Composition opportuniste de fragments d'IHM pour une interaction adaptative en environnement ambiant

Augustin Degas, Sylvie Trouilhet, Jean-Paul Arcangeli, Gaëlle Calvary, Joëlle Coutaz, Stéphane Lavirotte, Jean-Yves Tigli

Composition opportuniste de fragments d'IHM pour une interaction adaptative en environnement ambiant

Augustin Degas, Sylvie Trouilhet, Jean-Paul Arcangeli
IRIT, Université de Toulouse, Toulouse, France
Prenom.Nom@irit.fr

Gaëlle Calvary, Joëlle Coutaz
Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France
CNRS, LIG, F-38000 Grenoble, France
Prenom.Nom@imag.fr

Stéphane Lavirotte, Jean-Yves Tigli
Université Nice Sophia Antipolis
CNRS (UMR 7271), I3S, Sophia Antipolis, France
Prenom.Nom@unice.fr

RESUME

Dans ce papier nous proposons une approche basée sur un système multi-agents adaptatif, en utilisant les principes de Méta-Interactions Homme-Machine et d'Opportunisme dans le but de résoudre le problème de la Composition d'Interactions Homme-Machine dans les espaces interactifs ambiants. L'idée de cette approche est de voir chaque composant comme un agent capable d'interagir avec les autres composants pour composer de manière autonome et ainsi suggérer à l'utilisateur de manière opportuniste des compositions de son environnement ambiant interactif. Nous avons choisi de présenter principalement deux aspects de la composition d'interactions homme-machine, la contrôlabilité et l'objectif de la composition. Enfin nous illustrons notre approche avec des cas d'utilisation dans le cadre d'un projet nommé neoCampus.

Conceptes CSS

• Computer systems organization→Self-organizing autonomic computing

• Computing methodologies→Multi-Agent systems, Cooperation and coordination

• Human-centered computing→Ambient Intelligence, User interface management systems, User interface programming

Mots-clés

informatique ubiquitaire ; espaces ambiants interactifs ; interactions homme-machine ; composition opportuniste de composants ; méta-interactions homme-machine ; théorie des systèmes multi-agents adaptatifs

1. Introduction

L'objectif de l'intelligence ambiante est d'offrir aux utilisateurs un espace «intelligent» qui leur permette d'accéder aux informations et aux services numériques de façon appropriée et naturelle. L'environnement devient alors un espace interactif. Par «espace interactif», nous entendons tous les appareils et toutes les façons d'interagir avec eux, mais aussi toutes les applications logicielles disponibles pour l'utilisateur. L'intelligence ambiante découle de l'idée originelle d'"Ubiquitous Computing" de Mark Weiser [23], autrement dit, une technologie informatique si profondément intégrée dans notre vie quotidienne que nous ne prêtons plus attention à elle. Cette technologie devient, à l'image de la littérature, omniprésente pour tous et tous les jours. Deux éléments peuvent être soulignés dans la métaphore de la littérature. On peut noter que la limite entre le monde de la littérature et le monde humain est poreuse, amenant naturellement la littérature comme partie intégrante de notre vie. L'autre point est le nombre important d'occurrences d'objets littéraires qui peuplent notre vie quotidienne. L'intelligence ambiante présente également une frontière floue avec le monde de l'utilisateur et peuple ce dernier d'une multitude de logiciels et de dispositifs.

Le concept d'intelligence ambiante ainsi défini, se plaque tout naturellement sur notre domaine d'application. En effet, nous cherchons à inciter et à améliorer la participation et l'implication des étudiants pendant les cours ; cet enseignement est synchrone mais dispensé dans un environnement distribué : les étudiants sont dans plusieurs salles de cours et peuvent également être chez eux ou même sur le chemin du campus.

Dans ce contexte, de nombreux dispositifs sont disponibles, de nouveaux peuvent apparaître parce que les utilisateurs sont mobiles, et enfin d'autres dispositifs peuvent évoluer ou disparaître (par exemple à cause de batteries faibles).

Dans cet environnement en constant changement, l'utilisateur doit être au centre du processus et pouvoir agir en continu sur son espace. Cela signifie que les interactions Homme-Machine (IHM) doivent évoluer dynamiquement en fonction des changements de contexte, le contexte étant défini comme le triplet utilisateur-environnement-plateforme [4]. Cette capacité à évoluer en fonction de la dynamique de l'environnement est appelée plasticité des IHM [19]. En d'autres termes, l'IHM peut être vue comme un assemblage adapté et flexible qui a été composé en fonction de ce qui est disponible dans l'environnement et qui peut se recomposer pour prendre en compte des disparitions ou de nouvelles opportunités.

De plus, nous pensons que l'utilisateur n'est pas toujours capable d'identifier ses besoins, et cela semble d'autant plus vrai dans le domaine des IHM, où il peut être ardu d'expliciter ses besoins en matière d'interaction et par conséquent les composants d'interaction qui seraient les plus appropriés. Aussi, une assistance suggérant à l'utilisateur tout ou partie d'une interface pourrait être utile.

L'objectif de ce papier est dans un premier temps de définir les problématiques d'IHM plus spécifiques à un environnement ambiant puis de définir notre approche qui vise à assister l'utilisateur dans ce contexte et à lui proposer des applications susceptibles de lui être utiles.

La section qui suit introduit la problématique de la composition de composants d'IHM puis s'intéresse à deux aspects essentiels lorsque les éléments composés sont des fragments

d'IHM et aux solutions qui ont pu être proposées dans différents travaux. Le problème de la contrôlabilité (c'est-à-dire du degré de contrôle de l'utilisateur sur la composition) est étudié en premier lieu. Le second aspect examiné porte sur la finalité de la composition.

La section III présente notre approche qui en est encore à sa phase de conception. Elle tente de fournir des réponses aux problèmes mis en évidence dans la section II et est basée à la fois sur une méta-interface utilisateur, une composition opportuniste et un système multi-agent qui prend en charge la composition.

La quatrième et dernière section présente des cas d'utilisation pour illustrer la composition opportuniste de fragments d'IHM, les apports de cette dernière ainsi que les exigences à prendre en compte.

2. La composition de fragments d'IHM et ses exigences

2.1 Composition de fragments d'IHM

La composition de composants consiste à assembler des composants logiciels entre eux. Bien que la notion de composant logiciel ait été introduite il y a près de cinquante ans par Douglas McIlroy lors de la première conférence de SE (software enginering) en 1968 [17], la plupart des travaux de recherche sur ce sujet ont été effectués ces vingt dernières années [22]. Ces différents travaux « restent généraux, non encore appliqués aux interactions homme-machine » [3].

La notion de composition intéresse les concepteurs d'Interface Homme-Machine (IHM) pour de multiples raisons. Elle présente en particulier un intérêt pour le concepteur d'IHM qui souhaite réutiliser dans ses nouveaux projets ce qui a été produit précédemment, pour accélérer la phase de développement. Elle semble aussi intéressante dans le domaine de l'informatique mobile et ambiante, pour permettre à des systèmes de s'adapter dynamiquement à la fois à la tâche de l'utilisateur et au contexte d'usage [4] (c'est-à-dire à l'environnement, à l'utilisateur et à la plate-forme). L'ingénierie de l'IHM plastique s'intéresse ainsi à l'adaptation d'une IHM à son contexte d'usage [19]. Un exemple simple illustrant cette interaction adaptative est celui du voyageur arrivé en gare qui descend de son train et veut chercher un hôtel. Il se retrouve devant un écran affichant la carte de la ville. Et quand l'écran détecte son smartphone dans sa poche, on lui propose de manipuler la carte par l'intermédiaire du smartphone.

Dans le cadre de la composition d'IHM, un composant peut être aussi bien un composant du noyau fonctionnel (NF) qu'un composant d'interface (c'est-à-dire un élément graphique d'IHM). Dans la suite, un composant d'interface est appelé un fragment d'IHM. Un fragment d'IHM peut être un widget, par exemple un bouton, une partie d'IHM, ou une IHM complète. Ces composants peuvent donc être de granularité quelconque et de niveau d'abstraction quelconque. L'ensemble des composants possibles est l'union de l'ensemble des sous-parties du NF et de l'ensemble des sous-parties de l'IHM.

De plus, la composition d'IHM peut être le résultat d'un assemblage de fragments d'IHM et de composants de noyau fonctionnel. Cela peut être aussi un désassemblage d'IHM existantes puis un ré-assemblage de certains morceaux ; il s'agit alors d'une fusion.

Au travers de plusieurs systèmes de composition d'IHM étudiés, nous avons pu mettre en évidence deux aspects importants pour une interface plastique en environnement ambiant, le premier étant le degré de contrôle de la composition par l'utilisateur et le second l'objectif de la composition.

2.2 Contrôlabilité de la composition

La contrôlabilité est le degré de contrôle qu'opère l'utilisateur ou le concepteur sur la composition. Ces derniers ont souvent un contrôle partiel ou total du contrôle de la composition, aussi bien à la conception qu'à l'exécution.

La composition peut être totalement automatique comme dans *Compose* [12] ; en effet, la composition est déclenchée par l'explicitation d'un besoin par l'utilisateur mais par la suite, la composition est entièrement prise en charge par le système.

La contrôlabilité est un peu moins faible si le système de composition gère un peu moins de choses. *Task Tree Merge* [16] et *Alias* [14] sont deux exemples où la composition est majoritairement faite par le système, mais l'initiative vient du concepteur, puisqu'il va choisir les applications qui vont être fusionnées. Il va également gérer les conflits.

A l'inverse, la composition peut s'effectuer sous le contrôle complet du concepteur comme pour *ComposiXML* [15] ou de l'utilisateur comme pour *On-the-fly-services* [24]. Ils sont à l'initiative de la composition et en sont aussi les acteurs du début à la fin.

La contrôlabilité est ainsi soit très faible, soit très forte. Pourtant, le système de composition pourrait se positionner entre ces deux extrêmes et proposer de façon proactive des compositions, en laissant la validation à l'utilisateur. Le système serait présent à chaque étape de la composition pour anticiper sur le besoin de l'utilisateur (on pourrait comparer cela à l'auto-complétion qui assiste l'utilisateur dans la frappe de son texte). Un exemple de ce contrôle hybride est celui *d'ONTOCOMPO* [1] : le concepteur choisit les composants à ajouter ou à enlever et le système l'assiste en gérant les dépendances entre composants.

Nous pouvons relever deux points importants à partir de ces systèmes. Les systèmes de composition ont tendance soit à exclure l'utilisateur du processus d'assemblage, soit à laisser à sa charge la totalité du processus. Le second point à noter concerne la localisation des composants sélectionnés. Les applications sont créées à partir de ce qui est disponible sur la station de travail de l'utilisateur et non à partir de tout son espace interactif. Les systèmes n'utilisent donc pas toutes les possibilités offertes par les dispositifs disponibles dans l'environnement. Les systèmes pourraient aller plus loin et être capables de prendre en compte la dynamique de l'environnement pour maintenir les assemblages.

2.3 Objectif de la composition

Les systèmes de composition d'IHM étudiés sont guidés par deux buts différents, introduisant alors deux méthodes pour composer. Le premier but consiste à réutiliser les composants en vue de faciliter et d'accélérer la conception des applications. L'objectif dans ce cas est de fusionner des applications existantes ou des parties d'applications pour en faire une seule rassemblant les caractéristiques des parties fusionnées. Cela peut être aussi fait pour construire une application statique, c'est-à-dire une application qui n'évoluera plus même si le contexte d'utilisation change. Les systèmes *ComposiXML*, *Task Tree Merge*, *ONTOCOMPO* et Alias sont conçus dans cet objectif. Ce type de composition permet de diminuer le coût de développement et le temps de déploiement.

Le second but est de faire des applications capables de s'adapter dynamiquement en fonction du contexte d'utilisation et de la tâche de l'utilisateur. Il s'agit alors de laisser l'utilisateur final développer des applications *(On-the-fly service composition*, *SOAUI* [21]), ou bien encore de développer automatiquement des applications (*Compose*).

Notre objectif est de concevoir une interface capable d'évoluer au fur et à mesure des changements de l'environnement,

c'est-à-dire de proposer une interface plastique. Dans un contexte mobile et ubiquitaire, il s'agit d'utiliser ce qui peut être utilisé et de changer en fonction de ce qui devient utilisable. En effet, les composants apparaissant et disparaissant au fil du temps, cela évite d'avoir une application qui ne soit plus opérationnelle parce qu'un composant pré-sélectionné n'est pas disponible.

Un dernier aspect non négligeable et concernant l'objectif de la composition est lié à l'explicitation du besoin de l'utilisateur. Dans plusieurs systèmes, la composition est déclenchée et guidée par un besoin (*ComposiXML*, *ONTOCOMPO*, *On-the-fly service composition*, *SOAUI*). Pour d'autres, comme *Task Tree Merge* et *Alias*, la composition s'effectue à partir de ce que l'utilisateur a décidé de fusionner et est donc elle aussi guidée par un besoin (pour éviter dans ce cas la redondance d'information ou d'actions). Enfin, d'autres systèmes font un assemblage à partir d'un besoin explicité par l'utilisateur (*Compose*). La composition ne peut donc se faire sans avoir identifié et explicité les besoins de l'utilisateur final. Cependant, si un concepteur est capable d'identifier les exigences à satisfaire pour une application, ce n'est pas toujours le cas pour un utilisateur. Ce dernier peut éprouver des difficultés pour formuler de façon précise ce dont il a besoin et identifier les composants d'interface entrant en jeu dans l'application, d'où l'intérêt de l'assister dans cette tâche.

2.4 Synthèse

Nous avons, dans cette section, voulu caractériser la composition d'IHM et mis en évidence trois exigences à satisfaire dans le cas de composition d'IHM en environnement ambiant.

La première exigence est liée à la contrôlabilité de la composition. L'utilisateur doit être intégré dans le processus de composition et doit pouvoir à la fois observer et contrôler son espace interactif.

La seconde exigence est liée à la contrôlabilité mais aussi à l'objectif de la composition. Le système à concevoir doit être sensible au contexte d'utilisation mais aussi capable d'évoluer en fonction de ce contexte.

La troisième et dernière exigence est liée à l'objectif de la composition, en visant à partiellement automatiser la composition et à assister l'utilisateur tout au long des compositions et des recompositions.

Maintenir l'utilisateur dans la boucle, tout en automatisant la composition semble a priori antagoniste. Cependant, aucun de ces deux aspects ne doit être exclu dans le système de composition envisagé qui doit trouver un juste milieu entre un contrôle total de l'utilisateur et un contrôle total du système..

La section qui suit tente de donner des éléments de réponse pour satisfaire ces exigences. L'approche que nous envisageons est une association entre une méta-IHM et un moteur multi-agent de composition opportuniste.

3. Une proposition de composition d'IHM en environnement ambiant

Cette section décrit les principes de notre approche basée sur une méta-interface utilisateur et une approche de composition opportuniste. L'implémentation est à base d'une système multi-agent adaptatif dont nous présentons les caractéristiques.

3.1 Méta-interface utilisateur

La méta-interface utilisateur ou méta-IHM peut être une réponse à la nécessité de permettre à l'utilisateur d'observer et de contrôler son espace ambiant interactif. La notion de méta-IHM a été introduite par Joëlle Coutaz [8]. Le point de départ de cette notion est le même que celui présenté précédemment pour les environnements ambiants :“users are not limited to the system and applications of a single computer […] [. U]sers, services, and resources discover other users, services and resources, and integrate them into an ambient interactive space” [8]. Cet environnement interactif est amené à évoluer et n'est pas limité à une seule station de travail. Aussi, dans un tel environnement, les solutions usuelles ne sont plus appropriées (tels les scripts shell pour une station) et il est nécessaire de pouvoir intégrer l'utilisateur pour lui permettre d'agir sur son espace. C'est ainsi que [8] définit le concept de méta-IHM comme “the set of functions (along with their user interfaces) that are necessary and sufficient to control and evaluate the state of interactive ambient spaces”. Pour synthétiser, il s'agit de recenser les entités qui peuvent être présentes dans un espace interactif, de savoir comment on peut les manipuler et ce qu'il est possible de faire avec elles.

Ainsi, dans notre système, nous souhaitons proposer à l'utilisateur une méta-IHM qui lui permettre d'observer l'état de la composition, c'est-à-dire les composants disponibles. Cependant, l'utilisateur n'aurait pas en charge toute la composition qui serait partiellement automatisée. En effet, la méta-IHM serait présente au moment des compositions et des recompositions pour lui proposer les choix les plus pertinents et prendre en compte ses choix préférentiels.

3.2 Approche opportuniste

Dans les méthodes traditionnelles de développement, les logiciels sont généralement mis au point pour répondre à des besoins explicités et pré-établis ainsi qu'aux exigences des différentes parties prenantes. La composition descendante de composants logiciels permet de prendre en compte ces besoins. Cependant, il devient de plus en plus essentiel de pouvoir prendre en compte l'évolution du contexte d'utilisation parce que les besoins et exigences restent rarement inchangés. Une solution envisageable pour prendre en compte cette dynamique, consiste à concevoir une application qui tienne compte de tous les contextes d'utilisation. Malheureusement, dans un environnement ambiant, il est impossible de prédire exhaustivement toutes les situations possibles.

C'est de ce constat que part l'idée de composition opportuniste : comme il est impossible de décrire chaque situation possible et d'y associer l'action adéquate, on préfère faire ce qui est possible en fonction de ce qui est disponible... plutôt, faisons un moteur de composition capable de composer une application en fonction de la situation du moment, c'est-à-dire en fonction des composants disponibles dans l'environnement à un instant t. La recomposition consiste à refaire une application lorsque la situation change. Le processus de développement de logiciels traditionnels est alors inversé. Alors que le processus traditionnel commence par l'analyse des besoins, l'approche opportuniste est déclenchée par ce qui est disponible dans l'environnement pour construire une application, et examiner ensuite l'intérêt d'une telle application. Cette approche ascendante permet de faire émerger une application de l'environnement et de la faire évoluer en fonction de nouvelles opportunités.

Une approche ascendante de composition proche de ce que nous envisageons a été utilisée dans [9]. Dans cette approche, les composants sont inclus dans des conteneurs appelés Service Lightweight Component Architecture (SLCA). Chaque composant est soit un composant logiciel, soit un proxy vers un service. Une application est décrite par un ensemble de règles qui peuvent être dynamiquement implémentées à l'exécution. Ces règles sont prises en charge par un tisseur d'aspect d'assemblage. L'opportunisme de cette approche porte sur le choix des règles ; en effet le système sélectionne les aspects d'assemblage les plus appropriés au contexte. Cependant, nous voudrions que notre approche soit

indépendante de toutes règles ou modèles à instancier de façon à pouvoir fonctionner même si on se trouve face à une situation pour laquelle aucune règle n'a été prévue. C'est un principe qui nous semble nécessaire pour avoir un système sensible au contexte et à ses évolutions.

3.3 Systèmes Multi-Agents Adaptatifs pour la Composition

Nous avons donc pensé à un moteur de composition opportuniste de fragments d'interface basé sur la technologie mulit-agent. J.-P. Briot avait déjà noté l'intérêt d'utiliser les agents comme assistants à la composition de composants [2]. C'est cette idée que nous reprenons en nous appuyant sur un système multi-agent adaptatif. Un agent est une entité autonome qui a une vision locale de l'environnement qui l'entoure et qui agit avec et sur son environnement pour atteindre un but local. Parce que les agents communiquent et coopèrent entre eux pour accomplir des actions globales, ils forment un système. Le système est adaptatif car il possède la capacité d'évoluer par lui-même en fonction des changements dans l'environnement. Cependant, aucun élément du système n'a une vision globale de ce qu'il se passe ou de l'algorithme général. Ce ne sont que les actions locales des agents qui amènent à la résolution collective.

Pour cela chaque agent du système a un comportement nominal. Il fonctionne selon le cycle "perception – décision – action". Il peut arriver que le comportement d'un agent ne soit pas coopératif ; sept situations non coopératives génériques sont identifiées dans la théorie des AMAS [5]. Si une de ces situations est détectée, l'agent change son comportement pour revenir dans une situation coopérative.

Notre système est constitué de trois types d'agents construits selon le modèle AMAS : les agents composants, les agents instances de composants et les agents services. Un agent composant prend en charge la gestion d'un composant du noyau fonctionnel ou d'interface. Un composant fonctionnel ou d'interface peut être considéré comme une entité autonome. De plus, les composants sont naturellement décentralisés.

Chaque composant est donc agentifié et correspond à un agent de type composant et des agents de type instance. Ces agents doivent interagir et communiquer avec leur environnement pour connecter leurs interfaces requises et fournies. Ce sont les agents du dernier type, à savoir les agents services, qui prennent en charge la connexion des interfaces, plus précisément, la connexion d'une interface, si elle n'était pas encore connectée ou sa déconnexion puis sa reconnexion.

Nous ne devons pas perdre de vue, que l'utilisateur doit trouver sa place dans cette architecture : il doit pouvoir suivre les différentes compositions et pouvoir intervenir sur les propositions que peut lui faire le moteur multi-agent de composition opportuniste (puisque nous proposons une contrôlabilité médiane, c'est-à-dire un contrôle partiellement automatisé mais en maintenant possible une intervention de l'utilisateur, comme préconisé dans [8]).

Nous cherchons donc à assister l'utilisateur, ce dernier n'étant pas toujours capable d'expliciter ses besoins et donc de choisir les composants d'interaction adaptés. Le moteur pourrait ainsi lui faire des suggestions qui pourraient être des parties de la composition finale (l'assemblage de deux composants, par exemple), mais également une application complète.

Comme les applications émergent du travail coopératif des agents du système, les suggestions sont différentes compositions possibles de composants. Nous envisageons de permettre à l'utilisateur de choisir parmi ces différentes possibilités par le biais d'une méta-IHM.

On peut penser que les suggestions peuvent être nombreuses et donc difficiles à présenter dans la méta-IHM. Cependant, les agents sont dotés de capacités d'adaptation et capables de faire des choix différents en fonction du contexte d'utilisation, donc plus ils auront été confrontés à de nouveaux contextes d'utilisation, plus ils pourront sélectionner parmi toutes les possibilités celles qui sont le plus appropriées.

La section qui suit présente quelques exemples illustrant la composition d'IHM dans le cadre d'une expérience pédagogique visant à renforcer l'implication des étudiants lors de cours à distance.

4. Cas d'utilisation

Ce cas d'utilisation se déroule dans deux salles de cours et dans la chambre d'un étudiant. On va dans un premier temps montrer des compositions possibles, puis illustrer à partir de ces compositions possibles les points précédemment évoqués, pour finir en évoquant de nouveaux points à prendre en compte.

Dans cet exemple, de nombreux composants et dispositifs peuvent être présents. Le cadre de l'exemple étant un cours se passant dans une salle, retransmis dans une autre salle de cours et dans la chambre des étudiants, on peut trouver par exemple des caméras dans la première salle de cours (pour regarder le professeur ou le tableau) mais aussi dans la deuxième salle de cours (pour voir un élève qui pose une question par exemple). On peut aussi retrouver d'autres dispositifs, comme des tableaux interactifs et des microphones dans les différentes salles, mais aussi des enceintes, des écrans, des tablettes, des téléphones portables et des ordinateurs portables. La figure 1 illustre ses différents dispositifs en les répartissant en fonction de leur localisation : les trois rectangles représentent les trois localisations, le rectangle de gauche représente la salle de cours avec le professeur, celui en haut à droite la salle sans professeur et le dernier représente la chambre de l'étudiant. Les différentes flèches symbolisent l'inter-connexion des différentes salles.

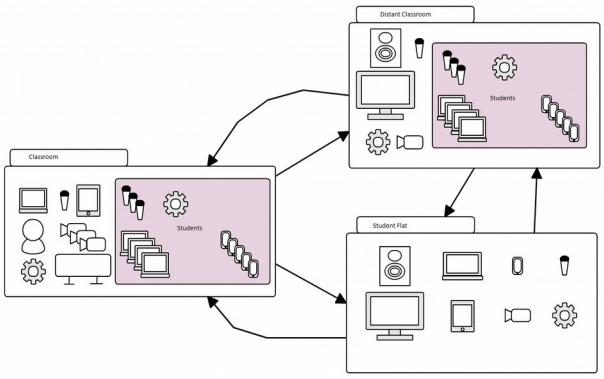

Figure 1. Un cours se tenant dans une salle de cours, retransmis dans une autre salle et la chambre d'un étudiant

Dans cet environnement ambiant, de nombreux exemples peuvent être pris pour illustrer ce qu'on entend par une composition opportuniste d'IHM, mais comme on peut l'imaginer, le nombre possible de compositions est assez important. Ainsi, pour simplifier, nous allons nous concentrer sur un petit exemple avec une caméra et une tablette (néanmoins le choix de ces éléments peut aussi être des compositions finales présélectionnées et suggérées à l'utilisateur par le système de composition, parmi de nombreuses possibilités).

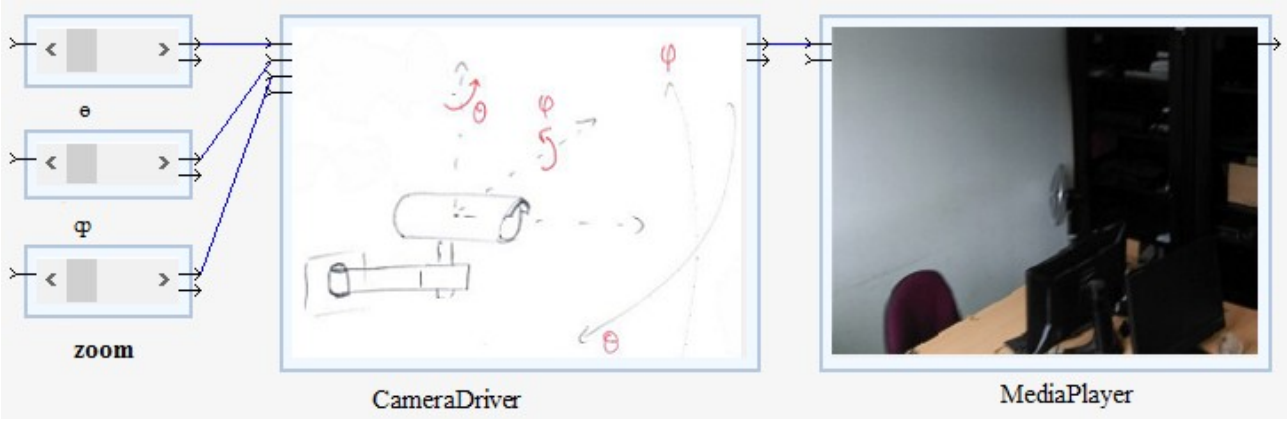

Figure 2. Illustration de la composition (1) : - > sont des interfaces fournies et > - sont des interfaces requises

Dans notre exemple, le driver de la caméra est un composant qui requiert la gestion de ses trois commandes : ses deux angles de rotation (ɵ pour l'angle de lacet et φ pour l'angle de roulis – le dessin du driver de la caméra de la Fig.3. illustre cela) qui sont des *floats*, et de son zoom qui est aussi un *float*. Ce même driver fournit un flux vidéo, qui est ce que la caméra est en train de filmer. Ce composant va demander de manière autonome dans son environnement si d'autres composants fournissent des moyens de gérer ses commandes et/ou requièrent un flux vidéo.

Les composants de la tablette étant les seuls présents dans notre exemple, ils seront aussi les seuls à répondre au composant driver. Pour l'exemple, disons pour simplifier que les seuls composants de la tablette sont des *sliders* et un composant capable d'afficher un flux vidéo. Pour simplifier encore un peu, on va dire que les *sliders* ont tous la même apparence excepté leur orientation qui peut être verticale ou horizontale. Dans cet exemple très simplifié, on a déjà 54 compositions possibles (3 contrôles requis avec 3 états possibles : non connecté, connecté à un *slider* vertical ou connecté à un *slider* horizontal, et un flux vidéo fourni qui est connecté ou pas) ou 8 compositions possibles si on considère que chaque requis et chaque fourni de la caméra doit être connecté. On va par la suite considérer quatre exemples de composition finale, qui peuvent être des suggestions faites à l'utilisateur via la méta-IHM :

• Deux *sliders* horizontaux gèrent φ et le zoom et un *slider* vertical gère ɵ . (1)

• Trois *sliders* horizontaux gèrent ɵ, φ et le zoom. (voir Figure 2) (2)

• Deux *sliders* horizontaux gèrent ɵ et le zoom et un *slider* vertical gère φ. (3)

• Un *slider* horizontal gère ɵ et deux *sliders* verticaux gèrent φ et le zoom. (4)

Et dans toutes ces suggestions, un composant de la tablette gérera le flux vidéo.

Dans ces différentes suggestions, l'une est particulièrement mauvaise pour l'utilisateur, une est plus acceptable mais sous-optimale, et deux sont acceptables. Par acceptable, nous entendons ,entre autre, que ces suggestions sont ergonomiques, aident l'utilisateur à savoir comment l'application marche, ne l'induisent pas en erreur et essayent d'être facile d'utilisation. On retrouve cette notion sous le terme de « Value » [6] ou de « Worth » [7], qui est en fait la valeur auquel un utilisateur associe une IHM. En bref, une IHM acceptable, qui a de la valeur, qui vaut le coût, est une IHM qu'on a envie d'utiliser et de recommander à d'autres.

La suggestion (1) est particulièrement mauvaise parce qu'elle induit l'utilisateur en erreur : alors qu'il s'attendrait au choix « logique » qu'est un *slider* vertical pour gérer φ parce que cet angle est situé dans un plan vertical, le *slider* vertical gère un angle situé dans le plan horizontal (ɵ). La deuxième est plus acceptable car elle n'induit pas l'utilisateur en erreur, en revanche elle ne l'aide pas à déterminer ce que manipule chaque *slider.* Les deux dernières, (3) et (4), sont les plus acceptables car elles n'induisent pas l'utilisateur en erreur et l'aident même à deviner comment l'application fonctionne. On peut dire que la dernière (4) est la plus appropriée parce qu'elle respecte les habitudes des utilisateurs (le zoom est en général représenté verticalement).

Cet exemple montre que, pour la composition d'IHM, on doit prendre en compte d'autres critères en plus de ceux que nous avons définis dans [20] pour la composition opportuniste, qui sont : décentralisation, adaptation dynamique, optimisation combinatoire, recomposition, apprentissage et conscience du contexte, utilité du résultat et enfin capacité à évoluer sans expliciter les besoins de l'utilisateur. Ceci implique donc que, dans la composition d'IHM, la simple association d'interfaces (un requis avec un fourni qui lui correspond) n'est pas suffisante et que l'utilisateur doit participer activement à la composition. Par exemple, dans notre cas, d'un point de vue strictement fonctionnel, les deux *sliders* ont la même qualité, mais du point de vue de l'utilisateur, utiliser le *slider* vertical pour φ est plus approprié. Contrairement à des critères qu'on peut quantifier, et à partir desquels on peut associer une valeur et distinguer deux éléments, certains critères sont dépendants de l'appréciation d'un utilisateur. Par exemple on peut distinguer deux écrans de même taille par leur résolution. On peut dire qu'utiliser un *slider* vertical pour φ est plus logique parce que dans les deux cas on trouve une notion de verticalité, en revanche le choix du *slider* vertical pour le zoom est dépendant de l'habitude des utilisateurs et pas de la sémantique. C'est parce que l'utilisateur est habitué à cette représentation qu'il continue de l'utiliser.

Supposons maintenant qu'on ait la possibilité d'utiliser des *sliders*, mais aussi d'autres objets numériques. Par exemple on peut associer le zoom au gestes de *pinch*/*spread* de la tablette, φ à un *swipe* vertical et ɵ à un *swipe* horizontal. Si on permet à un concepteur d'IHM d'utiliser ces gestes tactiles sur une tablette, il va les utiliser, mais pourquoi ? Est-ce par habitude ou pour utiliser moins de place sur ces petits écrans ? Si c'est par habitude, pourquoi n'a-t-on pas gardé les *sliders* des ordinateurs, et si c'est pour utiliser moins de place, pourquoi n'utilise-t-on pas les gyroscopes des tablettes ? Au delà du fait que chaque tablette n'a pas forcément un gyroscope suffisamment précis, il y a aussi une question de pratique : en forçant l'utilisateur à incliner la tablette,

on dégrade sa vision de l'écran. De nouveau, on rencontre des critères utilisateurs ; les concepteurs voulaient utiliser moins d'espace sur les écrans qu'ils voulaient petits et ont choisi ce qui semblait le plus approprié pour les deux côtés (homme et machine). D'une certaine façon c'est compréhensible puisque une IHM est une interface entre l'homme et la machine.

Dans la composition d'IHM, la simple association d'interface n'est pas suffisante et l'utilisateur doit donc être un acteur actif de la composition de son espace interactif. Néanmoins, on a beaucoup simplifié l'exemple, et même dans ces conditions, beaucoup de suggestions sont possibles. Certains critères peuvent être appris pas la machine : si un utilisateur choisit la même suggestion plusieurs fois parmi un même ensemble de suggestion (ou un ensemble évoluant), le système peut l'apprendre. Ces préférences apprises par le système peuvent restreindre le nombre de suggestions, fournissant ainsi une meilleure aide à l'utilisateur pour qu'il contrôle son environnement ambiant et interactif, et contribuant au besoin d'optimisation combinatoire [20].

On a aussi introduit brièvement une notion qu'on veut utiliser dans notre composition d'interfaces, qui est l'abstraction de composant d'interface utilisateur par ce qu'ils peuvent gérer, par exemple un *slider* peut gérer un *float*. Cette idée n'est pas vraiment nouvelle, on peut par exemple la trouver dans [11], un article datant de plus de trente ans, cependant cette abstraction est une idée intéressante pour ce qu'on veut faire, en particulier car elle ouvre de nombreuses possibilités : un nouveau composant qui est capable de gérer un *float* est plus facilement inclus dans une nouvelle composition qu'un composant qui se présente comme « *slider* » ou « *track bar* », ce qui nécessite de la part des autres composants de savoir ce qu'il peut faire. Cette idée ouvre aussi des compositions insolites ou inattendues. Par exemple on a utilisé précédemment des *sliders*, ces *sliders* peuvent être abstraits comme étant capables de gérer une variable *float* monodimensionnelle dans un intervalle, mais aussi différents booléens (si le *slider* est cliqué ou pas, s'il a le focus ou pas…). Cette abstraction peut aussi être utilisée pour représenter des moyens d'interaction non conventionnels, on peut par exemple dire qu'une porte peut communiquer son angle d'ouverture (disons entre 0 et 90) et qu'ainsi elle peut gérer un *float* monodimensionnel dans un intervalle, mais aussi des booléens (si la porte est ouverte ou non…). Des volets roulants peuvent être aussi représentés de la sorte, et gérer un *float* monodimensionnel dans un intervalle (par exemple un volet a moitié ouvert correspond à 50 dans [0;50]). Et si on permet de telles abstractions, une personne qui ne bénéficie d'aucun autre moyen d'interaction avec la caméra, comme un étudiant sans ordinateur avec un téléphone cassé, peut gérer ɵ avec la porte et ɋ avec le volet roulant.

5. Conclusion

Les environnements interactifs ambiants demandent de satisfaire différentes exigences pour obtenir des applications par composition de composants logiciels ou d'IHM. Ces environnements par nature sont dynamiques et ainsi les applications basées sur ces environnements doivent évoluer avec eux. Néanmoins, une telle évolution ne peut pas exclure l'utilisateur de la boucle. Pour inclure l'utilisateur dans le processus de composition, on doit rendre ce dernier accessible et non chronophage. En partant du faible couplage des différentes entités impliquées dans les environnements ambiants interactifs, une utilisation des composants et de la composition semble appropriée. Cependant il est nécessaire de changer les usages et d'utiliser les composants différemment, car l'utilisation qui en est faite actuellement ne répond pas aux spécificités des environnements interactifs ambiants (les besoins des utilisateurs évoluent dynamiquement, l'environnement évolue, et l'utilisateur doit être impliqué dans le processus…).

Au besoin de s'adapter on propose un approche basée sur l'opportunisme : dans cette approche les applications émergent de la composition de ce qui est disponible dans l'environnement et ainsi va évoluer avec lui dynamiquement. Au besoin d'impliquer l'utilisateur dans le processus de composition, on propose de rendre le système de composition capable de faire des suggestions à l'utilisateur. Ces suggestions, peuvent être de petites étapes, des parties de la composition finale quand l'utilisateur est capable d'exprimer ses besoins , ou des compositions finales s'il n'est pas capable de les exprimer.

D'un point de vue expérimental, notre travail est toujours en cours. Nous utilisons pour notre modèle multi-agent, la théorie AMAS. Un premier moteur de composition a déjà été implémenté, on veut faire une version plus évoluée et nous sommes en train de travailler avec WCOMP [10] et la technologie UPnP. L'architecture du moteur de composition est à trois niveaux : le niveau d'agent service, le niveau d'agent composant, le niveau d'agent type. Pour chaque interface requise ou fournie il y a un agent service associé, les agents composants coordonnent les agents service agentifiant les interfaces requises et fournies du composant. Par exemple le driver de caméra aura quatre agents service, un pour chacun de ses angles, un pour son zoom et un pour son flux vidéo. Les agents type quant à eux gèrent les agents composants de leur type, et collectent les expériences de connexions de ceux-ci dans le but de suggérer de meilleures compositions. Le comportement de ces différents agents est en cours de spécification. Pour le plus bas niveau, à savoir les agents services, le cycle de vie et les situations de non coopération sont identifiées. Ces agents services seront ceux utilisant l'abstraction par type, présentés dans la section IV. Cependant, un alignement par type n'est pas suffisant dans des conditions réelles, par exemple un *slider* qui gère le zoom d'une caméra (Fig.3) doit être capable de connaître l'intervalle de l'angle et la valeur initiale, et l'encodage du *float* peut être différent du *slider* à la caméra. L'interopérabilité sémantique et syntaxique des interfaces est donc aussi un des axes de travail dans notre projet, en plus du problème de contrôlabilité et de celui de l'adaptation au contexte.

6. Remerciements

Ce travail est soutenu par le projet neoCampus, un projet transdisciplinaire qui rassemble dix laboratoires de l'Université Paul Sabatier (Toulouse, France), et voit les campus comme des petites villes connectées. Ce travail est issu d'une collaboration entre trois laboratoires IRIT à Toulouse, IIHM à Grenoble et I3S à Nice.

7. Références

[1] Brel, C., Gonin, P. R., Giboin, A., Riveill, M., & Dery, A. M. (2014, September). Reusing and Combining UI, Task and Software Component Models to Compose New Applications. In Proceedings of the 28th International BCS Human Computer Interaction Conference on HCI 2014-Sand, Sea and Sky-Holiday HCI(pp. 1-10). BCS.

[2] Briot J.-P. (2009) Composants logiciels et systèmes multi-agents. In A. El Fallah Seghrouchni and J.-P. Briot, eds, Technologies des systèmes multiagents et applications industrielles, Chapitre 5. Lavoisier, Paris, France, p. 147-187.

[3] Calvary, G., Dery-Pinna, A. M., Occello, A., Renevier, P., & Gabillon, Y. (2013). Composition of User Interfaces. Computer Science and Ambient Intelligence, 203-224.

[4] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., & Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces. Interacting with computers, 15(3), 289-308.

[5] Capera, D., Georgé, J. P., Gleizes, M. P., & Glize, P. (2003, June). The AMAS theory for complex problem solving based on self-organizing cooperative agents. In Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on (pp. 383-388). IEEE.

[6] Cockton, G. Value-centred HCI. In Proceedings of the third Nordic conference on Human-computer interaction, NordiCHI '04, pp 149–160, New York, NY, USA, 2004. ACM

[7] Cockton, G. Designing worth is worth designing. In Proceedings of the 4th Nordic confer-ence on Human-computer interaction: changing roles. NordiCHI '06, pp.165–174, New York, NY, USA, 2006. ACM

[8] Coutaz, J. (2006). Meta-user interfaces for ambient spaces. In Task Models and Diagrams for Users Interface Design (pp. 1-15). Springer Berlin Heidelberg.

[9] Ferry, N., Hourdin, V., Lavirotte, S., Rey, G., Tigli, J. Y., & Riveill, M. (2010). Models at runtime: service for device composition and adaptation.

[10] Ferry, N., Hourdin, V., Lavirotte, S., Rey, G., & Tigli, J. Y. (2013). Wcomp, middleware for ubiquitous computing and system focused adaptation.Computer Science and Ambient Intelligence, 89-120.

[11] Foley, J. D., & Wallace, V. L. (1974). The art of natural graphic man—Machine conversation. Proceedings of the IEEE, 62(4), 462-471.

[12] Gabillon, Y., Petit, M., Calvary, G., & Fiorino, H. (2011, February). Automated planning for user interface composition. In IUI 2011-International Conference on Intelligent User Interfaces (p. 5p).

[13] Georgé, J. P., & Gleizes, M. P. (2005). Experiments in emergent programming using self-organizing multi-agent systems. In Multi-Agent Systems and Applications IV (pp. 450-459). Springer Berlin Heidelberg.

[14] Joffroy, C., Caramel, B., Dery-Pinna, A. M., & Riveill, M. (2011, June). When the functional composition drives the user interfaces composition: process and formalization. In Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems (pp. 207-216). ACM.

[15] Lepreux, S., Vanderdonckt, J., & Michotte, B. (2006). Visual design of user interfaces by (de) composition. In Interactive Systems. Design, Specification, and Verification (pp. 157-170). Springer Berlin Heidelberg.

[16] Lewandowski, A., Lepreux, S., & Bourguin, G. (2007). Tasks models merging for high-level component composition. In Human-Computer Interaction. Interaction Design and Usability (pp. 1129-1138). Springer Berlin Heidelberg.

[17] McIlroy, M. D., Buxton, J. M., Naur, P., & Randell, B.(1968, October). Mass-produced software components. In Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany (pp. 88-98). sn.

[18] Szyperski, C., Bosch, J., & Weck, W. (1999, June). Component-oriented programming. In Object-oriented technology ecoop'99 workshop reader (pp. 184-192). Springer Berlin Heidelberg.

[19] Thevenin, D., & Coutaz, J. (1999, August). Plasticity of user interfaces: Framework and research agenda. In Proceedings of INTERACT (Vol. 99, pp. 110-117).

[20] Triboulot, C., Trouilhet, S., Arcangeli, J-P., & Robert, F(2015). Opportunistic software composition: benefits and requirements. In Int. Conf. on Software Engineering and Applications (ICSOFT-EA), INSTICC, p. 426-431.

[21] Tsai, W. T., Huang, Q., Elston, J., & Chen, Y. (2008, October). Service-oriented user interface modeling and composition. In e-Business Engineering, 2008. ICEBE'08. IEEE International Conference on (pp. 21-28). IEEE.

[22] Vale, T., Crnkovic, I., de Almeida, E. S., Neto, P. A. D. M. S., Cavalcanti, Y. C., & de Lemos Meira, S. R. (2016). Twenty-eight years of component-based software engineering. Journal of Systems and Software, 111, 128-148.

[23] Weiser, M. (1991). The computer for the 21st century. Scientific american,265(3), 94-104.

[24] Zhao, Q., Huang, G., Huang, J., Liu, X., & Mei, H. (2008, December). A web-based mashup environment for on-the-fly service composition. In Service-Oriented System Engineering, 2008. SOSE'08. IEEE International Symposium on (pp. 32-37). IEEE.