PRIAM : affichage dynamique d'informations par des écrans coopérants en environnement mobile

Christophe Jacquet
Supélec
&
LIMSI-CNRS
Christophe.Jacquet
@supelec.fr

Yacine Bellik
LIMSI-CNRS
BP 133
91403 Orsay Cedex
France
Yacine.Bellik@limsi.fr

Yolaine Bourda
Supélec
Plateau de Moulon
91192 Gif-sur-Yvette Cedex
France
Yolaine.Bourda@supelec.fr

Résumé
Cet article présente un scénario d'interaction dans lequel des écrans d'affichage sont utilisés pour fournir des informations pertinentes aux utilisateurs de lieux publics lors de leurs déplacements. Les écrans publics peuvent ainsi collaborer de façon dynamique de sorte à minimiser la surcharge et la redondance des informations. Nous analysons les contraintes d'utilisabilité de ce scénario en termes de positionnement des informations sur les écrans. De ces contraintes, nous déduisons un algorithme basé sur une architecture décentralisée dans laquelle les écrans et les utilisateurs sont représentés par des agents logiciels. Nous présentons alors les résultats obtenus par simulation de ce système, appelé PRIAM (PRésentation d'Informations dans l'AMbiant), ainsi que des perspectives pour les travaux à venir.

Mots-clefs
Informatique ubiquitaire, Mobilité

Abstract
This article introduces an interaction scenario in which display screens are used to provide relevant information to users of public places as they move. Public monitors can thus dynamically cooperate so as to minimize overload and information redundancy. We analyse the usability constraints of this scenario in terms of information positioning on the screens. From these constraints, we deduce an algorithm based around a decentralized architecture in which screens and users are represented by software agents. Finally, we present the first results obtained from a simulation of this system, as well as perspectives for future work. The underlying software platform is called PRIAM (PResentation of Information in AMbient intelligence).

Categories and Subject Descriptors
H.5.2 [**User interfaces**]: Interaction styles, Theory and methods

UbiMob'06, September 5-8, 2006, Paris, France.

General Terms
Algorithms, Design, Human factors

Keywords
Ubiquitous computing, Mobility

1. INTRODUCTION
Lorsqu'on se retrouve dans un environnement inconnu, comme une gare, un aéroport, un centre commercial, etc., on éprouve parfois des difficultés à obtenir les informations dont on a besoin. En effet, si de tels lieux sont souvent équipés d'écrans d'information, ces derniers présentent des informations non ciblées (à destination de tout le monde). En conséquence, ils sont le plus souvent surchargés d'informations, ce qui rend leur lecture difficile. Or, une personne donnée est généralement intéressée par une et une seule information : la rechercher parmi une multitude d'informations inintéressantes est alors une tâche longue voire fastidieuse.

Pour améliorer cette situation, et partant du principe qu'il ne sert à rien de présenter une information qui n'intéresse personne, nous avons conçu un système d'informatique ubiquitaire qui peut utiliser de multiples dispositifs d'affichage pour fournir des informations personnalisées à des utilisateurs mobiles. Par exemple, des écrans d'information disposés au hasard dans un aéroport pourraient fournir aux passagers situés à proximité des informations sur leurs vols. De cette façon, seules les informations à destination des passagers stationnés devant l'écran seraient affichées, de façon à réduire sa surcharge.

Si de nombreuses personnes se rassemblent devant un écran, celui-ci va rapidement subir une surcharge d'informations. En conséquence, les utilisateurs vont devoir rechercher leur information parmi une longue liste d'informations non pertinentes. On peut alors penser à amener un *deuxième* écran, à côté du premier, de façon à augmenter la surface d'affichage disponible. Cependant, si les deux écrans ne coopèrent pas, le second va se contenter de recopier le contenu du premier, ne diminuant ainsi en rien la surcharge du premier. En d'autres termes, le deuxième écran sera quasiment inutile. La solution de ce problème réside dans une judicieuse *répartition du contenu* entre les deux écrans (voir fig. 1).

Dans cet article, nous introduisons une architecture à agents dans laquelle les dispositifs d'affichage voisins peuvent co-

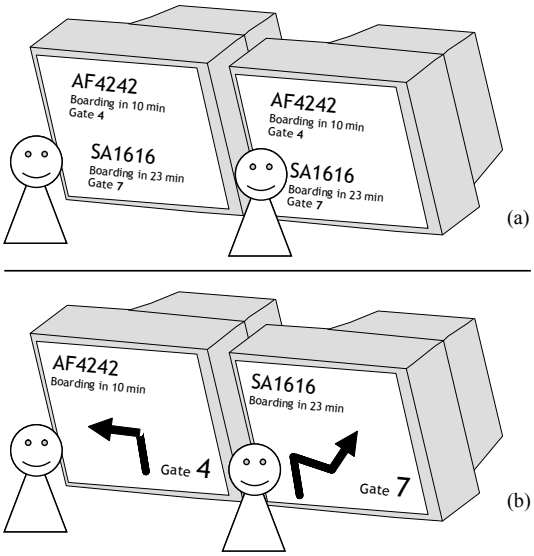


Figure 1: Deux écrans voisins dans un aéroport. Dans le cas (a), ils se contentent de dupliquer leur contenu. Dans le cas (b), ils coopèrent entre eux.

opérer pour réduire la surcharge globale, *sans qu'on leur ait communiqué explicitement à l'avance leurs positions relatives*. Ainsi, il n'est pas nécessaire de configurer manuellement le système. En particulier, il devient ainsi possible de déplacer les moniteurs alors même qu'ils sont en train de fonctionner, sans qu'il soit nécessaire de modifier la configuration logicielle. Ceux-ci adapteront dynamiquement leurs affichages en fonction des utilisateurs et des écrans alentour.

Dans une première partie, nous présentons des travaux connexes qui nous permettent d'introduire notre propre approche. Ensuite, nous formalisons le problème, et dressons une liste de contraintes d'utilisabilité auxquelles doit se conformer un système coopératif d'affichage d'informations. Nous en déduisons notre solution : un algorithme à agents distribué sur les dispositifs d'affichage et les utilisateurs. Cet algorithme n'a pas besoin de processus centralisés : les agents coopèrent entre eux *au niveau local* pour construire une solution. Finalement, nous décrivons l'implémentation de cet algorithme, sous forme d'un système nommé PRIAM, pour PRésentation d'Informations dans l'AMbiant. Cette implémentation comprend un simulateur qui nous a permis de tester l'algorithme. Nous concluons sur un ensemble de perspectives pour les travaux futurs.

2. TRAVAUX CONNEXES

Plusieurs systèmes ont déjà été proposés pour fournir des informations contextuelles à des utilisateurs mobiles lors de leurs déplacements. Par exemple, le système CoolTown [5] montre des pages web aux utilisateurs en fonction de leur position. Dans la plupart des cas, ces informations contextuelles sont fournies aux utilisateurs par l'intermédiaire de petits dispositifs portables : par exemple, le Cyberguide [6], un guide touristique pour les musées, était basé sur l'assistant personnel (PDA) d'Apple (Newton). Plus généralement, on se rend compte que la plupart des systèmes sensibles au contexte se basent sur des dispositifs portables qui sont censés fournir des informations aux utilisateurs sur leur environnement, par exemple, *"où se trouve la pizzeria la plus proche ?"* [4].

Cependant, d'autres applications d'informatique ubiquitaire (selon le terme de Mark Weiser [13]) n'exigent plus des utilisateurs qu'ils portent des PDA. Il est ainsi possible d'utiliser des systèmes publics d'affichage. C'est ce que fait, par exemple, le Gossip Wall [11]. Dans ce type de réalisations, les systèmes publics d'affichage jouent plusieurs rôles : fournir des informations d'intérêt général quand personne n'est à proximité immédiate, et fournir des informations plus personnelles lorsqu'un utilisateur engage une interaction explicite. Ce comportement n'est pas sans soulever des problèmes de respect de la vie privée [12].

Quant à notre système, bien qu'il utilise des systèmes publics d'affichage, il ne fournit pas d'informations *privées* aux utilisateurs, ce qui élimine, au moins partiellement, les problèmes de respect de la vie privée. En effet, il fournit des informations *publiques* (par exemple sur les vols d'avions), mais uniquement si elles sont pertinentes pour les personnes situées à proximité. Ce qui fait l'originalité de notre système est que les dispositifs d'affichage n'ont pas de connaissance *a priori* de leurs conditions de fonctionnement. Ils peuvent être placés à n'importe-quel endroit sans configuration préalable, et fournissent alors des informations pertinentes aux utilisateurs situés à proximité. De plus, ils collaborent entre eux de façon à améliorer le confort des utilisateurs (réduction des redondances et optimisation de l'espace).

Bien entendu, notre définition de la "proximité" est liée à la distance entre les utilisateurs et les dispositifs d'affichage, mais elle n'est pas forcément limitée à cette distance. Par exemple, si un utilisateur tourne le dos à un moniteur tout en ayant une conversation téléphonique, lui afficher de l'information sur le moniteur en question n'aurait aucun sens. Pour cette raison, la notion de "proximité" telle que nous l'entendons inclut également l'orientation des utilisateur : si un utilisateur ne *peut manifestement pas* voir un écran, on considérera qu'il n'est pas "proche" de cet écran.

Dans une réalisation pratique, la proximité sera mesurée par des dispositifs physiques. Nous verrons plus loin qu'il est possible de trouver des dispositifs qui tiennent compte de l'orientation correcte des utilisateurs.

3. POSITION DU PROBLÈME

Pour des raisons de simplicité, nous supposons que chaque utilisateur souhaite disposer d'une information particulière que nous appelons son *unité sémantique* (u.s.). Par exemple, l'u.s. d'un passager dans un aéroport pourrait être sa porte d'embarquement. Nous appelons *charge* d'un écran le nombre d'unités sémantique qu'il affiche. Pour le moment, l'u.s. d'un utilisateur est définie une fois pour toutes, mais cette restriction pourra être levée (voir section 6).

Nous considérons que lorsqu'un utilisateur est proche d'au moins un écran, il doit nécessairement pouvoir consulter son unité sémantique. C'est ce que nous appelons la *contrainte de complétude*. Nous souhaitons également que les informations soient distribuées de façon optimale entre les différents écrans. Pour ce faire, la charge d'un écran doit tendre à être

minimale, de façon à réduire la surcharge et limiter la confusion des utilisateurs. C'est la *contrainte d'optimisation* de l'espace de présentation. Si nous ne tenions compte que de ces deux contraintes, le problème se résumerait à la résolution d'un système de contraintes distribuées (complétude), tout en minimisant un paramètre de charge (optimisation de l'espace). Ainsi, le problème pourrait être considéré comme étant une instance de DCOP (Distributed Constraint Optimization Problem) ; plusieurs algorithmes existent pour résoudre un tel problème [7].

Cependant, les algorithmes de résolution de contraintes sont conçus pour trouver en une seule fois une solution à un problème, donné lui aussi en une seule fois. À l'opposé, notre problème est construit pas à pas à partir d'une situation initiale. En effet, nous pouvons supposer qu'au commencement aucun utilisateur n'est proche d'aucun écran. À partir de là, deux sortes d'événements peuvent survenir :

1. un utilisateur s'approche d'un écran dont il était éloigné ;
2. un utilisateur s'éloigne d'un écran dont il était proche.

De cette façon, toute situation peut être construite par une suite d'événements 1 et 2, débutant par un 1. Si nous supposons que nous disposons d'une solution convenable à un moment donné, et si nous sommes capables de construire une nouvelle solution après qu'un événement 1 ou 2 est survenu, alors nous sommes capables de résoudre le problème à tout moment (principe de récursion). Ainsi, un algorithme incrémental sera très efficace dans cette situation. Nous verrons que la solution que nous introduisons à la section 4 est incrémentale.

Optimiser l'espace occupé à l'écran est certes un but important, mais le respecter aveuglément peut conduire à la réalisation de systèmes inutilisables. En effet, si on décide d'éviter absolument toute surcharge des écrans, on finit par réorganiser constamment l'affichage des informations. Ceci a une conséquence néfaste : les utilisateurs risquent de voir leurs u.s. "sauter" d'un écran à l'autre à chaque fois qu'un autre utilisateur s'approche ou s'éloigne d'un écran. Ils passeraient alors leur temps à rechercher l'information qu'ils sont en train de lire, perdraient le fil, et finiraient par ne plus s'y retrouver du tout. En résumé, ce serait bien plus fastidieux et perturbant d'utiliser un tel système que de devoir rechercher une u.s. donnée sur un écran surchargé...

Au contraire donc, l'affichage des informations devrait rester *stable* lorsque des événements se produisent, pour ne pas perturber les utilisateurs. En effet, si un utilisateur donné ne se déplace pas, il est fort probable qu'il soit en train de lire son u.s., et donc il s'attend à ce que celle-ci reste en place, et non qu'elle disparaisse brusquement pour réapparaître ailleurs. Inversement, lorsqu'un utilisateur se déplace, il est peu probable qu'il soit en train de lire des informations en même temps, et donc on peut déplacer son u.s. sans que cela le dérange.

Ces considérations nous conduisent à prendre en compte les trois contraintes suivantes :

- **Contrainte C_1 (complétude).** Lorsqu'un utilisateur est proche d'un certain nombre d'écrans, son u.s. doit lui être fournie par l'un (au moins) de ces dispositifs ;
- **Contrainte C_2 (stabilité).** Lorsqu'un utilisateur ne se déplace pas, son u.s. doit rester fixe. En particulier, elle ne doit pas passer d'un écran à un autre ;
- **Contrainte C_3 (optimisation de l'espace d'affichage).** Pour éviter que les moniteurs ne soient surchargés, on évite au maximum la duplication des informations. Cela signifie que la somme des charges des écrans utilisés doit être minimale.

Pour des raisons d'utilisabilité, nous considérons que la contrainte C_1 est plus forte que la contrainte C_2, qui à son tour est plus forte que la contrainte C_3. En conséquence, l'espace d'affichage utilisé ne sera généralement pas complètement optimisé, de façon à préserver la stabilité de l'affichage. De même, lorsqu'un écran commencera à être saturé, certaines de ses u.s. seront déplacées sur un autre écran, afin qu'il puisse afficher une nouvelle u.s. à destination d'un nouvel utilisateur. Ainsi, on s'autorise à transgresser la contrainte de stabilité de façon à respecter la contrainte de complétude.

4. SOLUTION

Dans cette section, nous introduisons un algorithme capable de résoudre le problème d'affichage des informations, tout en respectant les trois contraintes énoncées ci-dessus. Cet algorithme est distribué sur les écrans et les utilisateurs : chacune de ces entités est représentée par un agent logiciel.

4.1 Formalisation mathématique

Lorsqu'il est question d'afficher une u.s. sur un écran, il faut évaluer le *coût* de cette opération. On introduit donc trois coûts, le *coût statique* de l'affichage d'un ensemble d'unités sémantiques sur un écran, le *coût dynamique* de l'ajout d'une u.s. sur un écran, et le *coût de migration* engendré par le déplacement d'une u.s. d'un écran à un autre.

Soit $\tilde{c}$ une fonction sur $\mathbb{R}^+$, strictement convexe et strictement croissante (nous verrons plus loin les raisons de ces restrictions). $\tilde{c}(\ell)$ représente le *coût statique* de l'affichage d'un écran de charge ℓ. Ainsi, pour un écran donné s dont la charge est ℓ_s, on définit $\mathbf{c}(s)$ comme étant $\tilde{c}(\ell_s)$. $\mathbf{c}(s)$ est appelé le *coût statique* de l'écran s avec son affichage courant.

Supposons qu'on veuille ajouter δ u.s. ($\delta \neq 0$) à un écran donné s (ou bien retirer $-\delta$ u.s. à l'écran s si $\delta < 0$). Le *coût dynamique* de l'opération, noté $\mathbf{d}(s, \delta)$, est défini comme étant :

$$\begin{aligned} \mathbf{d}(s,\delta) &= \mathbf{c}(s)_{\text{apres}} - \mathbf{c}(s)_{\text{avant}} \\ \mathbf{d}(s,\delta) &= \tilde{c}(\ell_s + \delta) - \tilde{c}(\ell_s) \end{aligned}$$

Nous avons signalé plus haut que $\tilde{c}$ est strictement croissante et strictement convexe. En conséquence, δ étant donné, si

$\ell_2 > \ell_1$ alors $\tilde{c}(\ell_2 + \delta) - \tilde{c}(\ell_2) > \tilde{c}(\ell_1 + \delta) - \tilde{c}(\ell_1)$. Ainsi, coût dynamique $\mathbf{d}$ est conçu de façon à croître lorsque ℓ_s augmente. Avec cette définition du *coût dynamique*, nous formalisons l'idée que *plus d'u.s. sont affichés sur un écran, plus il est coûteux d'afficher une u.s. supplémentaire.* En effet, s'il se trouve une seule u.s. sur un écran (ou même si l'écran est vide), ajouter une u.s. n'augmente pas le temps de recherche nécessaire à un utilisateur pour trouver son u.s. sur l'affichage. Mais si un écran est déjà surchargé, trouver une nouvelle u.s. parmi une multitude d'u.s. non pertinentes sera très long, difficile, et éventuellement stressant.

Par exemple, supposons que sur un système donné, $\tilde{c}$ soit défini par $\tilde{c}(x) = x^2$ pour deux écrans appelés a et b. Ce choix pour $\tilde{c}$ est complètement arbitraire : en pratique, $\tilde{c}$ doit être choisi pour chaque moniteur, de façon à refléter la difficulté d'y retrouver des informations au fur et à mesure que sa charge augmente. Si à un moment donné, l'écran a affiche deux u.s., alors $\mathbf{c}(a) = 2^2 = 4$; si l'écran b affiche au même moment quatre u.s., alors $\mathbf{c}(b) = 4^2 = 16$. Si l'on veut ajouter une u.s. à l'un de ces écrans, quels sont les coûts dynamiques associés à chacun d'entre eux ? Par le calcul, on trouve $\mathbf{d}(a, 1) = (2+1)^2 - 2^2 = 9 - 4 = 5$; de même, $\mathbf{d}(b, 1) = (4+1)^2 - 4^2 = 25 - 16 = 9$. Ainsi, si on a le choix, on préférera que la nouvelle u.s. soit affichée sur l'écran a, car le coût dynamique est moindre. Ceci correspond à l'intuition que l'on peut avoir de la situation. Notons bien qu'ici, les deux écrans partagent la même fonction de coût statique. On peut imaginer des situations où ce ne serait pas le cas : chaque écran peut définir sa propre fonction de coût statique, correspondant à ses caractéristiques propres.

Dans l'exemple, nous avons arbitrairement utilisé une fonction de coût $\tilde{c}(x) = x^2$. En pratique, cette fonction de coût devra être choisie de façon a refléter la réalité des comportements des utilisateurs. Nous prévoyons de mener des expériences grandeur nature qui nous permettront de déterminer quelles sont les fonctions de coûts réalistes.

Nous introduisons également des *coûts de migration*. Un coût de migration est comptabilisé lorsqu'une u.s. u est déplacée d'un écran à un autre. Chaque utilisateur U_i intéressé par l'u.s. u subit une gêne, formalisée par un coût de migration partiel $m(U_i, u)$. Le coût de migration total est la somme de tous les coûts de migration partiels fournis par chaque utilisateur :

$$m(u) = \sum_i m(U_i, u)$$

4.2 Architecture à agents

Nous pouvons envisager de construire une solution basée sur une architecture *centralisée*. Cependant, nous pensons qu'une telle méthode présente un certain nombre d'inconvénients, principalement :

- sa *fragilité* : si les serveurs centraux tombent en panne, tous les écrans cessent de fonctionner ;
- et sa *rigidité* : il est impossible de déplacer les écrans à volonté, du moins sans travail de reconfiguration.

À l'inverse, nous souhaitons donner au personnel des lieux publics la possibilité de déplacer les écrans, d'en apporter de nouveaux si un événement particulier survient, d'en enlever, etc., tout ceci sans être obligé de configurer quoi que ce soit. Les écrans doivent être capables de s'adapter d'eux-mêmes aux changements, sans qu'une intervention humaine soit nécessaire.

Notre implémentation est basée sur des agents logiciels. Les écrans sont représentés par des *agents d'affichage* ; les utilisateurs sont représentés par des *agents utilisateurs*. Nous supposons que chaque agent sait quels autres agents se trouvent à proximité, et peut communiquer avec eux. Ces suppositions nous semblent réalistes : la proximité peut être détectée par des lecteurs d'étiquettes RFID[1] [8], ou encore des systèmes Bluetooth ou infrarouges (comme dans le système Parctab [10, 9]). Les communications peuvent passer par des réseaux sans fils de type WiFi, maintenant monnaie courante, ou encore des réseaux de téléphonie mobile. Nos agents sont *réactifs* : ils accomplissent des actions en réponse à des événements. Un agent donné a peut réagir à trois sortes d'événements :

1. un autre agent b vient de s'approcher a ;
2. un agent b, auparavant proche de a, vient de s'éloigner ;
3. a vient de recevoir un message par le réseau, en provenance d'un agent d quelconque, qui n'est pas nécessairement proche de a.

Le concept général de notre système est donc le suivant : lorsqu'un agent utilisateur détecte la proximité d'un agent d'affichage, il lui demande d'afficher son u.s. De cette façon, si par exemple un passager s'approche d'un écran dans un aéroport, il pourra y lire des informations utiles, par exemple le numéro de sa porte d'embarquement. Un écran n'affiche que les informations à destination des passagers situés à proximité, de façon à limiter son encombrement visuel.

Notons que si un seul passager se trouve à proximité d'un écran, cela peut soulever un problème de respect de sa privée, car quiconque passe à proximité peut connaître sa destination. Dans ce cas particulier, on peut alors envisager de faire figurer une ou deux u.s. supplémentaires, aléatoires, de façon à tromper les personnes malveillantes.

4.3 Algorithme

Nous pouvons maintenant passer à la description plus précise du comportement de l'algorithme. Tout d'abord, on spécifie les agents utilisateurs de telle sorte qu'ils puissent s'attribuer chacun un *écran principal*, c'est-à-dire l'écran où l'u.s. de l'utilisateur est affichée à coup sûr. Au démarrage tous les écrans principaux sont non définis.

Au niveau global, l'algorithme fonctionne ainsi : lorsqu'un agent utilisateur s'approche (i) ou s'éloigne (ii) d'un écran, il envoie des *requêtes d'évaluation* aux agents d'affichage situés à proximité, afin de connaître le *coût* certaines opérations (affichage d'u.s., migration d'u.s.), que nous décrivons plus

[1]Identification par radiofréquences.

loin. Les agents d'affichage répondent à ces requêtes (*iii*, *iv*) et mémorisent les opérations soumises à évaluation. L'agent utilisateur a alors le choix entre *confirmer* ou *annuler* chacune de ces opérations évaluées.

Dans les figures ci-dessous, les agents sont représentés par des ovales. Le lien entre un agent utilisateur et l'agent qui correspond à son écran principal est représenté par une ligne pointillée. Les messages entre agents sont représentées par des flèches, généralement étiquetées. Les flèches en pointillés représentent les réponses aux requêtes.

[*i*] Lorsqu'un agent utilisateur doté d'une u.s. u s'approche d'un écran s :

- *s'il possède déjà un écran principal*, il envoie une requête `évaluation-migration`(s) à son écran principal (voir fig. 2). Si le résultat (coût dynamique) de la requête est négatif, alors l'agent la confirme. Sinon, il l'annule ;
- *sinon*, il envoie une requête `évaluation-affichage`(u) à s, et la confirme systématiquement, de façon à satisfaire la contrainte C_1 (complétude), voir fig. 3.

[*ii*] Lorsqu'un agent utilisateur (dont l'u.s. est u) s'éloigne de son *écran principal*, il commence par envoyer une notification `éloignement` à son écran principal, puis :

- *si d'autres écrans sont situés à proximité*, il envoie une requête `évaluation-affichage`(u) à chacun d'entre eux, et choisit celui pour lequel le coût dynamique est le plus faible (voir fig. 4). Il en fait son écran principal (contrainte C_3, optimisation de l'espace d'affichage). Il envoie alors un message de confirmation à cet écran, et un message d'annulation à tous les autres ;
- *sinon*, son écran principal est réputé non défini (et par conséquent son u.s. n'est plus affichée nulle part).

[*iii*] Lorsqu'un agent d'affichage reçoit une requête `évaluation-affichage`(u) :

- *s'il y a encore de la place pour l'u.s.* u, il l'ajoute à la liste des u.s. à afficher : lorsque la contrainte C_1 (complétude) est satisfiable, l'écran essaie de satisfaire la contrainte C_2 (stabilité) ;
- *sinon*, il essaie de déplacer l'une des u.s.. qu'il affiche sur un autre écran. Plus précisément, pour chaque u.s.. affichée v, il envoie récursivement une requête `évaluation-affichage`(v) à chaque écran vu par tous les agents a_i intéressés par v (voir fig. 5). Le coût d'une migration possible (si l'appel récursif correspondant n'échoue pas) est alors défini comme étant le coût renvoyé (noté d), plus le coût de migration associé, c'est-à-dire :

$$d + \sum_i m(a_i, v)$$

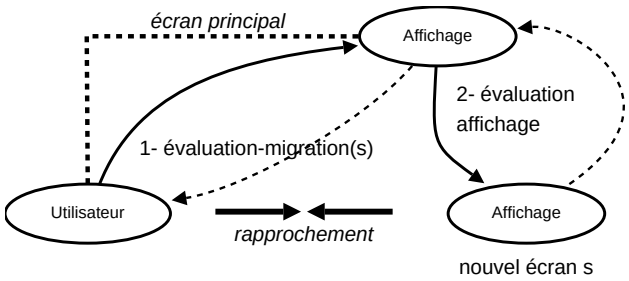


Figure 2: Lorsqu'un utilisateur s'approche d'un écran, il essaie d'y faire migrer son u.s.

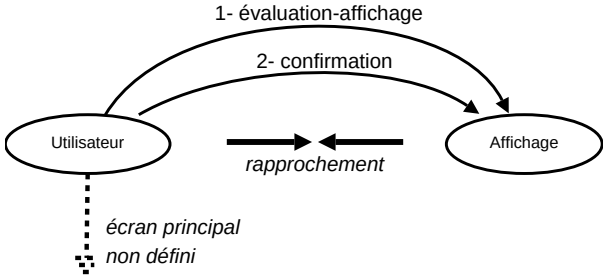


Figure 3: Lorsqu'un utilisateur qui n'a pas d'écran principal s'approche d'un écran, il lui demande systématiquement d'afficher son u.s.

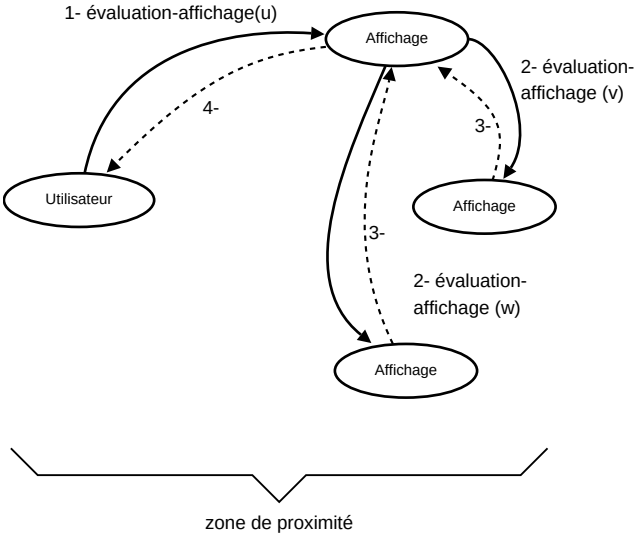


Figure 5: Lorsqu'un écran est surchargé, il essaie de faire migrer certaines des u.s. qu'il affiche sur des écrans visibles de tous les utilisateurs intéressés.

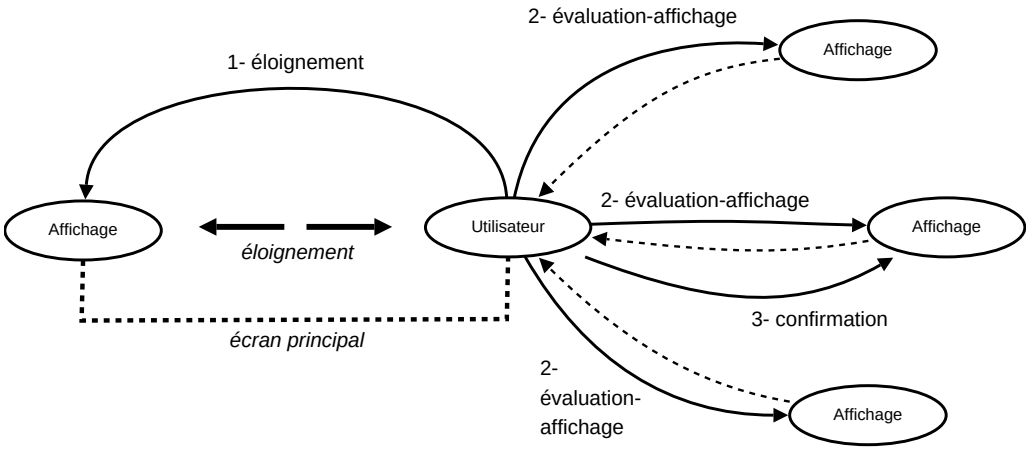


Figure 4: Lorsqu'un utilisateur s'éloigne de son écran principal, il essaie de faire afficher son u.s. par un autre écran situé à proximité.

Si au moins l'un de ces appels récursifs n'échoue pas, l'agent choisi le moins coûteux, le confirme, et annule les autres. Sinon, il déclare à son tour échouer. Si la contrainte C_2 (stabilité) n'est pas satisfiable, l'écran essaie toujours de respecter la contrainte C_1 (complétude), mais ce faisant, il continue à optimiser l'espace d'affichage (contrainte C_3). La règle C_1 n'est enfreinte qu'en dernier lieu, c'est-à-dire si tous les écrans à proximité n'ont plus d'espace disponible.

[*iv*] Lorsqu'un agent d'affichage reçoit une requête `évaluation-migration`(s) concernant une u.s. u :

- si plus d'un agent utilisateur intéressé par u a défini cet écran comme écran principal, la requête échoue (complétude et stabilité) ;
- sinon, l'agent d'affichage envoie à s une requête `évaluation-affichage`(u), et note le coût associé d_1 (voir fig. 2). Il évalue le "coût" de la suppression de u de sa surface d'affichage. Ce coût, négatif, est appelé d_2. Il calcule alors le coût de migration associé, et le note m. Alors, il renvoie $d_1 + d_2 + m$. On considère que la migration est souhaitable si cette quantité est négative.

Nous n'avons décrit ici que le comportement de base de l'algorithme. Les autres opérations, comme les confirmations et les annulations, sont définies de façon simple.

5. IMPLÉMENTATION ET PREMIERS RÉSULTATS

Nous appelons PRIAM l'implémentation en Java de cette architecture à agents. Les agents sont alors des objets distribués sur un réseau, qui communiquent entre eux via le mécanisme des RMI[2], intégré à Java. Ainsi, le réseau est considéré comme transparent.

[2]Remote Method Invocation.

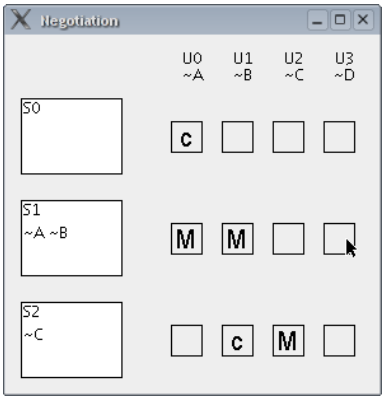
Des agents *utilisateurs* et *afficheurs* ont été implémentés. Ils mettent en œuvre l'algorithme décrit ci-dessus. Nous avons réalisé un simulateur qui nous permet de tester le système sur machine, afin de vérifier la validité de l'algorithme.

La figure 6 montre deux copies d'écran successives du simulateur. La simulation représentée met en jeu quatre utilisateurs (appelés `U0`, `U1`, `U2` et `U3`) et trois écrans (appelés `S0`, `S1` et `S2`). Les u.s. des quatre utilisateurs sont respectivement `A`, `B`, `C` et `D`. Dans cette simulation, on utilise des écrans spécifiés de telle sorte qu'ils ne peuvent afficher au plus que deux unités sémantiques.

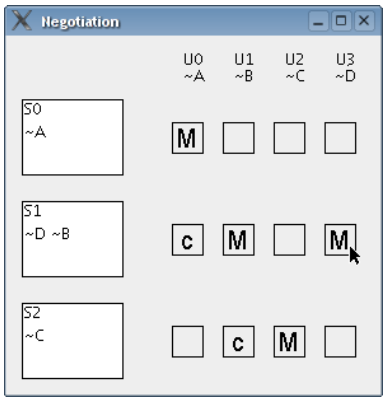
La matrice des 3×4 carrés au centre de la fenêtre du simulateur représente les relations de proximité : si un utilisateur n'est pas proche d'un écran, le carré est vide. Si un écran donné est l'écran *principal* d'un utilisateur, un "M" est dessiné dans le carré. Si un écran est proche d'un utilisateur, mais n'est pas pour autant son écran principal, un "c" est dessiné dans le carré. Les écrans sont représentés sur la gauche : ils indiquent quelles sont les u.s. affichées.

Dans un premier temps, considérons la situation initiale représentée sur la figure 6a. `U0` et `U1` sont proches de l'écran `S1`, et `S1` est leur écran *principal*. Ainsi, `S1` affiche les u.s. `A` et `B`. `U0` est également proche de l'écran `S0`, `U1` est également proche de l'écran `S2`, mais ce ne sont pas leurs écrans principaux. L'écran principal de l'utilisateur `U2` est `S2`. Pour le moment, `U3` n'a pas d'écran principal.

À un moment donné, l'utilisateur `U3` s'approche de l'écran `S1`. Afin de déclencher cette action dans le simulateur, l'expérimentateur clique sur la case correspondante de la "matrice" (i.e. là où se trouve le pointeur de souris sur la figure 6a). Pour satisfaire la contrainte C_1 (complétude), l'écran `S1` devrait afficher l'u.s. `D` en plus de `A` et `B`, mais ceci est impossible, car il ne peut afficher au maximum que deux u.s. En conséquence, `S1` choisit de transgresser la contrainte C_2 (stabilité) en faveur de la contrainte C_1, et de faire migrer l'une des deux u.s. `A` ou `B` vers respectivement l'écran `S0` ou l'écran `S2`. Afin de décider quelle migration effectuer, l'écran `S1` envoie donc des demandes d'évaluation des migrations à



(a)



(b)

Figure 6: Le simulateur propose une interface graphique qui permet de manipuler les relations de proximité.

`S0` et `S2`, qui lui renvoient les coûts correspondants :

1. si on fait migrer l'u.s. `A` sur `S0`, la charge de ce dernier passera de 0 à 1. Le coût, défini comme dans l'exemple de la section 4, vaut $1^2 - 0^2 = 1$;
2. si on fait migrer l'u.s. `B` sur `S2`, la charge de ce dernier passera de 1 à 2. Le coût vaut $2^2 - 1^2 = 3$.

`S1` choisit donc de faire migrer l'u.s. `A` sur l'écran `S0`. C'est bien ce résultat qui est observable sur la figure 6b. Parmi les deux migrations possibles, c'est bien *intuitivement* celle qui provoque le moins de surcharge des écrans.

Les test effectués grâce à cette implémentation nous ont permis de vérifier le comportement de l'algorithme dans diverses situations. Nous avons ainsi pu vérifier que le cahier des charges est bien suivi : les trois contraintes sont respectées, avec les priorités voulues. L'utilisabilité pratique du système sera évalué lors de la phase suivante des essais : des expérimentations en grandeur réelle.

En taille réelle, le fonctionnement du système sera analogue (des agents fonctionneront sur un réseau), à ceci près que la détection de proximité ne sera plus simulée, mais mesurée à l'aide de capteurs physiques.

Notons que si les agents modélisent des entités positionnées relativement dans l'espace (utilisateurs, écrans), ces agents ne seront eux-mêmes pas forcément disséminés dans l'espace. Ainsi, dans le cas de l'aéroport, il est tout à fait concevable que tous les agents utilisateurs et tous les agents d'affichage fonctionnent sur un ou plusieurs serveurs centraux. Cependant, il est également concevable que chaque agent d'affichage fonctionne sur l'écran qui lui correspond, et que de même, les agents utilisateurs fonctionnent sur les PDA ou les téléphones des utilisateurs. L'architecture de PRIAM ne définit pas de restriction ou de règle à ce sujet, mais laisse le libre choix aux concepteurs des applications.

6. PERSPECTIVES

6.1 Multimodalité

Dans cet article, tous les dispositifs de présentation d'information étaient des écrans : nous avons donc privilégié la modalité visuelle. Or, les informations dispensées par un tel système seraient particulièrement utiles pour les non-voyants, qui ont souvent du mal à trouver leur chemin dans des lieux inconnus : il faudrait donc adapter le système au handicap de ces utilisateurs potentiels.

C'est pourquoi la version du système PRIAM que nous finalisons actuellement sera capable de prendre en charge des dispositifs *multimodaux* en sortie, et non plus uniquement visuels. On ne parle plus alors de d'agents d'*affichage*, mais d'agents de *présentation* au sens large. Nous voulons ainsi être capables de fournir les informations selon des modalités différentes (texte visuel, synthèse vocale, texte Braille, images, etc.). Il ne s'agit cependant pas d'utiliser plusieurs modalités *à la fois*, mais plutôt de choisir quelle est la modalité la plus adaptée à un utilisateur donné : on parle de *multimodalité exclusive* [2].

Dans ce cas, les utilisateurs définissent des préférences, non seulement sur les u.s. qui les intéressent, mais aussi sur les modalités qu'ils acceptent en entrée. Par exemple, les utilisateurs aveugles refuseront les modalités visuelles, mais accepteront les modalités sonores ou texte en Braille. Une borne d'information devra alors forcément utiliser une modalité non visuelle, par exemple une synthèse vocale.

De plus, pour une modalité donnée, les utilisateurs peuvent souhaiter exprimer des préférences quant aux *attributs* de cette modalité. Par exemple, un utilisateur myope pourra indiquer une taille minimale pour le texte à afficher ; de même, les personnes qui rencontrent des problèmes d'audition pourront indiquer un niveau sonore minimum pour la synthèse vocale. Ces attributs seront donc utilisé lors de l'*instanciation* [1, 3] des unités sémantiques. Cette extension a des répercussions sur le calcul des coûts : par exemple, l'espace occupé à l'écran dépend de la taille des caractères lors d'une instanciation textuelle d'une u.s.

Nous envisageons de résoudre le problème du choix d'une modalité et d'attributs adaptés par un mécanisme de *profils* : les capacités d'interaction des utilisateurs, aussi bien que les dispositifs de présentation, seraient décrits dans des *profils*, un peu comme ce qui se fait dans le cadre de CC/PP[3].

[3]Composite Capabilities/Preferences Profile, une norme du W3C. Voir `http://www.w3.org/Mobile/CCPP/`.

6.2 Unités sémantiques

Dans ce qui précède, une u.s. était affectée *statiquement* à chaque utilisateur. Nous n'avons pas évoqué la façon dont les utilisateurs pouvaient obtenir ces unités sémantiques. Dans l'exemple de l'aéroport, la relation entre un passager et son u.s. est définie de façon triviale si l'u.s. en question porte les informations concernant son vol. Cependant, des cas moins simples sont plus problématiques.

C'est pourquoi nous pensons introduire des *agents informateurs*, chargés de fournir aux utilisateurs des informations utiles, éventuellement lorsqu'ils s'approchent d'un endroit précis, ou selon des critères d'*abonnement* et de *préférences* définis par les utilisateurs.

Nous avons également considéré qu'une seule unité sémantique était affectée à chaque utilisateur, ce qui est restrictif. Par exemple, dans un aéroport, un passager peut être intéressé à la fois par la localisation de son comptoir d'embarquement et par celle des toilettes les plus proches. Nous prévoyons donc de pouvoir affecter un nombre quelconque d'unités sémantiques à chaque utilisateur. Cet ajout complexifie les procédures, mais ne change pas la structure générale de l'algorithme tel qu'il a été décrit plus haut.

6.3 Notion de proximité

Dans cet article, la notion de proximité est *binaire* : deux agents quelconques sont soit proches, soit éloignés. En réalité, il pourrait être intéressant de définir plusieurs degrés de proximité, ou même une mesure de distance. Ces degrés ou distances pourraient alors être utilisés comme paramètres du processus d'instanciation des modalités. Par exemple, il peut être souhaitable que le texte affiché sur un écran soit plus gros lorsque les utilisateurs sont plus éloignés.

7. CONCLUSION

Dans cet article, nous avons présenté un scénario novateur d'interaction en mobilité : alors que l'on fournit des informations personnalisées aux utilisateurs lors de leurs déplacements, les écrans coopèrent dynamiquement de façon à réduire la surcharge d'information et donc améliorer l'utilisabilité du système. Nous avons analysé les diverses contraintes de ce scénario, ce qui nous a conduits à proposer une solution basée sur une architecture multi-agents décentralisée.

En simulation, cette architecture se montre efficace et adaptée. La prochaine étape consistera en des études pratiques *in situ*, basées sur une implémentation grandeur nature.

Comme nous l'avons expliqué, le système PRIAM peut être utilisé pour fournir des informations dans un aéroport ou une gare. En réalité, il doit être possible de l'adapter à toutes sortes de situation dans lesquelles on doit fournir des informations à des utilisateurs mobiles. Par exemple, lorsqu'on veut communiquer à des étudiants leurs résultats à un examen, on installe généralement des panneaux sur lesquels on affiche de longues listes de notes. Pour un étudiant, trouver son nom est une tâche fastidieuse et stressante. En utilisant notre système, on pourrait concevoir des panneaux qui n'afficheraient les résultats des étudiants situés à proximité : nul doute que la situation devant les panneaux d'affichage deviendrait alors moins chaotique.

8. BIBLIOGRAPHIE

[1] E. André. The generation of multimedia presentations. In R. Dale, H. Moisl, and H. Somers, editors, *A Handbook of Natural Language Processing*, pages 305–327. Marcel Dekker, 2000.

[2] D. Archambault and D. Burger. From multimodality to multimodalities: the need for independent models. In C. Stephanidis, editor, *Proceedings of HCI International '2001*, volume 3, pages 227–231. Lawrence Erlbaum, August 2001.

[3] M. K. H. Bunt, M. Maybury, and W. Wahlster. *Fusion and Coordination for Multimodal Interactive Information Presentation*, volume 27 of *Text, Speech and Language Technologies*, pages 325–340. Kluwer Academic Publishers, Dordrecht, Feb 2005.

[4] R. Hull, P. Neaves, and J. Bedford-Roberts. Towards situated computing. In *ISWC '97*, page 146, Washington, DC, USA, 1997. IEEE Comp. Soc.

[5] T. Kindberg and J. Barton. A Web-based nomadic computing system. *Computer Networks (Amsterdam, Netherlands: 1999)*, 35(4):443–456, 2001.

[6] S. Long, R. Kooper, G. D. Abowd, and C. G. Atkeson. Rapid prototyping of mobile context-aware applications: The cyberguide case study. In *Mobile Computing and Networking*, pages 97–107, 1996.

[7] R. Mailler and V. Lesser. Solving Distributed Constraint Optimization Problems Using Cooperative Mediation. In *Proceedings of AAMAS 2004*, pages 438–445. IEEE Computer Society, 2004.

[8] K. Römer, T. Schoch, F. Mattern, and T. Dübendorfer. Smart identification frameworks for ubiquitous computing applications. *Wirel. Netw.*, 10(6):689–700, 2004.

[9] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz, CA, US, 1994.

[10] B. N. Schilit, N. Adams, R. Gold, M. Tso, and R. Want. The PARCTAB mobile computing system. In *Proceedings Fourth Workshop on Workstation Operating Systems (WWOS-IV)*, pages 34–39. IEEE, October 1993.

[11] N. A. Streitz, C. Röcker, T. Prante, R. Stenzel, and D. van Alphen. Situated interaction with ambient information: Facilitating awareness and communication in ubiquitous work environments. In *HCI International 2003*, June 2003.

[12] D. Vogel and R. Balakrishnan. Interactive public ambient displays: transitioning from implicit to explicit, public to personal, interaction with multiple users. In *UIST '04*, pages 137–146, New York, NY, USA, 2004. ACM Press.

[13] M. Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–84, 1993.