Nikunj R. Mehta, Nenad Medvidovic, and Sandeep Phadke. Towards a taxonomy of software connectors. In *Proceedings of the 22nd international conference on Software engineering - ICSE '00*, pages 178–187, New York, New York, USA, June 2000. ACM Press.
- [Moo95] James D. Mooney. Portability and reusability. In *Proceedings of the 1995 ACM 23rd annual conference on Computer science - CSC '95*, pages 150–156, New York, New York, USA, February 1995. ACM Press.
- [Moo96] Melody M Moore. Rule-based detection for reverse engineering user interfaces. In *Reverse Engineering, 1996., Proceedings of the Third Working Conference on*, pages 42–48. IEEE, 1996.
- [MPV11] Gerrit Meixner, Fabio Paternò, and Jean Vanderdonckt. Past , Present , and Future of Model-Based User Interface Development. *i-com*, 10 :2–11, 2011.
- [MR97] Melody Moore and Spencer Rugaber. Using knowledge representation to understand interactive systems. In *Program Comprehension, 1997. IWPC'97. Proceedings., Fifth International Workshop on*, pages 60–67. IEEE, 1997.
- [MVG06] Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. *Electron. Notes Theor. Comput. Sci.*, 152 :125–142, March 2006.
- [MVL06] José Pascual Molina Massó, Jean Vanderdonckt, and Pascual González López. Direct manipulation of user interfaces for migration. In *Proceedings of the 11th international conference on Intelligent user interfaces - IUI '06*, page 140, New York, New York, USA, January 2006. ACM Press.
- [NC93] Laurence Nigay and Joëlle Coutaz. A design space for multimodal systems. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '93*, pages 172–178, New York, New York, USA, May 1993. ACM Press.

- [Nig94] Laurence Nigay. *Conception et modélisation logicielles des systèmes interactifs : application aux interfaces multimodales*. PhD thesis, UNIVERSITÉ JOSEPH FOURIER - GRENOBLE 1, 1994.
- [NM90] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '90*, pages 249–256, New York, NY, USA, 1990. ACM.
- [Pat11] Fabio Paternò. *Migratory interactive applications for ubiquitous environments*. Springer, 2011.
- [PMM97] Fabio Paternò, Cristiano Mancini, and Silvia Meniconi. Concurtasktrees : A diagrammatic notation for specifying task models. In *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction*, volume 96, pages 362–369, 1997.
- [PRI02] James Patten, Ben Recht, and Hiroshi Ishii. Audiopad : a tag-based interface for musical performance. In *Proceedings of the 2002 conference on New interfaces for musical expression, NIME '02*, pages 1–6, Singapore, Singapore, 2002. National University of Singapore.
- [PSS09] Fabio Paternò, Carmen Santoro, and Lucio Davide Spano. MARIA : A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments. *ACM Transactions on Computer-Human Interaction*, 16(4) :1–30, November 2009.
- [PZ10] Fabio Paternò and Giuseppe Zichittella. Desktop-to-mobile web adaptation through customizable two-dimensional semantic redesign. In *Human-Centred Software Engineering*, pages 79–94. Springer, 2010.
- [Que01] Whitney Quesenberry. Building a Better Style Guide. In *Proceedings of the Usability Professionals Association*, pages 1–10, 2001.
- [RBB⁺95] L Alperin Resnick, Alex Borgida, Ronald J Brachman, Deborah L McGuinness, Peter F Patel-Schneider, C Isbell, and K Zalondek. CLASSIC Description and Reference Manual For the COMMON LISP Implementation. *AI Principles Research Department, AT&T Bell Laboratories*, 1995.
- [RCPsC05] Nicolas Roussel, Olivier Chapuis, Université Paris-sud, and Orsay Cedex. Metisse : un système de fenêtrage hautement configurable et utilisable au quotidien. In *IHM '05 : Proceedings of the 17th international conference of the Association Francophone d'Interaction Homme-Machine*, 2005.
- [Res13] Microsoft Research. NUI : Natural User Interface, 2013.
- [RFJ08] Daniel Ratiu, Martin Feilkas, and Jan Jurjens. Extracting Domain Ontologies from Domain Specific APIs. *2008 12th European Conference on Software Maintenance and Reengineering*, 1 :203–212, 2008.
- [RSP11] Yvonne Rogers, Helen Sharp, and Jenny Preece. *Interaction design : beyond human-computer interaction*. Wiley, 2011.
- [SC12] Carlos Eduardo Silva and José Creissac Campos. Can GUI Implementation Markup Languages Be Used for Modelling ? In *Human-Centred Software Engineering*, pages 112–129, 2012.
- [Sca86] Dominique L. Scapin. Guide ergonomique de conception des interfaces homme-ordinateur. Technical report, Institut National de Recherche en Informatique et en Automatique, 1986.

- [SVFR04] Chia Shen, Frédéric D. Vernier, Clifton Forlines, and Meredith Ringel. DiamondSpin : an extensible toolkit for around-the-table interaction. In *Proceedings of the 2004 conference on Human factors in computing systems - CHI '04*, pages 167–174, New York, New York, USA, April 2004. ACM Press.
- [SWND03] Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis. *OpenGL programming guide : the official guide to learning OpenGL, version 1.4*. Addison-Wesley Professional, 4th ed edition, 2003.
- [TC02] David Thevenin and Joëlle Coutaz. Adaptation des IHM : Taxonomies et Archi. Logicielle. In *IHM 2002*, pages 207–210, 2002.
- [TKB78] Andrew Tanenbaum S., Paul Klint, and Wim Bohm. Guidelines for Software Portability. *Software-Practice And Experience*, 8(6) :681–698, November 1978.
- [TKR10] Thiago Tonelli, Krzysztof, and Ralf. Swing to swt and back : Patterns for api migration by wrapping. In *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, ICSM '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.
- [TS99] K. Tucker and R. E. Kurt Stirewalt. Model based user-interface reengineering. In *Proceedings of the Sixth Working Conference on Reverse Engineering*, WCRE '99, pages 56–, Washington, DC, USA, 1999. IEEE Computer Society.
- [UI97] Brygg Ullmer and Hiroshi Ishii. The metaDESK : Models and Prototypes for Tangible User Interfaces. In *UIST '97 Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 223 – 232, 1997.
- [USJ⁺08] Brygg Ullmer, Rajesh Sankaran, Srikanth Jandhyala, Blake Tregre, Cornelius Toole, Karun Kallakuri, Christopher Laan, Matthew Hess, Farid Harhad, Urban Wiggins, and Shining Sun. Tangible Menus and Interaction Trays : Core tangibles for common physical / digital activities. In *TEI '08*, pages 209–212. ACM, 2008.
- [Van97] Jean Vanderdonckt. Conception assistée de la présentation d'une interface homme-machine ergonomique pour une application de gestion hautement interactive. *Doctoral dissertation. Facultés Universitaire Notre-Dame de la Paix, Namur*, 1997.
- [vD97] Andries van Dam. Post-WIMP user interfaces. *Communications of the ACM*, 40(2) :63–67, February 1997.
- [vdVNHF11] Bram JJ van der Vlist, Gerrit Niezen, Jun Hu, and Loe MG Feijs. Interaction primitives : Describing interaction capabilities of smart objects in ubiquitous computing environments. In *AFRICON, 2011*, pages 1–6. IEEE, 2011.
- [VLM⁺04] Jean Vanderdonckt, Quentin Limbourg, Benjamin Michotte, Laurent Bouillon, Daniela Trevisan, and Murielle Florins. USIXML : a User Interface Description Language for Specifying Multimodal User Interfaces The Reference Framework used for Multi-Directional UI Development. *Language*, pages 19–20, 2004.
- [W3C03] W3C. W3C Multimodal Interaction Framework, 2003.
- [WB03] Mike Wu and Ravin Balakrishnan. Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. In *Proceedings of the 16th annual ACM symposium on User interface software and technology - UIST '03*, pages 193–202, New York, New York, USA, November 2003. ACM Press.
- [Weg97] Peter Wegner. Why Interaction Is More Powerful Than Algorithms, 1997.

- [WGM08] Xin Wang, Yaser Ghanam, and Frank Maurer. From Desktop to Tabletop : Migrating the User Interface of AgilePlanner. In *Engineering Interactive Systems 2008*, pages 263–270, 2008.

Appendices

Application des modèles de l'UI et des PI

Cette annexe a pour objectif de présenter d'abord les tableaux de correspondances pour décrire les éléments d'une bibliothèque graphique à l'aide du modèle de type de composants graphiques présenté au chapitre 5. Elle présente ensuite les ressources utilisées par les mécanismes d'abstraction d'une UI finale vers les modèles d'instance. Enfin, cette annexe présente l'application des règles d'identification des PI pour les bibliothèques XAML et XAMLSurface.

A.1 Comment décrire une instance du modèle de type de composants graphiques ?

Nous décrivons dans cette section le processus qui nous a permis d'instancier les *Widgets* des bibliothèques graphiques XAML Desktop et XAML Surface pour Microsoft PixelSense. Dans l'environnement .Net, les composants graphiques sont identifiés à partir des *Assemblies* qui appartiennent à des DLL (Dynamic Library Link).

La classe *Widget* du modèle de type de composant, correspond à toutes les classes de l'*Assembly PresentationFramework* pour XAML et de l'*Assembly Microsoft.Surface.Presentation* qui respectent les conditions suivantes :

- les composants graphiques sont des classes publiques de l'Assembly,
- ils héritent de *System.Windows.UIElement*.

A.1.1 Identification d'un Widget à partir d'un Assembly

Les attributs de la classe *Widget* sont identifiés à partir des composants graphiques identifiés dans un Assembly.

- *Widget.name* correspond au nom du type du composant graphique : par exemple pour la classe *Microsoft.Surface.Presentation.Controls.LibraryBar* de l'*Assembly PresentationFramework*, *Widget.name='LibraryBar'*.
- *Widget.cardinality* est déterminé à l'aide du tableau d'identification des cardinalités (cf tableau A.2)
- *Widget.contentType* est déterminé à l'aide du tableau d'identification de type de données (cf tableau A.1)

A.1.2 Identification des comportements

Les attributs de la classe *Event* des Widgets sont identifiés comme suit :

- l'attribut *name*
- l'attribut *eventType* est déterminé en parcourant tous les événements et les attributs d'un composant graphique et en utilisant le tableau d'identification des types d'événements (cf tableau A.3)
- l'attribut *propertyType* est identifié en recherchant parmi les propriétés d'un composant graphique et en utilisant le tableau d'identification des types de propriété (cf tableau A.4)

DataType	XAML et XAML Surface
Boolean	Si le composant graphique implémente l'attribut <i>Checked</i> de type booléen
Integer	Si le composant graphique implémente l'attribut <i>Value</i>
String	Si le composant graphique implémente les attributs <i>Text</i> , <i>Password</i> , <i>SelectedValue</i> ou la méthode <i>AddText</i>
Image	Si le composant graphique implémente l'attribut <i>ImageSource</i>
MediaElement	Si le composant graphique implémente un attribut de nom Uri
Object	Si le composant graphique implémente un attribut de nom Content
Widget	Si le composant graphique implémente un attribut de nom Children
Null	Si aucun des cas précédent n'est satisfait

TABLE A.1 – Table d’identification de type de données du modèle

Cardinality	XAML Desktop et XAML Surface
0	Si le type de données du Widget est Null
1	Si le composant graphique implémente l'attribut <i>Child</i>
N	Si le composant graphique implémente la méthode <i>AddChild</i>
N,M	Si le composant graphique implémente les attributs <i>Columns</i> , <i>Rows</i>

TABLE A.2 – Table d’identification des cardinalités

- l’attribut *inputDeviceType* est identifié en utilisant le tableau d’identification des types de propriété (cf tableau A.5).

A.1.3 Remarque

Cette approche a permis d’instancier les modèles de type pour les bibliothèques graphiques XAML Desktop pour la source et XAML Surface pour la cible.

A.2 Abstraction des UI finales

Nous présentons dans cette section les mécanismes d’abstraction d’une UI finale XAML vers le modèle d’instance de l’UI décrit au chapitre 5.

A.2.1 Identification des UIComponent à partir d’un fichier XAML

La classe *UIComponent* est identifiée à partir d’un code source XAML en considérant toutes les balises qui représentent les *Widgets* et qui ne contiennent pas d’autres balises de type *Widget*.

eventType	XAML Desktop et XAML Surface
Call	Si l’attribut suivant existe : <i>AddHandler</i>
Select	<i>Selected</i> , <i>SelectedText</i> , <i>SelectItem</i>
Change	<i>AllowDrop</i> , <i>TextChanged</i> , <i>OnValueChanged</i> , <i>EditMode</i> , <i>CanRotate</i> , <i>LocationChanged</i> , <i>CanMove</i> , <i>ResizeMode</i> , <i>CanScale</i>

TABLE A.3 – Table d’identification des types de comportements

propertyType	XAML Desktop et XAML Surface
Content	Si les attributs suivant existent : TextChanged, OnValueChanged, EditingMode, Selected, SelectedText, SelectItem, Checked, AllowDrop
Size	Si les attributs suivant existent : ResizeMode, CanScale
Position	Si les attributs suivant existent LocationChanged, CanMove
Orientation	Si l' attribut suivant existe CanRotate
WidgetStructure	Si l' attribut suivant existe Visible

TABLE A.4 – Table d’identification des types de propriétés

inputDeviceType	XAML Desktop et XAML Surface
DirectManipulationType	Si les événements de la souris, de l’écran tactile, du stylo existent (MouseDown, TouchDown, ContactDown, StylusDown , etc.)
SequentialManipulationType	Si les événements du clavier existent (Key)
Null	Si aucun événement lié à une dispositif d’entrée n’existent

TABLE A.5 – Table d’identification des types de propriétés

Une balise est de type *Widget* si son nom correspond à celui d’un Widget de la bibliothèque graphique de départ.

Les attributs de la classe container sont identifiées comme suit :

- *id* est un identifiant unique généré par le processus d’abstraction
- *name* correspond à la propriété *Name* de la balise relative à un composant graphique simple
- *type* correspond au Widget du même nom que la balise

A.2.2 Identification des Containers à partir d'un fichier XAML

La classe *Container* est identifiée à partir d'un code source XAML en considérant toutes les balises qui représentent les *Widgets* et qui contiennent d'autres balises de type *Widget*.

Une balise est de type *Widget* si son nom correspond à celui d'un Widget de la bibliothèque graphique de départ.

Les attributs de la classe container sont identifiés comme suit :

- *id* est un identifiant unique généré par le processus d’abstraction
- *name* correspond à la propriété *Name* de la balise relative à un container
- *typeContainer* est identifié à partir des éléments du container
- *type* correspond au Widget du même nom que la balise.

A.2.3 Identification de la classe Content à partir d'un fichier XAML

La classe *Content*, correspond aux données graphiques d'une UI à migrer. L'attribut *value* correspond à la valeur de la donnée graphique. Par exemple pour un bouton “Valider” en XAML, *value=’Valider’* et *propertyName=’Content’*. L'attribut *propertyName* correspond au nom de la propriété correspondante à la donnée. Pour les bibliothèques graphiques comme XAML et XAML Surface, le tableau A.6 permet d’identifier les noms de ces propriétés.

XAML Desktop et XAML Surface	
propertyName	Content, Value, Text

TABLE A.6 – Table d’identification des propriétés de contenus

eventType	XAML Desktop et XAML Surface
Change	AllowDrop,TextChanged,OnValueChanged,EditMode,CanRotate, LocationChanged,CanMove,ResizeMode,CanScal
Select	Selected,SelectedText,SelectedItem
Call	Si les attributs suivant existent : AddHandler
Delete	Methode Remove de la propriété Items
Update	Methode Add de la propriété Items, Méthodes AddChild, AddText de ItemsControl, Propriétés Content, Text, DataSource,

TABLE A.7 – Table d’identification des types de comportements

A.2.4 Identification de la classe ImplementedEvent à partir d'un fichier XAML

Les attributs de la classe *Event* des Widgets sont identifiés comme suit :

- l’attribut *name* correspond au nom d’un événement.
- l’attribut *type* est déterminé en parcourant tous les événements et les attributs d’un composant graphique et en utilisant le tableau d’identification des types d’événements (cf tableau A.7).
- l’attribut *property* est identifié en recherchant parmi les propriétés d’un composant graphique et en utilisant le tableau d’identification des types de propriété (cf tableau A.8).
- l’attribut *inputDeviceType* est identifié en utilisant le tableau d’identification des types de propriété (cf tableau A.9).

A.2.5 Exemple d’UIStructure XAML

Le listing A.1 décrit la structure et le positionnement des éléments de l’UI de l’application CBA. Les «event» correspondent à l’événement *Click* (cf les lignes 19, 44, 47, 50 du listing A.1 par exemple).

Listing A.1 – Exemple UIStructure

```
<Window x:Class="ExempleApplicationTools2012.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="326" Width="831">
    <Grid >
```

propertyType	XAML Desktop et XAML Surface
Content	Si les attributs suivants existent : TextChanged, OnValueChanged, EditingMode, Selected, SelectedText, SelectedItem, Checked, AllowDrop
Size	Si les attributs suivants existent : ResizeMode, CanScale
Position	Si les attributs suivants existent LocationChanged, CanMove
Orientation	Si l’attribut suivant existe CanRotate
WidgetStructure	Si l’ attribut suivant existe Visible

TABLE A.8 – Table d’identification des types de propriétés

inputDeviceType	XAML Desktop et XAML Surface
DirectManipulationType	Si les événements de la souris, de l'écran tactile, du stylo existent (MouseDown, TouchDown, ContactDown, StylusDown, etc.)
SequentialManipulationType	Si les événements du clavier existent (Key)
Null	Si aucun événement lié à un dispositif d'entrée n'existent

TABLE A.9 – Table d'identification des types de propriétés

```

6   <Grid.RowDefinitions>
    <RowDefinition Height="30"></RowDefinition>
    <RowDefinition Height="258*></RowDefinition>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="73*></ColumnDefinition>
    <ColumnDefinition Width="610*></ColumnDefinition>
    <ColumnDefinition Width="126.03*></ColumnDefinition>
</Grid.ColumnDefinitions>
<Grid Name="MenuPane" Grid.ColumnSpan="3" Grid.Row="0">
16  <Menu>
    <MenuItem Header="File">
        <MenuItem Header="Close" Click="Close_Click"></MenuItem>
    </MenuItem>
    <MenuItem Header="Edit"></MenuItem>
</Menu>
</Grid>
<Grid Name="RessourcePane" Grid.Column="0" Grid.Row="1" >
26  <Grid.RowDefinitions>
    <RowDefinition></RowDefinition>
    <RowDefinition></RowDefinition>
    <RowDefinition></RowDefinition>
</Grid.RowDefinitions>
<Grid Name="ControlPanc" Grid.Row="0" VerticalAlignment="Top">
    <Grid.RowDefinitions>
        <RowDefinition Height="20"></RowDefinition>
        <RowDefinition Height="15"></RowDefinition>
        <RowDefinition Height="* "></RowDefinition>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition></ColumnDefinition>
        <ColumnDefinition></ColumnDefinition>
        <ColumnDefinition></ColumnDefinition>
        <ColumnDefinition></ColumnDefinition>
        <ColumnDefinition></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <Label Grid.Row="0" Grid.ColumnSpan="5" >Browser</Label>
    <Button Name="ScreenShoot" Grid.Column="0" Grid.Row="1" Click="ScreenShoot_Click">
        <Image Source=".\\Ressources/screenshot.png"></Image>
46    </Button>
    <Button Name="Balloon" Grid.Column="1" Grid.Row="1" Click="Balloon_Click">
        <Image Source=".\\Ressources/balloon1.png"></Image>
    </Button>
    <Button Name="Text" Grid.Column="2" Grid.Row="1" Click="Text_Click">
        <Image Source=".\\Ressources/text1.png"></Image>
    </Button>
    <Button Name="Label" Grid.Column="3" Grid.Row="1">
        <Image Source=".\\Ressources/info.png"></Image>
    </Button>
    <Button Name="Favoris" Grid.Column="4" Grid.Row="1">
56    <Image Source=".\\Ressources/info.png"></Image>
    </Button>

```

```

    </Grid>
    <Grid Margin="0,35,0,8" Grid.RowSpan="3">
        <ListBox Name="RessourceList" Margin="0,0,0,-15" VerticalAlignment="Top" PreviewMouseLeftButtonDown=""
            <ListBox_PreviewMouseLeftButtonDown>
                <ListBox.ContextMenu>
                    <ContextMenu>
                        <MenuItem Header="File"></MenuItem>
                    </ContextMenu>
                </ListBox.ContextMenu>
            </ListBox>
        </Grid>
    </Grid>
    <Grid Grid.Column="1" Grid.Row="1" Background="DarkGray" >
        <Grid.RowDefinitions>
            <RowDefinition /><RowDefinition />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition /><ColumnDefinition />
            <ColumnDefinition /><ColumnDefinition />
        </Grid.ColumnDefinitions>

        <Canvas Name="canvas1" Grid.Column="0" Grid.Row="0" Margin="8,8,8,8" Background="White" AllowDrop="True"
            Drop="Canvas_Drop" >
        </Canvas>
        <Canvas Name="canvas2" Grid.Column="1" Grid.Row="0" Margin="8,8,8,8" Background="White"></Canvas>
        <Canvas Name="canvas3" Grid.Column="0" Grid.Row="1" Margin="8,8,8,8" Background="White"></Canvas>
        <Canvas Name="canvas4" Grid.Column="1" Grid.Row="1" Margin="8,8,8,8" Background="White"></Canvas>
    </Grid>
    <Grid>
        </Grid>
    </Grid>
</Window>

```

A.2.6 Exemple de UIBehavior C#

Le listing A.2 décrit le comportement des éléments de l'UI de l'application CBA.

Listing A.2 – Exemple UIBehavior

```

namespace ExempleApplicationTools2012
{
    /// <summary>
    /// Logique d'interaction pour Window1.xaml
    /// </summary>
    public partial class Window1 : Window
    {
        public Window1()
        {
            InitializeComponent();
            _controler = new Controler.Controler();
            canc = new CanvasClass(this.canvas1, this.canvas2, this.canvas3, this.canvas4);
            _controler._view.RessourceList = RessourceList;
        }
        private CanvasClass canc = null;
        private Controler.Controler _controler;
        private void ListBox_DragLeave(object sender, DragEventArgs e)
        {
        }
        private void Canvas_Drop(object sender, DragEventArgs e)
        {
            Canvas parent = (Canvas)sender;
            _controler.DropCanvas(parent);
        }
        private void ListBox_DragEnter(object sender, DragEventArgs e)

```

```

27     { }
27     private void ListBox_PreviewMouseLeftButtonDown(object sender, MouseButtonEventArgs e)
27     {
27         ListBox parent = (ListBox)sender;
27         _controler.DragOutList(parent, e.GetPosition((ListBox)parent));
27     }
27     private void Text_Click(object sender, RoutedEventArgs e)
27     {
27         _controler.getItemList("text");
27     }
27
37     private void Balloon_Click(object sender, RoutedEventArgs e)
37     {
37         _controler.getItemList("balloon");
37     }
37 }

```

A.2.7 Exemple d'AbstractView C#

Le listing A.3 décrit une vue abstraite de l'application CBA. La méthode *updateRessourceList* (cf ligne 7 du listing A.3) représente les flèches de type «*updateView*» pour les interactions en sortie car elle permet une notification de la vue par le modèle ou le contrôleur.

Listing A.3 – Exemple AbstractView

```

using System;
namespace ComicBook.View
{
    interface IAView
    {
        void canvas_Drop(object parent);
        void dragOutList(object parent, System.Windows.Point position);
        void updateRessourceList(System.Collections.ObjectModel.ObservableCollection<System.Windows.Controls.Image> coll);
9    }
}

```

Le listing A.4 est une implémentation de la vue abstraite selon les éléments de la bibliothèque graphique. Cette classe est modifiée pendant la migration car elle n'est pas indépendante des instruments d'interactions.

Listing A.4 – Exemple implémentation d'AbstractView

```

namespace ComicBook.View
{
    public class AView : ComicBook.View.IAView
    {
        //store the "source" listbox
        Object data = null;
        CanvasClass canc = null;
        public ListBox RessourceList;
        public AView()
        {
        }
        public void canvas_Drop(Object parent)
        {
            if (data != null)
                canc.addBalloon((Canvas)parent, (UIElement)data);
        }
        public void updateRessourceList (ObservableCollection<Image> coll)
        {
            RessourceList.ItemsSource = coll;
        }
        public void dragOutList(Object parent,Point position)
        {

```

```
//get the object source for the selected item
object data1 = GetObjectDataFromPoint((ListBox)parent, position);

//if the data is not null then start the drag drop operation
if (data1 != null)
{
    data = data1;
    DragDrop.DoDragDrop((ListBox)parent, data1, DragDropEffects.Copy);
}

private static object GetObjectDataFromPoint(ListBox source, Point point)
{
    UIElement element = source.InputHitTest(point) as UIElement;
    if (element != null)
    {
        //get the object from the element
        object data = DependencyProperty.UnsetValue;
        while (data == DependencyProperty.UnsetValue)
        {
            // try to get the object value for the corresponding element
            data = source.ItemContainerGenerator.ItemFromContainer(element);
            if (data == DependencyProperty.UnsetValue)
                element = VisualTreeHelper.GetParent(element) as UIElement;
            if (element == source)
                return null;
        }
        if (data != DependencyProperty.UnsetValue)
            return data;
    }
    return null;
}
```


A.3 Primitives d'interactions

A.3.1 Liste des Widgets

Composant graphique et Illustration	PI Intrinsèques	Type de données et Cardinalité
Input Field 	Data Edition Data Selection Data Display DataMoveIn/Out Widget Selection Navigation Widget Display Activation	Datatype=String Cardinality=1
Text Area 	Data Edition Data Selection Data Display DataMoveIn/Out Widget Selection Navigation Widget Display Activation	Datatype=String Cardinality=1
List Déroulant 	Data Edition Data Selection Data Display DataMoveIn/Out Widget Selection Navigation Widget Display Activation	Datatype=String, Cardinality=N
List / List Item 	Data Display DataMoveIn/Out Data Selection Widget Selection Navigation Widget Display Activation	List: Container Datatype=Widget Cardinality=N ListItem: UIComponent Datatype=String, Image Cardinality=1
Table 	Data Display DataMoveIn/Out Data Selection Widget Selection Navigation Widget Display Activation	Table:Container Datatype=Widget Cardinality=N,M Col: Container Datatype= Cell Cardinality=N Row: Container Datatype= Cell Cardinality=M Cell: UIComponent Datatype=String, Image Cardinality=1
Check Box 	Data Edition Data Display Widget Selection Navigation Widget Display Activation	DataType=Boolean, Cardinality=1

Radio Button  Radio button  Radio button selected	Data Edition Data Display Widget Selection Navigation Widget Display Activation	DataType=Boolean, Cardinality=1
Spinner 	Data Selection Data Display Widget Selection Navigation Widget Display	DataType=Integer, Cardinality=N
Calendar 	Data Selection Data Display Widget Selection Navigation Widget Display Activation	DataType=String, Cardinality=N
Color Choser 	Data Selection Data Display Widget Selection Navigation Widget Display Activation	DataType=Integer, Cardinality=N, M
Button 	Navigation Widget Display Activation	DataType=String, Cardinality=1
Menu / Menu Item 	Navigation Widget Display Activation	Menu : Container DataType=String, Cardinality=N MenuItem : UIComponent DataType=String, Cardinality=1
Image 	Data Display Navigation Widget Display Activation	DataType=Image, Cardinality=1
Slider 	Data Selection Navigation Widget Selection Activation	DataType=Integer, Cardinality=N
Canvas 	Data Edition Data Selection Data Display DataMoveIn/Out Widget Selection Navigation Widget Display Activation	DataType=Object, Cardinality=N

Table des figures

2.1	Applications sur les marchés de téléchargement	10
2.2	Description de la fenêtre principale de l'application CBA	11
2.3	Migration de la structure d'une UI pour desktop sur une table interactive	14
2.4	Espace problèmes de la migration des UI vers les tables interactives	16
3.1	Modèle d'interactions instrumentale d'une table interactive	24
3.2	Table interactive DiamondTouch	25
3.3	Instanciation physique des éléments GUI dans TUI	26
3.4	Instances des containers DiamondSpin	28
3.5	Exemple de ScatterView	29
3.6	Modèle d'interactions abstraites de Gellersen	32
3.7	Illustration de la propriété 360°	39
3.8	Représentation synthétique des guidelines selon les trois dimensions des UI	40
3.9	Représentation synthétique des guidelines suivant deux dimensions	42
4.1	Application de consultation des contacts	48
4.2	Synthèse de la migration manuelle des UI	48
4.3	Synthèse des approches de portage des UI sur tables interactives	52
4.4	Exemple de transformation USIXML	54
4.5	Service de migration des UI	55
4.6	Synthèse des approches automatiques de migration des UI	57
4.7	Processus de migration avec MORPH	58
4.8	Synthèse de l'approche semi automatique	62
4.9	Synthèse des approches de migration des UI	63
5.1	Un artefact d'une UI	72
5.2	Boîte de dialogue	73
5.3	Types et instances de composants graphiques	74
5.4	Modèle de types de composants graphiques	76
5.5	Modèle de structure d'instance d'une UI	82
5.6	Illustration des types de container	85
5.7	Modèle abstrait et UI finale	90
6.1	Processus de migration assistée	100
6.2	Critères ergonomiques de conception et guidelines pour favoriser la collaboration	102
6.3	Critères ergonomiques de conception et guidelines pour des UI tangibles	103
6.4	Utilisations effectives des guidelines par les mécanismes de migration des UI	104
6.5	Modèle de règles de substitution	105
6.6	Modèle de règles de concrétisation	106
6.7	Exemple des groupes d'éléments graphiques à transformer	110
6.8	Exemple de <i>ControlGroup</i>	112
6.9	Exemple de <i>ControlGroup</i> sur table interactive Microsoft PixelSense	112
6.10	Exemple de <i>DisplayGroup</i>	114
6.11	Exemple de <i>DisplayGroup</i> migré sur une table interactive	115

6.12 Illustration d'un <i>UpdateGroup</i>	117
6.13 Exemple d'un <i>UpdateGroup</i> sur une table interactive	117
6.14 Représentation de l'UI CBA pour tables interactives	118
6.15 Modèle UI CBA	130
7.1 Processus semi automatique de migration d'une UI vers les tables interactives	138
7.2 Architecture des applications migrées vers les tables interactives	139
7.3 Exemple de cadre de sélection des éléments graphique dans l'éditeur du prototype . .	146
7.4 Exemple de substitution de <i>LibraryBar</i> par <i>SurfaceListBox</i>	147
7.5 Menu d'association d'un tag avec un container	147
7.6 Éditeur graphique du prototype	148
7.7 Fiche d'évaluation de la migration des applications	151
7.8 Application agenda	151
7.9 Regroupement des éléments de l'application Album Photo	152
7.10 Regroupement des éléments de l'application de dessin	153
7.11 Regroupement des éléments de l'application de calculatrice	154
7.12 Synthèse de l'étude de la migration des quatre applications	155

Liste des tableaux

3.1	Synthèse des dispositifs physiques d'interactions pour 3 tables interactives	27
3.2	Type d'UI pour les tables interactives	35
3.3	Affinement des critères ergonomiques de Scapin	37
3.4	Migration de l'aspect visuel d'un formulaire	42
4.1	Synthèse des technologies de portage des UI sur tables interactives	51
4.2	Table d'équivalences	56
4.3	Récapitulatif de MORPH	61
5.1	Exemples de primitives d'interactions en entrée	72
5.2	Exemple de primitives d'interactions en sortie	74
5.3	Types de container	84
6.1	Caractéristiques des <i>ControlGroups</i>	110
6.2	Règle de substitution des groupes de contrôle	111
6.3	Règle de concrétisation des groupes de contrôle	111
6.4	Caractéristiques des groupes d'affichage de contenus	113
6.5	Règle de substitution des groupes <i>DisplayGroups</i>	114
6.6	Caractéristiques des groupes de modification de contenus	115
6.7	Règle de substitution des <i>UpdateGroups</i>	116
6.8	Caractéristiques des groupes mixtes	117
6.9	Synthèse des problèmes traités par les transformations des groupes	120
6.10	Caractéristiques des interacteurs de données en sortie	121
6.11	Caractéristiques des interacteurs de données en entrée et en sortie	122
6.12	Caractéristiques des interacteurs d'activation	122
6.13	Synthèse des problèmes traités par les transformations des interacteurs	123
6.14	Poids des PI par rapport aux guidelines	126
6.15	Poids des types de données par rapport aux guidelines	127
6.16	Coût des interventions manuelles	129
6.17	Widgets équivalents	131
A.1	Table d'identification de type de données du modèle	172
A.2	Table d'identification des cardinalités	172
A.3	Table d'identification des types de comportements	172
A.4	Table d'identification des types de propriétés	173
A.5	Table d'identification des types de propriétés	173
A.6	Table d'identification des propriétés de contenus	174
A.7	Table d'identification des types de comportements	174
A.8	Table d'identification des types de propriétés	174
A.9	Table d'identification des types de propriétés	175

Résumé

Dans le domaine du génie logiciel pour les Interactions Homme Machine (IHM), la migration des interfaces utilisateurs (UI) est un moyen pour réutiliser des applications sur des plateformes ayant des modalités d’interactions différentes des environnements de départ. Les approches existantes de migration des UI sont manuelles dans le cadre des approches spécifiques, elles sont automatiques dans le cadre des services d’adaptation des UI aux contextes d’usage, ou elles sont semi automatiques dans le cadre d’une migration flexible dirigée par un concepteur.

Dans cette thèse nous nous intéressons à la migration semi automatique des UI vers une cible comme une table interactive dans l’objectif de transformer des UI Desktop en UI qui favorisent la collaboration et l’utilisation des objets tangibles. Les tables interactives sont des plateformes qui disposent des instruments d’interactions permettant de décrire des UI tangibles et multi-utilisateurs. En considérant que le noyau fonctionnel (NF) des applications de départ peut être réutilisé sur les cibles sans changement, les UI des applications sont caractérisées par la dimension des dialogues entre les utilisateurs et le système, la dimension de la structure et du positionnement des éléments graphiques et la dimension du style des éléments visuels. La migration d’une UI dans ces conditions consiste à transformer ou à recréer les différentes dimensions d’une UI de départ pour la cible tout en considérant les critères de conception des UI pour les tables interactives.

Nous proposons dans cette thèse un modèle d’interactions abstraites pour établir les équivalences entre les dialogues et la structure des UI indépendamment des modalités d’interactions des plateformes source et cible. Les primitives d’interactions et la structure des composants graphiques permettent de décrire des opérateurs d’équivalences pour retrouver et classer les éléments graphiques équivalents en prenant en compte les guidelines des tables interactives. Nous proposons aussi des règles de substitution et de concrétisation pour accroître l’accessibilité des éléments graphiques et favoriser l’utilisation des objets tangibles.

Mots clés : migration des interfaces utilisateurs, équivalences des plateformes, modalités d’interactions, critères de conception, guidelines

Abstract

In software engineering, in the field of human computer interaction (HCI), the migration of user interface (UI) is a way to reuse existing applications on platforms with different interactions modalities. The existing approaches for UI migration can be manual (for specific applications), they can be automatic (for services which adapt UI based on context aware), or they can be mix of the previous - semi automatic (providing a flexible migration process driven by the person in charge).

This thesis proposes a semi automatic process for migration of UI from a desktop to interactive table for the purpose of transforming the UI of desktop to support further collaboration and usage of tangible objects. The interactive tables are platforms with interactions instruments which allow the description of tangible and multi users UIs. Considering that the functional core (FC) of source applications can be reused on target platform without transformation, any UI can be characterized with three dimensions : the first dimension concerns the dialogues between the users and the system, the second dimension concerns the structure and the layout of graphical components, and the third dimension concerns the visual style of graphical elements. In this context, the problematic regarding the UI migration is how to transform or re inject these different dimensions of source UI into the target, while considering the UI design criteria for interactive tables.

This thesis proposes an abstract interactions model for establishing equivalences (independent of modalities of interactions) between the source and the dialogue and structure of the target. The primitives of interaction and the structure of graphical components are used to describe equivalence operators to find and to rank equivalent elements on interactive tables. Furthermore, this thesis proposes substitution and concretization rules to increase the accessibility of graphical elements and to facilitate the usage of tangible objects. The ranking process and the transformation rules are based on guidelines for UI migration to interactive tables which are interpreted from design criteria.

Keywords : user interface migration, équivalence of plateform, design criteria, guidelines