Composition d'Interfaces Homme-Machine : Opérateurs de composition et propriétés de composabilité

Yoann Gabillon
Luxembourg Institute of Science and Technology (LIST)
L-4362 Esch/Alzette, Luxembourg
yoann.gabillon@list.lu

RÉSUMÉ
La composition automatique d'Interfaces Homme-Machine (IHM) à l'aide d'opérateurs de composition est une des approches qui permet d'améliorer le développement d'IHM, de produire une IHM adaptative et de maitriser le développement d'IHM de grande taille. Dans cet objectif, la vérification de la correction d'une composition d'IHM est un challenge important. Ce travail s'intéresse à la vérification de propriétés de composabilité de composants IHM et de leurs services selon des opérateurs de composition définis à différents niveaux d'abstraction. Les principales contributions de cet article sont la proposition d'un modèle de composants permettant de réaliser une composition hiérarchique, l'identification des principaux aspects intervenant dans la composition d'IHM, la formalisation de nouveaux opérateurs et la vérification de propriétés de composition.

Mots Clés
Interface Homme-Machine, Composant, Composition, Opérateur de Composition (Composeurs), Propriété d'Assemblage, Services, Propriété de Composabilité.

ACM Classification Keywords
H.5.2. User Interfaces: Graphical user interfaces (GUI), Theory and methods, User-centered design

INTRODUCTION

Un logiciel à base de composants est une entité réutilisable, de haut niveau se définissant à l'aide d'interfaces requises et fournies. Un logiciel comprend un partie fonctionnelle et une partie Interface Homme-Machine (IHM). Le développement d'IHM est une tâche difficile dans le but de réaliser des logiciels centrés utilisateur. La composition d'IHM réfère à l'assemblage de composants IHM pour produire une IHM plus grande. La composition de composants permet d'augmenter la réutilisabilité [21]. La composition automatique d'IHM est effectuée par des opérateurs de composition (composeurs).

http://dx.doi.org/10.1145/2820619.2820632

Les propriétés de composition permettent de calculer automatiquement une composition correcte de composants IHM. Une composition correcte est, par exemple, la composition de composants sans conflit entre eux. Plus précisément, deux IHM vocales ne peuvent être composées en parallèle si leur utilisation nécessite les même mots.

Une IHM adaptative est une IHM capable de s'adapter elle-même ou d'être adaptée automatiquement à l'exécution, à un changement du contexte d'usage [3]. Le contexte d'usage comprend les propriétés représentant l'utilisateur, sa ou ses plate(s)-forme(s) et son environnement d'interaction. Le développement d'IHM adaptative reste une tâche difficile. Le projet CAMELEON [3] propose des modèles, comme framework, définis à différents niveaux d'abstraction. Ces niveaux distinguent le niveau domaine (tâches utilisateur et concepts), l'Interface Utilisateur Abstraite (IUA), l'Interface Utilisateur Concrète (IUC) et l'Interface Utilisateur Finale (IUF).

La *motivation* de ce travail est liée au développement d'une composition correcte d'IHM grâce à des opérateurs de composition dans le but soit d'accélérer le développement d'IHM, soit de produire une IHM adaptative automatiquement ou soit de composer automatiquement une IHM de grande taille. Lau et Rana [12] identifient quatre mécanismes de composition : l'endiguement, l'extension, la connexion et la coordination. Ils définissent la composition par endiguement comme réfèrant à une composition où des composants sont encapsulés, composés à l'intérieur d'un autre (« refers to putting units of behaviour inside the definition of a larger unit »). Alors qu'actuellement, la composition de composants IHM [10, 22, 18, 4] et les opérateurs de composition IHM définis [19, 13, 2, 15, 11] réalisent une composition par connexion, ce travail propose une approche par endiguement dans le but de définir de nouveaux opérateurs de composition. Au contraire des travaux précédents focalisant sur l'IUC [19, 14], l'IUA[13, 2] ou le Modèle de Tâches [15]), cet article définit de nouveaux opérateurs de composition à tous niveaux d'abstraction. Ces niveaux d'abstraction permettent une séparation des préoccupations pour développer une IHM adaptative. Même si les propriétés de composition sont identifiées comme un challenge primordial pour développer des logiciels par composition [23, 1], aucun travail à notre connaissance ne se concentre sur la composabilité de composants IHM. Pourtant, une meilleur IHM peut permettre une meilleure utilisabilité et de rendre un système centré-utilisateur. Quelques travaux vérifient la composabilité en

considérant que des composants IHM fournissent un service (appelé un service IHM) [9, 20, 7] mais en prenant en compte seulement deux opérateurs (séquence et interleaving).

L'*objectif* de ce travail est de fournir au développeur d'IHM un modèle à composants de haut niveau par endiguement et de lui proposer de nouveau opérateurs de composition associés à des propriétés de composabilité réutilisables pour d'autres opérateurs.

Les principales *contributions* de ce travail sont un **modèle à composants** permettant une composition par endiguement et la vérification de propriétés de composabilité. Cette vérification dépend des opérateurs de composition utilisés. En conséquence, nous proposons d'**opérateurs de composition d'IHM** définis à tous niveaux d'abstraction grâce à l'identification des principaux **aspects impactant la composition d'IHM**. Dans notre travail, un composant IHM fournit un service IHM. L'utilisation de services est centrale pour vérifier les propriétés de composabilité lors de l'assemblage de composants [1]. Nous définissons une propriété de composabilité en considérant le lien entre les éléments du modèle de l'IHM (propriétés d'assemblage) mais également entre les prédicats décrivant les services IHM. La composabilité des composants IHM est automatiquement vérifiée par un **système de composition** nommé UICONDUCTOR. Nous réalisons également une **étude utilisateur** pour évaluer l'utilité des aspects de la composition et si les prédicats sont appropriés pour décrire les services IHM.

Cet article est structuré comme suit. La Section 2 présente le modèle à composants à travers la description des composants IHM et des services IHM. Le modèle à composants est illustré grâce à un exemple. La Section 3 introduit les aspects impactant la composition d'IHM. La Section 4 est dévouée à la définition de nouveaux composeurs à tous niveaux d'abstraction et réalisant une composition par endiguement. Dans la Section 5, nous définissons les propriétés de composabilité associées aux composeurs proposés. La Section 6 montre brièvement deux logiciels produits par UICONDUCTOR, notre système de composition utilisant les composeurs et les composants IHM pour produire une IHM composée. Dans la Section 6, nous présentons également le résultat d'une étude utilisateur réalisée dans le but d'évaluer l'intérêt des aspects impactant la composition et d'évaluer l'intérêt des prédicats pour décrire les services IHM.

MODÈLE À COMPOSANTS IHM

Pour définir le modèle à composants, nous formalisions un modèle IHM à tous niveaux d'abstraction.

Modèle IHM

Une IHM est modélisée par un *Modèle IHM* (UI Model) à différents niveaux d'abstraction : le *Modèle de Tâches* [16], l'*IUA* [17] et l'*IUC* [13]. Chaque niveau d'abstraction peut être représenté par un graphe où les sommets sont les tâches, espaces de travail et interacteurs (i.e. widgets comme des boutons ou des champs texte) et où les arcs sont des liens (relations) entre ces éléments.

Modèle de Tâches : Le Modèle de Tâches peut être représenté par un graphe complet, asymétrique et orienté $\langle T, E_T \rangle$ tel que l'ensemble des sommets T est l'ensemble des tâches $T = \{t_1, t_2, \dots t_n\}$ du Modèle de Tâches et l'ensemble des arcs E_T est un ensemble de liens entre tâches. Il existe deux types de liens entre tâches : les relation de dépendance et l'es opérateurs en tâches. Une *relation de descendance* notée $d(t_i, t_j)$ (où $t_i, t_j \in T$ et $t_i \neq t_j$) exprime que la tâche t_i est le père de la tâche t_j. L'ensemble des relations de descendance du Modèle de Tâches est noté D_T. Un *opérateur entre tâches* $op_T(t_i, t_j)$ (où $t_i, t_j \in T$ et $t_i \neq t_j$) exprime un opérateur entre tâches de CTT [16] $op_T \in \{enabling, interleaving, orderindependency, choice\}$ entre les deux tâches t_i et t_j. En d'autres termes, les tâches t_i et t_j sont frères, si et seulement si elles ont le même et unique père et s'il existe un opérateur entre tâches entre elles. Dans ce cas, la tâche t_i est notée le *frère de gauche* et t_j le *frère de droite*. L'ensemble des opérateurs entre tâches est noté OP_T.

L'ensemble des arcs E_T est constitué par l'ensemble des relations de descendance D_T et des opérateurs entre tâches OP_T : $E_T = D_T \cup OP_T$.

IUA et IUC : De la même manière que le Modèle de Tâches, l'IUA et l'IUC peuvent aussi être représentées par un graphe complet, asymétrique et orienté tel que $AUI = \langle W, E_W \rangle$ où $E_W = D_W \cup OP_W$ avec $OP_W \in \{\emptyset, navigation\}$; $CUI = \langle I, E_I \rangle$ où $E_I = D_I \cup OP_I$ avec $OP_I \in \{\emptyset, navigation\}$.

Modèle IHM : Un Modèle IHM $UIModel$ est un graphe $\langle V, E \rangle$ avec $V = T \cup W \cup I$ et $E = E_T \cup E_W \cup E_I \cup E_{TW} \cup E_{WI}$ où E_{TW} est un ensemble de relations de descendance (arcs) entre une tâche $t \in T$ et un espace de travail $w \in W$ tel que $d(t, w) \in E_{TW}$; et E_{WI} est un ensemble de relations de descendance entre un espace de travail $w \in W$ et un interacteur $i \in I$ tel que $d(w, i) \in E_{WI}$.

Spécification d'un composant IHM

Un composant IHM est défini par un nom, les paramètres, un modèle IHM, les éléments requis et fournis et optionnellement un opérateur de composition. Un service est composé de préconditions, postconditions et conditions interactives pour permettre la vérification des propriétés de composabilité. Nous utilisons des prédicats de la logique du premier ordre pour décrire ces conditions.

Définition [Composant IHM] : Un composant IHM c est un 6-tuple $\langle Name, Param, UIModel, R_c, P_c, op \rangle$ avec $Name$ le nom unique du composant ; $Param$ est un ensemble de variables initialisées ; $UIModel$ un modèle IHM ; R_c un ensemble d'éléments requis par le composant ; P_c un ensemble d'éléments fournis par le composant ; *op* un opérateur de composition (composeur) qui est une fonction chargée d'assembler les Modèles IHM des sous-composants dans l'$UIModel$ de ce composant.

Il existe deux types de composants : un *composant élémentaire* et un *composant composite*. Un composant élémentaire comprend un Modèle d'IHM complet (exécutable) et aucun opérateur de composition. Un composant

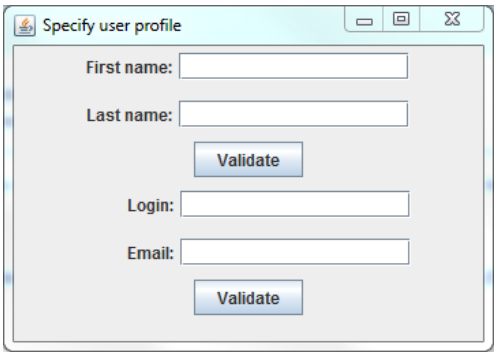

Figure 1: Les composants $C1$ et $C2$ composés dans $C3$, i.e., une fenêtre avec un Layout vertical.

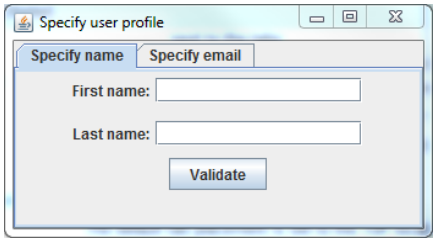

Figure 2: Les composants $C1$ et $C2$ composés sous la forme d'onglets par $C4$.

composite comprend un Modèle IHM partiel et un opérateur de composition.

Définition [Service IHM] : Un service IHM s fourni par un composant IHM c est un 5-tuple $\langle Name, Param, Precond, Post, InteractivePost \rangle$ où $Name$ est le nom du service ; $Param$ est l'ensemble des variables initialisées ; $Precond$ est l'ensemble des préconditions, i.e., un ensemble de prédicats positifs (resp. negatifs) de la logigue de premier ordre $Precond^+$ (resp. $Precond^-$) qui doivent être vrais (resp. faux) avant l'exécution du service ; $Post$ est l'ensemble des postconditions, i.e., un ensemble de prédicats positifs ($Post^+$) et négatifs ($Post^-$) qui doivent être vrais (resp. faux) à la fin de l'exécution du service ; $Interact$ est l'ensemble des conditions interactives devant être réalisées pendant l'utilisation par l'utilisateur du service, i.e., un ensemble de prédicats positifs ($Interact^+$) (resp. négatifs ($Interact^-$)) qui doivent être vrais (resp. faux) pendant l'utilisation du service. Par exemple, pour une IHM vocale, un haut-parleur doit être disponible durant l'interaction (mais pas nécessairement durant l'exécution entière s'il existe un bouton "start" pour démarrer l'IHM vocale par exemple). A la fin de l'utilisation du service, le haut-parleur sera à nouveau disponible.

Exemple illustratif

Comme exemple illustratif, nous supposons qu'il existe deux composants $C1$ et $C2$. Le composant $C1$, intitulé "Specify name", est une fenêtre contenant deux étiquettes, deux champs texte permettant à l'utilisateur de spécifier son prénom et son nom, et un bouton "Validate" (figure 3) au niveau IUC. Le composant $C2$, intitulé "Specify user name", est une fenêtre contenant deux étiquettes ($Lb1$ et $Lb2$), deux champs texte ($Tf1$ et $Tf2$) permettant à l'utilisateur de spécifier son login et son email, et un bouton "Validate" ($VB1$). Ces deux composants peuvent être assemblés dans la même fenêtre (figure 1), dans un onglet (figure 2) ou en une séquence temporelle (i.e. quand $C1$ est validé, $C2$ est executé) dans le but de permettre à l'utilisateur de spécifier des informations sur son profil. Ces composants sont assemblés dans un composant composite pour produire une IHM composée. Par exemple, dans la figure 1, le composant composite $C3$ compose les sous-composants $C1$ et $C2$ dans une fenêtre comportant un Layout vertical. L'assemblage des Modèles IHM est effectué par un composeur. Par exemple, dans la figure 3, l'opérateur "InterleavingTab" ajoute des liens entre l'onglet ($I'4$) et $I1$ de $C1$ puis $I2$ de $C2$ pour produire l'IHM composée de la figure 2. Les éléments requis de $C4$ sont deux tâches (ici $T1$ et $T2$), deux espaces de travail ($W1$ et $W2$) et deux interacteurs $I1$ et $I2$. Les éléments fournis sont $T4$, $W4$ et $I4$ pour permettre à $C4$ d'être également composé.

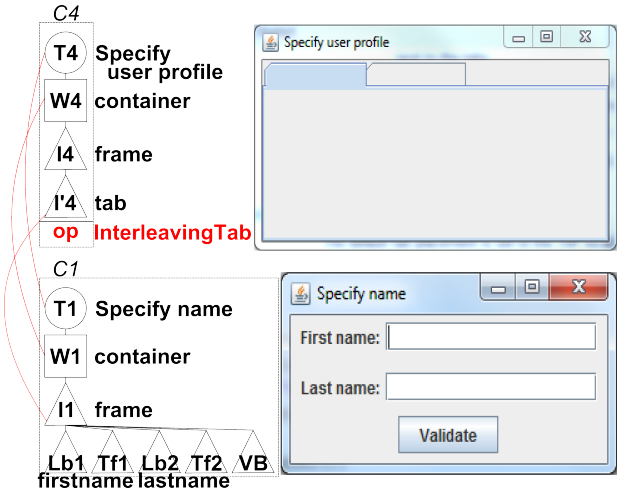

Figure 3: Exemple de modèles de $C4$ et $C1$ et des liens (relations de descendance) ajoutés par le composeur.

ASPECTS DES COMPOSEURS

Il existe différentes manières de composer deux IHM. Par exemple, deux IHM peuvent être composées en onglets, mais aussi horizontalement ou verticalement dans une même fenêtre, etc. Nous identifions, grâce à la littérature [13, 19] et aux niveaux d'abstraction séparant différentes préoccupations, trois aspects principaux qui impactent la composition d'IHM : les aspects *logico-temporel*, *spatial* et *sémantique*.

Aspect logico-temporel

L'*aspect logico-temporel* caractérise les relations entre IHM à composer dans un intervalle de temps. Cet aspect est mis en évidence par le niveau Tâches [16]. Si nous ne considérons pas l'aspect fonctionnel ou les informations échangées, nous pouvons identifier les opérateurs de composition suivants comme exprimant un type de composition unique : enabling (incluant with or without informa-

Algorithm 1: MergeKeepNB - Fusionne les éléments équivalents en gardant les éléments équivalents du $(nb)^{ème}$ sous-composant.

Input: $ui_0, ui_1, \ldots, ui_n$ un ensemble de composants IHM, ui le modèle de l'IHM composée, nb le $(nb)^{ème}$ élément à garder dans ui_{nb}, EQ et SUB les relations d'équivalence et subsumes.

Output: ui le modèle de l'IHM composée, $null$ sinon.

1 **foreach** $eq \in EQ$ **do**
2 **if** $\exists v \in eq, v \notin ui_{nb}.V$ **then**
3 $ui.V \leftarrow ui.V - v$;
4 **end**
5 **end**
6 **foreach** $sub(v1, v_2, \ldots, v_n) \in SUB$ **do**
7 $ui.V \leftarrow (ui.V - v_2, \ldots, -v_n) \cup v1$;
8 $ui.E \leftarrow ui.E \cup d(father(v2), v1)$;
9 **end**
10 //Les arcs qui contiennent un élément supprimé sont également supprimés
 $ui.E \leftarrow SuppessEdgeWithoutVertex(ui.E, ui.V)$;
11 **Return** ui ;

tion passing, suspend-resume and disabling), interleaving (incluant synchronisation), order independence et choice.

Aspect spatial

L'*aspect spatial* caractérise les relations entre IHM à composer dans une ou plusieurs régions d'exécution. Par exemple, la région d'exécution d'une IHM graphique est l'écran. La région d'exécution d'une IHM vocale est la région de propagation du son. Cet aspect reflète la volonté du développeur de composer une IHM à un endroit spécifique de la perception de l'utilisateur. Cet aspect est souligné par le niveau IUC. [5] identifie les relations spatiales suivantes : Meets, Overlaps, Equals, Covers, Contains. Ces relations peuvent aider le concepteur de composeurs.

Aspect sémantique

L'*aspect sémantique* caractérise les opérations possibles sur les éléments *équivalents* et *subsumés* entre deux modèles d'IHM à composer. $eq(v1, v2, \ldots vn)$ signifie que les sommets sont équivalents. $sub(v0, v1, v2, \ldots, vn)$ signifie que le sommet $v0$ subsume $v1, v2, \ldots vn$. Les relations d'équivalence sont déjà introduites par ComposiXML [13] et Amusing [19] (excepté la possibilité de sélectionner le $(nb)^{ème}$ sommet à garder), mais les relations subsumes et le déplacement vers le haut d'un élément sont des contributions de cet article. Nous identifions deux stratégies pour fusionner deux éléments : "MergeKeepNB" qui fusionne les éléments équivalents/subsumés, et "MergeKeepNBUP" qui fusionne les élements equivalents/subsumés et factorise (déplacer l'élément sélectionné vers le haut de l'arbre des modèles). MergeKeepNBUP est une contribution de cet article.

$MergeKeepNB(ui_0, ui_1, \ldots, ui_n, ui, nb, EQ, SUB)$ fusionne les éléments équivalents et subsumés. Les éléments équivalents du $(nb)^{ème}$ composant sont gardés (avec $0 \leq nb \leq n$ et $nb \in \mathbb{N}$) et les autres sont supprimés. Par exemple, dans la figure 1, l'IHM comporte deux boutons "Validate" (VB). L'IHM pourrait être meilleure si le premier bouton était supprimé et fusionné avec le second grâce à $MergeKeepNB((C0, C1, C2), ui, 1, eq(C1.VB, C2.VB))$. Les éléments subsumés de SUB sont supprimés et remplacés par l'élément qui les subsume. Par exemple, les étiquettes "first name" et "login" peuvent être supprimées et remplacées par l'étiquette les subsumant "first name or login". ui est le modèle de l'IHM composée, en conséquence, cette fusion doit être effectuée après l'assemblage du modèle de l'IHM composée. L'algorithme 1 réalise cette fusion. Les paramètres de cet algorithme sont un ensemble de modèles d'IHM déjà composés $ui_0, \ldots, ui_n$ dans ui, un modèle de l'IHM composée ui, le numéro de l'élément équivalent ou subsumé que l'on va garder nb (nb désignant le numéro du composant) et l'ensemble des relations entre les éléments équivalents EQ et subsumés SUB. Dans un premier temps, si un élément d'une relation d'équivalence est dans le $(nb)^{ème}$ sous-composant, cet élément est gardé dans l'IHM composée, sinon cet élément est supprimé dans les autres (ligne 3 de l'algorithme 1). Dans un deuxième temps, l'algorithme supprime tous les éléments subsumés (ligne 7) et remplace le $(nb)^{ème}$ par l'élément qui les subsume (ligne 8). Quand toutes les relations d'équivalence et subsumes sont traitées, l'algorithme supprime les arcs qui pointent vers des sommets supprimés (ligne 10). Une fois terminé, l'algorithme retourne l'IHM composée.

$MergeKeepNBUP(ui_0, ui_1, \ldots, ui_n, ui, nb, up, EQ, SUB)$ fusionne les éléments équivalents et subsumés. Les éléments équivalents sont supprimés mais le $(nb)^{ème}$ élément est déplacé vers le haut dans l'arbre des niveaux d'abstraction (avec $0 \leq nb \leq n$ et $n \in \mathbb{N}$). De la même manière, les éléments subsumés sont aussi supprimés et l'élément qui les subsume est ajouté au $up^{ème}$ père dans les niveaux d'abstraction. Cette méthode est utilisée quand il est nécessaire de déplacer vers le haut un interacteur dans une fenêtre plus haute par exemple. Dans notre exemple illustratif, $MergeKeepNBUP((C0, C1, C2), 0, 2, eq(C2.bV, C3.bV))$ produit une IHM où le bouton "Validate" de "C1" (VB dans la figure 3) est supprimé et le bouton "Validate" de "C2" est déplacé 2 fois (car $UP = 2$) vers le haut dans la fenêtre de $C3$ ($I3$ dans la figure 3) contenant l'onglet. L'algorithme 2 réalise cette fusion et ce déplacement vers le haut. Les paramètres de cet algorithme sont un ensemble de modèles d'IHM à composer $ui_0, \ldots, ui_n$, un modèle de l'IHM composée ui, le numéro de l'élément équivalent ou subsumé que l'on va garder nb (nb désignant le numéro du composant), le nombre de fois que les éléments vont être déplacés vers le haut (up) et l'ensemble des relations entre les éléments équivalents EQ et subsumés SUB. Dans un premier temps, pour chaque relation d'équivalence, chaque élément est supprimé dans l'IHM composée ui (ligne 18 de l'algorithme 2). Le $(nb)^{ème}$ élément équivalent de ui_{nb} est déplacé vers le haut up fois (ligne 15). Avant de réaliser ce déplacement, l'algorithme vérifie que l'élément qui va devenir le père de l'élément déplacé vers le haut est du même type (interacteur, espace de travail ou tâche ; ligne 11). Par exemple, un bouton doit être déplacé seulement sous un

autre interacteur comme une fenêtre (mais pas sous un espace de travail ou une tâche).

Algorithm 2: MergeKeepNBUP - Fusionne les éléments équivalents en gardant l'élément de la $(nb)^{ème}$ IHM et déplace vers le haut cet élément up fois.

Input: $ui_0, ui_1, \ldots, ui_n$ des composants IHM, ui une IHM composée, nb et up des entiers positifs et EQ et SUB deux ensembles des relations d'équivalence et subsumes.
Output: ui le modèle de l'IHM composée, $null$ sinon.

1 **foreach** $eq \in EQ$ **do**
2 //Pour chaque élément équivalent
3 **foreach** $v \in eq$ **do**
4 **if** $v \in ui_{nb}.V$ **then**
5 //Le $(up)^{ème}$ père est stocké dans x
6 $x \leftarrow father(v)$;
7 **for** $int\ i = 1; i < up, i++$ **do**
8 $x \leftarrow father(x)$;
9 **end**
10 //Vérifie si v peut être déplacé vers le haut up fois
11 **if** $type(x) != type(v)$ **then**
12 **Return** null;
13 **end**
14 //Un arc entre le $(up)^{ème}$ père de v et v est ajouté
15 $ui.E \leftarrow ui.E \cup (x, v)$;
16 **end**
17 //Les éléments équivalents sont supprimés
18 $ui.V \leftarrow ui.V - v$;
19 **end**
20 **end**
21 **foreach** $sub(v_1, v_2, \ldots, v_n) \in SUB$ **do**
22 //Tous les éléments subsumés sont supprimés
 $ui.V \leftarrow (ui.V - v_2 \cdots - v_n) \cup v1$;
23 //Le $(up)^{ème}$ père est stocké dans x
 $x \leftarrow father(v_{nb})$;
24 **for** $inti = 1; i < up, i++$ **do**
25 $x \leftarrow father(x)$;
26 **end**
27 //Vérifie si v_1 peut être déplacé vers le haut up fois
28 **if** $type(x) != type(v)$ **then**
29 **Return** null;
30 **end**
31 //Un arc entre le $(up)^{ème}$ père x et v_1 est ajouté
32 $ui.E \leftarrow ui.E \cup d(x, v_1)$;
33 **end**
34 $ui.E \leftarrow SuppessEdgeWithoutVertex(ui.E, ui.V)$;
35 **Return** ui ;

OPÉRATEURS DE COMPOSITION D'IHM

Parmi le grand nombre de composeurs qui peuvent être définis, nous essayons d'identifier ceux qui nous semblent être les plus utiles. Évidemment, nous ne pouvons pas prétendre être exhaustifs. Cependant, les opérateurs entre tâches peuvent être un bon moyen pour identifier les différents opérateurs de haut niveau. Ainsi, d'après notre expérience et les opérateurs entre tâches, nous choisissons de définir les composeurs suivants : InterleavingTab, InterleavingVertical, Enabling, OrderIndependence et Choice. Le composeur InterleavingVertical existe déjà dans la littérature [19, 13] mais les autres sont des contributions de ce travail.

InterleavingTab

"InterleavingTab" est en charge d'assembler un ensemble d'IHM dans un système à onglets où chaque composant est placé dans un onglet indépendant. Dans un premier temps, l'algorithme 3 sélectionne les trois éléments dans $ui_0.P_C$ ($T4$, $W4$ et $I4$) qui vont devenir les pères des éléments fournis par les sous-composants (lignes 3, 4 et 5 de l'algorithme 3). En effet, ces trois éléments deviennent liés par une relation de descendance avec les éléments fournis par $ui_1, \ldots, ui_n$ (avec $ui.D_T \leftarrow ui.D_T \cup d(fatherT, sonT)$; ligne 9). Puis, l'algorithme ajoute l'opérateur entre tâches "Interleaving" entre les tâches requises ($ui.OP_T \leftarrow ui.OP_T \cup Interleaving(leftBrother, sonT)$; ligne 12). Finalement, ce composeur fusionne et déplace vers le haut les éléments équivalents et subsumés grâce à l'algorithme $MergeKeepNBUP$ 2 (ligne 25). En conséquence, le bouton "Validate" de $C1$ est supprimé et le boutton de $C2$ et déplacé sous $I4$ dans la figure 2 pour permettre à l'utilisateur de valider les deux onglets avec un seul bouton.

InterleavingVertical

"InterleavingVertical" assemble un ensemble d'IHM verticalement dans une même fenêtre. L'algorithme sélectionne d'abord les trois éléments dans $ui_0.P_C$ ($T3$, $W3$ et $I3$) qui vont devenir les pères des éléments fournis par les sous-composants. En effet, ces trois éléments seront liés par une relation de descendance avec les éléments fournis par $ui_1, \ldots, ui_n$ ($ui.D_T \leftarrow ui.D_T \cup d(fatherT, sonT)$) de la même manière que le composeur "InterleavingTab". Finalement, cet opérateur fusionne les éléments équivalents en gardant le dernier élément ($MergeKeepNB(\{ui_0, \ldots, ui_n\}, ui, n, EQ, SUB)$).
Par exemple, l'IHM de la figure 1 sera produite mais le premier bouton "Validate" sera supprimé et fusionné avec le second.

Enabling

"Enabling" assemble un ensemble d'IHM séquentiellement. Par exemple, quand l'utilisateur termine le composant $C1$, le composant $C2$ sera exécuté. Dans un premier temps, l'algorithme sélectionne les trois éléments dans $ui_0.P_C$ qui vont devenir les pères des éléments fournis par les sous-composants. En effet, ces trois éléments sont liés par une relation de descendance avec les éléments fournis par les sous-composants. Finalement, l'algorithme ajoute les opérateurs entre tâches requis ($ui.OP_T \leftarrow ui.OP_T \cup Enabling(leftBrother, sonT)$).
Optionnellement, l'algorithme peut ajouter un bouton "Previous" dans le second composant pour permettre à l'utilisateur de revenir au précédant (par exemple, dans $C2$ pour permettre à l'utilisateur de revenir à $C1$). Cet opérateur n'effectue pas de traitement sémantique parce que les IHM à composer ne sont pas exécutées dans le même intervalle de temps.

Algorithm 3: InterleavingTab - Compose des IHM entrelassées (interleaved) dans un onglet.

Input: $ui_1, \ldots, ui_n$ un ensemble d'IHM à composer dans ui_0, EQ et SUB.
Output: ui le modèle de l'IHM composée.

$ui.V \leftarrow ui_0.V \cup \cdots \cup ui_n.V$;
$ui.E \leftarrow ui_0.E \cup \cdots \cup ui_n.E$;
$fatherT \leftarrow$la tâche dans $ui_0.P_C$;
$fatherW \leftarrow$l'espace de travail dans $ui_0.P_C$;
$fatherI \leftarrow$l'interacteur dans $ui_0.P_C$;
$leftBrotherT \leftarrow \emptyset$;
foreach *UI Model* $ui_i \in ui_1, \ldots, ui_n$ **do**
  foreach *task* $sonT \in ui_i.P_c$ **do**
    $ui.D_T \leftarrow ui.D_T \cup d(fatherT, sonT)$;
    //L'opérateur entre tâches est ajouté
    if $leftBrotherT \neq \emptyset$ **then**
      $ui.OP_T \leftarrow ui.OP_T \cup Interleaving(leftBrotherT, sonT)$;
    end
    $leftBrotherT \leftarrow sonT$;
  end
  foreach *workspace* $sonW \in ui_i.P_c$ **do**
    $ui.D_W \leftarrow ui.D_W \cup d(fatherW, sonW)$;
    //Il n'y a pas d'opérateur entre espaces de travail à ajouter.
  end
  foreach *interactor* $sonI \in ui_i.P_c$ **do**
    $ui.D_I \leftarrow ui.D_I \cup d(fatherI, sonI)$;
  end
end
//Fusion et déplacement vers le haut des éléments équivalents et subsumés
$ui \leftarrow MergeKeepNBUP(ui_0, \ldots, ui_n, ui, n, 2, EQ, SUB)$;
Return ui;

OrderIndependence

"Order Independence" assemble un ensemble d'IHM de la même manière que "InterleavingTab" ou "InterleavingVertical" sauf que l'utilisateur est obligé de terminer son interaction avec un composant avant de pouvoir interagir avec l'autre. Par exemple, l'utilisateur doit entrer son nom et prénom entièrement avant de pouvoir spécifier son login (figure 2). L'algorithme est le même que "InterleavingTab" excepté que les boutons "Validate" ne sont pas fusionnés car les IHM doivent être validées indépendamment.

Choice

"Choice" est en charge d'assembler les IHM en ajoutant, pour chaque composant, un bouton pour exécuter le composant correspondant. Par exemple, cette IHM permet à l'utilisateur de sélectionner $C1$ ou $C2$ grâce à l'IHM de la figure 4. Avec cet opérateur, le modèle de l'IHM ui_0 comprend une étiquette et deux boutons correspondant au deux composants $C1$ et $C2$. L'algorithme est presque identique à "InterleavingTab" sauf qu'il ajoute un bouton par sous-composants ($ui.OP_I \leftarrow ui.OP_I \cup navigation(leftBrotherI, sonI)$).

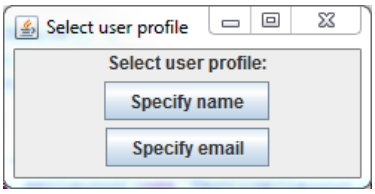

Figure 4: "Choice" permet à l'utilisateur de choisir entre $C1$ et $C2$.

PROPRIÉTÉS DE COMPOSITION

Dans cette section, nous étudions les propriétés d'assemblage et de composabilité selon les différents composeurs proposés. Seules les propriétés d'"Interleaving" et de "OrderIndependance" ont été traitées dans [7] sans tenir compte des conditions interactives.

Propriété d'assemblage

Assembler des composants IHM dans un composant composite consiste à la vérification d'une correspondance entre les éléments requis par le composant composite et l'élément fournis par les sous-composants. Grâce à notre formalisation, nous pouvons définir une relation d'assemblage :

Définition [Assemblage de composants] : Un ensemble de composants IHM $c_1, c_2, \ldots, c_n$ (avec $n \in \mathbb{N}$) peuvent être assemblés dans c_0 si et seulement si $R_{c_0} \cap (P_{c_1} \cup P_{c_2} \cup \cdots \cup P_{c_n}) \subseteq (P_{c_1} \cup P_{c_2} \cup \cdots \cup P_{c_n})$.

Composabilité d'Interleaving

La propriété de composabilité dépend de l'aspect temporel. En conséquence, tous les composeurs utilisant l'opérateur entre tâches "Interleaving" comme "InterleavingTab" ou "InterleavingVertical" doivent vérifier la propriété de composabilité suivante. En effet, l'opérateur "Interleaving" est utilisé si les IHM à composer peuvent être exécutées dans un même intervalle de temps. Ainsi, un ensemble de composants IHM et de services IHM sont composables avec "Interleaving" si les composants peuvent être assemblés et que leurs services n'ont pas de conflits. Plus formellement, un ensemble de services $s_1, s_2, \ldots, s_n$ fournit par les composants $c_1, c_2, \ldots, c_n$ peuvent être composés dans un service composite s_0 de c_0 avec "Interleaving" si et seulement si $c_1, c_2, \ldots, c_n$ peuvent être assemblés dans c_0 ; et les services peuvent être composés sans conflit : $\forall s_i, s_j \in s_1, s_2, \ldots, s_n$ avec $c_i \neq c_j$: $((Post^+(s_j) \cup Interact^+(s_j) \cup Precond^+(s_j)) \cap ((Post^-(s_i) \cup Interact^-(s_i) \cup Precond^-(s_i)) = \emptyset)$ et $((Post^-(s_j) \cup Interact^-(s_j) \cup Precond^-(s_j)) \cap ((Post^+(s_i) \cup Interact^+(s_i) \cup Precond^+(s_i)) = \emptyset)$.

Intuitivement, deux services sont indépendants si leurs postconditions et les conditions interactives ne sont pas en contradiction (conflit) avec les préconditions des autres. Par exemple, si un service active une connexion internet et qu'un autre déconnecte internet, les deux services ne peuvent pas être exécutés en même temps sinon il a un conflit. Cette propriété permet de vérifier l'indépendance de deux services quand il sont exécutés dans le même intervalle de temps.

Composabilité d'Enabling

Parce que les sous-composants peuvent être composés en séquence avec "Enabling", il est nécessaire de vérifier que les prochains sous-composants ne sont pas en conflit avec les précédents. Un ensemble de services $s_1, s_2, \ldots, s_n$ peuvent être composés dans un service s_0 avec "Enabling" si et seulement si $c_1, c_2, \ldots, c_n$ peuvent être assemblés dans c_0 ; et s'ils peuvent être *composés en séquence* : $\forall s_i \in s_1, s_2, \ldots, sc_n$ et $PostPrevious = \emptyset$, $(PostPrevious^+ \cup Post^+(s_i)) \cap Precond^-(s_{i+1}) = \emptyset$ et $(PostPrevious^- \cup Post^-(c_i)) \cap Precond^+(c_{i+1}) = \emptyset$ et $PostPrevious = (PostPrevious^+ - Post^-(s_i)) \cup (PostPrevious^- - Post^+(s_i))$. Par exemple, supposons s_0 qui déconnecte internet et s_1 qui nécessite une connexion internet, s_1 ne pourra pas être composé après s_0 grâce à cette propriété (exepté si entre temps, un autre service réactive internet dans ($PostPrevious$)).

Composabilité d'OrderIndependence

Parce que les sous-composants sont composés avec "Order Independence", ils ne peuvent pas être utilisés en même temps même s'ils sont exécutés en même temps. En conséquence, un ensemble de composants $c_1, \ldots, c_n$ peuvent être composés dans c_0 avec "Order Independence" si et seulement si $c_1, c_2, \ldots, c_n$ peuvent être assemblés dans c_0 ; et les services ne sont pas en conflit sans considérer les conditions interactives, i.e., $\forall s_i, s_j \in s_1, s_2, \ldots, s_n$ avec $s_i \neq s_j$: $((Post^+(s_j) \cup Precond^+(s_j)) \cap ((Post^-(s_i) \cup Precond^-(s_i)) = \emptyset)$ et $((Post^-(s_j) \cup Precond^-(s_j)) \cap ((Post^+(s_i) \cup Precond^+(s_i)) = \emptyset)$.

Par exemple, si les services de $C1$ et $C2$ permettent d'écouter de la musique, ces services peuvent être composés en parallèle avec "Order Independence" car les deux services ne seront pas utilisés en même temps. En effet, leur musique ne sera ainsi pas diffusée en même temps (même si les composants sont exécutés en même temps).

Composabilité de Choice

Parce que les sous-composants sont composés avec "Choice", ils ne seront pas exécutés ou utilisés en même temps. En conséquence, il n'est pas nécessaire de vérifier les conflits entre eux.

IMPLÉMENTATION ET ÉVALUATION

Cette section présente une implémentation d'un système de composition UICONDUCTOR utilisé pour développer deux logiciels et une évaluation qui prend la forme d'une étude utilisateur.

Implémentation

Nous avons implémenté UICONDUCTOR, un système capable de gérer des composants JAVA définis en accord avec le modèle à composants proposé dans ce travail. Chaque composant a au moins une tâche, un espace de travail et un interacteur. Parce qu'il est nécessaire de modifier chaque composant à partir d'un composant composite, chaque composant et ses éléments sont inspectables et modifiables et peuvent être exécutés/stoppés, visibles/cachés, activés/désactivés (i.e. le composant est visible mais l'utilisateur ne peut l'utiliser). UICONDUCTOR est capable de produire une IHM composée à partir d'un ensemble de composants et de services. Le système vérifie la composabilité des services et compose leurs composants (ou retourne une erreur sinon). UICONDUCTOR permet de produire cette IHM composée à la conception (pour accélérer le développement d'IHM ou permettre le développement d'une IHM de grande taille) ou à l'exécution (pour permettre à une IHM d'être adaptative). Par exemple, UICONDUCTOR a été utilisé pour composer un logiciel permettant de trouver un voyage dans [8] à l'exécution, et pour produire une IHM adaptative d'un logiciel de visualisation à l'exécution [6]. Pour développer ces deux logiciels, nous avons utilisé tous les composeurs définis dans ce papier ainsi qu'un schéma de composition. Un schéma de composition est une description des composants à composer incluant la mise en correspondance des interfaces. Le schéma de composition prend la forme d'une description XML car notre modèle à composants est hiérarchique (endiguement).

Evaluation

Nous voulons évaluer les deux hypothèses suivantes :
H1 : Les aspects des composeurs sont utiles pour au moins un développeur d'IHM lors de la définition de nouveaux composeurs.
H2 : Les prédicats de la logique du premier ordre sont utilisables pour définir les préconditions, conditions interactives et postconditions d'un service.

Protocole : Pour valider ces deux hypothèses, nous avons réalisé une étude utilisateur. Les sujets sont 10 étudiants en seconde année de Master. Nous considérons que ces sujets sont représentatifs de futurs développeurs car il sont à 1 ans et demi de la fin de leurs études. La moyenne d'âge est de 23 ans. Les sujets devaient développer un composant composite (avec un composeur) en 40 minutes après avoir lu les aspects de composition pendant 10 minutes. Les sujets devaient aussi décrire 7 services (préconditions, postconditions et conditions interactives) en 40 minutes après avoir découvert la définition et l'utilité de ces descriptions pour développer une IHM adaptative. Après ces travaux, les sujets devaient à chaque fois remplir un questionnaire à choix multiple durant 10 minutes pour recueillir leur avis.

Résultat : Au questionnaire réalisé après le développement d'un composant composite et de son composeur, à la question, "Est-ce que ces aspects vous ont aidé pour développer ce composant ? Oui, Non ou Je ne sais pas", 6/10 se sont servis de ces aspects et aimeraient les réutiliser lors de futurs développements. De plus, les aspects temporel et sémantique, qui sont des contributions de cet article, sont identifiés comme plus utiles (7/10 sujets les trouvent utiles) que l'aspect spatial (pour lequel 6/10 sujets pensent qu'il est utile). En conséquence, nous pouvons dire que l'hypothèse H1 est validée par les résultats de cette étude parce qu'une majorité (donc au moins 1) de sujets (6/10) ont utilisé et veulent réutiliser ces aspects pour composer des IHM.

Nous évaluons ensuite la description de 7 services associés à 7 composants. Selon une échelle d'évaluation de la qualité des productions des sujets allant de 0 à 5, la

moyenne des préconditions (3.7), conditions interactives (3.5) et des postconditions (4.1) ont été correctement définies par les sujets, i.e., les descriptions ont permis d'éviter les conflits entre composants. Concernant l'opinion des sujets recueillie par le questionnaire, à la question "Comment avez-vous trouvé la difficulté de décrire les services par des prédicats de la logique du premier ordre ? Très dur, dur, facile ou très facile.", 6 sujets sur 10 évaluent comme facile ou très facile l'utilisation de ces prédicats. Parmi ces définitions, les préconditions (8/10) et les postconditions (5/10), sont identifiées comme plus faciles à définir que les conditions interactives (4/10). En conséquence, nous pouvons dire que l'hypothèse H2 est validée car les prédicats sont appréciés et ont été correctement utilisés.

CONCLUSION

Les contributions principales de ce travail sont : un **modèle à composants** pour faire une composition par endiguement ; une **identification des aspects principaux impactant la composition** d'IHM. Plus précisément, nous identifions comme contributions les aspects temporel et sémantique (relation subsume) et une nouvelle stratégie de fusion en remontant l'élément fusionné ; un **ensemble de nouveaux opérateurs** formalisés selon les niveaux d'abstraction et intégrant les aspects spatial et sémantique ; la **définition de propriétés d'assemblage et de composition** associées aux composeurs proposés ; une **étude utilisateur** qui confirme l'utilité des aspects des composeurs et des prédicats pour définir les services IHM.

Cependant, ce travail comporte certaines limites. En effet, même si les niveaux d'abstraction permettent de modéliser tous types d'IHM, les implémentations effectuées ont été limitées à des IHM vocales et graphiques (WIMP). De plus, le modèle de composants est limité aux éléments des niveaux d'abstraction sans inclure leur comportement (évènements). En outre, les prédicats nécessitent d'être étendus par des prédicats numériques pour permettre une description des services plus précise. Pour terminer, le nombre de sujets de l'étude utilisateur est limité à 10, or, répéter l'expérience avec plus de sujets permettrait de valider l'hypothèse H2.

Comme perspective, nous envisageons de définir d'autres opérateurs de composition et d'étendre le modèle à composants pour gérer les prédicats numériques. Nous envisageons également d'étudier les propriétés ergonomiques garanties grâce à l'utilisation des composeurs proposés dans ce travail.

BIBLIOGRAPHIE

1. Attiogbé C., André P. & Ardourel G. Checking component composability. Springer (2006), 18–33.
2. Brel C., Renevier-Gonin P., Pinna-Déry A. M. & Riveill M. Application and UI composition using a component-based description and annotations. In *Proceedings - 38th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2012* (2012), 204–207.
3. Calvary G., Coutaz J., Thevenin D., Limbourg Q., Bouillon L. & Vanderdonckt J. A unifying reference framework for multi-target user interfaces. *Interacting with Computers 15*, 3 (2003), 289–308.
4. Davidyuk O., Gilman E., Milara I. S., Mäkipelto J., Pyykkönen M. & Riekki J. iCompose: context-aware physical user interface for application composition. *Central European Journal of Computer Science 1*, 4 (2011), 442–465.
5. Egenhofer M. J. & Franzosa R. D. Point set topological relations. *International Journal of Geographical Information Systems 5* (1991), 161–174.
6. Gabillon Y., Biri N. & Otjacques B. Designing an adaptive user interface according to software product line engineering. In Proc. *ACHI'15*, IARIA (2015), 86–91.
7. Gabillon Y., Calvary G. & Fiorino H. PLACID: A Planner for Dynamically Composing User Interfaces Services. In Proc. *EICS '14*, ACM (2014), 223–228.
8. Gabillon Y., Lepreux S. & de Oliveira K. M. Towards ergonomic user interface composition: A study about information density criterion. In *Human-Computer Interaction. Human-Centred Design Approaches, Methods, Tools, and Environments*, Springer (2013), 211–220.
9. Gajos K. Z., Weld D. S. & Wobbrock J. O. Automatically generating personalized user interfaces with supple. *Artificial Intelligence 174*, 12 (2010), 910–950.
10. Iribarne L., Padilla N., Criado J., Asensio J.-A. & Ayala R. A Model Transformation Approach for Automatic Composition of COTS User Interfaces in Web-Based Information Systems, 2010.
11. Karuzaki E. & Savidis A. Yeti: Yet another automatic interface composer. In Proc. *EICS '15*, ACM, 12–21.
12. Lau K. K. & Rana T. A taxonomy of software composition mechanisms. In *Proceedings - 36th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2010* (2010), 102–110.
13. Lepreux S., Hariri A., Rouillard J., Tabary D., Tarby J.-C. & Kolski C. Towards multimodal user interfaces composition based on usiXML and MBD principles. In Proc. *Lecture Notes in Computer Science*, J. A. Jacko, Ed., vol. 4552 of *Lecture Notes in Computer Science*, Springer (2007), 134–143.
14. Lepreux S. & Vanderdonckt J. Towards A support of user interface design by composition rules. In *CADUI*, G. Calvary, C. Pribeanu, G. Santucci, and J. Vanderdonckt, Eds., Springer (2006), 231–244.
15. Lewandowski A., Lepreux S. & Bourguin G. Tasks models merging for high-level component composition. In Proc. *Lecture Notes in Computer Science*, J. A. Jacko, Ed., vol. 4550 of *Lecture Notes in Computer Science*, Springer (2007), 1129–1138.
16. Paternò F., Mancini O. & Meniconi S. Concurtasktrees: a diagrammatic notation for specifying task models. In *Proceedings of IFIP INTERACT'97: Human-Computer Interaction* (1997), 362–369.
17. Paternò F., Santoro C. & Spano L. D. MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Transactions Computer-Human Interact 16*, 4 (2009).
18. Pietschmann S., Voigt M., Rümpel A. & Meiß ner K. CRUISe: Composition of rich user interface services. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5648 LNCS (2009), 473–476.
19. Pinna-Dery A.-M., Fierstone J. & Picard E. Component model and programming: A first step to manage human computer interaction adaptation. In Proc. *Lecture Notes in Computer Science*, L. Chittaro, Ed., vol. 2795 of *Lecture Notes in Computer Science*, Springer (2003), 456–465.
20. Ponnekanti S. R., Lee B., Fox A., Hanrahan P. & Winograd T. Icrafter: A service framework for ubiquitous computing environments. In *Ubicomp 2001: Ubiquitous Computing*, Springer (2001), 56–75.
21. Sametinger J. *Software engineering with reusable components*. Springer Science & Business Media, 1997.
22. Tsai W.-T., Huang Q., Elston J. & Chen Y. Service-oriented user interface modeling and composition. IEEE (2008), 21–28.
23. Xie F. & Browne J. C. Verified systems by composition from verified components, 2003.