Vers un Prototypage des Interfaces Graphiques Incluant Vraiment l'Utilisateur Final

Jean Vanderdonckt, Adrien Coyette

Belgian Laboratory of Computer-Human Interaction (BCHI), Information Systems Unit (ISYS)
Louvain School of Management (IAG), Université catholique de Louvain (UCL)
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgique)
{jean.vanderdonckt, adrien.coyette}@uclouvain.be – http://www.isys.ucl.ac.be/bchi, http://www.usixml.org

RESUME

Le besoins en prototypage des interfaces homme-machine graphiques varient en fonction du moment où ils interviennent dans le cycle de vie de développement de l'application interactive. Dans ce but, la notion de niveau de fidélité du prototypage est introduite, définie et illustrée de façon à être supportée par des outils adéquats. Pour chaque niveau de fidélité, les forces et les faiblesses de ces outils sont développées de façon à identifier quel outil peut le mieux convenir en fonction de la phase de développement. Les niveaux de fidélité basse, modérée et élevée sont respectivement supportés par des outils idoines développés à cet égard : respectivement, un outil d'esquisse d'interface baptisé SketchiXML, un outil de dessin vectoriel, baptisé VisiXML, et un éditeur d'interface avancé, baptisé GrafiXML. Ces outils logiciels sont interopérables par l'échange de spécifications d'une interface homme-machine, rédigées dans le langage UsiXML (USer Interface eXtensible Markup Language), un langage de description d'interface homme-machine basé sur XML.

MOTS CLES : Approche dirigée par les modèles, conception assistée par ordinateur, langage de description d'interface, niveau de fidélité, outil d'esquisse, prototypage, USer Interface eXtensible Markup Language.

ABSTRACT

The requirements for prototyping graphical user interfaces vary depending on the moment they are considered during the development life cycle of the interactive application. For this purpose, the notion of 'level of fidelity' is introduced, defined, and illustrated so as to be supported by appropriate tools. For each level of fidelity, the strengths and the weaknesses of these supporting tools are discussed in order to identify which one is estimated the most appropriate for each step of the development life cycle. The levels of fidelity, respectively low, moderate, and high are supported by individual softwares which have been developed for this purpose: respectively a sketching tool, named SketchiXML, a vectorial drawing tool, named VisiXML, and an advanced interface editor, named GrafiXML. These tools are interoperable by exchanging the specifications of a user interface written in the UsiXML language (USer Interface eXtensible Markup Language), a XML-compliant user interface description language.

KEYWORDS : computer-aided design, level of fidelity, model-driven engineering, prototyping, sketching tool, user interface description language, USer Interface eXtensible Markup Language.

INTRODUCTION

La plupart des acteurs principaux intervenant dans l'équipe de développement d'une application interactive s'accorde généralement à dire que le prototypage de l'application en général [6] et de son interface homme-machine (IHM) en particulier [2] sont des activités à encourager fortement. Notamment pour découvrir le plus rapidement possible les écarts entre les besoins des utilisateurs finaux et les spécifications de l'interface, du système et pour bien d'autres raisons encore [4,25,27]. En revanche, quand il s'agit des moyens à déployer pour assurer la réalisation de ce prototypage et garantir son retour sur investissement, les moyens varient largement en forme, en méthode, en ressource mobilisée, en durée. Et les acteurs si prompts à favoriser le prototypage se retrouvent bien vite démunis quant au moyen le plus approprié pour leur projet et respectant leur budget. Puisque différents moyens de prototypage existent mettant en œuvre des moyens à coût variable, il nous paraît opportun de dégager des grandes familles de méthodes de prototypage d'interface en fonction de différents critères. Tel est l'objectif de cette communication.

On considère généralement que le prototypage rapide de l'interface est une méthode particulière qui se situe dans le cycle général de vie de développement de cette interface. Ou bien: "une stratégie spécifique pour conduire l'élicitation des besoins desquels les besoins des utilisateurs finaux sont extraits, présentés, et raffinés sur base d'un modèle de travail de l'application" selon l'interprétation de Boar [6]. Pour ce qui est de l'interface en particulier, il s'agit d'aboutir le plus rapidement possible à des spécifications précises qui rencontrent les besoins exprimés ou perçus comme tels par les utilisateurs finaux. Cette démarche devrait en principe s'opérer de façon à ce que cette découverte soit la plus efficace possible: arriver le plus rapidement en consommant le moins de ressources possible, notamment en évitant de coder directement l'interface, une activité considérée comme coûteuse. Pour ce faire, on remplace l'interface à spécifier par une représentation, un modèle. Puisque l'interface est graphique par nature – ou du moins de modalité

Fidélité\Critère	Basse	Moyenne	Elevée
Phase de développement	Elicitation des besoins, conception préliminaire, conceptualisation, début de l'application	Conception continue, validation de l'ergonomie du prototype, application en cours de route	Conception détaillée, application en fin de spécification
Contenu	Présentation surtout	Présentation, contenu, layout, début de la navigation	Présentation et navigation, contenu, layout, fonctionnalités
Usage	Exploration, découverte, évocation, communication	Simulation, raffinement, itération, amélioration, validation de l'utilisabilité, test utilisateur	Propagation générale à l'application, spécification finale, documentation, marketing
Type de prototypage	Horizontal	Diagonal	Vertical
Type d'approche	Ascendante (bottom up)	Expansive (middle-out)	Descendante (top down)
Facilité de changement	Elevée	Modérée	Très faible
Coût	Faible	Modéré	Elevé
Temps requis	Faible	Modéré	Elevé
Naturalité de la représentation	Très élevée	Modérée	Faible
Niveau de détail	Faible	Modéré	Elevé
Fréquence d'itération	Très élevée	Elevée	Faible
Niveau d'interactivité	Faible	Modéré	Elevé
Représentation	Esquisse	Dessin, dessin vectoriel	Présentation et navigation réelles
Convient pour…	Des applications de grande taille	Des applications moyennes	Des applications de taille réduite ou des fragments d'autres types d'application
Niveau des spécifications	Abstrait	Mixte	Concret
Outils en général	Denim [14], FreeForms [24], GUILayout [5], Paper [27], JavaSketchIt [8], SILK [17], SketchREAD [1]	EtchaPad [22], ExcelProto [3], MidFi [12], ProtoMixer [23]	Editeurs d'interface fournis avec les environnements intégrés de développement
Outils UsiXML	**SketchiXML [9,10,11]**	**VisiXML [30]**	**GrafiXML [19,20], FormiXML [30]**
URL	http://www.usixml.org/index.php?view=page&idpage=29	http://www.usixml.org/index.php?view=page&idpage=11	http://www.usixml.org/index.php?view=page&idpage=10

***Tableau 1**:* Comparaison des caractéristiques des différents niveaux de fidélité.

graphique majoritaire –, il est fondamental que cette représentation soit aussi graphique. Autrement, il est difficile, voire impossible, pour les utilisateurs finaux de réagir sur une autre représentation, de donner leur avis et de valider les spécifications en cours. Des spécifications abstraites, même si elles sont recommandables [10], ne permettent pas de faire intervenir pleinement l'utilisateur car la non-connaissance du langage abstrait demeurera toujours pour eux une barrière infranchissable. Il est donc primordial de d'abord déterminer quelle représentation manipuler, ce qui est abordé dans ce qui suit.

UN RÉFÉRENTIEL POUR LE PROTOTYPAGE D'INTERFACES

Comme les besoins en prototypage rapide d'interface varient en fonction du projet et des ressources qui lui sont allouées, nous proposons de choisir un prototypage basé sur la notion de fidélité du prototypage. La *fidélité du prototypage* exprime la similarité entre la représentation de l'interface prototypée et l'interface finale. On dira que la fidélité est *élevée* (en anglais, *high fidelity* [26]) si la représentation du prototype est la plus proche possible de celle de l'interface finale, pour ne pas dire qu'elles sont identiques. Dans ce cas, un prototype à un niveau de fidélité élevée devrait être aussi proche que possible de l'interface finale en termes de présentation (quels sont les objets interactifs utilisés), de navigation globale (comment naviguer entre les espaces d'information), de navigation locale (comment naviguer à l'intérieur d'un espace d'information), de contenu, de layout et de fonctionnalités. On dira que la fidélité est *faible* (en anglais, *low fidelity* [26]) si la représentation du prototype évoque l'interface finale sans la représenter totalement en détails. Entre les deux, on exprimera que la fidélité est *modérée* (en anglais, *mid fidelity* [11,12]).

En général, on considère que la fidélité d'un prototype reste constante et ne fait pas intervenir différentes représentations. Cependant, on peut très bien imaginer disposer d'un prototypage mélangeant différents niveaux de fidélité si le prototype est composé de fragments de fidélité différentes. Ceci peut arriver lorsqu'un prototype est imaginé à partir de nouvelles esquisses (fidélité faible), de recopies d'écran estimées applicables (fidélité élevée), ou de dessins issus de logiciel de présentation assistée par ordinateur (fidélité moyenne). Petrie & Schneider [23] présentent ProtoMixer, un outil de support au prototypage *mixte* mêlant plusieurs niveaux.

Le tableau 1 compare les trois niveaux de fidélité ainsi définis sur base d'une même grille d'analyse constituée de critères. Dans la suite du texte, nous reprendrons seulements les critères considérés comme les plus importants de cette grille d'analyse.

Le niveau de fidélité basse est applicable surtout au cours des étapes préliminaires du cycle de vie de développement d'une application interactive, principalement lorsque les spécifications de l'interface sont inconnues, incomplètes, restent à découvrir, à explorer. Dans ce cas, l'objectif est surtout d'explorer des conceptions alternatives possibles sans trop les raffiner afin de déterminer les grands contours de l'interface. Puisque cette étape peut être itérée fréquemment, il est fondamental que son coût et son temps de production demeurent faibles. Ceci au prix d'une représentation basse souvent centrée sur la présentation : pas question de développer ici déjà les fonctionnalités avancées de l'application si on n'est pas encore sûr qu'elles correspondent aux besoins des utilisateurs finaux.

Au contraire, le niveau de fidélité élevée convient principalement pour les étapes avancées du cycle de vie de développement d'une application interactive. Soit parce que le domaine est suffisamment connu et maîtrisé pour proposer une interface dont on pense raisonnablement qu'elle va s'approcher favorablement de l'interface finale. Soit parce que les itérations répétées du prototypage à un niveau de fidélité plus bas ont permis de dégager les conceptions acceptables et qu'il importe maintenant de les raffiner jusqu'à obtention de l'interface finale. Dans ce cas, l'objectif est surtout de peaufiner l'interface pour qu'elle soit la plus finalisée possible. Par conséquent, son coût et son temps de production sont élevés. Il est donc souhaitable de répéter le moins possible un prototypage à un niveau de fidélité élevée.

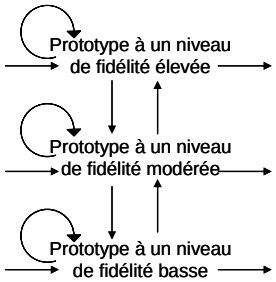

***Figure 1**:* Trajectoires possibles de prototypage.

Le prototypage à un niveau de fidélité élevée peut être consécutif à un prototypage à un niveau de fidélité moins élevé comme soulevé ci-dessus, mais pas nécessairement. Toutes les trajectoires de prototypage son théoriquement possibles entre les différents niveaux de fidélité [4,14,23] comme la figure 1 le représente. En principe, le prototypage peut être initié à n'importe quel niveau de fidélité, pour autant que cela corresponde aux besoins des utilisateurs finaux et se terminer à n'importe quel niveau (figure 1). En pratique cependant, on observe très fréquemment le passage d'une fidélité basse ou moyenne à une fidélité élevée pour supporter un prototypage réellement progressif et itératif. Le prototypage peut être itéré à chaque niveau, mais les itérations devraient être moins fréquentes à un niveau de fidélité élevé qu'à un niveau de fidélité bas ou modéré. Il n'est pas nécessaire non plus de traverser chacun des niveaux pour obtenir le prototypage souhaité. Tout dépend de la couverture de l'interface à prototyper. Pour cela, on peut s'appuyer sur différents cadres de référence permettant d'identifier quel type de prototypage est souhaitable.

Le cadre de référence de Nielsen (figure 2) distingue deux types de prototypage en fonction du niveau d'interactivité couvert (tableau 1). On peut décomposer une application interactive complète en trois couches : l'interface homme-machine (IHM), la couche d'abstraction et de contrôle de l'application et le noyau fonctionnel comportant les fonctions sémantiques de l'application ; ceci est typique du patron MVC (Model-View-Controller) ou PAC (Présentation-Abstraction-Contrôle) de J. Coutaz.

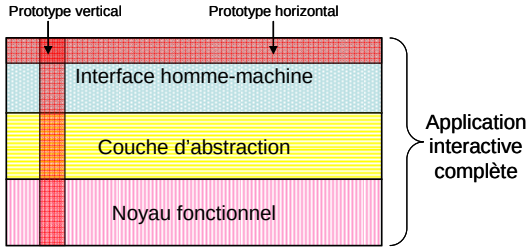

***Figure 2**:* Prototypages vertical vs. horizontal.

Le type de prototypage est alors dit *horizontal* si on souhaite prototyper le maximum des fonctionnalités de l'application au travers de son interface. L'interface est alors prototypée à un niveau de fidélité faible ou modéré pour vérifier que toutes les fonctionnalités sont bien identifiées et couvertes. En revanche, le prototypage n'est pas profond puisque seuls les aspects primordiaux (p. ex. la présentation) priment sur les aspects détaillés (p. ex. les fonctions réellement implémentées). Lorsque la première couche horizontale est finalisée, par exemple au travers d'une validation par les utilisateurs finaux, le prototypage peut se propager vers les niveaux inférieurs de la figure 2. L'interface est d'abord complétée, la couche d'abstraction est entamée, puis les fonctions.

Le type de prototypage est dit *vertical* si on souhaite se concentrer sur quelques fonctionnalités de l'application et que celles-ci doivent être suffisamment avancées pour valider l'approche retenue. L'interface est alors prototypée à un niveau de fidélité élevée pour que ces fonctionnalités soient les plus détaillées possibles. En revanche, le prototypage n'est pas large puisque seuls les aspects détaillés (p. ex. jusqu'à la navigation locale) d'un petit nombre de fonctionnalités comptent. Lorsque la colonne de prototypage vertical est aboutie, le prototypage peut se propager latéralement pour couvrir d'autres fonctionnalités non encore traitées auparavant.

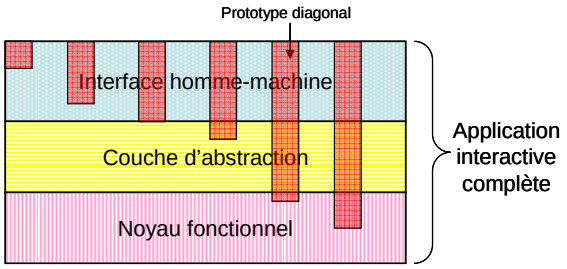

***Figure 3**:* Prototypage diagonal.

Dans la pratique, cependant, on observe souvent des projets où certains ensembles de fonctionnalités sont tantôt bien maîtrisés (p. ex. suite à l'expérience acquise dans le passé au travers d'autres applications interactives) tantôt presque inconnus (p. ex. aucun autre système semblable n'existe, aucune expérience préalable ou retour d'utilisation n'existe). Dans ce cas, il est possible de combiner les deux types de prototypage dans le prototypage que nous appelons *diagonal* [10,11] (figure 3). Dans ce type de prototypage, les parties bien maîtrisées sont soumises à un type de prototypage vertical tandis que les parties mal maîtrisées sont soumises à un type de prototypage horizontal. On peut ainsi marier les avantages des deux types sans en souffrir des conséquences négatives. La propagation du prototypage est alors dite *expansive* puisqu'elle peut s'étendre dans toutes les directions. C'est pourquoi le prototypage à un niveau modéré convient bien pour un type d'approche *expansive* (tableau 1), par opposition au niveau bas pour une approche *ascendante* et au niveau élevé pour une approche *descendante*.

Si maintenant nous poursuivons la décomposition de la couche interface homme-machine en trois parties (la présentation, la navigation globale et la navigation locale), le type de prototypage est alors être précisé et des nouvelles trajectoires de prototypage peuvent apparaître (figure 4). Le cas le plus fréquemment pratiqué consiste à initialiser le prototypage de l'interface par la partie la plus facile, la plus visuelle et la plus naturelle pour l'utilisateur final : la présentation des informations. On dit alors qu'il s'agit d'un *prototype présentationnel* d'abord (figure 4a). Il peut n'avoir aucune navigation ni globale ni locale, auquel cas il s'agit d'une simulation statique. La figure 4a montre une couverture plus large de la présentation pour refléter la partie qui est effectivement prototypée. En général, le prototypage par papier [27] convient bien et est largement éprouvé. Il peut aussi comporter un embryon de navigation, bien souvent globale d'abord, locale ensuite car on préfère spécifier les grands traits de la navigation globale avant de préciser ceux de la navigation locale qui en découlent presque directement. A nouveau, le niveau de fidélité du prototypage est fonction de la couverture à prototyper.

Moins fréquemment, on observe le *prototype navigationnel global* d'abord (figure 4b) dans lequel on élabore une architecture d'unités d'interaction ou d'information dont on ne spécifie que le but et que l'on relie en fonction des besoins informationnels de l'utilisateur final. Au fur et à mesure que la navigation globale se précise, on peut s'attaquer à la présentation particulière de l'unité d'interaction résultant ainsi de la navigation et en préciser quelques éléments de sa navigation locale. La figure 4b montre que la navigation globale est la plus touchée dans ce cas, puis la présentation, puis la navigation locale.

Enfin, encore moins fréquemment apparaît le *prototype navigationnel local* d'abord (figure 4b). Dans celui-ci, on précise quelle interaction l'utilisateur final souhaite avoir avec les différents éléments d'information précis (p. ex. quel filtre de recherche, quel ordre d'encodage, quelle suite de boîte de dialogue dans un « wizard »). Ceci étant précisé, on représente alors physiquement ces éléments d'information (p. ex. au travers des objets interactifs, du contenu) au sein de la présentation et on boucle le tout par spécifier les grands traits de la navigation globale. La figure 4c montre que, dans ce cas, la navigation locale a attiré le centre d'intérêt du prototypage, puis la présentation, puis la navigation globale. Après quoi, un prototypage vertical ou horizontal peut être poursuivi.

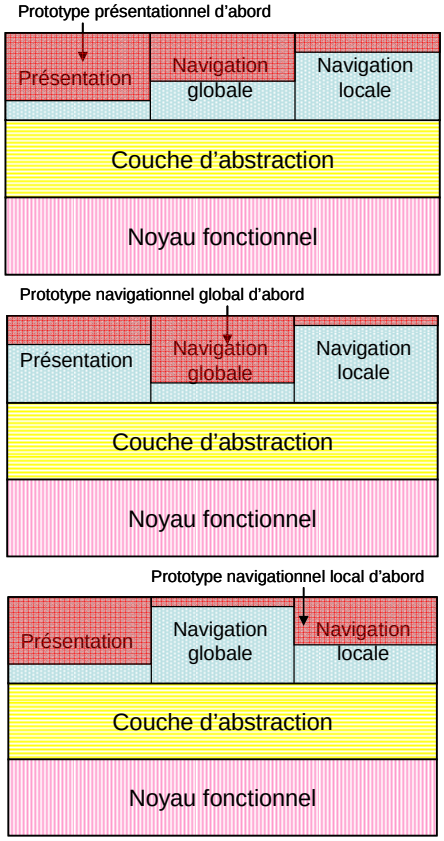

***Figure 4**:* Différents prototypages de l'interface : présentationnel (a), navigationnel global (b), navigationnel local (c).

Après avoir analysé quels sont les types de prototypage possibles et après avoir dégagé certaines tendances, voyons à présent comment supporter au mieux chaque

niveau de fidélité par des outils logiciels appropriés. Pour chaque niveau de fidélité, nous passerons en revue d'abord les grandes caractéristiques du prototypage à supporter, puis nous citerons quelques outils représentatifs avant de présenter l'outil logiciel que nous avons développé pour chaque niveau.

L'ensemble de ces outils est interopérable : ils communiquent entre eux tous les spécifications de l'interface en cours de prototypage au moyen d'un fichier rédigé en UsiXML (USer Interface eXtensible Markup Language, http://www.usixml.org) [19,20]. Ce langage de description d'interface utilisateur (en anglais, *User Interface Description Language* - UIDL) possède une vocation descriptive (spécifier au mieux les aspects de l'interface en cours de développement) et une vocation générative (autoriser la génération automatique de code ou un rendu automatique de l'interface à partir de ses spécifications). Ce langage de type XML a été créé au sein du projet européen Caméléon (http://giove.cnuce.cnr.it/cameleon.html) et est poursuivi actuellement au sein du réseau d'excellence Similar (http://www.similar.cc). Il couvre notamment les aspects de tâche, du domaine, de l'interface abstraite, de l'interface concrète et du contexte d'utilisation (c'est-à-dire l'utilisateur, sa plate-forme logicielle et matérielle et l'environnement physique dans lequel il est plongé). Dans la suite, seule l'interface concrète sera utilisée. Pour plus de détails, voir [19,20] et le site. Il existe d'autres langages similaires, avec des couvertures plus ou moins différentes et des objectifs différents: l'étude [28] procure un aperçu de ces langages de description sous la forme d'une grille d'analyse.

Voyons à présent quels sont les outils disponibles en commençant par le niveau de fidélité élevée puisqu'il est censé être le plus proche de l'interface finale que nous espérons. Nous verrons ensuite progressivement les niveaux de fidélité inférieurs. Il est aussi convenable de débuter par ce niveau car il permet de voir comment les différentes représentations de l'interface finale se transforment dans les niveaux de fidélité inférieurs.

PROTOTYPAGE À UN NIVEAU DE FIDÉLITÉ ELEVÉE

A ce niveau, l'objectif est de maximiser la proximité entre l'interface prototypée et l'interface finale, à obtenir et produire. Quand on parle de proximité, il s'agit de couvrir un maximum des aspects de l'interface (présentation, navigations globale et locale), mais aussi l'ordonnancement dans le temps et dans l'espace de l'utilisation des fonctionnalités. Pour cela, il faudrait que le prototype puisse accepter des entrées au départ des dispositifs d'interaction supportés (le plus souvent, le clavier et la souris) afin de les traiter et de les refléter dans le test des fonctionnalités. Certains prototypes peuvent débuter sur un jeu de données restreint, non nécessairement connectées à une base de données, afin de simplifier le prototype. D'autres, au contraire, mettent en jeu des sous-ensembles utiles de bases de données.

La plupart du temps, les concepteurs et les développeurs se reposent sur les outils qui sont fournis en standard avec les environnements intégrés de développement, tels que les éditeurs d'interface (en anglais, *interface builders*) d'origine. Par exemple, Microsoft Visual Builder dans l'environnement éponyme.

D'un côté, ces éditeurs sont ceux qui offrent la présentation et la navigation native de la plate-forme sur laquelle l'application interactive est développée ; on peut représenter directement l'interface désirée en tirant d'une palette d'objets ceux qui vont constituer l'interface et les placer dans une surface de travail, ce qui constitue un avantage indéniable tant du point de vue fonctionnel que visuel (l'interface graphique étant visuelle par nature). D'un autre côté, ces éditeurs se restreignent souvent à l'édition d'interface graphique basée sur des objets interactifs statiquement prédéfinis. Dès lors qu'il s'agit d'avoir des objets interactifs non natifs, non standards ou dynamiquement gérés (p. ex. une barre de menu variable, une boîte de dialogue fonction du contenu), l'avantage de l'éditeur cesse : il n'est plus possible d'éditer une telle interface et il faut recourir à la programmation manuelle. Il en va de même pour tous les aspects de navigation.

Par conséquent, les éditeurs d'interface s'avèrent appropriés pour les prototypes présentationnels, surtout en horizontal, mais très peu pour les prototypes navigationnels, particulièrement en vertical. En effet, dès qu'il s'agit d'un comportement à développer, il faut recourir à la programmation, dont le coût n'est évidemment pas souhaitable en phase de prototypage. Par analogie, le prototypage diagonal est difficile à supporter.

En raison de ces limites, plusieurs personnes se tournent vers des outils ne présentant pas les mêmes lacunes, tels que les outils dits de *façade* [2,4,27]. Ces outils permettent de prototyper une interface en la construisant sans faire appel à du codage, soit sans nécessairement devoir développer la couche d'abstraction et/ou le noyau fonctionnel. Des outils hypermédia (comme HyperCard), des outils de présentation assistée par ordinateur (comme Microsoft PowerPoint, Aldus Persuasion), des outils auteur pour des applications multimédia (comme Macromedia Director) ont déjà été considérés bien des fois. Effectivement, il est possible de produire une interface à niveau de fidélité élevée avec un minimum de navigation : le prototypage horizontal est mieux supporté et le prototypage vertical ou diagonal, partiellement, les prototypes présentationnel et navigationnel sont mieux supportés dans cette approche. Hélas, l'effort de production est **perdu** lorsqu'on passe à l'environnement de développement intégré prévu pour l'application complète. Même si un tel outil génère automatiquement du code, en tout ou en partie, ce code généré ne sera pas récupérable pour l'interface finale. Le développeur recommence le développement à zéro, avec une interface à haute fidélité.

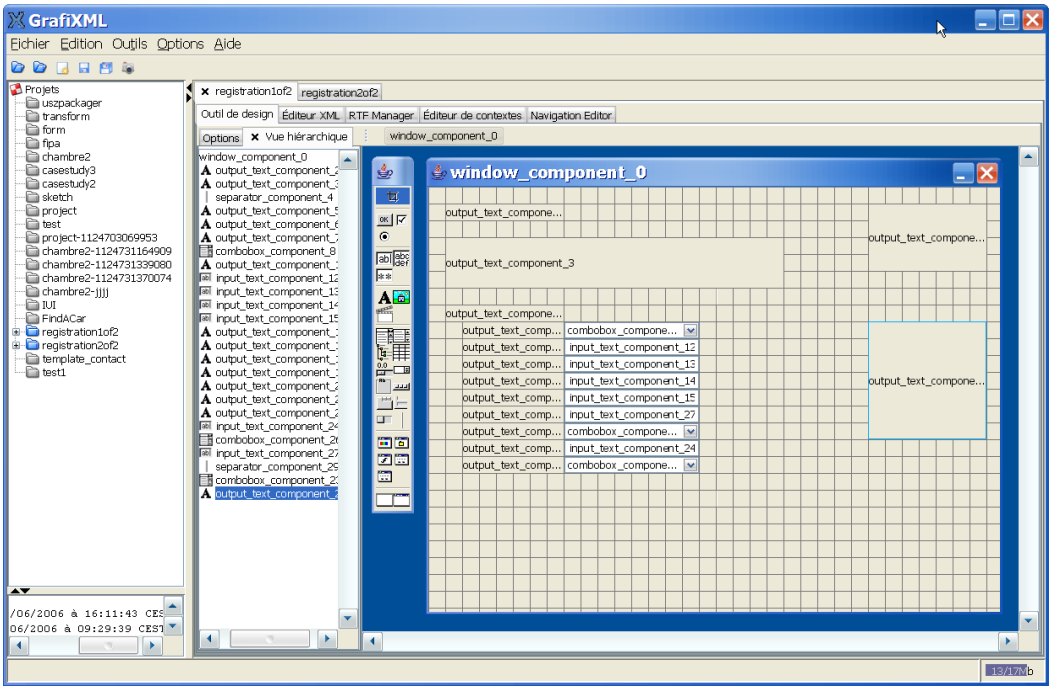

***Figure 5**:* Interface de GrafiXML en mode de composition de la présentation (prototypage présentationnel).

Pour répondre à ces défis, nous avons développé GrafiXML (figure 5), un éditeur d'interface à niveau de fidélité élevée fondé sur le langage UsiXML. A l'instar des éditeurs d'interface, GrafiXML permet de représenter une interface graphique en positionnant les objets interactifs souhaités, que ce soit des objets standards (p. ex. un bouton radio, une liste de sélection) ou des objets ajoutés (p. ex. la saisie d'une heure, un calendrier, un conteneur). Au lieu de générer automatiquement ou semi-automatiquement le code de l'interface ainsi représentée, GrafiXML produit automatiquement des spécifications fonctionnelles et opérationnelles rédigées en UsiXML. Ces spécifications peuvent être dépendantes ou indépendantes d'un contexte d'utilisation. Il est possible aussi de décliner différentes interfaces possibles pour plusieurs contextes d'utilisation associés au même projet. Ceci est particulièrement approprié lorsqu'il faut développer une version multi-plate-forme d'une interface. La figure 5 présente l'outil en fidélité élevée au moment de la spécification de la présentation. Etant donné que cette interface peut être multi-plate-forme, seule une représentation logique est affichée. Pour obtenir en temps réel une interface réelle, il suffit de demander la prévisualisation de l'interface (qui ne fonctionne qu'en Java Swing). Grâce à un système d'export, GrafiXML peut actuellement générer automatiquement le code de l'interface dans plusieurs langages cibles, tels que Java, XHTML ou XUL (http:// www.mozilla.org/ projects/xul/). Par exemple, la figure 6 correspond à l'interface générée en XHTML à partir de celle spécifiée à la figure 5, mais sans feuille de style associée. Il est possible d'adjoindre une feuille de style ultérieurement afin de rendre la présentation adaptée au contexte d'utilisation en fonction du travail du designer graphique. Cette opération est réalisée extérieurement à l'environnement de GrafiXML. Logiquement, GrafiXML peut simultanément couvrir plusieurs langues pour une même interface en spécifiant les ressources qui varient en fonction de l'utilisateur (p. ex. les ressources textuelles, les images, les libellés, le texte de contenu). Dans ce cas, il ne faut pas dessiner plusieurs interfaces – une seule suffit –, mais bien spécifier les variations de contenu propres à chacune des versions dans les langues différentes. Une autre différence significative de GrafiXML par rapport à d'autres éditeurs concerne sa capacité à accueillir des plug-ins destinés à couvrir des ensembles étendus de fonctionnalités. Par exemple, des plug-ins existent pour transformer une interface existante pour un PDA, pour (dé)composer des interfaces entre elles, pour évaluer leur qualité ergonomique en cours de conception, etc.

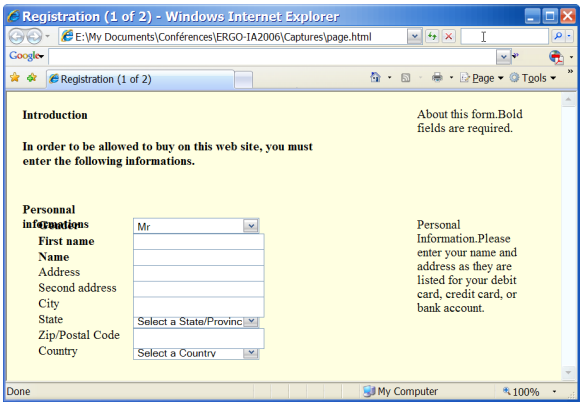

***Figure 6**:* Interface finale résultant de GrafiXML.

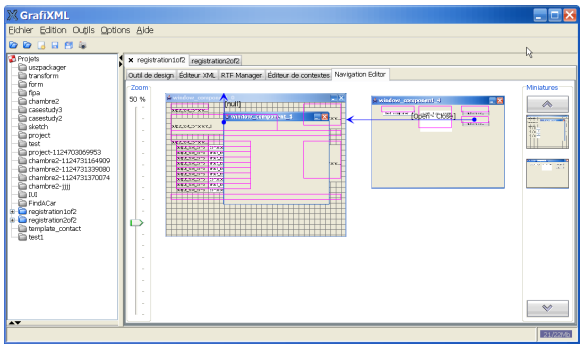

***Figure 7**:* Interface de GrafiXML en mode de spécification de la navigation (prototypage navigationnel).

Enfin, pour supporter le prototypage navigationnel, GrafiXML offre un mode de spécification de la navigation (figure 7). Dans ce dernier, chaque conteneur (cela peut être une page web, une boîte de regroupement, une fenêtre, une boîte de dialogue) est affiché par sa miniature et les miniatures du projet en cours sont rassemblées sur la partie droite. Il est alors possible de tirer des flèches d'un objet repris dans un conteneur vers un autre conteneur afin de spécifier des navigations globales élémentaires, du genre « fermer la première fenêtre et ouvrir la suivante », « désactiver la première fenêtre, la réduire sous forme iconique et activer la seconde fenêtre ». Le siège de cette navigation globale (entre conteneurs) peut résider dans le chef d'un objet interactif en particulier (p. ex. un bouton de commande) soit dans le chef du conteneur lui-même, signifiant par là un comportement attaché à ce conteneur. Ici aussi, l'outil génère les spécifications détaillées en UsiXML V1.6.4 correspondant au comportement souhaité, tant du point de vue de la présentation et de la navigation, que du contenu et du layout (actuellement, quatre types de layout sont disponibles pour la plupart des situations rencontrées). Actuellement, GrafiXML ne supporte que la navigation globale. Nous sommes en train de développer la partie relative à la spécification de la navigation locale. Ce qui est bien plus complexe étant donné la variation et la complexité des navigations possibles (p. ex. si je sélectionne telle langue dans la liste de sélection, alors activer la valeur X dans le radio bouton et activer le bouton de commande du conteneur). Il est ardu de définir une notation graphique permettant d'exprimer un ensemble significatif de ces comportements. Aussi, nous espérons nous limiter à une stylistique simple.

Remarquons enfin que GrafiXML ne dispose pas actuellement de la capacité à incorporer un objet interactif non prédéfini dans UsiXML. Actuellement, il y a une trentaine d'objets, mais cet ensemble peut toujours évoluer en importance. Par conséquent, GrafiXML souffre de la même lacune que les éditeurs concernant les objets métier. Nous verrons cependant qu'une solution partielle à ce problème a été instaurée dans le prototypage à un niveau de fidélité basse.

PROTOTYPAGE À UN NIVEAU DE FIDÉLITÉ MODÉRÉE

Tant qu'il s'agit de représenter graphiquement la présentation et/ou la navigation dans GrafiXML, le coût et le temps de production semblent rester au moins égal à ceux encourus dans un éditeur d'interface courant, si ce n'est la capacité d'expression plus importante. Ceci mis à part, les spécifications rédigées en UsiXML produites par cet outil (ou par d'autres d'ailleurs) restent toujours disponibles et permettent en principe de ne pas perdre l'effort de prototypage lorsqu'il s'agit de passer à la phase de développement. Si on peut récupérer les spécifications afin de produire du code (capacité générative), un tel code peut être produit par diverses techniques de génération (p. ex. par squelette, par analyse statique de code, par « generative programming »).

Pour ce qui est des propriétés avancées de la présentation et de la navigation qui ne font pas l'objet d'une spécification graphique dans GrafiXML, le concepteur a la possibilité de les spécifier dans des feuilles de propriétés ou de laisser le système générer automatiquement des valeurs par défaut pour lui (qui sont réglables et paramétrables). Cette approche convient lorsque l'on connaît bien l'interface finale ou tout au moins que l'on est proche de son résultat. Si on n'est pas trop sûr de la mouture finale de cette interface, on peut vite regretter que le coût et le temps de création deviennent prohibitifs face au retour sur investissement. C'est pourquoi il peut s'avérer judicieux de passer à un niveau de fidélité modérée.

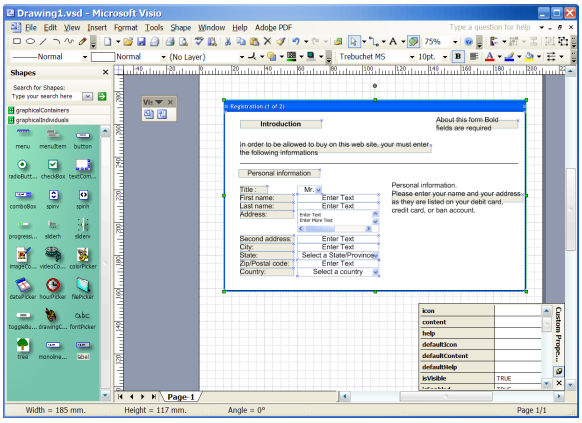

***Figure 8**:* Interface de VisiXML en mode de spécification de la présentation (prototypage présentationnel).

Dans ce but, nous avons développé VisiXML (figure 8), un éditeur d'interface à niveau de fidélité modérée permettant de se restreindre aux aspects visuels uniquement sans passer par de la programmation ou du codage supplémentaire. VisiXML se présente sous la forme d'un plug-in développé au sein de l'environnement Microsoft Visio Pro agrémenté d'un stencil comportant les icônes de dessin de tous les conteneurs et objets interactifs prévus dans une interface concrète spécifiable par UsiXML

(voir partie gauche de la figure 8). Dans VisiXML, le concepteur se résume à dessiner l'interface souhaitée en sélectionnant les conteneurs et objets interactifs individuels souhaités. La différence fondamentale entre GrafiXML (fidélité élevée) et VisiXML (fidélité modérée) réside dans le fait qu'ici, il ne s'agit que d'un dessin vectoriel, en principe plus facilement modifiable, mais dont la présentation est liée à celle d'une plate-forme logicielle cible. Au contraire, dans GrafiXML, l'édition n'est pas liée à un environnement particulier.

Comme VisiXML est incorporé à l'environnement Microsoft Visio, on bénéficie de tout ce qui est supporté par cet environnement : on peut dessiner n'importe quel autre objet vectoriel de base tel que de la décoration, du texte, des figures, des dessins. Ceci permet enfin de spécifier des éléments de présentation qui ne seraient pas définis en standard dans UsiXML, mais ces dessins sont perdus dans l'export : quand le concepteur pense avoir terminé sa conception, il peut cliquer sur une première icône qui détermine automatiquement une hiérarchie des objets présentés, puis sur une autre icône permettant de générer automatiquement les spécifications en UsiXML. Lorsqu'il s'agit d'autres éléments de dessin prévus par Visio, mais pas par VisiXML, ces éléments sont sauvegardés dans le prototype de fidélité modérée, mais perdus dans les spécifications générées. Ceci engendre une incohérence latente entre la représentation externe (l'interface dessinée) et la représentation conceptuelle (l'interface spécifiée en UsiXML). En contrepartie, la facilité d'édition est plus importante, permettant de modifier plus rapidement et moins en détails l'interface en cours de prototypage. En effet, le prototypage est restreint aux seules propriétés physiques et pour la plupart des propriétés physiques ayant un impact visuel, des valeurs par défaut sont spécifiées afin de minimiser l'apport du concepteur. Si néanmoins, le concepteur souhaite spécifier des paramètres jugés fort fins pour ce niveau (p. ex. la police de caractères d'un texte, sa taille, sa couleur), il peut le faire via la feuille de propriétés située en bas à droite de la figure 8. Ces propriétés sont retenues dans la génération des spécifications en UsiXML. Là où la capacité de VisiXML s'arrête, c'est lorsqu'il s'agit de passer à une autre plate-forme (ici, seul Microsoft Windows XP est supporté) ou de prototyper à des fins d'exploration de conceptions alternatives. Certes, VisiXML comporte la faculté de dessin, mais sa représentation liée à une plate-forme ne permet pas de faire penser qu'il s'agit d'une interface en pleine évolution : son niveau formel [14] de détails procure l'impression qu'il s'agit déjà d'une interface presque finale, en tout cas dans les yeux de l'utilisateur final.

En conclusion, VisiXML, même s'il est restreint à du dessin vectoriel sur une plate-forme donnée, permet de supporter le prototypage présentationnel et navigationnel global. Le prototypage navigationnel local n'est aucunement supporté.

Dans la même veine, Berger [4] a imaginé un outil de prototypage d'interface à un niveau de fidélité modérée au sein de l'environnement Microsoft Excel avec la particularité que des comportements minimaux sur les données peuvent être effectués grâce au langage de macro-commande d'Excel.

PROTOTYPAGE À UN NIVEAU DE FIDÉLITÉ BASSE

Pour ne plus véhiculer l'impression d'une interface dont la représentation est quasi finale, même si sa conception l'est, il est judicieux de passer à un niveau de fidélité basse. Lorsqu'on se situe à un niveau de fidélité basse, la représentation de l'interface que l'on manipule doit être la plus naturelle possible pour ne pas entraver le processus de création, de conception exploratoire. La représentation graphique de l'interface manipulée au sein de l'éditeur est donc fondamentale pour donner l'impression à l'utilisateur qu'il s'agit d'une interface en cours de conception et non d'une interface finale. Le caractère informel [14] des spécifications est à préserver.

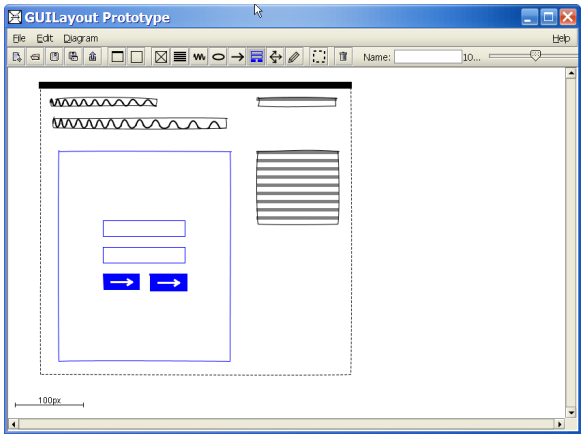

***Figure 9**:* Interface de GUILayout en mode de spécification de la présentation (prototypage présentationnel).

Au bas de l'échelle des représentations graphiques se trouve GUILayout [4] dont l'interface est reproduite à la figure 9. Nous y voyons que le concepteur peut définir des zones contenant des types d'information différents : texte, graphique, vidéo, formulaire. Ces zones informationnelles de base peuvent être récursivement composées au sein de conteneurs de niveau immédiatement supérieur. Pour représenter l'interface finale de la figure 6, le concepteur a ici dessiné un conteneur reprenant les zones de texte (destinées à contenir les informations de guidage) et la zone de formulaire. La représentation graphique manipulée par GUILayout est celle du niveau de fidélité le plus bas que nous connaissons aujourd'hui. Pour donner un aperçu avant développement d'une interface, cette représentation suffit. Mais dès qu'il s'agit de préciser le contenu des zones dessinées, l'utilité de l'outil s'interrompt. Nous jugeons que cette représentation est de niveau trop bas pour être exploitée de manière continue dans la suite du cycle de vie de développement.

A un stade moins bas que GUILayout résident les travaux de Meyer [21] avec son outil SketchaPad [22]. Meyer estime pour sa part que même une représentation graphique de l'interface finale qui serait indépendante de la plate-forme (p. ex. la partie gauche de la figure 10) reste insuffisante et risque tout de même de donner l'impression que la conception de l'interface est suffisamment aboutie, précise pour donner l'impression à l'utilisateur final que l'interface est déjà obtenue, finalisée. C'est pourquoi il a développé un algorithme graphique permettant d'affecter automatiquement des contours flous aux représentations graphiques des objets interactifs représentés (la partie droite de la figure 10).

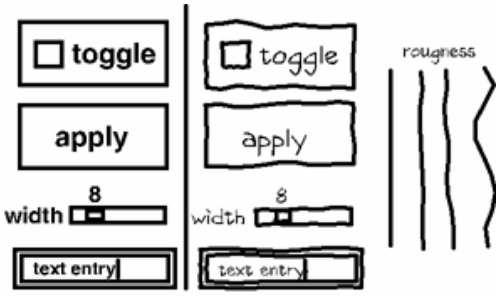

***Figure 10**:* Deux représentations graphique d'une interface en prototypage à niveau de fidélité basse [21].

A un niveau plus élevé qu'EtchaPad réside une famille d'outils d'esquisse d'interfaces manipulant une représentation dessinée (« esquisse », en anglais, *sketching*) de l'interface en cours de prototypage. Les représentants les plus caractéristiques de cette famille sont dans l'ordre alphabétique : Denim [14], EtchaPad [22], FreeForms [24, 25], JavaSketchit [8] basé sur la bibliothèque de reconnaissance de forme CALI [13], SILK [16,17], SketchRead [1]. Ces outils partagent un mode de fonctionnement presque commun : au lieu de manipuler une représentation graphique de l'interface, proche ou éloignée comme dans [21], ils permettent au concepteur de dessiner librement l'interface en cours de conception comme dans un outil de dessin graphique libre. Prototyper une interface à un niveau de fidélité basse permet de découvrir autant de problèmes qu'à un niveau plus élevé.

Premièrement, un utilisateur final possède autant la capacité de dessiner une interface qu'un concepteur ou qu'un développeur : dans une étude préliminaire [11], nous n'avons pas décelé de différence statistiquement significative entre la capacité d'un utilisateur final et celle d'un concepteur d'interface pour dessiner une interface. Voici donc un moyen d'inclure vraiment l'utilisateur final dans le prototypage puisqu'il peut lui-même dessiner aussi bien que son concepteur. Il est intéressant d'étudier le binôme (concepteur, utilisateur final) en situation collaborative de conception, mais ceci requiert une autre infrastructure qui est celle d'un environnement collaboratif de conception (figure 11).

***Figure 11**:* Un environnement collaboratif de conception basé sur plusieurs niveaux de fidélité concurrents (source : http://www.isys.ucl.ac.be/bchi/members/jgo/Research.html)

Dans cet environnement, un ou plusieurs niveaux de fidélité peuvent se côtoyer : *mono-fidélité* si un seul des trois niveaux est mobilisé, *multi-fidélité* si plus d'un niveau de fidélité est mobilisé. On dira que la multi-fidélité est *mixte* (comme dans [23]) lorsqu'une représentation mélange plusieurs niveaux de fidélité simultanément ou *distribuée* lorsque plusieurs représentations ayant chacune une fidélité donnée se côtoient.

Deuxièmement, ce n'est pas parce que le niveau de fidélité du prototypage s'exprime à un niveau inférieur à celui de la réalité (autrement dit, à un niveau de fidélité moyenne ou basse) que la capacité du prototype à révéler ses avantages et ses inconvénients diminue. En particulier, il a été montré que le nombre de problèmes ergonomiques identifiés à l'aide d'une évaluation heuristique n'est pas moindre avec une fidélité basse qu'avec une fidélité élevée.

Troisièmement, plusieurs principes fondamentaux président au prototypage à un niveau de fidélité basse :

- **Principe de naturalité** : il faut que l'interface en cours de dessin soit la plus naturelle possible et que son dessin soit le plus naturel possible pour ne pas limiter les capacités exploratrices de son utilisateur, même si sa représentation n'est pas immédiatement similaire à celle de l'interface finale.
- **Principe de non-obstrusion** : comme corollaire au principe de naturalité, il faut que le système de support à l'esquisse soit le moins obstrusif possible et perturbe le moins possible le concepteur durant cette phase de prototypage. La représentation en fidélité basse ne doit donc en aucun cas introduire des tâches ou des actions qui soient étrangères à la nature originale de l'activité de prototypage.
- **Principe de continuité** : il faut que le système de support à l'esquisse supporte continûment le dessin quelle que soit la nature de l'élément sujet au prototypage (p. ex. un objet interactif, du texte, du dessin, du contenu multimédia). L'utilisateur ne devrait pas changer de mode de dessin si un élément de nature différente doit être représenté.

- **Principe de récupération** : l'effort obtenu suite à l'esquisse ne doit pas être perdu pour le reste du cycle de vie de développement de l'application interactive. En principe, pour minimiser les coûts, l'effort consenti durant ce prototypage, même s'il est de niveau de fidélité basse, devrait être perdu le moins possible ou récupéré le plus possible dans la suite.

L'ensemble de ces outils adhère plus ou moins aux trois premiers principes dans une certaine mesure, mais généralement pas au dernier. C'est pourquoi, nous avons développé SketchiXML [9,10,11], un éditeur d'interface à niveau de fidélité basse fondé sur le langage UsiXML, dont l'interface est reproduite à la figure 12.

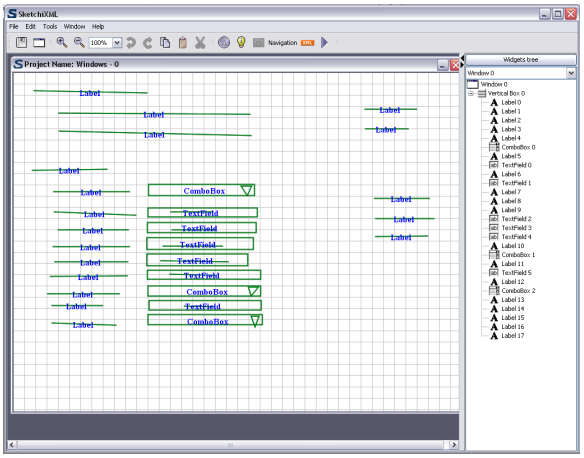

***Figure 12**:* Interface de GUILayout en mode de spécification de la présentation (prototypage présentationnel).

Dans cet outil, le concepteur peut spécifier ou non un profil exprimant un contexte d'utilisation cible (c'est-à-dire pour un utilisateur, une plate-forme et un environnement donné). Ensuite, il peut dessiner en dessin libre l'interface en cours de prototypage. Bien que SketchiXML puisse être aussi utilisé sur un ordinateur normal équipé d'un clavier et d'une souris, il trouve sa pleine utilisation sur des plateformes de type TabletPC ou Wacom Cintiq (cfr. http://www.wacom.com/lcdtablets/index .cfm) qui offrent à son utilisateur la capacité d'écrire et de dessiner directement sur l'écran, minimisant ainsi l'écart entre le lieu d'interaction et sa représentation. A tout moment, le concepteur peut demander au système de reconnaître le contenu de l'esquisse pour générer aussi automatiquement que possible des spécifications de base d'une interface en UsiXML. Pour ce faire, chaque objet interactif de base (p. ex. une liste de sélection) a été encodé dans une grammaire graphique sous la forme de trois représentations préférentielles. Ces trois représentations sont issues de tests utilisateur qui ont permis de déterminer quelle sont les trois représentations graphiques préférées des personnes (concepteurs et non-concepteurs) pour chaque objet interactif. Pour les autres, seule l'esquisse demeure et ne donne pas lieu à de la reconnaissance.

SketchiXML tente donc d'adhérer aux quatre principes fondamentaux introduits ci-avant de la façon suivante :

- **Principe de naturalité** : le dessin et l'écriture du concepteur forment les seules entrées possibles de base pour réaliser une esquisse, l'objectif étant de maximiser la métaphore de l'esquisse papier. Ces deux formes d'expression sont largement reconnues comme supportant largement un processus de conception, hautement créatif par nature [25,27].
- **Principe de non-obstrusion** : si le concepteur le préfère, aucun traitement de l'esquisse n'est effectué en cours de conception. En particulier, aucune reconnaissance des objets interactifs et de l'écriture ne sont opérées. Celles-ci ne sont déclenchées qu'à la demande du concepteur afin de préserver le contrôle explicite et de ne pas perturber le concepteur. La figure 12 montre que certaines formes ont été reconnues et enrichis par le nom de l'objet correspondant.
- **Principe de continuité** : tant que le concepteur est en train d'effectuer une esquisse, il reste dans ce mode. Il ne doit pas passer d'un mode à l'autre pour exprimer tantôt un objet interactif, tantôt du contenu, tantôt du texte, tantôt une commande de manipulation de l'esquisse (par exemple, effacer un fragment de l'esquisse, déplacer un fragment). A cette fin, SketchiXML est équipé d'un moteur de reconnaissance de formes basé sur une grammaire graphique existant sous la forme de la bibliothèque CALI [13], d'un moteur de reconnaissance de gestes basé sur le traitement du signal émis par le stylet et d'une bibliothèque de reconnaissance d'écriture.
- **Principe de récupération** : contrairement à Denim [14] qui ne supporte aucune reconnaissance et plus loin que FreeForms [24] qui reconnaît moins d'objets, SketchiXML reconnaît environ une trentaine d'objets interactifs, soit tous ceux qui sont définis dans l'interface concrète en UsiXML [19]. Il peut donc exporter le résultat de l'esquisse dans un fichier UsiXML qui est alors récupéré dans l'outil GrafiXML (figures 5, 7) afin d'être affiné jusqu'au niveau de détails demandé.

Au terme de ce parcours d'outils, la figure 1 peut être revisitée à la lumière des avancées effectuées et permet de couvrir à présent les trajectoires de la figure 13.

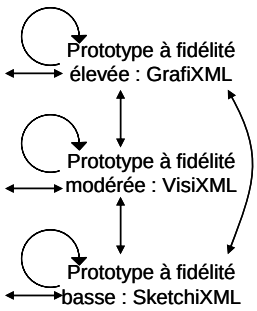

***Figure 13**:* Trajectoires possibles de prototypages en UsiXML.

PROTOTYPAGE AVEC D'AUTRES MODALITES

Les outils analysés jusqu'ici se basent évidemment tous sur une même hypothèse de travail : l'interface de l'application à prototyper est de modalité graphique dominante. La question concernant l'adaptation de ces réflexions à d'autres modalités d'interaction ou à des types particuliers d'applications interactive est une question en cours de recherche. Par exemple, SUEDE [15] permet de prototyper rapidement une application vocale en manipulant une abstraction de questions et réponses. La question du niveau de fidélité se pose ici en d'autres termes et n'a pas encore fait l'objet de réflexion. De même, MultiXML [29] permet de prototyper relativement rapidement une interface multimodale (graphique, vocale, tactile ou combinant celles-ci) pour le web, mais la question du niveau de fidélité n'est pas encore résolue non plus. Enfin, Topiary [18] exploite l'expérience acquise dans le développement d'applications sensibles au contexte pour proposer des représentations de natures différentes utiles au prototypage de telles applications.

CONCLUSION

Au terme de ce parcours, nous avons montré qu'à l'aide de la trilogie GrafiXML-VisiXML-SketchiXML, il est possible de supporter différents niveaux de fidélité du prototypage d'une interface graphique (élevée, modérée, basse) et qu'il est possible de passer de n'importe quel niveau de fidélité à un autre en principe. VisiXML ne supporte que l'interface graphique pour une plate-forme de type MS Windows au contraire des deux autres outils. Les trajectoires de prototypage de la figure 13 sont couvertes. En comparant les différents outils de support au prototypage, nous avons pu identifier quels sont ceux qui conviennent pour un prototypage horizontal, vertical ou diagonal, présentationnel, navigationnel global ou local. L'expérience acquise jusqu'ici semble révéler que le prototypage à un niveau de fidélité basse est promis à un avenir certain tant la naturalité est importante : le concepteur, l'utilisateur final lui-même ou bien les deux ensembles peuvent pour la première fois collaborer directement au prototype en cours de conception. Gageons que ces nouvelles formes de prototypage procureront à l'utilisateur final un réel sentiment d'implication.

REMERCIEMENTS

Les auteurs remercient particulièrement le projet ReQuest (Rapid prototyping of e-commerce applications, Programme «WIST» Wallonie Information Science & Technology, Convention n°315592), financé par la Région Wallonne (Belgique), ainsi que le réseau d'excellence SIMILAR (The European research taskforce creating human-machine interfaces SIMILAR to human-human communication, http://www.similar.cc, FP6-IST1-2003-507609), financé dans le cadre du 6ième Programme Cadre de la Commission Européenne. Ensuite, nous remercions WACOM Research Europe. Nous remercions aussi chaleureusement A. Caetano, N. Goulart, M. Fonseca, et J.A. Jorge (INESC-ID, Portugal) pour nous avoir permis d'utiliser leur bibliothèque CALI.

BIBLIOGRAPHIE

1. Alvarado, Ch. and Randall, D. SketchREAD: A Multi-domain Sketch Recognition Engine. In *Proc. of 17th Annual ACM Symposium on User Interface Software and Technology UIST'2004* (Santa Fe, October 24-27, 2004), ACM Press, New York, 2004, pp. 23-32.
2. Baümer, D., Bischofberger, W.R., Lichter, H., and Züllighoven, H. User Interface Prototyping - Concepts, Tools, and Experience. In *Proc. of 18th Int. Conf. on Software Engineering ICSE'96* (Berlin, March 25-29, 1996). IEEE Computer Society, Washington, 1996, pp. 532-541.
3. Berger, N. *The Excel Story*. Interactions, Vol. 13, No. 1, January-February 2006, pp. 14-17.
4. Berkun, S. *The Art of User Interface Prototyping*, November 2000, accessible at http://www.scottberkun.com/essays/essay12.htm
5. Blankenhorn, K. *A UML Profile for GUI Layout*, Master thesis, University of Applied Sciences Furtwangen, Furtwangen, 2004.
6. Boar, B.H. *Application prototyping: a requirements definition strategy for the 80s*. John Wiley & Sons, New York, 1984.
7. Buxton, W. Sketching and Experience Design. In *Proc. of 10th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2005* (Rome, 12-16 septembre 2005), M.-F. Costabile, F. Paternò (eds.), Lecture Notes in Computer Science, Vol. 3585, Springer-Verlag, Berlin, 2005, p. 1.
8. Caetano, A., Goulart, N., Fonseca, M. and Jorge, J. JavaSketchIt: Issues in Sketching the Look of User Interfaces. In *Proc. of the 2002 AAAI Spring Symposium on Sketch Understanding* (Palo Alto, March 2002), AAAI Press, Menlo Park, 2002, pp. 9-14.
9. Coyette, A., Faulkner, S., Kolp, M., Limbourg, Q., and Vanderdonckt, J. SketchiXML: Towards a Multi-Agent Design Tool for Sketching User Interfaces Based on UsiXML. In *Proc. of 3rd Int. Work-shop on Task Models and Diagrams for user interface design TAMODIA'2004* (Prague, November 15-16, 2004), Ph. Palanque, P. Slavik, M. Winckler (eds.), ACM Press, New York, 2004, pp. 75-82.
10. Coyette, A., Vanderdonckt, J., Faulkner, S., and Kolp, M. Generating Abstract User Interfaces from an Informal Design. In *Proc. of 17th Int. Conf. on Software Engineering and Knowledge Engineering SEKE'2005* (Taipei, July 14-16, 2005), IJSEKE Press, Taiwan, 2005.
11. Coyette, A. and Vanderdonckt, J. A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces. In *Proc. of 10th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2005* (Rome, 12-16 septembre 2005), M.-F.

Costabile, F. Paternò (eds.), Lecture Notes in Computer Science, Vol. 3585, Springer-Verlag, Berlin, 2005, pp. 550-564.

12. Engelberg, D. and Seffah, A. A Framework for Rapid Mid-Fidelity Prototyping of Web Sites. In *Proc. of the IFIP 17th World Computer Congress - TC13 Stream on Usability: Gaining a Competitive Edge WC'2002* (August 25-29, 2002), Kluwer Academic Press, Dordrecht, 2002, pp. 203-215.
13. Fonseca, M.J., Pimentel, C. and Jorge, J.A. CALI: An Online Scribble Recognizer for Calligraphic Interfaces. In *Proc. of the 2002 AAAI Spring Symposium on Sketch Understanding* (Palo Alto, March 2002), AAAI Press, Menlo Park, 2002, pp. 51-58.
14. Hong, J.I., Li, F.C., Lin, J., and Landay, J.A. End-User Perceptions of Formal and Informal Representations of Web Sites. In *Extended Abstracts of Proc. of ACM Conf. on Human Factors in Computing Systems CHI'2001* (Seattle, March 31-April 5, 2001). ACM Press, New York, 2001, pp. 385-386.
15. Klemmer, S.R., Sinha, A.K., Chen, J., Landay, J.A., Aboobaker, N., and Wang, A. Suede: a Wizard of Oz Prototyping Tool for Speech User Interfaces. In *Proc. of the 13th Annual ACM Symposium on User Interface Software and Technology UIST'2000* (San Diego, November 6-8, 2000), ACM Press, new York, 2000, pp. 1-10.
16. Landay, J.A. and Myers B.A. Interactive Sketching for the Early Stages of User Interface Design. In *Proc. of ACM Conf. on Human Factors in Computing Systems CHI'95* (Denver, May 7-11, 1995), ACM Press, New York, 1995, pp. 43-50.
17. Landay, J. and Myers, B.A. *Sketching Interfaces: Toward More Human Interface Design*. IEEE Computer, Vol. 34, No. 3, March 2001, pp. 56-64.
18. Li, Y., Hong, J.I., and Landay, J.A. Topiary: a Tool for Prototyping Location-enhanced Applications. In *Proc. of the 17th Annual ACM Symposium on User Interface Software and Technology UIST'2004* (Santa Fe, October 24-27, 2004), ACM Press, New York, 2004, pp. 217-226.
19. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Lopez, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proc. of 9th IFIP Working Conf. on Engineering for HCI jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004* (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 200-220.
20. Limbourg, Q. and Vanderdonckt, J. UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. In Matera, M., Comai, S. (Eds.), *Engineering Advanced Web Applications*, Rinton Press, Paramus, 2004, pp. 325-338.
21. Meyer, J. *Creating Informal Looking Interfaces*, 2005, accessible at http://www.cybergrain.com/tech/pubs/lines_technote.html
22. Meyer, J. EtchaPad – Disposable Sketch Based Interfaces. In *Proc. of ACM Conf. on Human Factors in Computing Systems CHI'96* (Vancouver, April 13-18, 1996), Conference Companion, ACM Press, New York, 1996, pp. 195-198.
23. Petrie, J.N. and Schneider, K.A. Mixed-Fidelity Prototyping of User Interfaces. In *Proc. of 13th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2006* (Dublin, 26-28 July 2006), G. Doherty and A. Blandford (eds.), Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2006.
24. Plimmer, B.E. and Apperley, M. Software for Students to Sketch Interface Designs. In *Proc. of 9th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2003* (Zurich, 1-5 September 2003), M. Rauterberg, M. Menozzi, J. Wesson (eds.), IOS Press, Amsterdam, 2003, pp. 73-80.
25. Plimmer B., and Apperley M. Interacting with Sketched Interface Designs: An Evaluation Study. In *Proc. of ACM Conf. on Human Factors in Computing Systems CHI'2004* (Vienna, April 2004), ACM Press, New York, 2004, pp. 1337-1340.
26. Rudd, J., Stern, K., and Isensee, S. *Low vs. High-Fidelity Prototyping Debate*. Interactions, Vol. 3, No. 1, January 1996, pp. 76-85.
27. Snyder, C. *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces*. Series in Interactive Technologies, Morgan Kaufmann, 2002.
28. Souchon, N. and Vanderdonckt, J., A Review of XML-Compliant User Interface Description Languages. In *Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003* (Madeira, June 4-6, 2003), J. Jorge, N.J. Nunes, J. Cunha (eds.), Lecture Notes in Computer Science, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 377-391.
29. Stanciulescu, A. and Vanderdonckt, J. Design Options for Multimodal Web Applications. In *Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2006* (Bucharest, 6-8 June 2006), Springer-Verlag, Berlin, 2006, pp. 41-56.
30. Vanderdonckt, J., A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In *Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05* (Porto, 13-17 juin 2005), O. Pastor & J. Falcão e Cunha (eds.), Lecture Notes in Computer Science, Vol. 3520, Springer-Verlag, Berlin, 2005, pp. 16-31.