Conservatoire National des Arts et Métiers

Centre de Lyon – Rhône-Alpes

PROBATOIRE

Spécialité: Informatique

Présenté par Pascal Joyeux

en juin 2006

IHM pour des produits industriels :

conception automatisée, principes, avantages, inconvénients

Professeur responsable: Jacques Kouloumdjian
Contact: Bertrand David

Je tiens à remercier Madame Joëlle Coutaz et Messieurs Bertrand David, Guillaume Masserey, Kinaan Samaan, Julien Micoud et Olivier Champalle pour l'aide qu'ils m'ont apportée.

Sommaire

Introduction

Il y a encore quelques décennies les ordinateurs étaient cantonnés à des rôles de supercalculateur ou de centralisation des données. Ils étaient puissants et très volumineux. Il n'était en effet pas rare de les voir remplir une pièce entière. Ils n'étaient utilisés que par des professionnels avertis dans des buts spécifiques. Le PC a sorti l'ordinateur de son isolement pour le faire entrer dans nos foyers et commencer son émancipation. De nos jours une nouvelle étape est franchie, l'ordinateur se miniaturise et envahit nos équipements, notre quotidien. Notre voiture, notre lave-vaisselle, notre chaîne hifi, notre téléphone, les bornes de plans dans les villes, les panneaux publicitaires sont autant d'appareils devenus intelligents, capables de communiquer et d'interagir avec leur environnement. Cette interaction utilise les moyens de communication existants comme le Web, les boucles Wan, les boucles Lan et les différents réseaux propriétaires et domestiques. Tous ces moyens de communication présentent un inconvénient majeur : ils sont câblés, donc coûteux et difficiles à mettre en œuvre. L'avènement des technologies sans fils, satellites, UMTS[1], WiFi[2], Bluetooth permet de démocratiser l'utilisation d'équipements intelligents. C'est surtout l'arrivée sur le marché de nombreux équipements personnels nomades, systèmes informatiques petits, mobiles et performants comme les téléphones portables, PDA[3], tablet PC et autres pocket PC qui font prendre à l'ordinateur un nouveau virage. Dans ce monde où tout est informatisé, l'usager se retrouve face à une énorme source d'information dont il veut pouvoir exploiter toutes les ressources. Pour cela, l'information devra pouvoir s'adapter au type de support utilisé par l'usager, mais aussi le suivre dans ses déplacements pour assurer la continuité de l'information et anticiper ses besoins en information afin de lui proposer la bonne information au bon moment.

Pour expliquer cela, je décrirai donc dans un premier temps les différents types d'interfaces capables de répondre à ces besoins, plusieurs méthodes d'adaptation aux supports sur lesquels elles seront utilisées dans le cadre d'une conception automatisée en dégageant les qualités et défauts de chacune avant d'expliquer comment elles seront mises en œuvre, Comment cette interaction se met-elle en œuvre ? Comment s'adapte l'interface en fonction du contexte dans lequel se trouve l'utilisateur, de ses centres d'intérêts, de ses capacités ?

Différents types d'interfaces

Trois grands types d'interfaces pour les équipements mobiles ou "nomades" se dégagent de l'ensemble de ce qui existe : les interfaces préprogrammées ou préconçues, les interfaces à base de modèles et les interfaces générées automatiquement.

Les interfaces préprogrammées, préconçues

Certes quelque peu en marge du sujet, il me semble néanmoins important de mentionner ce type d'interface ne fusse que pour mettre en évidence les contrastes existants avec les autres types d'interfaces et les problèmes que peuvent poser leur utilisation.

Les interfaces préprogrammées

Souvent définies par le constructeur de l'équipement qu'elles vont contribuer à piloter, les interfaces préprogrammées sont rigides et peu ou pas adaptables aux besoins réels ou aux habitudes de

[1] **UMTS : U**niversal **M**obile **T**elecommunications **S**ystem.
[2] **WiFi : Wi**reless **Fi**delity (réseau local Ethernet sans fil).
[3] **PDA : P**ersonal **D**igital **A**ssistant (assistant personnel).

l'utilisateur ; je dis peu adaptables, car l'initié pourra, en utilisant un logiciel spécialement conçu, modifier cette interface selon ses besoins et la transférer ensuite dans l'appareil qu'il utilisera pour piloter ses équipements. C'est le cas, par exemple, de la télécommande "universelle" développée par Philips pour piloter les équipements de sa gamme [I1] le "Pronto". La figure 1 montre bien que l'interface est générique, multi-équipements, rigide et peu conviviale.

La conception des interfaces préprogrammées ne permet évidemment pas d'envisager leur utilisation dans le cadre d'une conception automatisée.

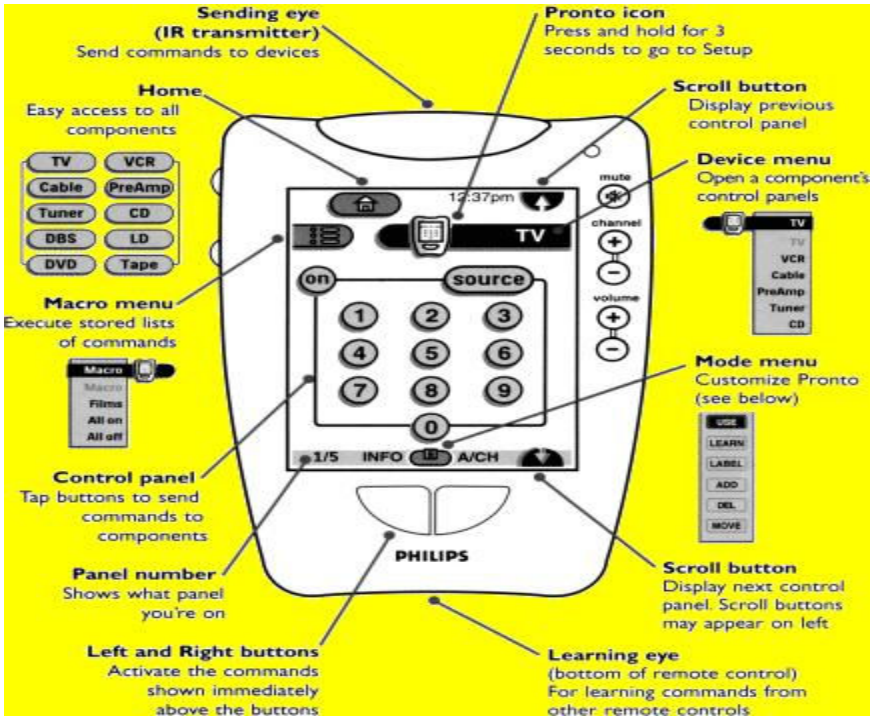


Figure 1 : Description de la face avant du Pronto

Les interfaces préconçues

Les interfaces préconçues sont elles aussi définies par le constructeur de l'équipement qu'elles vont contribuer à piloter, mais elles ne sont pas directement incluses dans un appareil dédié à leur pilotage. Elles sont stockées dans l'équipement qu'elles devront piloter et l'usager, pour les utiliser devra se connecter à l'appareil et les transférer sur la "télécommande" de son choix. C'est une méthode utilisée par le système JINI de Sun et ICrafter (partiellement). Cela présente plusieurs inconvénients : il faut que le constructeur ait prévu une interface pour chaque type de "télécommande" potentielle. Si le type de "télécommande" de l'usager n'a pas été prévu par le constructeur soit par ce qu'il est de conception antérieure à l'appareil soit parce qu'il a été omis, l'usager ne peut espérer au mieux piloter l'équipement désiré avec sa "télécommande" qu'avec une interface inadaptée ; au pire, il ne pourra pas piloter cet équipement. Avec ce type d'interface on ne peut pas espérer piloter un photocopieur du début des années 1980 avec un Pocket PC[1] par exemple, sauf si une maintenance lourde et coûteuse a permis de mettre à jour les interfaces disponibles dans le photocopieur.

Les interfaces à base de modèles (patterns)

Définition

Un pattern est une solution récurrente qui décrit et résout un problème général dans un contexte particulier [L1]. La notion de modèle a été introduite en 1970 pour l'architecture par C. Alexander [A1] : il a créé des modèles décrivant l'environnement physique dans lequel les gens allaient vivre et travailler. Le but d'Alexander était de donner aux usagers un moyen de communiquer de façon à ce qu'ils puissent exprimer leurs avis quant au projet.

[1] Ordinateur de poche.

L'utilisation des modèles en IHM pour la conception de systèmes interactifs existe depuis 1998 et poursuit le même but que celui décrit par Alexander. Un modèle est une description abstraite de la réalité qu'il devra représenter. Il sera lisible par tous, spécialistes ou non.

En 2000, le "CHI workshop" a admis pour les patterns une définition stipulant que "la description d'un modèle doit représenter précisément la solution à un problème dans le but d'être réutilisé dans le cadre de systèmes interactifs. Il doit inclure le nom des composants, leur classement, des exemples clairs, le contexte, les justifications (explications, exemples), la solution, des croquis, des références (à d'autres patterns), une synthèse et les crédits"[1].

Exemple

Le modèle présenté ci-dessous a été écrit par M.Van Welie [I2] et il décrit la façon de créer un menu contextuel, dans quel cas avoir recours à cette solution, ces avantages.

Contextual Menu	
Author	Martijn van Welie
Problem	At any point in time, the user needs to know what her possibilities are in order to decide what to do.
Principle	User Guidance (Visibility)
Context	An application typically contains a lot of functionality and the user needs to know her possibilities at any point during use. The user is typically a novice or casual user and the functions are used infrequently.
Forces	• The user may not be familiar with the meaning of a possibility. • The number of possibilities may be large and the user needs to locate the desired one. • Not all possibilities may be available in the current context, but the user may nevertheless want to know about the existence of the possibilities. • Integrating learning and using, while not hindering the expert
Solution	**Put the choices in a menu.** Show a list of all functions that are available in the current context. Make the functions accessible in one action. If the number of functions is high (> ~7), group the functions and make the groups distinguishable. If the total number of functions is low (in general or within a group), the list should show all functions and show which ones are possible to select in the current context. For users who are not familiar with the function label there should be a description available that explains the function. Functions should be ordered according to one or more of the following criteria; semantics, similarity, frequency of usage or alphabetically. If the list of possibilities does not differ much between contexts, the list should show all possibilities and highlight the possibilities of the current context.
Rationale	The list of functions gives the user an immediate overview of all possibilities. Humans are familiar with such list (e.g. a dinner menu) and will quickly recognise its function. The solution improves memo ability and satisfaction. It decreases performance speed because extra actions are needed.

[1] "An HCI design pattern captures the essence of a successful solution to a recurring usability problem in interactive systems. It consists of the components name, ranking, sensitizing example, context, problem statement, evidence (rationale, examples), solution, sketch, references (to other patterns), synopsis, and credits."

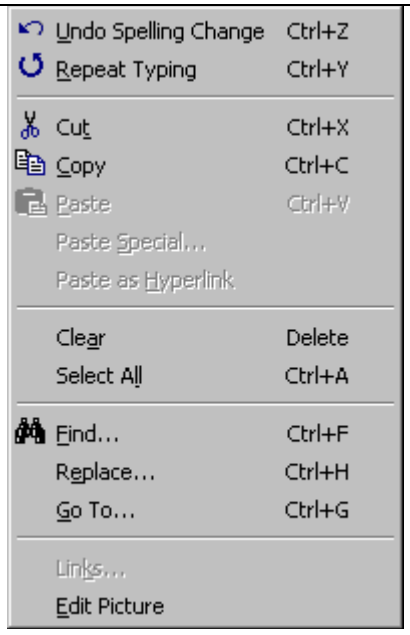
Examples	 This is the "Edit" submenu from Word97. This menu shows all the functionality related to editing things in a document. The functions are semantically grouped and only the possibilities of the current context are highlighted.
Known Uses	Any application with menu's,Web site menu's
Related Patterns	
Implementation	

Figure 3 : pattern de menu contextuel

Ce modèle montre qu'il propose une solution à un problème spécifique d'utilisabilité*[1] dans un contexte donné (utilisation de menus contextuels pour connaître à tous moments les actions que l'utilisateur peut effectuer).

Analyse

Ce type de moyens de créer une interface présente plusieurs avantages :

- Le respect du concept d'utilisabilité. Concept dont la définition en matière d'IHM est assez floue car plusieurs définitions existent, mais je retiens la vision de M. Welie [A4] et trouve que celle utilisée en ergonomie et définie dans la norme ISO 9241 est la plus adaptée à la conception d'IHM. Elle définit en effet trois buts principaux à atteindre pour une interface : l'efficacité, l'efficience et la satisfaction qui sont à mon avis les éléments essentiels d'une interface bien conçue. Dans ce cadre les patterns trouvent tout a fait leur place car réutiliser des modèles existants pour redéfinir une nouvelle interface permet à l'utilisateur de ne pas être stressé par quelque chose qu'il n'a jamais vu et d'être plus rapidement efficace ou d'atteindre plus rapidement son but.
- Il permet un gain de temps en terme de développement en permettant de réutiliser ce qui existe déjà.
- Le modèle n'est pas figé, un autre concepteur d'interface peut s'en inspirer, l'adapter au contexte particulier de son application. Utiliser, réutiliser un pattern et ainsi le compléter, l'enrichir en établissant des liens avec le modèle original et d'autres modèles semblables constituant ainsi un "langage de pattern" [L1]. Il ne s'agit pas là d'un langage de programmation pour produire des IHM, mais plutôt d'une collection de modèles décrits en langage naturel, interconnectés entre eux, constituant ainsi une sorte de réseau [A2], une sorte de "dictionnaire". Ces réseaux sont des patterns qui à tout niveau d'échelle, collaborent pour résoudre un problème complexe qui ne saurait être résolu par un seul pattern.

[1] L'astérisque indique que le mot est défini dans le lexique.

Le reproche qui pourrait être fait aux patterns tels qu'ils sont présentés précédemment c'est, une fois la description du pattern effectuée, d'être quelque peu figés : son utilisation reste limitée à une application précise dont le cadre est défini par le concepteur. L'utilisateur de l'interface ainsi conçue doit en accepter les règles et les modes de fonctionnement dus notamment à la plateforme matérielle pour laquelle elle a été conçue.

Le schéma ci-dessous résume le principe d'utilisation de pattern pour la création d'une interface et on voit bien que l'utilisateur ne peut espérer pouvoir utiliser cette application sur n'importe quelle plateforme matériel. Les interfaces générées sont en effet destinées à une seule plateforme matériel.

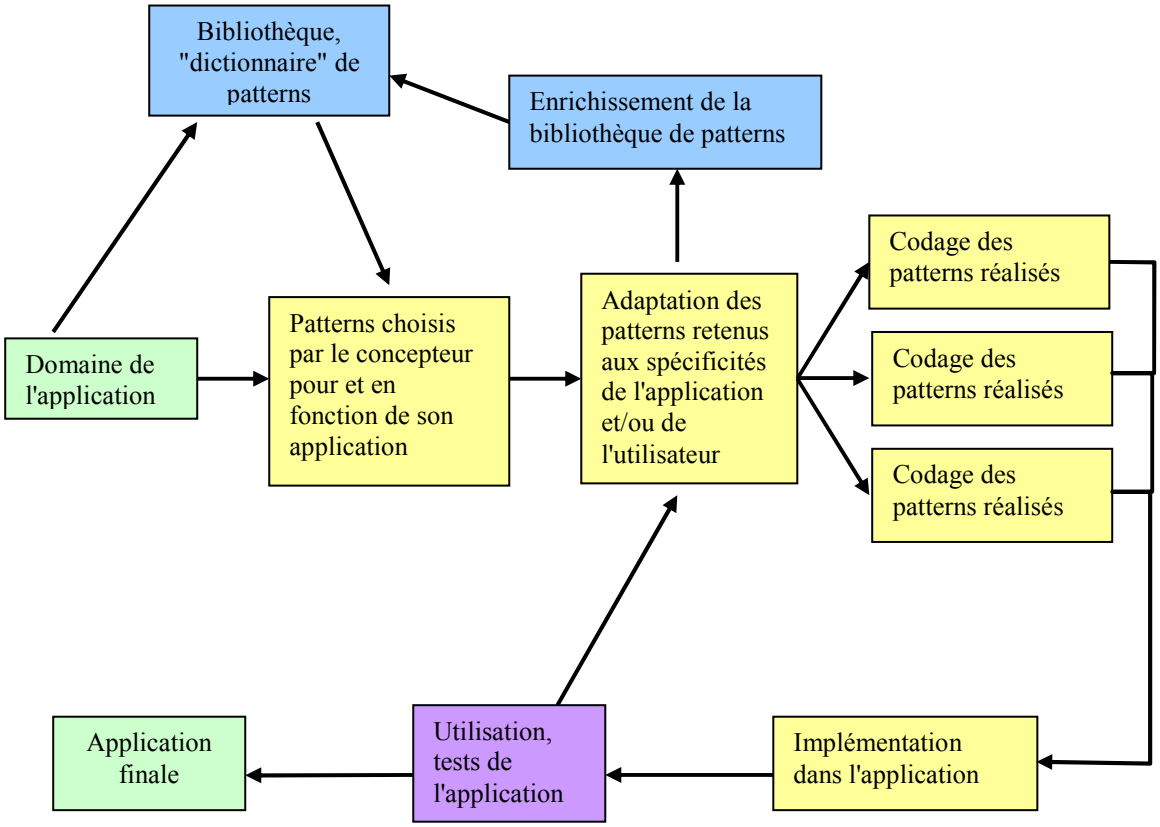


Figure 4 : Principe d'utilisation de patterns

Les interfaces autonomes

Définition

Ce sont des interfaces qui se constituent elles-mêmes en utilisant des données téléchargées depuis l'équipement qu'elles doivent piloter. Ces données ne comportent aucune consigne de mise en page ou de déclaration d'interfaces simplement une énumération des fonctions dont l'équipement dispose. Ensuite un "générateur d'interfaces" crée l'interface pour la plateforme cible[1] désirée l'adaptant ainsi, au moment de la création, aux capacités d'affichage de la cible. Cette méthode permet de créer des interfaces en accord avec les possibilités de la cible et les habitudes de l'utilisateur.

Si le "générateur d'interfaces" est implémenté directement sur la plateforme cible, l'interface sera immédiatement utilisable après génération (figure 5). En revanche, si le "générateur d'interfaces" est implémenté sur une plateforme tierce, il devra, après avoir récupéré les données de l'équipement à piloter et celles de la cible, générer l'interface puis la transférer sur la plateforme cible (figure 6).

Généralement la génération automatique d'interfaces utilise des langages à balises dérivés de XML[2]. UiML[1] ou XiML[2] sont deux exemples de langages orientés interface utilisateur basés sur XML

[1] "La plate forme cible" ou "la cible" est l'équipement destiné à recevoir l'interface.
[2] XML: e**X**tensible **M**arkup **L**angage.

[A2], [A3]. C'est sous cette forme que les fonctions de l'équipement à piloter vont être récupérées par le "générateur d'interfaces". Il utilisera ensuite la DTD[3] et la feuille de style XSL[4] ou un autre mécanisme équivalent propre à la cible pour générer l'interface *ad hoc*. En effet, la DTD est un document permettant de décrire un modèle de document XML. Une DTD est une sorte de grammaire de l'application XML. Elle permet de définir quelles balises sont autorisées, leur ordre et quels éléments elles peuvent contenir. Ainsi les balises qui ne concernent pas la cible seront écartées lors de la génération de l'interface. La feuille de style XSL est quant à elle un fichier qui décrit comment doivent être présentés, affichés ou lus les documents XML basés sur une même DTD.

Exemple

Les schémas présentés ci-dessous montrent pour les figures 5 et 6 comment s'établit le dialogue à travers le réseau dans les deux configurations matériel évoquées ci-dessus. La figure 7 montre un exemple de génération d'interface utilisant XiML.

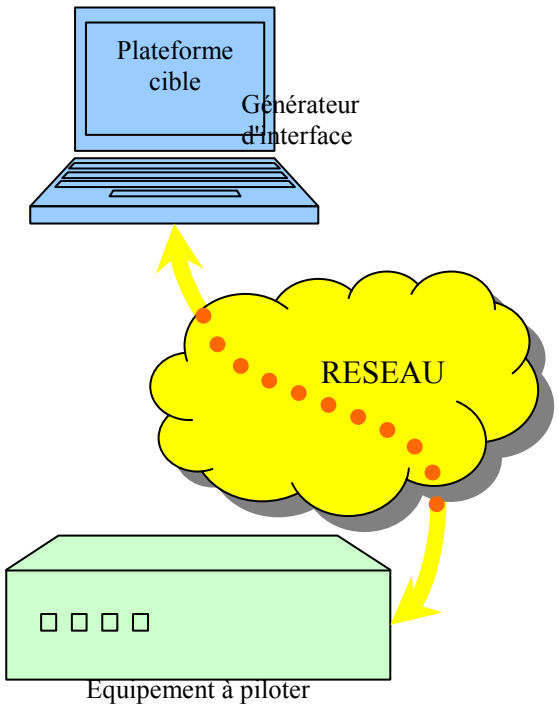


Figure 5 : Générateur d'interface implémenté sur la plateforme cible.

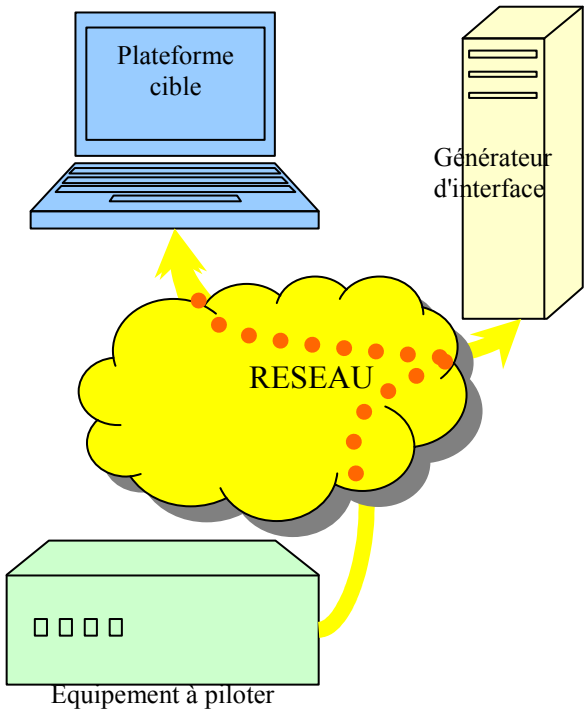


Figure 6 : Générateur d'interface implémenté sur une plateforme tierce.

[1] UiML: **U**ser **I**nterface **M**arkup **L**anguage.
[2] XiML: e**X**tensible **I**nterface **M**arkup **L**angage.
[3] DTD: **D**ocument **T**ype **D**efinition
[4] XSL: e**X**tensible **S**tylesheet **L**angage

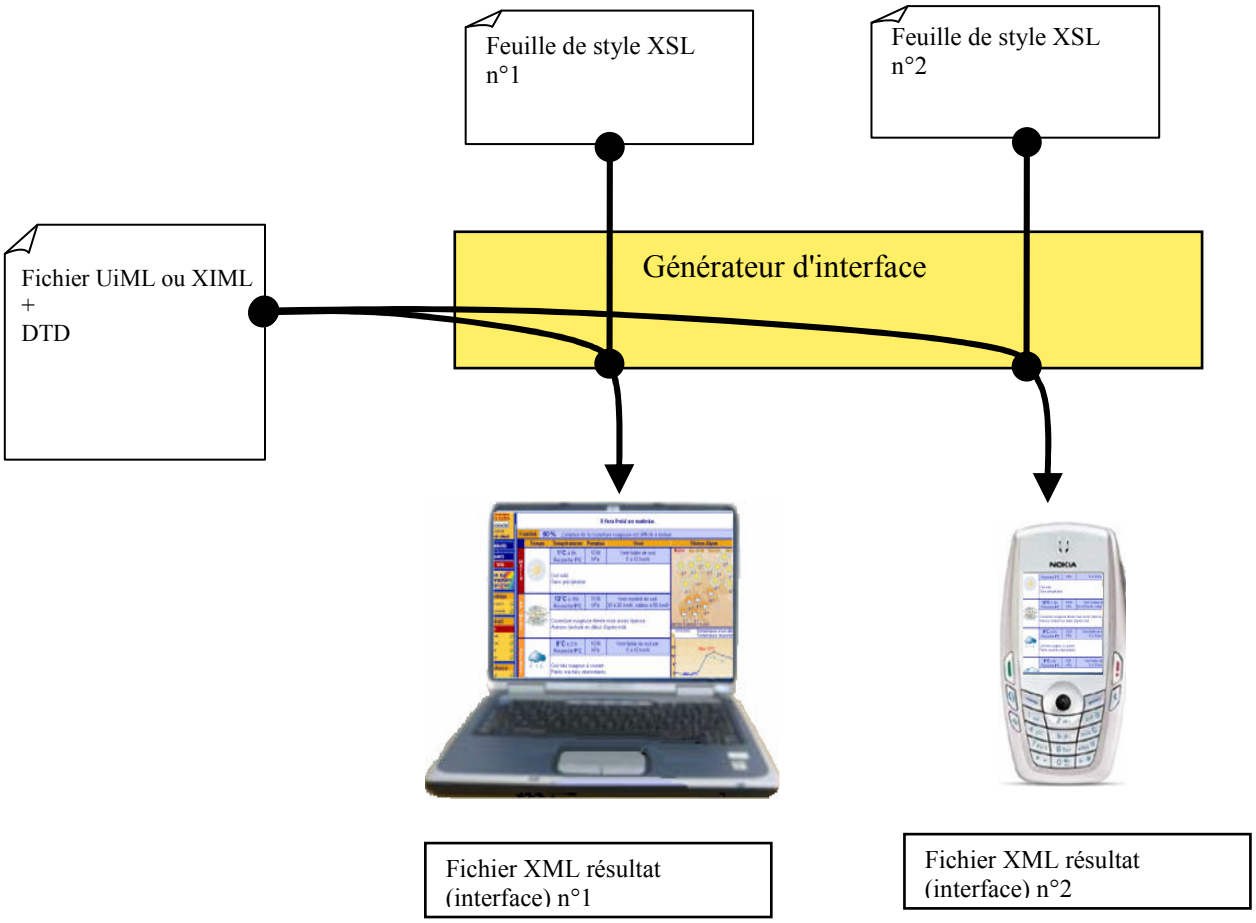


Figure 7 : Principe de génération d'une interface à partir d'un fichier UiML ou XiML et en utilisant DTD et XSL

Analyse

Les interfaces conçues de cette façon semblent être les plus à même de répondre aux exigences actuelles en terme de possibilité de réaction en face d'un nombre de cibles potentielles inconnues. L'utilisabilité y est aussi prise en compte grâce aux feuilles de style.

En revanche, cela laisse supposer une infrastructure réseau importante et aussi des modifications des équipements de la part des constructeurs. En effet, peu d'équipements sont à l'heure actuelle capables d'émettre des données dans le but d'une conception d'interface sur une cible externe.

Conception automatisée

Dans le contexte actuel, avec la multiplication des sources d'informations, des moyens d'accès à ces informations, avec les avancées de l'électronique qui permettent à nos équipements quotidiens de devenir "intelligents", avec la miniaturisation des équipements informatiques qui ont vu dans le même temps leurs performances s'accroître et, avec le développement des technologies sans fils, leur liberté et leurs moyens d'agir sur leur entourage décupler, il faut désormais penser les interfaces différemment. Des notions comme la multi-modalité*, l'adaptabilité*, l'adaptativité*, la plasticité*, l'utilisabilité*, le contexte*, les habitudes de l'utilisateur*, ses déplacements et les cibles nombreuses qu'il peut utiliser devront être pris en compte pour générer automatiquement une interface.

Méthodes de conception

La méthode SUPPLE

Supple [A11, 12, 13] est un outil de génération automatique d'interface qui aborde le problème de génération comme un problème d'optimisation d'interface dont le but est de minimiser les efforts de l'utilisateur pour utiliser l'interface dans un souci d'utilisabilité. Supple ne fait pas que construire l'interface, il l'adapte aussi aux moyens d'interaction de la cible. Cette méthode utilise trois modèles pour générer ses interfaces.

- Des spécifications fonctionnelles d'interfaces qui définissent le type de données qui doivent être échangées entre la cible et l'application.
- Un "device model" qui décrit les widgets[1] disponibles sur la cible ainsi que leur facilité d'utilisation en fonction de l'équipement cible.
- Un "user model" qui est une modélisation de la façon dont l'utilisateur utilise le plus souvent l'équipement cible.

Les spécifications fonctionnelles de l'interface (I) sont décrites sous la forme d'un arbre (figure 9) qui spécifie les fonctionnalités que doit présenter l'interface à l'utilisateur et non pas la façon dont elles doivent être représentées. C'est l'algorithme qui choisira cette dernière lors de la conception de l'interface. Une interface est définie comme étant $I \equiv \langle \varepsilon, C_I \rangle$ où ε est un ensemble d'éléments et C_I un ensemble de contraintes définies par le concepteur. La description des éléments de l'interface contient également des informations qui doivent être transmises lors de l'interaction et leur type sous la forme :

$$T \equiv \text{int} \mid float \mid string \mid bool \mid \langle T, C_T \rangle \mid vecteur(T) \mid \{T_i^{i \in 1 \ldots n}\} \mid T \mapsto null$$

int, float, string bool représentent les éléments de base. Par exemple, le widget simulant l'action sur un interrupteur pour allumer ou éteindre une lampe sera, comme dans l'exemple de la figure 9, représenté par un bool. $\langle T, C_T \rangle$ met en œuvre un type dérivé. C'est-à-dire le type lui-même T et les contraintes liées à ce type C_T. Dans l'exemple de la figure 9, l'intensité lumineuse ou la vitesse d'un ventilateur sont définis par des entiers limités dans leurs valeurs possibles (0 à 10 pour l'intensité de la lampe ou 0 à 3 pour la vitesse de la ventilation). $vecteur(T)$ représente une suite ordonnée (vide ou non) de valeurs. $\{T_i^{i \in 1 \ldots n}\}$ représente un ensemble d'éléments et $T \mapsto null$ représente un élément qui ne retourne rien de significatif pour Supple.

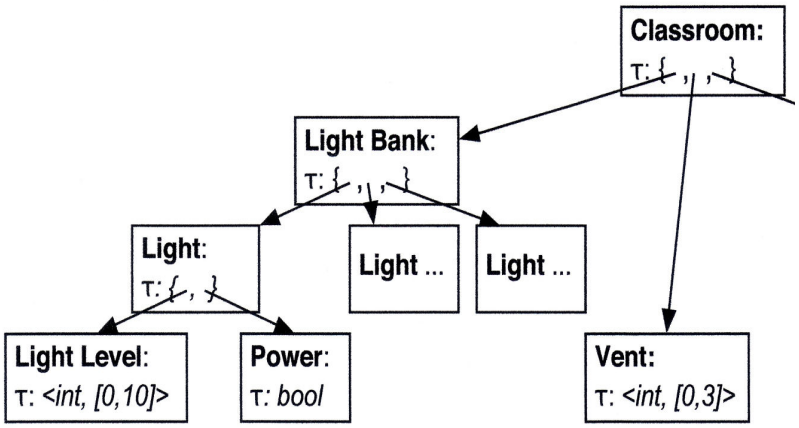


Figure 9 : Exemple d'arbre de description fonctionnelle de l'interface d'une salle de classe où il est possible de piloter plusieurs lampes et une ventilation.

[1] Widgets: moyens d'interaction.

De la même façon les capacités de la cible sont modélisées (D) sous la forme : $D \equiv \langle W, C_D, M, N \rangle$ où W est l'ensemble des widgets disponibles, C_D l'ensemble des contraintes spécifiques à la cible et M et N sont des fonctions qui permettent d'évaluer l'opportunité d'utiliser tel ou tel widget selon le contexte.

Le "user model" (T) est, quant à lui, alimenté par les habitudes d'interaction de l'utilisateur sous la forme $T \equiv \langle e_i, V_{oldi}, V_{newi} \rangle$ ou e_i est l'ensemble des éléments de l'interface manipulés lors de l'interaction, V_{oldi} leur ancienne valeur et V_{newi} leur nouvelle valeur consécutive à l'interaction. Il va sans dire que, lors de la première génération d'une interface, ces éléments sont remplacés par ceux fournis par le concepteur.

Tout au long du processus de création de l'interface, l'algorithme de Supple essaie d'attribuer un widget à un élément des spécifications de l'interface. Les éléments de base, qui correspondent réellement aux fonctions de l'application, sont associés avec des widgets capables de les afficher ou de les manipuler. Les éléments qui peuvent être logiquement groupés dans le modèle d'interface sont associés à des widgets qui permettront leur affichage sous forme de "pavé" horizontal ou vertical, de grilles ou encore en tableau. Il y a bien sûr une multitude d'associations possibles et la finalité est de construire l'interface qui correspond le mieux aux contraintes de la cible. Les ressources informatiques nécessaires à l'exécution de cet algorithme ne permettent pas d'envisager de l'implémenter sur une plateforme disposant de peu de moyens tel qu'un PDA ou un téléphone portable. L'algorithme est codé en JAVA. La figure 10 résume l'algorithme utilisé pour choisir les widgets dans chaque vue.

```
selectCommonActivities(interfaceModel, concreteInterface, trace)
1. initialize duplicatedFunctionality
2. while there is still a view with space for duplicated functionality
3. currentBestUtility ← 0
4. currentBestCandidate ← null
5. foreach view in concreteInterface that has space for duplicated functionality
6. foreach element in interfaceModel
7. temp ← duplicatedFunctionality + duplicate element into view
8. if utility(temp | concreteInterface, trace) > currentBestUtility
9. then currentBestCandidate ← duplicate element into view
10. if (currentBestCandidate != null)
11. then duplicatedFunctionality ← duplicatedFunctionality + currentBestCandidate
12. else return duplicatedFunctionality
13. return duplicatedFunctionality
```

Figure 10 : Résumé de l'algorithme de sélection

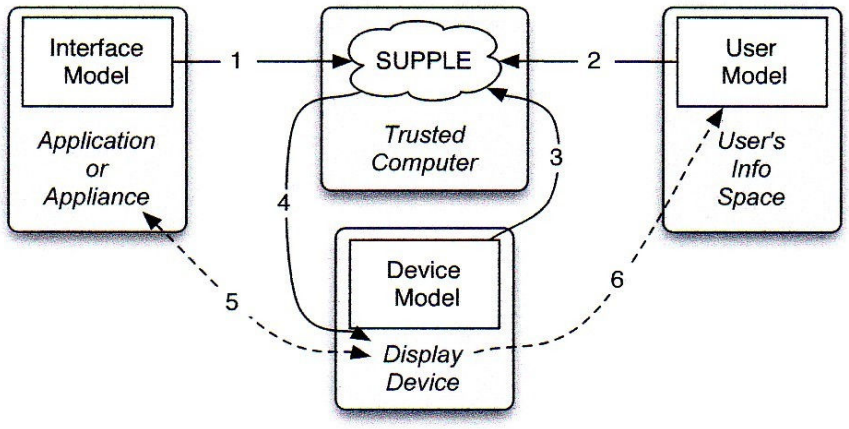


Figure 11 : Architecture de Supple.

La figure 11 montre comment ces modèles sont mis en œuvre.

D'un point de vue architectural, le "user model" est stocké sur un serveur propre à l'utilisateur (non représenté figure 11). Cela sous-entend qu'une machine autre que celle sur laquelle l'interface va être générée sera mise en œuvre si ce n'est pas cette machine qui sollicite l'interaction. L'"interface model" est fournie par l'application dont il va falloir établir une interface ou son développeur. Le "device model" est pour sa part fourni par la cible elle-même.

D'un point de vue création de l'interface, une fois l'exécution de l'application sollicitée par la cible, les modèles des spécifications fonctionnelles d'interfaces (flèche 1), de l'utilisateur (flèche 2) et de l'équipement cible (flèche 3) sont envoyés à la machine exécutant Supple. L'interface générée est envoyée à la cible (flèche 4). L'interaction entre l'équipement et la cible est alors fonctionnelle (flèche 5) et la cible met à jour le modèle de l'utilisateur sur son serveur (flèche 6).

En résumé Supple est un outil de génération d'interfaces qui génère des interfaces adaptées à la cible en se souciant de leur utilisabilité. En revanche, il est gourmand en ressources puisqu'il ne faut pas moins de quatre machines pour mettre en œuvre une interaction. Il n'est pas non plus très souple puisqu'il n'y a pas de passerelle prévue pour un éventuel changement de cible en cours d'interaction. Il ne donne pas de possibilité d'interaction autre que celle prévue par le concepteur et les habitudes de l'utilisateur dans un cadre donné. L'utilisateur ne peut pas passer sur une interaction vocale si cela n'a pas été prévu.

La méthode ICrafter

ICrafter [A14] est un framework, un environnement qui permet aux développeurs de concevoir des services ou applications et les interfaces de ces applications pour différentes cibles. ICrafter fonctionne selon trois axes.

- L'adaptabilité qui s'effectue selon deux directions.
  - L'adaptabilité aux équipements aussi bien dans le sens de supporter les différentes modalités d'un équipement que de pouvoir supporter plusieurs équipements ayant les mêmes moyens d'interaction.
  - L'adaptabilité au contexte. Le framework permet aux interfaces qu'il génère d'inclure des informations venant du contexte. Ces informations doivent être facilement exploitables par l'utilisateur.
- La "déployabilité" ou l'aptitude à évoluer et à se déployer facilement évoque un système qui doit répondre à trois exigences.
  - Le système n'oblige pas l'utilisation d'un langage de programmation particulier ni d'un langage spécifique pour les interfaces et non plus l'utilisation d'un OS[1] particulier. Le système ne gère pas l'affichage de l'interface sur la cible ; par conséquent les interfaces qu'il génère doivent pouvoir s'adapter immédiatement au moyen d'affichage de la cible, sur un navigateur web par exemple.
  - Le système utilise une palette, une bibliothèque de descriptions des interfaces car il est peu concevable de devoir décrire toutes les interfaces à la main pour chaque service et chaque équipement. Idéalement, quand un nouveau service est implémenté, il devrait être disponible sans qu'il y ait besoin de concevoir les interfaces manuellement pour chaque cible. De la même façon, ajouter une nouvelle cible ne doit pas obliger à décrire une interface pour chaque service. C'est pourquoi le système intègre la génération automatique ou semi-automatique d'interfaces et aussi la possibilité de personnaliser des interfaces existantes. Dans ce but, le système intègre une palette d'interfaces allant des interfaces complètement réalisées à la main aux interfaces conçues automatiquement qui lui permettent de transmettre à la cible soit des interfaces préconçues soit des interfaces générées automatiquement.
  - Le système est "robuste" et supporte sans trop de difficultés les pannes.

[1] **OS : O**pérating **S**ystem.

- L'agrégation, c'est-à-dire l'aptitude à générer des interfaces capables de gérer plusieurs services simultanément, non pas en accolant les interfaces de chaque service, mais en créant véritablement une interface incluant les fonctionnalités de chaque service utilisé. Par exemple, dans une salle de classe, l'enseignant peut avoir à piloter plusieurs postes d'éclairage et une ventilation : dans le principe d'agrégation, l'interface résultante devra respecter le principe d'utilisabilité et proposer à l'utilisateur l'interface la plus facile à utiliser.

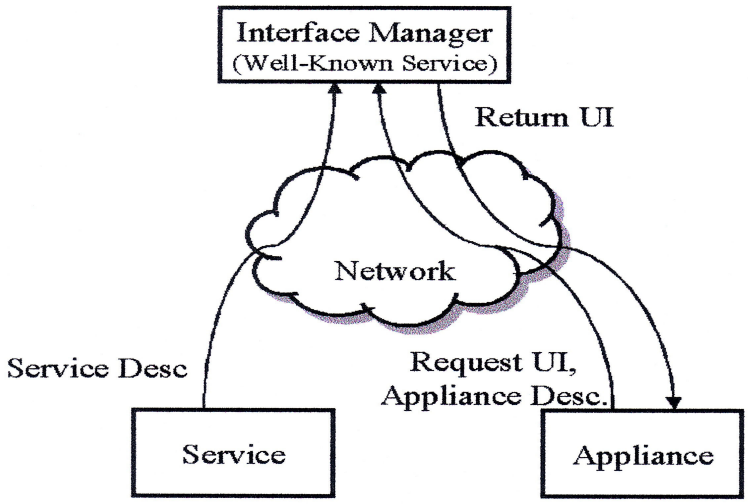


Figure 12 : Architecture générale de ICrafter

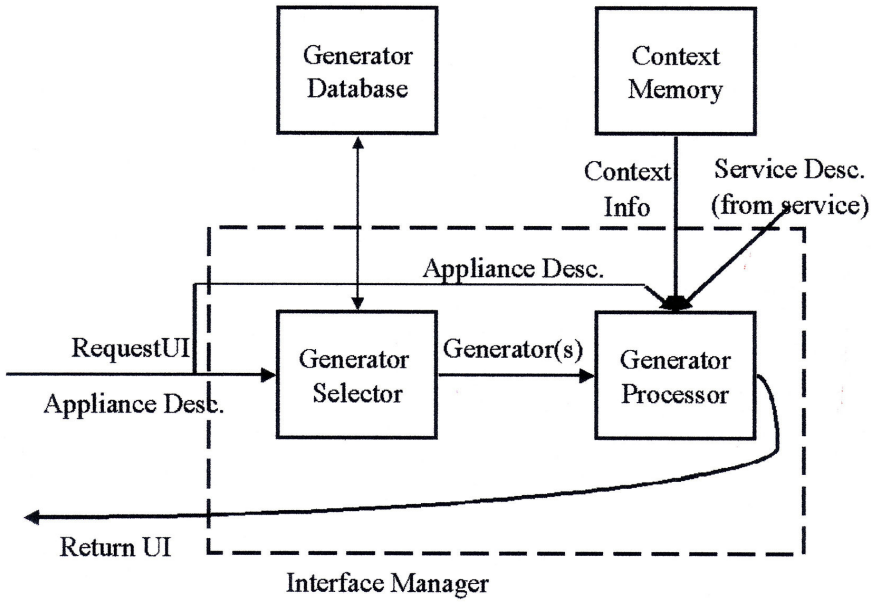


Figure 13 : Détails de l'**I**nterface **M**anager de ICrafter.

La figure 12 présente l'architecture générale de ICrafter. Les Services avec lesquels un utilisateur peut interagir signalent en permanence, un peu à la manière d'un sémaphore, leur présence. Ils utilisent pour cela un protocole de communication du style RMI/RPC[1]. Ils incluent dans ce signal une description des fonctionnalités du service. Lorsqu'une cible demande une interaction, elle sollicite l'"interface manager" (**IM**) et lui soumet une description de ses capacités. L'interface manager génère alors l'interface idoine et la transmet en retour à la cible.

[1] **RMI/RPC : R**emote **M**ethod **I**nvocation/**R**emote **P**rocedure **C**all.

La description des services est faite avec SDL[1], un langage basé sur XML et semblable à ISL[2] et à la méthode de description des services de UPnP[3]. Elle ne contient que la liste des opérations permises par le service.

La description des capacités de la cible consiste en une simple déclaration des langages d'interaction supportés par la cible comme HTML, VoiceXML, MoDAL[4] et SUIML[5] qui sont utilisés par ICrafter. Elle peut également contenir d'autres informations représentées sous la forme de couple nom/valeur, mais ce n'est pas obligatoire.

La figure 13 détaille le fonctionnement de l'"Interface Manager". Quand une demande d'interaction lui est soumise, il sélectionne dans un premier temps un ou plusieurs générateurs d'interfaces dans le "generator database" en fonction de la cible et du service avec lequel elle veut interagir. Un générateur d'interfaces est un logiciel qui peut créer une ou plusieurs interfaces pour la cible considérée en fonction du ou des services désirés.

Le "generator database" est une liste de langages (HTML, VoiceXML, MoDAL ou SUIML), de plateformes (Tcl, Python ou Java), de descriptions, de localisations et de patterns concernant le service ou l'équipement. Cette liste existe pour tous les générateurs d'interfaces disponibles. Des générateurs génériques sont même implémentés, ils peuvent répondre à tous les services ou équipements.

Dans un second temps, il génère l'interface avec le "compilateur" sélectionné en fonction des informations récupérées du "generator database" en prenant en compte la description des fonctionnalités du service, des capacités de la cible et du contexte. La description de la cible lui est fournie au moment de la demande de création d'interface et la description du contexte est accessible dans une base de données centrale appelée "context memory". Cette base de données est écrite en XML et interrogeable par un simple langage de requêtes.

Pour palier la déficience de ressources des équipements de type PDA ou téléphones, l'"interface manager" et toutes les ressources qui sont nécessaires à la génération d'interfaces sont implémentées sur une machine tierce.

Icrafter est donc une méthode très axée sur l'utilisation et l'adaptation de modèles plutôt que sur une véritable génération automatique. Cela ne lui permet pas d'utiliser une modalité autre que celle prévue initialement pour l'interaction. Il est par contre assez polyvalent, puisque ne se souciant guère du langage et de la plateforme utilisée sur la cible. Il n'est pas prévu pour changer de cible en cours d'interaction, mais peut générer des interfaces permettant d'interagir sur plusieurs équipements en même temps.

La méthode Pebbles

Pebbles est un projet [A15, 16, 17] cherchant à utiliser les équipements nomades pour interagir avec d'autres équipements nomades ou des équipements intelligents du type minichaîne, téléphone, magnétoscope et autre. Pour ce faire, un système baptisé PUC[6] a été mis au point. Ce système crée automatiquement des interfaces à partir d'un langage de spécification plutôt que de transférer sur la cible des interfaces préconçues ; il ne les adapte pas non plus d'un jeu d'interfaces pré-installé sur la cible. Cette caractéristique du système lui permet de créer des interfaces homogènes entre les différents équipements pilotés et de leur incorporer des éléments liés aux habitudes de l'utilisateur. Par exemple, si l'utilisateur veut piloter plusieurs magnétoscopes, il pourra retrouver les mêmes aspects et façons de fonctionner sur toutes les interfaces créées à cet effet.

Pour cela le système doit respecter certaines conditions.

- Une communication bilatérale de type peer-to-peer doit pouvoir s'établir entre le contrôleur et l'équipement à piloter. Ceci afin de permettre au contrôleur de charger

[1] **SDL : S**ervice **D**escription **L**anguage.
[2] **ISL : I**nterface **S**pecification **L**anguage.
[3] **UPnP : U**niversal **P**lug and **P**lay.
[4] **MoDAL : Mo**bile **D**ocument **A**pplication **L**anguage.
[5] **SUIML : S**wing **UI M**arkup **L**anguage.
[6] **PUC : P**ersonal **U**niversal **C**ontroller.

les spécifications de l'équipement avant de générer l'interface. Il est important que cette communication reste établie en permanence. Cela permet aux interfaces de donner des informations concernant des équipements qui ne sont pas ceux dont l'interface est actuellement affichée. Cela donne également la possibilité aux interfaces graphiques de désactiver des éléments de l'interface qui ne sont pas utilisables dans l'état actuel de l'équipement piloté. La connaissance de l'état dans lequel se trouve l'équipement est également très important pour les interfaces vocales.

- L'équipement doit pouvoir communiquer avec plusieurs contrôleurs en même temps afin de permettre l'utilisation, par exemple, de plusieurs modalités. Un utilisateur peut interagir avec un équipement à l'aide de l'interface graphique d'un contrôleur et avec l'interface vocale d'un autre selon la position qu'il occupe dans l'espace d'interaction ou plusieurs utilisateurs peuvent vouloir interagir selon des modes différents sur le même équipement.
- Les spécifications de l'équipement doivent comporter des informations sur les fonctionnalités de l'équipement et non pas sur la façon dont ces fonctionnalités doivent être portées à l'écran. Cela donne des spécifications plus courtes, cela permet de conserver la compatibilité avec les futurs modèles de contrôleur et cela donne la possibilité de choisir la modalité d'interaction.
- Les fonctions semblables doivent être groupées dans les spécifications. Cela permet à l'utilisateur de retrouver de manière intuitive une fonction et rend la création de l'interface plus aisée. L'utilisation d'un "arbre des dépendances" (figure 14) semble être la meilleure solution.
- Chaque action que l'utilisateur peut effectuer doit être décrite dans les spécifications de l'équipement. Les variables et commandes d'état sont un moyen simple de les représenter. Le système PUC déduit autant que possible les fonctions des variables, mais certaines utilisent aussi des commandes pour les actions qui ne peuvent pas être déduites de l'état dans lequel est l'appareil.
- Des informations de dépendance définissant quand des fonctionnalités ou des commandes particulières sont actives afin de pouvoir matérialiser leur état sur l'interface doivent être disponibles dans les spécifications de l'équipement. Si une commande ou une fonction n'est pas disponible elle peut être grisée à l'affichage par exemple rendant ainsi l'utilisation de l'interface plus aisée. Cela peut permettre, dans le cadre d'interfaces vocales, de lever des ambiguïtés en éliminant les actions qui ne sont pas possibles dans l'état actuel de l'équipement.
- Les textes pour les différents moyens d'interaction doivent aussi être explicites et sans ambiguïté pour générer une interface de bonne qualité. Cela est encore plus vrai pour les interfaces vocales où il n'y a pas de repère graphique pour guider l'utilisateur. Pour donner la possibilité au générateur d'interfaces de choisir l'étiquette la plus appropriée, sa description dans les spécifications doit comporter plusieurs textes et plusieurs annonces (speech text recording).
- Malgré toutes les conditions précédentes, il est des choses qui ne peuvent être écrites dans un fichier de description. Un concepteur, quand il crée une interface utilise aussi des conventions issues des habitudes, de l'expérience. Ce problème peut être contourné en implémentant ces informations dans chaque générateur d'interfaces en créant un jeu de descriptions de haut niveau de ces éléments dans les spécifications du langage. Cela permet au générateur, s'il est capable de les utiliser, de générer de meilleures interfaces en utilisant ces éléments ; sinon, le générateur demande à l'équipement des informations détaillées lui permettant de créer une interface utilisant les moyens de la cible. Cette approche a aussi pour bénéfice de réduire le temps de transfert initial des spécifications puisque seules les données de haut niveau sont transmises dans un premier temps.

Prenant en compte ces recommandations, l'équipe Pebbles a créé un langage de spécification et implémenté deux générateurs d'interfaces : l'un graphique est écrit en Java et l'autre, vocal est créé avec USI[1] [I3]. Ces deux possibilités associées aux descriptions de haut niveau des équipements permettent à l'utilisateur du PUC de choisir la méthode d'interaction qu'il désire, voire de changer de modalité en cours d'interaction. Le langage de spécification des équipements est basé sur XML et utilise une DTD pour les valider.

Les spécifications de l'équipement contiennent une représentation des fonctionnalités de l'équipement sous forme d'arbre où les fonctionnalités sont groupées par dépendance (figure 14). Ces spécifications contiennent également des dictionnaires d'étiquettes utilisables pour nommer les widgets d'interaction et aussi des représentations phonétiques de ces étiquettes. Cette représentation phonétique est constituée en utilisant ARPAbet, le jeu de phonèmes utilisé dans CMUDICT [I4], des balises signalant que le texte doit être "lu" selon les principes de SABLE [I5] et une URL pointant sur un enregistrement audio du texte.

Pour pouvoir connecter le PUC à tous les équipements, un adaptateur a dû être mis au point pour chaque type d'équipements, soit sous forme logicielle soit par modifications du matériel. Cet adaptateur a pour fonction de permettre aux équipements une communication bilatérale en peer-to-peer. La figure 15 développe l'architecture complète du système Pebbles.

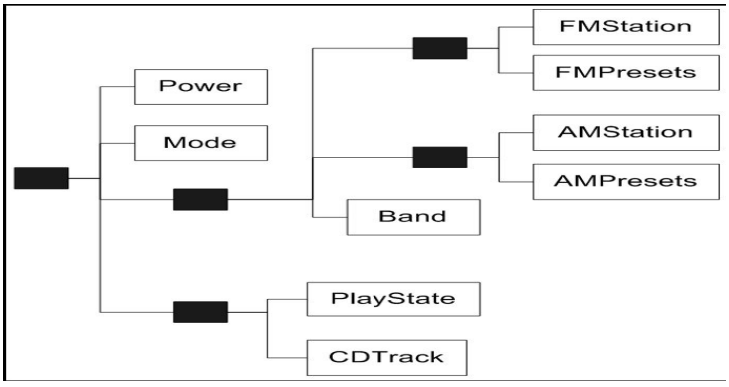


Figure 14 : Arbre de spécification d'une minichaîne où les fonctionnalités sont groupées par dépendance sous les rectangles noirs.

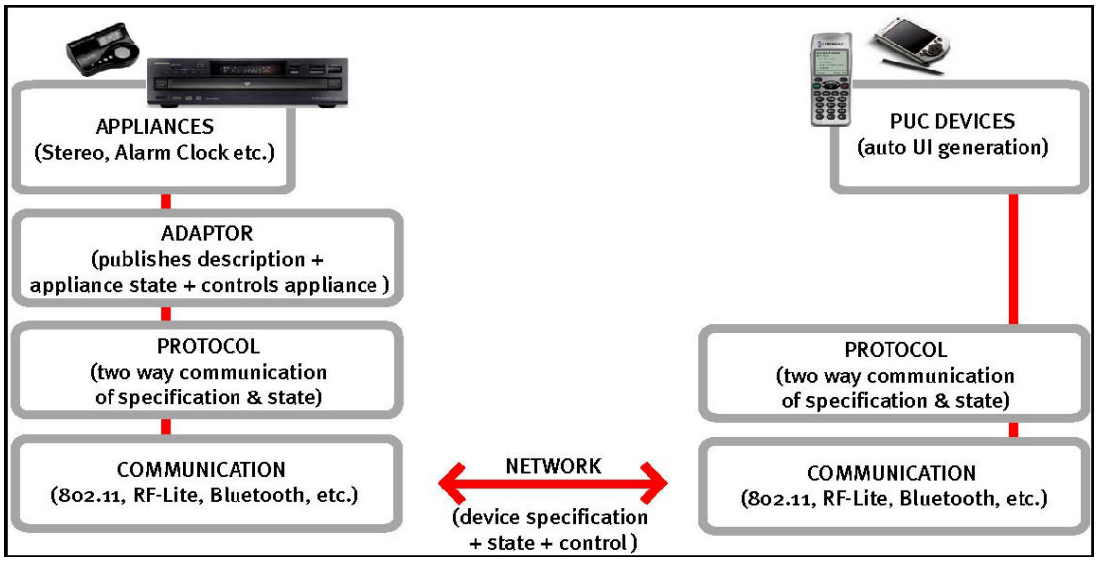


Figure 15 : L'architecture PUC.

[1] **USI : U**niversal **S**peech **I**nterface.

Récemment l'équipe Pebbles a implémenté dans son système PUC, UNIFORM[1], une nouvelle façon de garantir l'homogénéité des interfaces générées [A18], et donc de générer les interfaces. Deux études ont permis de dégager sept nouvelles règles permettant de construire des interfaces homogènes.

1. Les interfaces doivent permettre de manipuler des fonctions semblables de la même façon.
2. Des fonctions similaires doivent être localisées aux mêmes endroits sur des interfaces différentes.
3. Les interfaces doivent utiliser des étiquettes habituelles pour des fonctions analogues.
4. Deux interfaces doivent avoir des similitudes visuelles.
5. L'utilisabilité d'une fonction unique est plus importante que l'homogénéité de fonctions proches, c'est-à-dire qu'une fonction orpheline ne doit pas être reléguée au second plan si cela nuit à l'utilisabilité de l'interface.
6. Le générateur d'interface doit intégrer une méthode de recherche de fonctions similaires chez plusieurs équipements.
7. L'utilisateur doit pouvoir choisir avec quel type d'équipements il veut que son interface soit homogène.

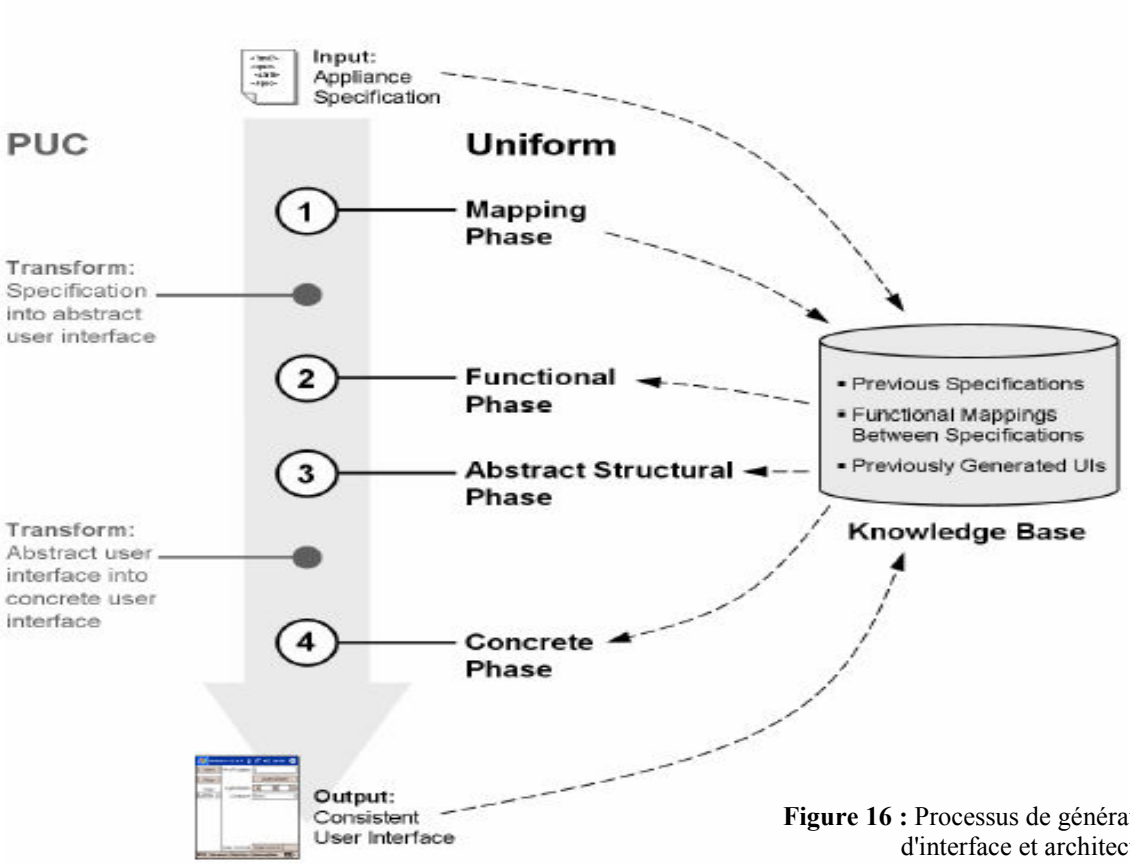


Figure 16 : Processus de génération d'interface et architecture

La figure 16 présente la nouvelle architecture du système PUC intégrant UNIFORM. La méthode de génération d'interfaces du système PUC utilise deux étapes : la première où les spécifications d'un équipement sont transformées en une interface abstraite ; la seconde où l'abstraction générée lors de la première étape est transformée en une interface concrète, spécifique à la cible et présentable à l'utilisateur. UNIFORM se dissocie du système de génération d'interfaces. Il influe sur la génération de l'interface en agissant sur les entrées, les sorties et les résultats intermédiaires. Il doit cependant connaître les structures de données utilisées par le générateur. L'homogénéité est assurée en quatre phases.

[1] **UNIFORM : U**sing **N**ovel **I**nterfaces **F**or **O**perating **R**emote that **M**atch.

- Une phase dite de "mapping"* où les spécifications de l'équipement dont on veut générer une interface sont comparées avec des spécifications connues et stockées dans la "knowledge base". Après cette comparaison, PUC transforme ces spécifications en une interface abstraite.
  La "knowledge base", littéralement base de connaissances, est un élément important dans l'architecture UNIFORM. Elle contient les spécifications des interfaces précédemment générées, le résultat des mappings effectués entre les différentes spécifications, des informations sur l'aspect des interfaces générées avec ces spécifications et, certainement le plus important, elle contient également le mapping des fonctions entre différents équipements.
- Alors que PUC vient de générer une interface abstraite, la "functional phase" vérifie les mappings fonctionnels issus de la première phase et éventuellement les modifie pour assurer l'homogénéité fonctionnelle de l'interface, c'est-à-dire pour s'assurer que le même jeu de moyens de contrôles (widgets) est utilisé dans d'autres interfaces pour assurer les mêmes fonctions.
- L'"abstract structural phase" où l'organisation de l'interface abstraite est modifiée pour vérifier que des fonctions similaires sont bien rassemblées dans un même groupe et assurer ainsi l'homogénéité structurelle de la future interface.
- Après que PUC a généré une interface concrète, elle est modifiée pour consolider l'aspect habituel de l'interface.

PUC, associé à UNIFROM, est donc un système capable de générer des interfaces homogènes multi-modales et multi-utilisateurs. C'est, des projets présentés, celui qui prend le plus en compte le souci d'utilisabilité des interfaces qu'il génère afin que l'utilisateur trouve un vrai bénéfice à utiliser le système Pebbles[1], un vrai lien entre tous les équipements qu'il peut piloter. D'un point de vue ressources, il est assez exigent puisqu'il faut modifier l'équipement avec lequel l'utilisateur veut interagir pour lui permettre une communication bilatérale. Il est plutôt destiné à des interactions "domotiques" avec des équipements du quotidien.

La méthode basée sur AMF

AMF[2] est un modèle d'architecture logicielle multi-agents multi-facettes [A10, 20] qui peut être vu comme une amélioration des modèles architecturaux PAC[3] [A7] et MVC[4] [A8]. Il associe la puissance des principes conceptuels des architectures multi-agents comme le fait PAC et les schémas de mise en œuvre opérationnelle qui ont fait le succès de MVC [A9].

Cette méthode propose une description plus générique du niveau abstrait de façon à ne pas imposer, comme dans les méthodes décrites précédemment de style d'interaction. Elle se décompose en trois étapes (figure 17).

- La première étape consiste à modéliser l'arbre des tâches génériques et à référencer les éléments de l'interface abstraite, c'est-à-dire les objets de l'application et les ressources possibles. Il faut ensuite établir les liens entre ces deux modèles et le modèle interactionnel de l'application. Cette étape est effectuée manuellement par le concepteur et n'est plus reprise. Le modèle des tâches doit être écrit de façon aussi abstraite que possible puis transcrit en XML. Ce langage correspond tout à fait à la représentation arborescente souhaitée pour décrire l'arbre des tâches. Cette modélisation de l'arbre des tâches est faite selon le principe CTT de Paterno avec l'outil CTTE.

[1] PUC + UNIFORM
[2] **AMF: A**gent **M**ulti-**F**acettes.
[3] **PAC: P**rentation **A**bstraction **C**ontroleur.
[4] **MVC: M**odel **V**iew **C**ontroleur

- La deuxième étape a pour but de construire des interfaces semi-concrètes, c'est-à-dire des interfaces concrètes paramétrables. Elle s'exécute automatiquement lorsqu'un utilisateur potentiel sollicite l'exécution de l'application en étant dans un environnement bien défini en utilisant du matériel reconnu. Pour générer dynamiquement des composants de l'interface adaptés à la cible, cette étape reprend les éléments issus de la première étape et les caractéristiques de la cible et les adapte pour créer des arbres des tâches spécifiques (ATS). Lors de l'exécution de cette étape, une seconde tâche est effectuée : c'est l'enrichissement du modèle d'interaction abstraite toujours en fonction de la cible. Cet enrichissement effectue trois opérations.
    - Il supprime du modèle initial les tâches non réalisables sur la cible soit parce qu'elles font appel à des fonctions qui n'existent pas sur la cible soit parce que les fonctions présentes ne peuvent être réalisées. Par exemple, la cible ne peut pas piloter un appareil initialement prévu ou elle peut le piloter, mais il n'est pas connecté. Il supprime également les éléments de l'interface dépendants des tâches qu'il vient de supprimer.
    - Il remplace chaque tâche abstraite par une tâche concrète connexe spécifique à la cible, issue de modèles de la "bibliothèque de templates".
    - En effectuant ce remplacement, il enrichit en même temps le modèle d'interaction abstraite AMF en y ajoutant les modèles d'interaction correspondant aux tâches remplacées.
- La dernière étape vise à produire l'interface concrète. Elle est responsable de la mise en page définitive de l'interface qui aura été préalablement adaptée à une cible donnée, c'est-à-dire au triplet plateforme, utilisateur, environnement.

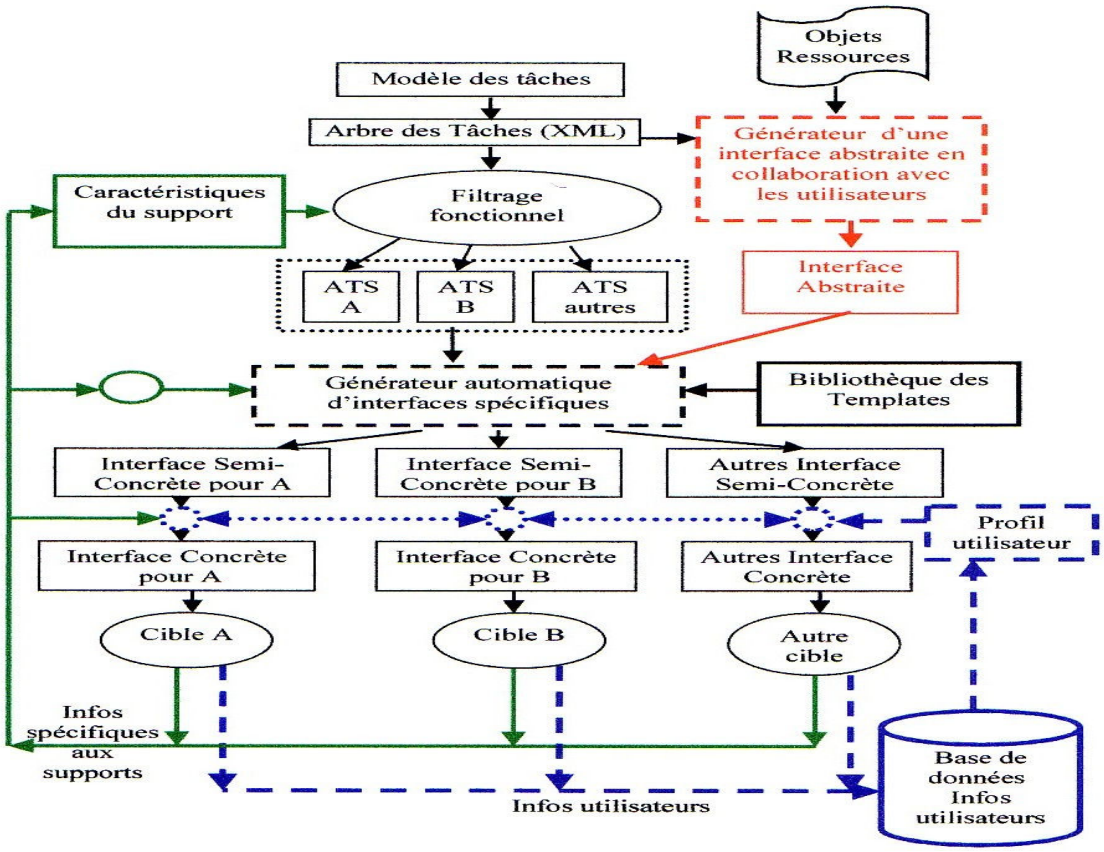


Figure 17: Processus de conception d'IHM multicibles AMF.

Comparaisons

Pour comparer ces différentes méthodes, j'ai choisi d'utiliser un tableau regroupant les différentes caractéristiques à analyser. Ces caractéristiques sont celles définies en début de paragraphe que doit utiliser une méthode pour générer une interface dans le contexte actuel.

	Interface constructeur type Pronto	Supple	ICrafter	Pebbles[3]	AMF
Multi-modal	Non	Non, pas une fois que l'interface a été générée	Non, pas une fois que l'interface a été générée	Oui	Oui
Multi-cibles	Non	Oui	Oui	Oui	Oui
Multi-utilisateurs	Non	Non	Oui	Oui	Oui
Adaptabilité	Oui, si l'utilisateur en a les capacités	Oui, si l'utilisateur en a les capacités	Oui	Pas au sens propre, mais l'utilisateur peut choisir avec quel type d'interface il veut être homogène	Pas à ma connaissance
Adaptativité	Non	Oui	Oui	Oui	oui
Utilisabilité	1[1]	3	2	5	
Homogénéité	oui	2, si les modèles des équipements ne diffèrent pas trop	3	5	4, utilisation d'une base de données et d'un profil utilisateur
Agrégation[2]	1	3[4]	5	3	3
Prise en compte du contexte	Non	Non	Oui	Oui	Oui
Prise en compte des habitudes	Non	Oui avec les "model user"	Non	Oui	Oui
Utilisation de modèles (niveau d'abstraction)	Non	Oui (3)	Oui (1)	Oui (4)	Oui (5)
Architecture	Aucune, communication unilatérale entre la télécommande et les équipements	Lourde, jusqu'à quatre machines mises en oeuvre	Simple, mais suppose que les équipements soient capables de communiquer	Simple, mais les équipements doivent supporter une communication bilatérale	Simple (peut s'alourdir lors de l'implémentation sur équipements nomades)
Domaine d'application privilégié	Interaction avec les équipements du même constructeur	Interaction multi-équipements	Interaction multi-users, multi interfaces,multi-équipements	Interaction multi users sur un même équipement multi-équipements depuis un seul utilisateur	Interaction multi-users, multi-équipements

1 : Pour les caractéristiques pouvant être évaluées, j'ai adopté un système de notation allant de 1 à 5 (5 étant la note maximale).
2 : Le système de référence étant celui de ICrafter.
3 : PUC + UNIFORM.
4 : Supple est capable de présenter une interface permettant de piloter plusieurs équipements mais pas de la générer à la volée.

Si un consensus relatif à l'utilisation de patterns à plusieurs étapes de la génération de l'interface semble être établi, il n'en ressort pas moins que de leur niveau d'abstraction résulte de la capacité qu'a l'interface générée à répondre aux critères définis au début du paragraphe. Plus le niveau d'abstraction est élevé, plus l'interface générée est capable d'accepter plusieurs modalités et de s'adapter à la cible. Le niveau d'abstraction permet aussi une intégration plus ou moins aisée des habitudes de l'utilisateur et du contexte. L'adaptabilité est toujours permise par un environnement de développement extérieur au processus de génération automatique et aboutit à des interfaces qui ne sont plus générables automatiquement. Il apparaît également que les équipements avec lesquels une interaction est envisagée ne sont pas équipés pour et qu'une adaptation est souvent nécessaire.

Mise en oeuvre de ces principes

Dans le contexte actuel, l'informatisation des équipements, l'augmentation des capacités des équipements nomades et la diminution de leur coût autorisent l'utilisateur à ne plus être rivé à un bureau. Le développement des technologies sans fils et des moyens d'interaction l'invite à sortir de chez lui. L'idée n'est plus de se connecter pour pouvoir interagir mais d'être connecté partout, toujours.

Les environnements pervasifs

Mise en place du réseau

L'émancipation de l'informatique et la mise en œuvre des technologies évoquées précédemment ne sauront se faire sans un support réseau adapté. Les équipements nomades ayant tout de même des capacités limitées, ils sollicitent souvent une machine tierce pour générer l'interface et interagir. Pour que cela puisse se faire, il faudra que cette machine tierce soit accessible partout et qu'elle puisse être informée du contexte dans lequel l'interaction va être mise en place. Il faudra également que l'application permettant l'interaction soit transparente et qu'elle puisse gérer des systèmes hétérogènes. C'est là la définition d'un environnement pervasif. Il doit être accessible partout (Ubiquitaire), intégrer les terminaux mobiles (Mobile) et prendre en compte le contexte d'exécution (Context-aware). La figure 18 illustre cette implémentation.

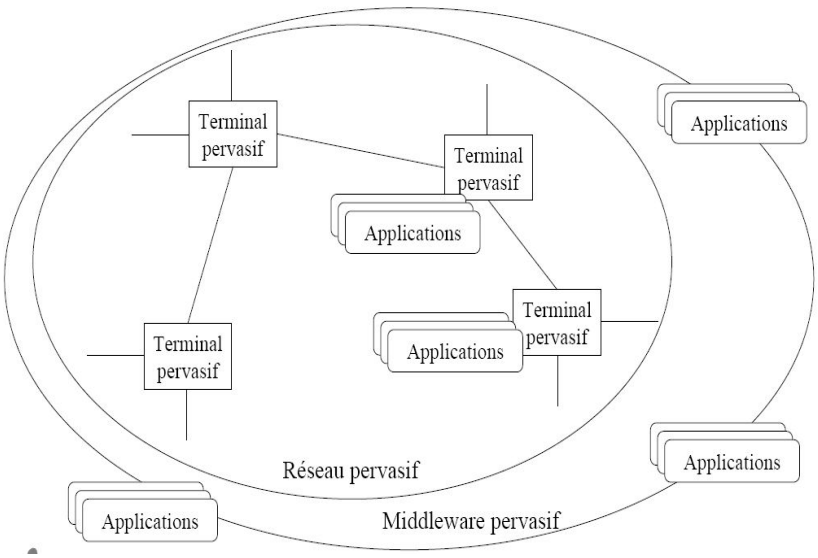


Figure 18 : Framework de système pervasif

La société Ozone [I6] installe à Paris et Rennes des réseaux pervasifs en WiFi. Des émetteurs placés sur les toits de particuliers volontaires servent à établir un maillage un peu à la façon d'un réseau peer-to-peer et les équipements nomades ou non peuvent se connecter au réseau.

Comment l'utilisateur s'insère-t-il dans ce milieu ?

Comme je l'ai déjà mentionné précédemment, il faudra que l'utilisateur soit connecté ou "connectable" à chaque instant. Il est hors de question d'envisager qu'avant de commencer une interaction un utilisateur doive entamer une procédure de connexion fastidieuse comme celles que nous connaissons à l'heure actuelle. Pour cela, il y a deux principes pour intégrer un utilisateur au système. Soit l'utilisateur se déclare, soit le système est capable de le découvrir.

La déclaration

Comme dans le système ICrafter, l'utilisateur émet un signal permettant au réseau de connaître l'existence d'un utilisateur potentiel. Cela permet également à l'utilisateur de donner au système ses caractéristiques qui en retour lui fournira une liste des services disponibles.

La localisation

Deux principes existent en terme de localisation. Le premier consiste à utiliser l'infrastructure réseau existante pour localiser l'utilisateur, le second consiste à implanter des capteur spécifiques.

Le système Aura [I7], développé par la Carnegie Mellon University, localise les utilisateurs et suit leurs déplacements en utilisant l'infrastructure existante. Une mesure de la qualité du signal reçu depuis plusieurs émetteurs permet de connaître la position de l'utilisateur (figure 19).

D'autres systèmes comme RADAR [I8], développé par Microsoft Research, et Cricket [I9], développé par le MIT, utilisent pour leur part des capteurs spécifiques implantés dans la zone de couverture (figure 20).

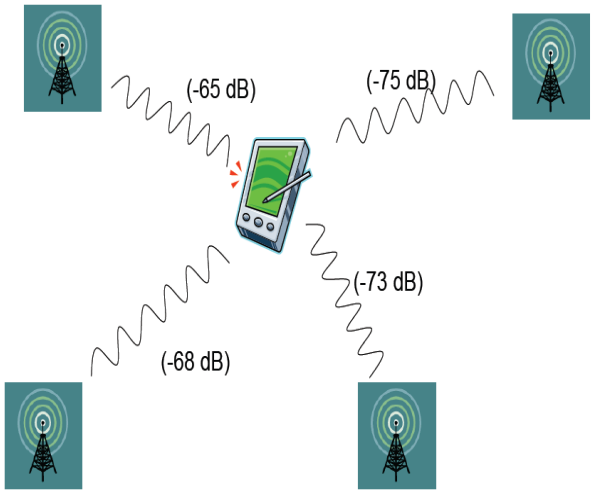


Figure 19 : Principe de localisation en utilisant l'infrastructure réseau.

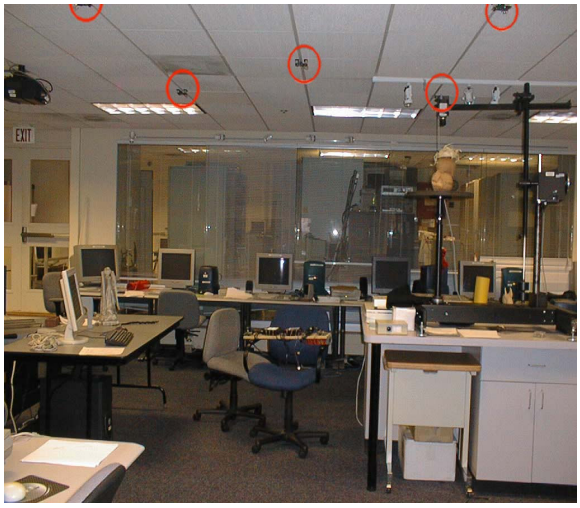


Figure 20 : Implantation des capteurs dans le cadre de Cricket.

Adaptation de l'interaction dans ce milieu

J'entends par milieu le triplet contexte-support-utilisateur. Je vais, à l'aide d'exemples, montrer comment l'interaction s'adapte à ces trois éléments dans un environnent intérieur public, dans un environnement fermé et dans un environnement extérieur.

Dans un environnement intérieur public

J'utiliserai pour illustrer mes propos un développement original du projet PEACH [A19], un guide interactif dans un musée. Ce développement exploite la possibilité de combiner les équipements nomades des visiteurs avec des équipements fixes simples. La méthode consiste à autoriser plusieurs

Figure 21 : Répartition des détecteurs infrarouges.

visiteurs à interagir avec les équipements fixes sur la base d'un système d'élections. La communication entre les différents éléments est assurée par un réseau sans fils. Les visiteurs sont localisés dans le musée à l'aide de détecteurs infrarouges (figure 21) qui permettent de déterminer quand un visiteur entre dans une zone d'interaction. Le visiteur est accompagné durant toute sa visite par un guide virtuel qu'il aura choisi en fonction de ses centres d'intérêt, guide virtuel neutre ou à caractère historique. Ce choix permet de classer les visiteurs. Lorsqu'il pénètre dans une zone d'interaction, le visiteur voit son guide s'activer, quitter son équipement mobile et réapparaître sur l'écran fixe, ceci dans le but de focaliser son attention sur l'écran fixe. Si d'autres visiteurs sont déjà à ce poste, le système, étant au courant des centres d'intérêts des différents visiteurs, va déterminer s'il est possible qu'ils interagissent ensemble ; sinon, il suggérera de passer à un autre poste. S'ils peuvent interagir simultanément c'est leur PDA qui va leur permettre d'interagir comme une télécommande. Cela présente plusieurs avantages. D'abord il ne gêneront pas les autres visiteurs en s'approchant de l'écran pour utiliser une dalle tactile, ensuite cela permet d'augmenter le nombre de visiteurs pouvant suivre ensemble la même présentation. L'arrivée d'un nouveau visiteur au poste entraîne automatiquement l'apparition de son guide sur l'écran qui informe les autres visiteurs de ses centres d'intérêts et la génération d'une liste de présentations résultant des intérêts de chacun. Cette liste peut donc être vide et dans ce cas, des présentations génériques sont proposées. Si tout le monde choisit la même présentation, le son est diffusé sur l'écran principal ; sinon, il est diffusé sur le PDA et écoutable avec une oreillette, le guide personnel du visiteur le guidant alors sur l'écran de présentation (figure22).
Dans ce type d'environnement, l'interaction doit évoluer en fonction de l'équipement mobile du visiteur, de ces centres d'intérêts et du nombre d'utilisateurs. Le réseau doit être omniprésent, le suivi du visiteur est assuré par des capteurs qui informent le système de sa position.

Figure 22 : Présentation partagée avec guides virtuels personnalisés.

Dans un environnement fermé

Je reprendrai, pour illustrer cette situation, l'exemple de la salle "iRoom", un prototype d'espace interactif du projet ICrafter évoqué dans la seconde partie. Dans cet exemple, un utilisateur entre dans l'iRoom et allume son moyen d'interaction : cela permet la transmission de ses caractéristiques au système (IM). En retour, IM lui renvoie une interface où figurent les équipements avec lesquels il peut interagir (figure 23). Il lui faut alors sélectionner ceux avec lesquels il veut effectivement interagir en fonction de ses besoins. Après avoir transmis ces informations au système, IM lui retourne une interface rassemblant tous les équipements correspondant à son choix (figure 24).

Dans la partie supérieure de l'interface figure une représentation de iRoom où sont matérialisés les équipements sélectionnés. Cela permet à l'utilisateur de déplacer les informations d'un équipement à l'autre par simple "copier-coller", lui donnant ainsi la possibilité de les passer directement de son écran sur le projecteur ou un tableau. La partie inférieure lui permet de sélectionner l'équipement en lui-même pour le piloter (allumer, éteindre, monter, descendre…) (figure 25). Il va sans dire que plusieurs personnes peuvent faire la même chose et interagir ensemble (figure 26).

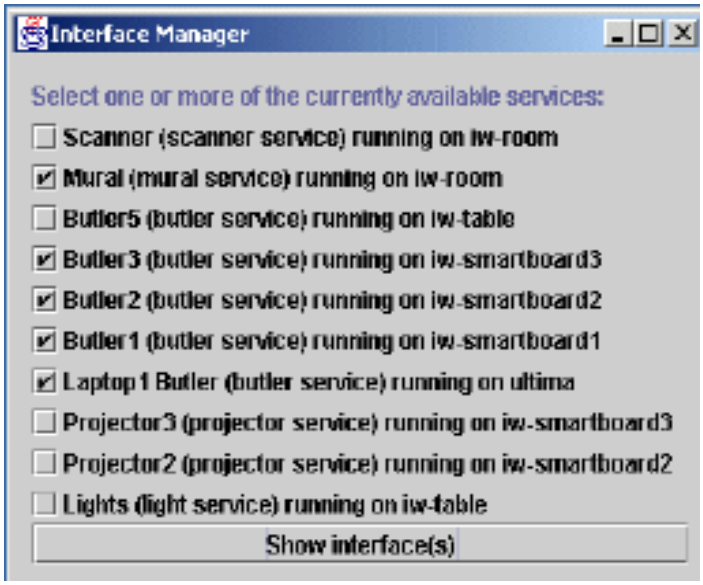


Figure 23 : Interface retournée par IM pour la sélection des équipements à mettre en œuvre dans l'interaction.

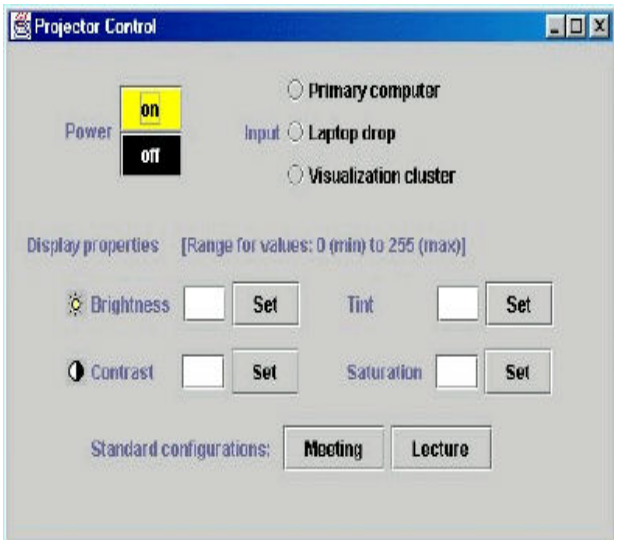


Figure 25 : Interface de pilotage d'un projecteur créée à partir d'un modèle la définissant complètement.

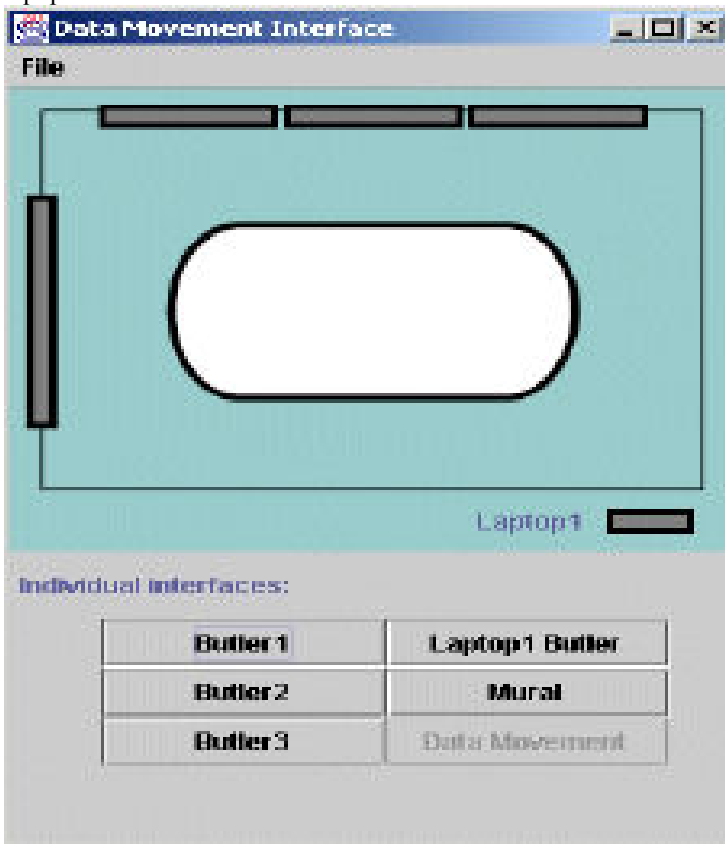


Figure 24 : Interface retournée par IM après la sélection des équipements à mettre en œuvre dans l'interaction avec la partie supérieure permettant le copier-coller et la partie inférieure permettant d'accéder aux interfaces de pilotage.

Figure 26 : Une interaction à plusieurs utilisateurs.

Dans ce type d'interaction, l'évolution s'effectue selon le nombre d'utilisateurs, d'équipements sollicités et de l'équipement avec lequel chaque utilisateur veut mettre en œuvre l'interaction. L'utilisateur étant statique, il n'est pas nécessaire d'assurer son suivi. En revanche, le système doit être capable de connecter dynamiquement tous les équipements mis en œuvre entre eux et avec tous les utilisateurs.

En extérieur

Je prendrai, pour illustrer une partie des interactions possibles en extérieur, le cas de la Brigade de Sapeurs-Pompiers de Paris qui, pour être assistée dans ses interventions, a été munie d'équipements mobiles informatiques (Figure 28). La figure 27 montre comment l'information pourra suivre les pompiers. Cela leur permet :

- de saisir en temps réel des renseignements ayant trait à l'intervention médicalisée ;
- d'éviter la double saisie de ces renseignements sur support papier et support informatique ;
- de faire fonctionner l'ensemble du programme en réseau avec accès simultané par tout poste informatique connecté. Une aide aux déplacements et aux interventions est possible, des alertes aux différents médecins peuvent être émises et les hôpitaux peuvent être prévenus avec envoi du dossier détaillé du patient ;
- de construire une base de données exhaustive pour les recherches épidémiologiques permettant notamment la recherche multi-critères (médico-légal) ;
- de constituer, en s'appuyant sur cette base de données, des états statistiques sur le nombre, le type d'interventions effectuées par les ambulances de réanimation ;
- d'assurer la traçabilité des patients et des soins prescrits ;
- de conserver les informations pour améliorer, puis évaluer la qualité des soins dans le cadre d'une démarche de qualité.

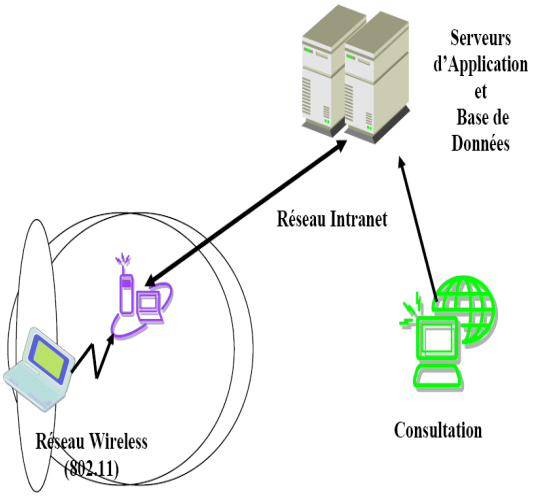


Figure 27 : Architecture mise en œuvre.

Figure 28 : L'équipement mobile fourni aux pompiers.

Dans cet exemple, on voit toutes les interactions dont disposent les pompiers. Dans un milieu ouvert tout est possible. On peut imaginer un utilisateur qui marche et décide d'aller au cinéma. Il consulte sur son PDA les cinémas qui diffusent le film qu'il veut voir et ensuite un plan lui indique le chemin à suivre pour se rendre au cinéma le plus proche. Un utilisateur peut consulter un plan de la ville et transférer les données directement sur son PDA. Puis il sera guidé dans son déplacement (figure 29) d'après le plan, y compris pour trouver un restaurant sur son chemin où son plat préféré l'attendra suite à sa commande.

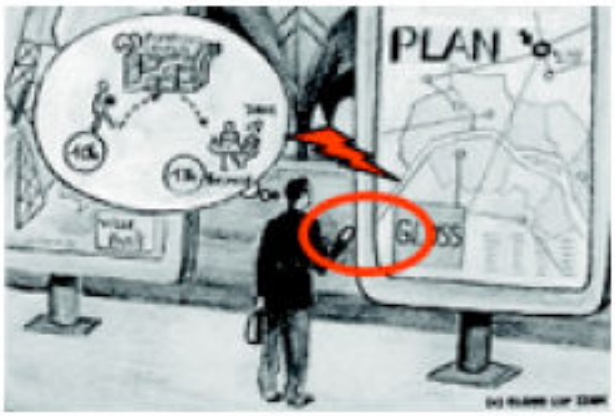


Figure 29 : Un utilisateur consulte un plan.

Conclusion

Dans ce monde où l'ordinateur devient évanescent au profit d'une informatique omniprésente, plus facilement intégrable dans notre vie de tous les jours et où l'utilisateur lambda va enfin pouvoir se servir d'elle pour faire autre chose que ses comptes et recevoir quelques mails, il serait préjudiciable que ces services ne soient pas facilement accessibles.

Dans cet environnement, l'Interaction Homme Machine a une place importante à tenir. Elle doit permettre à l'utilisateur d'accéder aux services qui lui sont proposés, de lui proposer les services qui correspondent à ses besoins, qu'ils soient explicites ou implicites, dans le cadre d'une pratique quotidienne, personnelle ou professionnelle.

L'utilisation de ces nouveaux moyens devra simplifier l'existence de l'utilisateur et leur mise en œuvre devra être transparente pour ce dernier. La conception automatique d'interfaces fiables, claires et facilement utilisables semble être la bonne démarche pour obtenir l'adhésion de l'utilisateur à ce nouveau concept. Il semble d'ailleurs qu'il soit déjà prêt à suivre cette voie si l'on considère le développement important et récent de la téléphonie mobile, le téléphone portable n'étant qu'un moyen d'interaction nomade permettant à l'utilisateur d'interagir avec la voix, l'image et le texte.

Lexique

Adaptabilité

L'adaptabilité est la propriété d'une interface utilisateur adaptable. La définition "d'interface utilisateur adaptable" donnée en 1994 par Opperman et Simm [A5] est toujours valide. Ils définissaient une interface utilisateur adaptable comme pouvant être modifiée par l'utilisateur en fonction de ses besoins.

Adaptativité

L'adaptativité concerne quant à elle la propriété d'une interface à se modifier automatiquement au cours de son exécution selon la définition donnée par Stephanidis en 2001 [A6].

Contexte

Le contexte est un couple formé par les éléments "cible / environnement" où la cible est considérée sous sa forme matérielle et logicielle comme un PDA ou un téléphone portable par exemple. L'environnement se réfère au milieu accueillant l'interaction. La luminosité, le bruit, la localisation géographique sont des facteurs qui peuvent influencer l'interaction.

Données utilisateur

Ce sont toutes les habitudes d'interaction de l'utilisateur, ses cibles usuelles, le type d'interface qu'il utilise le plus souvent, le type d'interaction dont il demande le plus la mise œuvre. Toutes ces données seront utilisées pour constituer de nouvelles interfaces en permettant à l'utilisateur de ne pas être stressé, toujours dans un souci d'utilisabilité.

Mapping

Qualifie l'établissement d'une relation entre des fonctions similaires des deux spécifications d'équipement différentes.

Multi-modalité

Littéralement une interface multi-modale pourrait être définie comme une interface permettant à l'utilisateur d'avoir plusieurs modes d'interactions en parallèle que ce soit un clavier, une souris, un écran tactile dans le cas de modalités classiques ou le geste, la parole, le regard dans le cas de modalités plus évoluées. Aujourd'hui il faut intégrer à cette définition la possibilité de faire la même chose de plusieurs façons. Par exemple proposer une sauvegarde soit par menu déroulant, soit par icône ou encore par raccourci clavier.
Pour la génération d'interface dans le contexte actuel les modalités seront multiples. Le clavier peut être physique ou virtuel avec un nombre plus ou moins important de touches. Le dispositif de pointage peut être une souris, un trackball, un joypad, un touchpad, ou encore une dalle tactile…

Plasticité

Cette notion regroupe en fait les deux précédentes. Par analogie avec le monde des matériaux, la plasticité d'une interface exprime son aptitude à s'adapter au contexte dans le respect de son utilisabilité. C'est-à-dire sa capacité à s'adapter aux caractéristiques physiques de la cible au moment de sa génération ou pendant l'exécution, que ces changements soient provoqués par le système ou l'utilisateur.

Dans le contexte actuel cela veut dire qu'une interface doit être capable de s'adapter avant sa création à la cible que va choisir l'utilisateur, pendant l'exécution à un nouveau choix de cible provoqué par l'utilisateur. Ce dernier peut choisir de changer de moyen d'interaction avec le système. Ce changement de cible peut aussi être provoqué par le système qui juge que sur la cible actuelle les ressources ne sont plus suffisantes pour garantir l'utilisabilité.

Utilisabilité

L'utilisabilité, en matière d'interface, est une notion proche de celle de l'ergonomie. La norme ISO 9241 la définit comme "le degré selon lequel un produit peut être utilisé, par des utilisateurs identifiés, pour atteindre des buts définis avec efficacité, efficience et satisfaction, dans un contexte d'utilisation spécifié." Les trois critères de l'utilisabilité soulignés par cette définition sont l'efficacité, l'efficience et la satisfaction. L'efficacité désigne le fait que le produit permet à ses utilisateurs d'atteindre le résultat prévu. L'efficience ajoute la notion de moindre effort ou de temps minimal requis pour atteindre ce résultat. La satisfaction finalement a trait au confort et à l'évaluation subjective de l'interaction pour l'utilisateur.

Abréviations

AMF	**A**gent **M**ulti-**F**acettes; modèle d'architecture logicielle
DTD	**D**ocument **T**ype **D**efinition
ISL	**I**nterface **S**pecification **L**anguage
MoDAL	**Mo**bile **D**ocument **A**pplication **L**anguage.
MVC	**M**odel **V**iew **C**ontroleur
PAC	**P**rentation **A**bstraction **C**ontroleur.
PDA	**P**ersonal **D**igital **A**ssistant (assistant personnel)
RMI	**R**emote **M**ethod **I**nvocation (invocation de méthode à distance, methode spécifiques à Java)
RPC	**R**emote **P**rocedure **C**all (appel de procedure distantes)
SDL	**S**ervice **D**escription **L**angage
SUIML	**S**wing **UI** **M**arkup **L**anguage.
UiML	**U**ser **I**nterface **M**arkup **L**anguage.
UMTS	**U**niversal **M**obile **T**elecommunications **S**ystem
UPnP	**U**niversal **P**lug and **P**lay.
USI	**U**niversal **S**peech **I**nterface. (un projet développé à l'université de Carnegie Mellon [I3]
WiFi	**Wi**reless **Fi**delity (réseau local Ethernet sans fil)
XiML	e**X**tensible **I**nterface **M**arkup **L**angage.
XML	e**X**tensible **M**arkup **L**angage.
XSL	e**X**tensible **S**tylesheet **L**angage

Références

Bibliographie

L1 Pierre-Alain Muller, Nathalie Gaertner "Modélisation objet avec UML"; deuxième édition 2000 Eyrolles.

Articles

A1 Christopher Alexander. "The Timeless Way of Building". Oxford University Press, 1979.

A2 Angel Puerta and Jacob Eisenstein "XIML: A Universal Language for User Interfaces"

A3 Marc Abrams and Constantinos Phanouriou "UIML: An XML Language for Building Device-Independent User Interfaces" XML '99, Dec. 1999, Philadelphia.

A4 Martijn van Welie, Gerrit C. van der Veer, Anton Eliëns "Patterns as Tools for User Interface Design" 2000.

A5 R. Opperman et H. Simm, Adaptability: User-initiated individualization, in R. Oppermann (ed), Adaptive User Support, Lawrence Earlbaum, 1994.

A6 C. Stephanidis, A. Paramythis, M. Sfyrakis et A. Savidis, A case Study in Unified User Interface Development: The AVANTI Web Browser. Paru dans [Stephanidis 2001] pp. 525-568.

A7 **J.** Coutaz : PAC, an Object Oriented Model for Dialog Design, in Proceedings Interact'87 (North Holland, 1987), 431-436.

A8 Krasner G.E., Pope S.T. A Cookbook For Using the Model-View-Controller User Interface Paradigm in The Smalltalk-80 System. Journal of Object Oriented Programming, 1988, 1, 3, pp. 26-49.

A9 K. Samaan/F. Tarpin-Bernard "The AMF Architecture in a Multiple User Interface Generation Process"(Workshop AVI04 Gallipoli (Lecce), Italy, 2004).

A10 F. Tarpin-Bernard/B. David "Amf : Un modèle d'architecture multi-agents multi-facettes"(Technique et science informatiques, volume 18, numéro 5, pages 555–586, 1999)

A11 Krzysztof Gajos and Daniel S. Weld "Automatically Generating User Interfaces For Ubiquitous Applications" (Workshop on Ubiquitous Display Environments, Nottingham, UK, 2004).

A12 K. Gajos/D. S. Weld "SUPPLE: Automatically Generating User Interfaces" (IUI'2004 Funchal, Portugal).

A13 K. Gajos/... "Fast And Robust Interface Generation for Ubiquitous Applications" (Ubicomp 2005, Tokyo, Japan).

A14 S. R. Ponnekanti/… "ICrafter : A Service Framework for Ubiquitous Computing Environments" (Ubicomp 2001).

A15 J. Nichols/B. Myers/… "Requirements for Automatically Generating Multi-Modal Interfaces for Complex Appliances" (IEEE Fourth International Conference on Multimodal Interfaces, Pittsburgh, PA. Oct 2002. pp. 377-382)

A16 J. Nichols/B. Myers/…"Generating Remote Control Interfaces for Complex Appliances" (UIST'02, 27-30 Oct. 2002, Paris, France)

A17 J. Nichols/B. Myers "Automatically Generating Interfaces for Multi-Device Environments". Ubicomp 2003 Workshop: Multi-Device Interfaces for Ubiquitous Peripheral Interaction.

A18 J. Nichols/B. Myers/B. Rothrock "UNIFORM: Automatically Generating Consistent Remote Control User Interfaces" (CHI 2006, April 22-27, 2006, Montréal, Québec, Canada).

A19 Michael Kruppa "The better remote control – Multiuser interaction with public displays" (MU3I 2004).
A20 G. Masserey/K. Samaan/F. Tarpin-Bernard/B. David "Implémentation du modèle AMF"(IHM 2006)

Internet

I1 Philips, Pronto Intelligent Remote Control. Philips Consumer Electronics, 2003. http://www.pronto.philips.com/
I2 http://www.welie.com/index.html
I3 http://www.cs.cmu.edu/~usi/
I4 http://www.speech.cs.cmu.edu/cgi-bin/cmudict
I5 http://www.bell-labs.com/project/tts/sabpap/sabpap.html
I6 http://www.ozone.net/
I7 http://www.cs.cmu.edu/~aura/
I8 http://research.microsoft.com/~padmanab/papers/infocom2000.pdf
I9 http://cricket.csail.mit.edu/

Figures

Figure 1 http://www.remotecentral.com/pronto/index.html
Figure 3 http://www.welie.com/patterns/gui/menu.html
Figure 7 Adaptée du cours de T. Accary-Barbier, http://liris.cnrs.fr/tiphaine.accary/teaching/XML_CNAM/xpath_xslt/3_XML_CNAM2005.pdf , diap69.
Figure 9 K. Gajos/D. S. Weld "SUPPLE: Automatically Generating User Interfaces" (IUI'2004 Funchal, Portugal).
Figure 10 K.Gajos/... "Fast And Robust Interface Generation for Ubiquitous Applications" (Ubicomp 2005, Tokyo, Japan).
Figure 11 K.Gajos/D.S.Weld "Automatically Generating User Interfaces For Ubiquitous Applications" (Workshop on Ubiquitous Display Environments, Nottingham, UK, 2004).
Figure 12 S. R. Ponnekanti/…. "ICrafter : A Service Framework for Ubiquitous Computing Environments"(Ubicomp 2001).
Figure 13 S. R. Ponnekanti/…. "ICrafter : A Service Framework for Ubiquitous Computing Environments"(Ubicomp 2001).
Figure 14, 15 J. Nichols/B. Myers/…"Generating Remote Control Interfaces for Complex Appliances" (UIST'02, 27-30 Oct. 2002, Paris, France).
Figure 16 J. Nichols/B. Myers/B. Rothrock "UNIFORM: Automatically Generating Consistent Remote Control User Interfaces" (CHI 2006, April 22-27, 2006, Montréal, Québec, Canada).
Figure 17 K. Samaan/O. Delotte/F. Tarpin-Bernard "Processus de génération d'IHM multicibles pour applications interactives"(IHM2002, Poitiers).
Figure 18 F. Laforest/F. Le Mouët "systèmes d'information pervasifs" (Cours: unité d'enseignement mutuelle)
Figure 20 http://cricket.csail.mit.edu/#overview
Figure 21, 22 Michael Kruppa "The better remote control – Multiuser interaction with public displays" (MU3I 2004).
Figure 23, 24 25, 26 S. R. Ponnekanti/B. Lee/A. Fox/P. Hanrahan/T. Winograd "ICrafter : A Service Framework for Ubiquitous Computing Environments"(Ubicomp 2001).
Figure 27, 28 http://www.mobilnew.info/mobilnew/IMG/pdf/medical1.pdf.