Modèle, méthodes et outils de support au prototypage multi-fidélité des interfaces graphiques

Model, Methods, and Tools for Supporting Multi-fidelity Prototyping of Graphical User Interfaces

Jean VANDERDONCKT, Adrien COYETTE

Belgian Laboratory of Computer-Human Interaction (BCHI)
Information Systems Unit (ISYS)
Louvain School of Management (LSM)
Université catholique de Louvain (UCL)
{Jean.Vanderdonckt, Adrien.Coyette}@uclouvain.be
http://www.isys.ucl.ac.be/bchi

Résumé. Le prototypage à fidélité mixte consiste à mélanger, simultanément ou non, différents niveaux de fidélité au sein d'un même prototype d'interface homme-machine. Le prototypage multi-fidélité est un prototypage à fidélité mixte en permettant au concepteur de fournir n'importe quel objet interactif en plusieurs niveaux de fidélité et d'assurer une transition dynamique et souple entre ces niveaux en cours de conception. L'approche par multi-fidélité permet d'intégrer des éléments d'interface livrés en un ou plusieurs niveaux de fidélité, mais surtout de constituer un prototype de l'interface cohérent par rapport au cycle de vie de développement de l'application interactive. L'article présente les modèles sous-jacents à un prototypage multi-fidélité. Les méthodes de conception mettant en jeu ces modèles sont esquissées et différents outils de support à ces méthodes viennent illustrer leur application.
Mots-clés. Approche dirigée par les modèles, conception assistée par ordinateur, langage de description d'interface, niveau de fidélité, outil d'esquisse, prototypage, prototypage multi-fidélité, USer Interface eXtensible Markup Language.

Abstract. Mixed-fidelity prototyping consists of mixing simultaneously or not various fidelity levels within the same user interface prototype. Multi-fidelity prototpying incorporates mixed-fidelity prototyping in that it enables the designer to express any interaction object in multiple fidelity levels and to ensure a smooth and dynamic transition between these levels at design-time. The multi-fidelity approach allows integrating user interface elements delivered in one or many fidelity levels in order to build a user interface prototype which evolves as the interactive application development life cycle is progressing. This paper presents models underpinning multi-fidelity prototyping. Design methods relying on these modes are defined and various software tools supporting these methods will examplify their application.
Keywords. Model-driven engineering, computer-aided design, user interface description language, fidelity level, sketching tool, prototyping, multi-fidelity prototyping, USer Interface eXtensible Markup Language.

1 Introduction

Le prototypage rapide d'application est aujourd'hui largement reconnu comme étant une activité sinon indispensable à la qualité du système futur, au moins hautement souhaitable. Si le prototypage couvre en principe l'application complète (Boar, 1984), l'Interface Homme-Machine (IHM) de cette application constitue un composant de premier choix pour engager le plus rapidement possible un prototypage (Rettig, 1994, Baümer *et al.*, 1996, Berkun, 2000) : l'IHM constitue la partie émergée de l'iceberg qu'il est possible de montrer à l'utilisateur final en vue de récolter ses commentaires, des propositions d'amélioration, mais aussi une tentative de validation du prototype courant. En général, l'utilisateur final peine à valider des spécifications, fonctionnelles ou non-fonctionnelles, de son futur système sur base de constructions qui ne sont pas exprimées dans son propre langage. Par contre, il est impressionnant de constater combien ce même utilisateur final, non nécessairement formé aux techniques de prototypage et encore moins formé aux techniques informatiques, est capable de réagir sur un prototype d'interface. Et surtout de réagir en fournissant un retour d'information permettant de faire évoluer effectivement le prototype courant vers l'interface finale, celle qui sera livrée.

Sumner *et al.* (1997) observent que le processus de prototypage rapide de l'interface, si on le structure en cycles de trois phases conception-prototypage-évaluation, se manifeste par trois propriétés :

1. **l'ouverture** : à tout moment dans le cycle, de nouvelles spécifications peuvent apparaître.
2. **l'itération** : il n'est pas possible de connaître *a priori* le nombre de cycles nécessaires pour obtenir un prototype de qualité suffisante pour passer à son développement.
3. **l'incomplétude** : un prototype reste par nature incomplet dans la mesure où il ne peut pas dégager toutes les spécifications, qu'elles soient initiales ou découvertes en cours de route. Cette incomplétude est intrinsèque.

Si les propriétés d'ouverture et d'itération semblent évidentes d'emblée de jeu, la propriété d'incomplétude demeure un problème permanent tant on ignore aujourd'hui comment minimiser cette incomplétude. Ce problème se manifeste notamment lorsqu'il s'agit d'identifier et de mettre en jeu les moyens nécessaire au prototypage et garantir son retour sur investissement. En effet, les moyens varient largement en forme (p. ex., cela peut aller d'un simple dessin à une maquette exécutable), en méthode (p. ex., cela peut suivre une démarche tout à fait libre ou largement structurée), en ressources mobilisées (p. ex., on peut impliquer un ou plusieurs utilisateurs futurs), en durée (p. ex. en termes de cycles conception-prototypage-évaluation). Puisque différents moyens de prototypage existent, il nous paraît opportun de dégager des grandes familles de méthodes de prototypage d'interface en fonction de différents critères. Nous le ferons principalement en examinant les deux premiers axes, à savoir la forme et la méthode, puis en examinant les outils de support appropriés.

On considère généralement que le prototypage rapide de l'interface est une phase du cycle de vie de développement (en anglais, "development life cycle") de cette interface en particulier ou de l'application interactive en général. Soit : "une stratégie spécifique pour conduire l'élicitation des besoins qui sont extraits, présentés, et raffinés sur base d'un modèle de travail de l'application" selon Boar (1984). Pour ce qui est de l'interface en particulier, il s'agit d'aboutir le plus rapidement possible à des spécifications précises conformes aux 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 (Snyder, 2002) : arriver le

plus rapidement en consommant le moins possible de ressources, notamment en évitant de coder directement l'interface, une activité coûteuse.

Pour ce faire, on remplace l'interface à spécifier par une *représentation* ou un *modèle*, sur lequel le processus de prototypage va se fonder (Rettig, 1994). Par conséquent, la représentation que l'on se donne de l'interface en cours de prototypage, ou de son modèle, devient prépondérante. Puisque l'interface est graphique par nature – ou du moins majoritairement de modalité graphique –, il paraît naturel que cette représentation soit aussi graphique. De plus, la nature graphique d'une représentation autorise des traitements multiples – p. ex. l'édition graphique de cette représentation, son interprétation, son amélioration (Alvarado et Randall, 2000) – dont le plus important nous paraît la capacité de dessiner cette représentation de manière la plus naturelle et non-contrainte possible. Pour caractériser cette représentation, la notion de fidélité d'un prototype a fait l'objet de nombreux travaux, notamment (Engelberg et Seffah, 2002, Landay et Myers, 1995, Newman *et al.*, 2003, Petrie et Schneider, 2006, Rudd *et al.*, 1996, Walker *et al.*, 2002). Le niveau de fidélité exprime à première vue la distance entre la représentation manipulée de l'interface prototypée et l'interface elle-même. Cependant, en la regardant de manière plus précise, McCurdy *et al.* (2006) estiment que la notion de fidélité n'est pas unidimensionnelle, mais est fonction de la valeur que l'on peut prendre suivant cinq dimensions à considérer conjointement :

1. le **niveau de la représentation visuelle** : cette dimension exprime la proximité, la ressemblance entre la représentation visuelle du prototype et celle de l'interface finale. Plus la représentation visuelle est proche de celle que l'on va finalement obtenir, plus ce niveau est élevé. Cette dimension est considérée comme fondamentale car elle est souvent assimilée entièrement à la notion de niveau de fidélité tant la représentation peut suggérer ou conditionner ce qu'on peut prototyper avec celle-ci. les dimensions suivantes montrent d'autres aspects à considérer que ce seul niveau de conformité.
2. la **couverture fonctionnelle** : cette dimension exprime la variété des présentations et des comportements que l'on peut prototyper sur la base de la représentation visuelle. Plus on peut prototyper d'aspects différents, plus la largeur fonctionnelle est élevée.
3. la **profondeur des fonctionnalités** : cette dimension exprime la finesse avec laquelle on peut prototyper des aspects de l'interface, à fonctionnalité constante. Plus finement on peut prototyper un aspect particulier, plus la profondeur fonctionnelle est élevée.
4. la **richesse d'interactivité** : cette dimension relate la faculté d'exprimer des interactions entre le prototype et l'utilisateur final. Cela peut aller d'une richesse basse (p. ex. un prototype purement statique) jusqu'à une richesse élevée (p. ex. une simulation d'une interface dynamique sur la base de comportements simplifiés prédéterminés). Plus on peut supporter des échanges, des entrées/sorties entre l'utilisateur final et le prototype, plus la richesse d'interactivité sera considérée comme élevée.
5. la **richesse du modèle des données** : cette dimension traduit jusqu'à quel point on peut intégrer des données dans le prototype, particulièrement dans la richesse d'interactivité. Cela peut aller de données saisies et/ou affichées statiquement (de manière prédéfinie) à des données issues d'une base de données de test permettant de se faire une idée plus précise de la manière d'interagir avec ces données. Plus les données sujettes à interaction sont évoluées (p. ex. saisie, affichage, vérification des contraintes syntaxiques, sémantiques), plus la richesse du modèle des données est considérée comme élevée.

Dans la suite de cet article, la première dimension (le niveau de la représentation visuelle) est prépondérante pour indiquer tout changement du niveau de fidélité (la qualité de la représentation visuelle change, la fidélité change de manière coordonnée), les trois autres dimensions (à savoir la largeur et la profondeur des fonctionnalités, la richesse d'interactivité) seront également abordées. La dernière dimension n'est pas traitée. La section 2 de cet article est consacrée à la définition du modèle de l'interface et au problème de sa représentation visuelle. Une fois le modèle défini, il est important de se définir une approche méthodologique dans laquelle la représentation visuelle trouve une place et montrant quand et comment la représentation du prototype peut être utilisée dans le cycle de vie du développement de l'interface. Définir un modèle sans la méthode qui l'accompagne ne permet pas de connaître et de parcourir les étapes nécessaires à son obtention. La section 3 est consacrée à la définition d'approches méthodologiques alternatives. Enfin, dès que la dimension méthodologique a été précisée, nous serons à même de concevoir des outils de support à ce type de prototypage, ce qui fera l'objet de la section 4.

2 Modèle pour le prototypage multi-fidélité

Pour qu'un modèle soit défini de façon rigoureuse, il importe de définir successivement sa sémantique (section 2.1), sa syntaxe concrète (section 2.2) et sa stylistique (section 2.3). La dernière sous-section fera l'objet d'une attention particulière dans la mesure où nous y définirons la représentation des objets du modèle de l'interface.

2.1 Sémantique du modèle de l'interface

Définir la sémantique du modèle de l'interface revient à énoncer la signification du modèle de l'interface que l'on souhaite prototyper. Cette sémantique est ici exprimée sous la forme d'un méta-modèle spécifié en termes d'un diagramme de classe de la méthode UML. Seule la partie statique est détaillée. La partie dynamique comprenant les méthodes est ici omise pour raison de simplification.

Notre modèle se situe au niveau de l'interface concrète telle que définie au sein du cadre de référence CAMELEON (Calvary *et al.*, 2003), l' interface « concrète » est définie comme une expression de l'interface finale indépendamment de toute plate-forme logicielle et matérielle au sein de laquelle elle peut être implémentée. Elle est qualifiée de concrète dans la mesure où la modalité graphique est déjà sélectionnée, mais pas la ou les plate(s)-forme(s) de développement.

Une *interface concrète* est modélisée (figure 1) comme étant une agrégation d'objets interactifs concrets (en anglais, "concrete interaction object" - CIO) liés par des relations concrètes (Limbourg *et al.*, 2004, Limbourg et Vanderdonckt, 2004b). Un CIO n'a pas de représentation visuelle en tant que tel et représente davantage une classe abstraite caractérisée notamment par le nom (name), son icône (icon), son contenu courant (content), son contenu par défaut (defaultContent), le fait qu'il soit facultatif (isMandatory). Une relation concrète lie tout ou partie d'un modèle de l'interface (source et target) via un lien de contrôle ou de présentation graphique (p. ex. la transition, l'adjacence, la contenance, la mise en évidence, l'alignement) lui-même qualifié par des attributs (Limbourg et Vanderdonckt, 2004a).

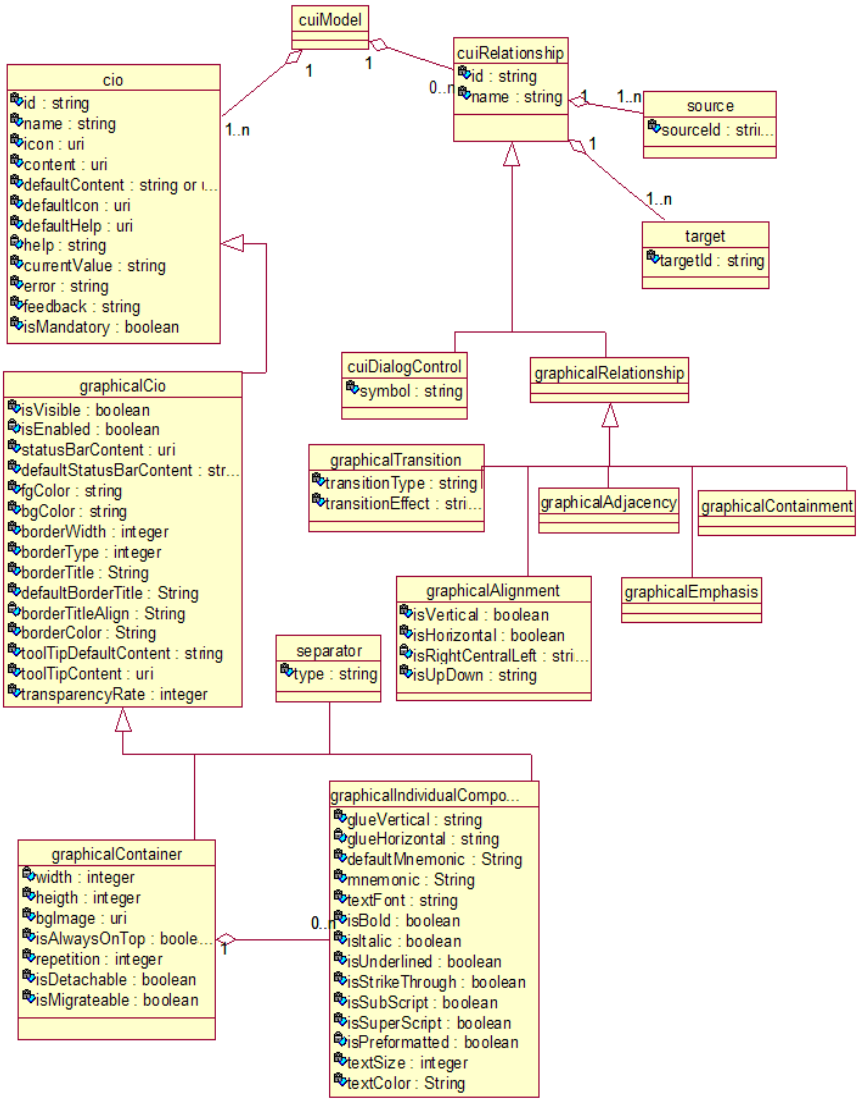


Figure 1. *Méta-modèle d'une interface concrète (Limbourg et Vanderdonckt, 2004a, 2004b)*

Puisque l'interface à prototyper ici est de nature graphique, seuls des CIO graphiques (graphicalCIO) sont mis en jeu. Ceux-ci sont caractérisés notamment par des attributs physiques (p. ex. la couleur d'avant-plan, d'arrière plan) et des attributs conceptuels (p. ex. sa visibilité, son activation, sa transparence). Il existe d'autres types de CIO (notamment pour les interfaces vocales et 3D, mais ceux-ci ne sont pas traités ici (pour plus de détails, voir (Vanderdonckt, 2005)). Trois catégories d'objets interactifs graphiques sont définies (Limbourg et Vanderdonckt, 2004a, 2004b) (figure 1) :

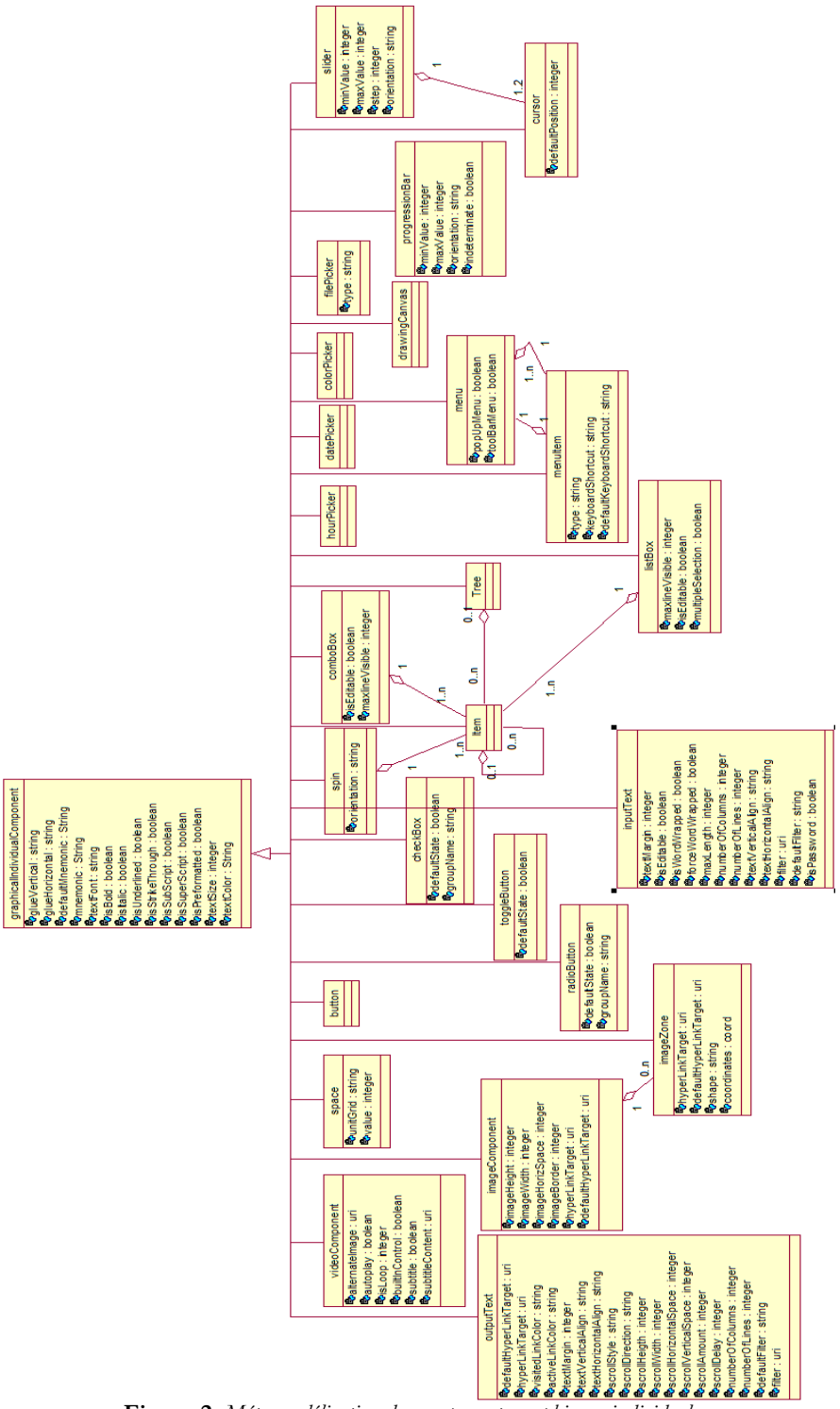


Figure 2. *Méta-modélisation des composants graphiques individuels*

1. la catégorie *composant graphique individuel* (graphicalIndividualComponent) comprend une abstraction de 25 objets interactifs usuels provenant des boîtes à outils et des gestionnaires de fenêtres (figure 2) : case à cocher, liste de sélection, liste de sélection déroulante, bouton de commande, image sensible, lien hypertexte, espace vidéo, bande sonore,… Cette abstraction a été obtenue en identifiant les attributs d'intérêt communs à une série d'environnements et représente le « plus grand commun diviseur ».
2. la catégorie *conteneur graphique* (graphicalContainer) comprend une abstraction de 14 objets à vocation de contenance de composants individuels : fenêtre, boîte de dialogue,…
3. la catégorie *séparateur* (separator) représente n'importe quel élément purement statique dont la raison d'être est de séparer visuellement les deux autres catégories précédentes.

A titre d'exemple, un lien permettant de passser d'une page web à une autre peut être modélisé par une transition graphique (graphicalTransition) associée à un outputText contenu dans une flowBox de départ pour arriver vers une flowBox d'arrivée. Modéliser précisément une interface en termes du modèle introduit constitue une activité aussi répétitive que compliquée ; c'est pourquoi il est plus facile de la confier à un éditeur qui produit automatiquement ce modèle dès qu'un prototype de l'interface est constitué. Ainsi, aucun effort supplémentaire n'est demandé au concepteur pour obtenir ce modèle outre la prise en main aisée de l'éditeur. A partir de ce modèle, une approche dirigée par les modèles (Ingénierie Dirigée par les Modèles – IDM) peut être adoptée pour effectuer un rendu de cette interface ou bien pour générer aussi automatiquement que possible le code de cette interface pour une plate-forme donnée. Il devient ainsi possible de produire des interfaces concrètes, mais indépendantes des plates-formes existantes (Coyette *et al.*, 2005).

2.2 Syntaxe du modèle de l'interface

Définir une syntaxe concrète de notre modèle d'interface concrète revient à préciser quelle grammaire formelle est utilisée pour modéliser une telle interface lors du prototypage. Dans ce but, le méta-modèle défini à la section 2.1 a donné lieu à un schéma XML permettant d'assurer la conformité d'une spécification par rapport au langage UsiXML (User Interface eXtensible Markup Language – http://www.usixml.org) dans sa version V1.6.4 (Limbourg et Vanderdonckt, 2004a, 2004b). Cette expression sous la forme d'un schéma XML a supplanté celle d'une DTD (Document Type Definition) en raison de la précision des types de données qu'elle autorise et de la rigueur dans l'expression du document. Dans l'expression d'une interface concrète en UsiXML, seules les feuilles de la figure 2 sont exprimées dans la spécification. Les nœuds intermédiaires ne le sont pas.

2.3 Stylistique du modèle de l'interface

Définir la stylistique du modèle pose la question de la représentation à associer aux différents objets. Dans chaque environnement intégré de développement, on trouve généralement un éditeur permettant de dessiner une interface dont sa représentation est calquée sur celle de la plate-forme accueillant l'environnement. La représentation graphique des objets interactifs est propre à chaque environnement et à chaque plate-forme, ce qui est contraire à notre souhait d'indépendance des plates-formes existantes. De plus, nous souhaitons ultérieurement fournir au concepteur un outil permettant de dessiner naturellement une interface concrète en manipulant de telles représentations graphiques.

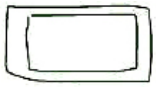
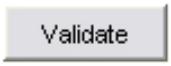
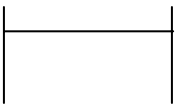
Objet interactif graphique	**Représentation graphique**	**Propositions d'esquisse**
Ancre		
Barre de progression		
Boîte à cocher		
Boîte de dialogue à onglets		
Boîte de re-groupement		
Bouton		
Bouton de déconnexion		
Bouton de réinitialisa-tion		
Bouton de validation		
Bouton radio		
Cadres (boî-tes)		
Champ de connexion		
Champ de recherche		

Figure 3. *Définition de la stylistique d'une interface concrète*

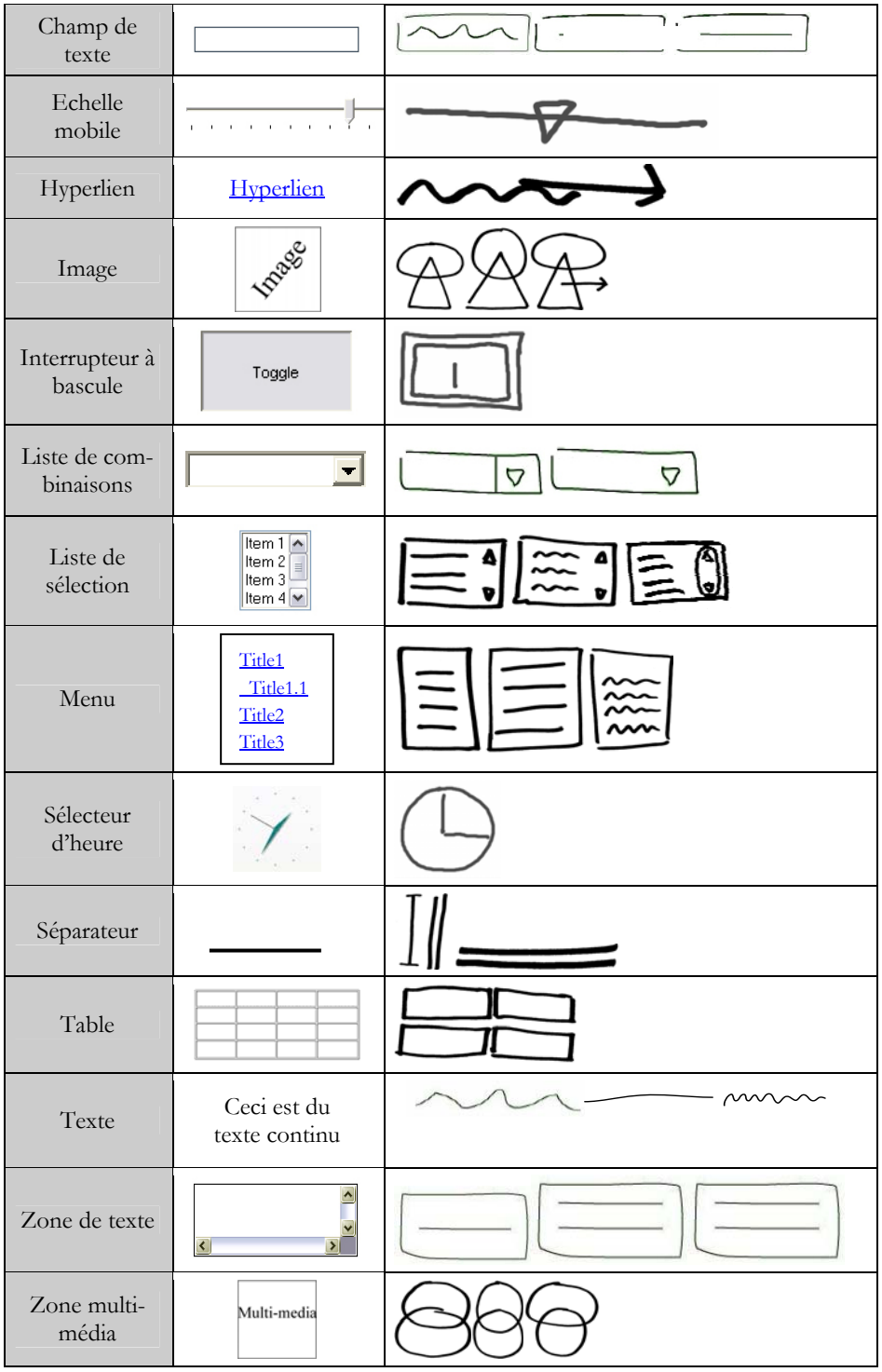


Figure 3. *Définition de la stylistique d'une interface concrète (suite)*

Pour déterminer les représentations graphiques les plus appropriées pour chaque objet interactif graphique (première colonne de la figure 3), une étude a été menée proposant des esquisses (Coyette et Vanderdonckt, 2005) alternatives (troisième colonne de la figure 3) pour chaque objet interactif graphique dont la représentation graphique est connue dans un environnement (deuxième colonne de la figure 3). Ces représentation furent proposées à deux groupes de participants avec ou sans expérience en conception d'IHM. Outre les objets de base de la figure 2, les participants ont marqué un intérêt pour manipuler aussi des objets particuliers non présents directement dans la figure 2, mais modélisables à l'aide des propriétés contenues dans le modèle. C'est le cas des boutons de déconnexion, de réinitialisation et de validation, ici considérés comme des instanciations partielles de la classe « Bouton » (une partie des valeurs des propriétés est instanciée). C'est aussi le cas des champs de connexion et de recherche, ici considérés comme des agrégations d'objets présents dans le modèle. Pour chaque objet ainsi identifié, un ensemble de propositions d'esquisse a été déterminé et testé tant par des concepteurs expérimentés que par des utilisateurs finaux afin d'identifier quelle représentation ils préfèrent dessiner et manipuler (Coyette et Vanderdonckt, 2005). Il en a résulté un classement par ordre décroissant de préférence de ces esquisses (les esquisses de la troisième colonne de la figure 3 sont ainsi présentées). Notons que ces esquisses ont été observées comme étant facilement associées aux objets concernés et qu'il n'y a pas eu de différence statistiquement significative entre les concepteurs et les utilisateurs finaux du point de vue de leur préférence. Enfin, les utilisateurs finaux ne sont pas moins habiles que les concepteurs pour dessiner une interface dont la représentation des objets interactifs graphiques est basée sur les représentations issues du test (Coyette et Vanderdonckt, 2005).

3 Méthodes pour le prototypage multi-fidélité

Cette section motive la notion de niveau de fidélité opérationnelle qui sera utilisée par la suite (section 3.1), motive pourquoi elle est importante (section 3.2) et montre des approches possibles de prototypage basées cette notion (section 3.3).

3.1 La fidélité du prototypage

Comme les besoins en prototypage rapide d'interface varient en fonction du projet et des ressources qui lui sont allouées, nous proposons de baser son prototypage sur la notion de fidélité du prototypage. La *fidélité du prototypage* exprime la similarité entre l'esquisse de l'interface prototypée et la représentation graphique de l'interface finale (Rettig, 1994, Hong *et al.*, 2001). On dira que la fidélité est *élevée*, en anglais, "high fidelity" (Rudd *et al.*, 1996, Walker *et al.*, 2002, Sefelin *et al.*, 2003), si la représentation du prototype est la plus proche possible de celle de l'interface finale. Dans ce cas, un prototype à un niveau de fidélité élevée devrait être aussi proche que possible de l'interface finale en termes de représentation (quels sont les objets interactifs graphiques utilisés), de navigation globale (comment naviguer entre les conteneurs graphiques), de navigation locale (comment naviguer à l'intérieur d'un conteneur graphique), de contenu (cfr. la richesse du modèle des données) et de fonctionnalités (cfr. la largeur et la profondeur de ces fonctionnalités selon McCurdy *et al.* (2006)). On dira que la fidélité est *faible*, en anglais, "low fidelity" (Rudd *et al.*, 1996), si la représentation du prototype évoque l'interface finale sans la représenter totalement en détails. Entre les deux, on exprimera que la fidélité est *moyenne*, en anglais, "mid fidelity" (Engleberg et Seffah, 2002). Jusqu'à présent, on a conservé l'habitude que la fidélité d'un prototype reste constante au cours du temps et ne fait pas intervenir différentes représentations.

Fidélité\critère	Basse	Moyenne	Elevée
Phase de développement	Enumération des besoins, conception préliminaire, conceptualisation, début de l'application	Conception continue, validation de l'ergonomie du prototype, application en cours	Conception détaillée, application en fin de spécification
Contenu	Présentation surtout	Présentation, layout, contenu, début de la navigation	Présentation et navigation, contenu, layout, fonctionnalités
Usage	Exploration, découverte, évocation, communication	Simulation, raffinement, itération, amélioration, validation de l'utilisabilité, test	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	Moyenne	Très faible
Coût	Faible	Moyen	Elevé
Temps requis	Faible	Moyen	Elevé
Naturalité de représentation	Très élevée	Moyenne	Faible
Détail	Faible	Moyen	Elevé
Fréquence d'itération	Très élevée	Elevée	Faible
Niveau d'interactivité	Faible	Moyen	Elevé
Représentation	Esquisse	Dessin, dessin vectoriel	Présentation et navigation réelles
Applications cibles	De grande taille	De taille moyenne	De taille réduite ou fragments
Niveau d'abstraction des spécifications	Abstrait (indépendante des modalités d'interaction et des plates-formes	Mixte (propre à une modalité, indépendante des plates-formes)	Concret (propre à une modalité d'interaction sur une plate-forme donnée)
Outils en général	Denim (Hong *et al.*, 2001), FreeForms (Plimmer et Apperley, 2003), GUILayout (Blankenhorn, 2004), Paper (Snyder, 2002), JavaSketchIt (Caetano *et al.*, 2002), SILK (Landay et Myers, 2001), SketchREAD (Alvarado et Randall, 2004)	EtchaPad (Meyer, 1996), ExcelProto (Berger, 2006), MidFi (Engelberg et Seffah, 2002), ProtoMixer (Petrie et Schneider, 2006), iRise (www.irise.com), MockupScreens	Editeurs d'interface fournis avec les environnements intégrés de développement

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

Cependant, on peut très bien imaginer disposer d'un prototypage mélangeant différents niveaux de fidélité si le prototype est composé d'objets interactifs graphi-

ques représentés avec des fidélités 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 d'un logiciel de présentation assistée par ordinateur (fidélité moyenne). Petrie et Schneider (2006) présentent ProtoMixer, un outil de support au prototypage mixte mêlant différents niveaux. Le tableau 1 compare les trois niveaux de fidélité ainsi définis sur la base d'une même grille d'analyse constituée de critères. Dans la suite du texte, nous reprendrons seulement les critères considérés comme importants.

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, indéterminées. Dans ce cas, l'objectif est surtout d'explorer des conceptions alternatives possibles (Tohidi *et* al., 2006) sans trop les affiner 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, sauf si un outil efficace vient supporter cette étape. Cette maîtrise du coût et du temps peut s'effectuer au prix d'une réduction du niveau de fidélité de la représentation (p. ex. en basse fidélité souvent centrée sur la présentation) : pas question de développer 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 s'approchant favorablement de l'interface finale, soit parce que les itérations répétées du prototypage à un niveau de fidélité inférieure ont permis de dégager des conceptions acceptables et qu'il importe maintenant de les affiner jusqu'à obtenir 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.

3.2 Etudes empiriques sur la fidélité du prototypage

Le prototypage à fidélité faible possède de nombreux atouts attestés par différentes études expérimentales qui ont été menées à son égard en le comparant par rapport à un prototypage à un niveau de fidélité plus élevé :

- La faculté de dessiner, d'esquisser une interface graphique récolte plus de suffrages en général qu'un simple éditeur d'interface graphique et davantage chez les utilisateurs finaux (qui ne doivent rien exprimer d'autre que leur aptitude à dessiner ce qu'ils souhaitent) que chez les concepteurs d'interface (Landay et Myers, 1995, Plimmer et Apperley, 2004, Buxton, 2005).
- Le fait de prototyper une interface à un niveau de fidélité plus bas ne diminue en rien sa capacité à servir de catalyseur pour découvrir des problèmes et des erreurs d'ordre ergonomique (Virzi *et al.*, 1996). On peut toujours identifier les faiblesses et les omissions d'un prototype.
- La fidélité basse par rapport à la fidélité élevée n'enlève rien au pouvoir expressif du prototype, qu'il soit effectué sur papier ou à l'aide d'un outil de support (Walker *et al.*, 2002). Suivant cette même étude, la préférence va nettement vers un outil de support de façon à récupérer, éditer, conserver, et communiquer les résultats du prototypage.
- La fidélité basse fournit des résultats tout à fait comparables à ceux de la fidélité élevée tant en terme de qualité qu'en terme de quantité, mais à un prix moindre (Sefelin *et al.*, 2003). A nouveau, la préférence pour un outil de sup-

port est confirmée par cette étude comme elle a été identifiée dans la précédente étude d'opinion.

- La fidélité basse encourage davantage la découverte et la communication de conceptions alternatives pour une même interface qu'à n'importe quel autre niveau (Tohidi *et al.*, 2006). Elle s'avère complémentaire à la conduite de tests ergonomiques et supporte davantage la réflexion relative aux conceptions alternatives que simplement la réaction des utilisateurs vis-à-vis des prototypes produits.
- Comme la fidélité basse possède le coût minimal, rien ne la retient pour jeter plus facilement un prototype et en recommencer un autre (Petrie et Schneider, 2006, Bailey et Konstan, 2003).
- La fidélité moyenne peut préserver les qualités de la fidélité faible si elle maintient ses capacités d'expression, tout en offrant la possibilité de représenter une interface de manière « lissée » sans toutefois faire référence à une plateforme logicielle ou matérielle donnée (Engleberg et Seffah, 2002, Meyer, 2005).

3.3 Approches méthodologiques du 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 (Berkun, 2000, Hong *et al.*, 2001) comme la figure 4 le suggère. 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 4). 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 (Baümer *et al.*, 1996). 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 basse ou moyenne. 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.

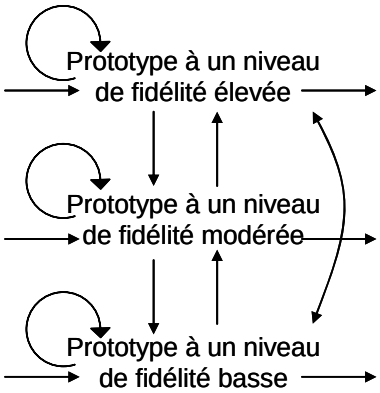


Figure 4. *Trajectoires possibles de prototypage*

Le cadre de référence de Nielsen (figure 5) distingue deux types de prototypage en fonction du niveau d'interactivité couvert (tableau 1) (Nielsen, 1993) : verti-

cal (figure 5a) et horizontal (figure 5b). On peut décomposer une application interactive complète en trois couches : l'interface homme-machine proprement dite, la couche d'abstraction de l'application et le noyau fonctionnel comportant les fonctions sémantiques de l'application.

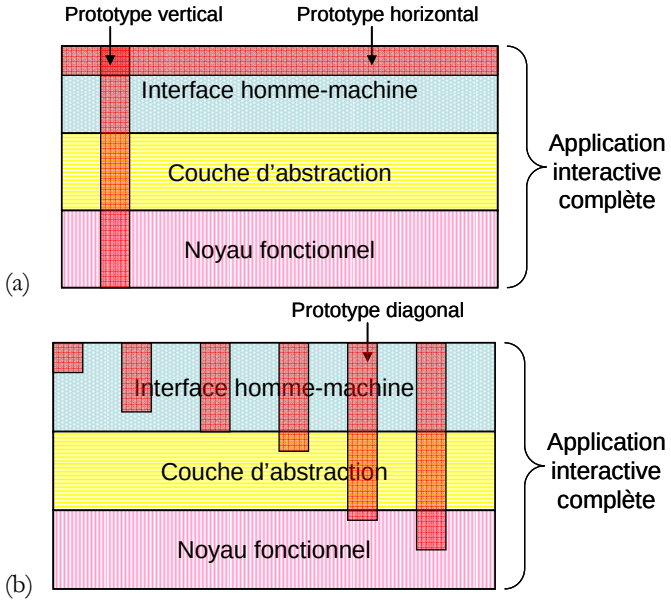


Figure 5. *Prototypage vertical (a) vs. horizontal (b) (Nielsen, 1993)*

Prototypage horizontal

Le type de prototypage est 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 basse ou moyenne 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 5b. L'interface est d'abord complétée (1), la couche d'abstraction est entamée ensuite (2), puis les fonctions sémantiques du noyau fonctionnel (3). Ceci correspond bien à une situation où l'interface constitue l'élément primordial à prototyper avant toute autre partie. Une fois celle-ci stabilisée, le comportement associé peut être développé sans crainte de le voir modifié trop rapidement.

Prototypage vertical

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

5a). Ainsi, on prototype successivement différentes parties de l'application interactive, dénotées par (1) à (4) à la figure 5a.

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'information n'existe). Dans ce cas, il est possible de combiner les deux types de prototypage dans le prototypage que nous appelons *diagonal* (Coyette *et al.*, 2005, Coyette et Vanderdonckt, 2005) (figure 6). 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 souffrir des conséquences négatives respectives. La propagation du prototypage (suivant la largeur et la profondeur des fonctionnalités) est dite *expansive* puisqu'elle peut s'étendre dans toutes les directions. C'est pourquoi le prototypage à un niveau de fidélité moyenne convient bien pour un type d'approche *expansive* (comme exprimé au tableau 1), par opposition à un niveau de fidélité basse pour une approche *ascendante* et à un niveau de fidélité élevée pour une approche *descendante*. La figure 6 montre différentes étapes d'un prototypage diagonal qui débute par l'interface (1 à 3), puis touche la couche d'abstraction (4 et 5) pour terminer par le noyau fonctionnel (6).

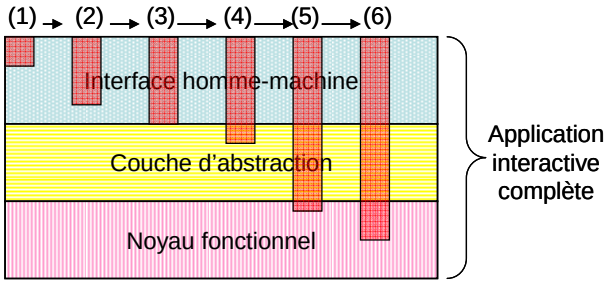


Figure 6. *Prototypage diagonal*

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 précisé et des nouvelles trajectoires de prototypage peuvent apparaître (figure 7). 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 qu'il s'agit d'un *prototype présentationnel* d'abord (figure 7a). Il peut n'y avoir aucune navigation (ni globale, ni locale), auquel cas il s'agit d'une simulation statique. La figure 7a affine la couche présentation refléter la partie qui est effectivement prototypée.

En général, le prototypage par papier (Snyder, 2002) 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 que le prototypage présentationnel, on observe le *prototype navigationnel global d'abord* (figure 7b) 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 7b 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 7c). 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 en spécifiant les grands traits de la navigation globale. La figure 7c 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.

Après avoir analysé 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. Nous commençons par le niveau de fidélité élevée puisqu'il est censé être le plus proche de l'interface finale envisagée. Nous examinons ensuite progressivement les niveaux de fidélité consécutifs (moyenne, élevée). 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 autres niveaux de fidélité.

L'ensemble de ces outils est interopérable : les outils communiquent entre eux en s'échangeant 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) (Limbourg *et al.*, 2004, Limbourg et Vanderdonckt, 2004). 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 européen SIMILAR (http://www.similar.cc). Le langage UsiXML a couvert initialement les interfaces graphiques multi-plate-forme au sein du projet Caméléon pour couvrir ensuite les interfaces vocales, haptiques, 3D et multimodales au sein du projet SIMILAR. Ce langage couvre notamment les aspects de tâche, du domaine, de l'interface abstraite (Coyette *et al.*, 2005), 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 (Calvary *et al.*, 2003). Dans le présent article, seule l'interface concrète est utilisée. Il existe d'autres langages similaires, avec des couvertures plus ou moins différentes et des objectifs différents : l'étude par Souchon et Vanderdonckt (2003) procure un premier aperçu de ces langages de description sous la forme d'une grille d'analyse, les deux les plus significatifs en termes d'avancement étant UIML (User Interface Markup Language, www.uiml.org) et XIML (eXtensible Interface Markup Language, www.ximl.org).

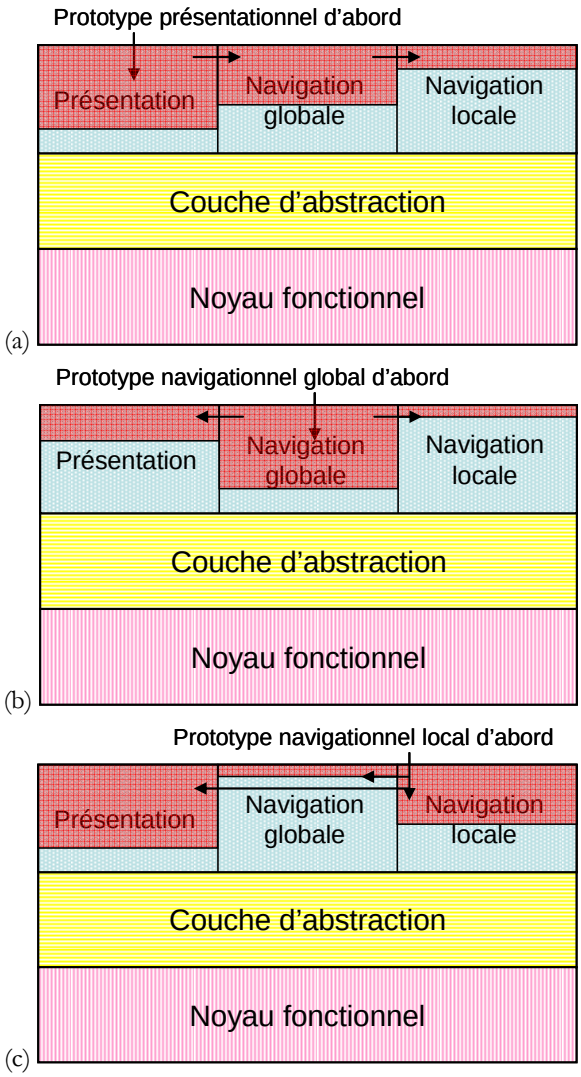


Figure 7. *Différents prototypages de l'interface : « présentationnel » (a), « navigationnel global » (b), « navigationnel local » (c).*

4 Outils logiciels de support au prototypage multi-fidélité

Voyons à présent quels sont les outils disponibles : pour chaque niveau de fidélité, nous passons en revue d'abord les grandes caractéristiques du prototypage à supporter, puis nous citons quelques outils représentatifs avant de présenter l'outil logiciel que nous avons développé pour ce niveau.

4.1 Prototypage à un niveau de fidélité élevée

A un niveau de fidélité élevée, 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 provenant de 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 Microsoft Visual Studio.

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. Pour les mêmes raisons, 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* (Berkun, 2000, Snyder, 2002). Ces outils permettent de prototyper une interface en la construisant sans faire appel à du codage et 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 parce que ce code est généré dans un langage auteur qui n'est pas celui de l'interface finale (souvent, un langage de script, de marquage ou de programmation). Le développeur recommence le développement à zéro, avec une interface à haute fidélité.

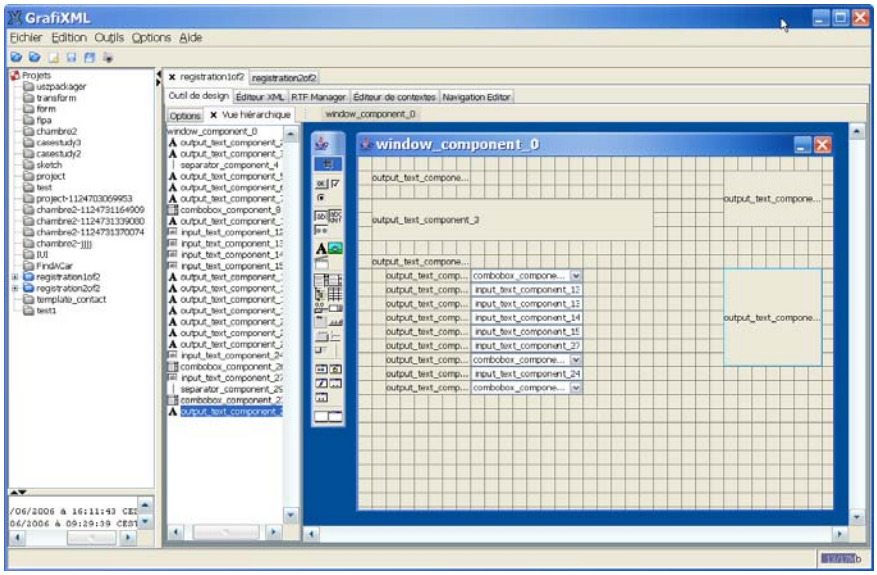


Figure 8. *Interface de GrafiXML en mode de composition de la présentation (prototypage présentationnel)*

Pour répondre à ces défis, nous avons développé GrafiXML (figure 8), 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 soient 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 rédigées en UsiXML. Ces spécifications peuvent être liées ou non à un contexte d'utilisation. Il est possible aussi de décliner différentes interfaces possibles pour plusieurs contextes d'utilisation associés au même projet. Ceci est approprié lorsqu'il faut développer une version multi-plate-forme d'une interface. La figure 8 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 visuelle indépendante de ces plates-formes est affichée. Pour obtenir en temps réel une interface finale, 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 9 correspond à l'interface générée en XHTML à partir de celle spécifiée à la figure 8, mais sans feuille de style associée. Une feuille de style peut être adjointe 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.

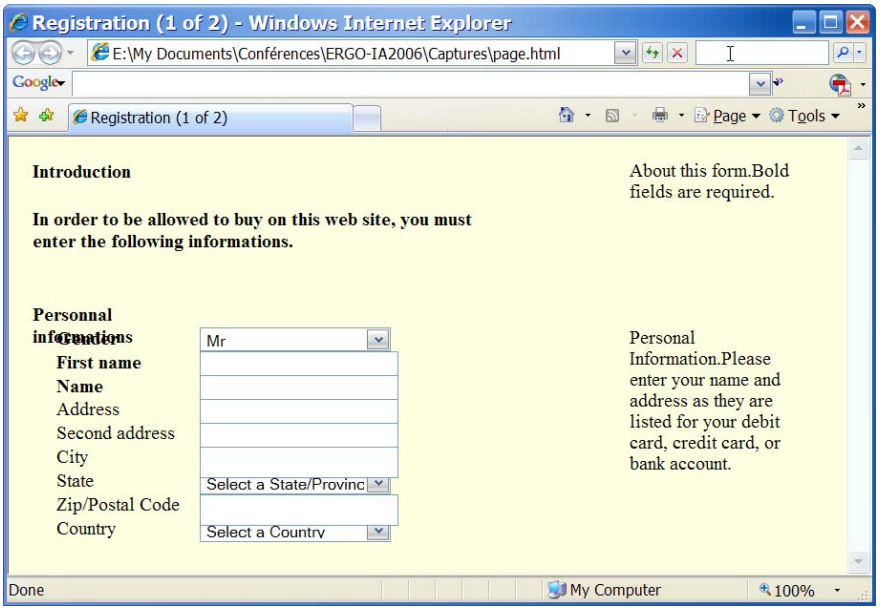


Figure 9. *Interface finale résultant de la génération à partir de GrafiXML*

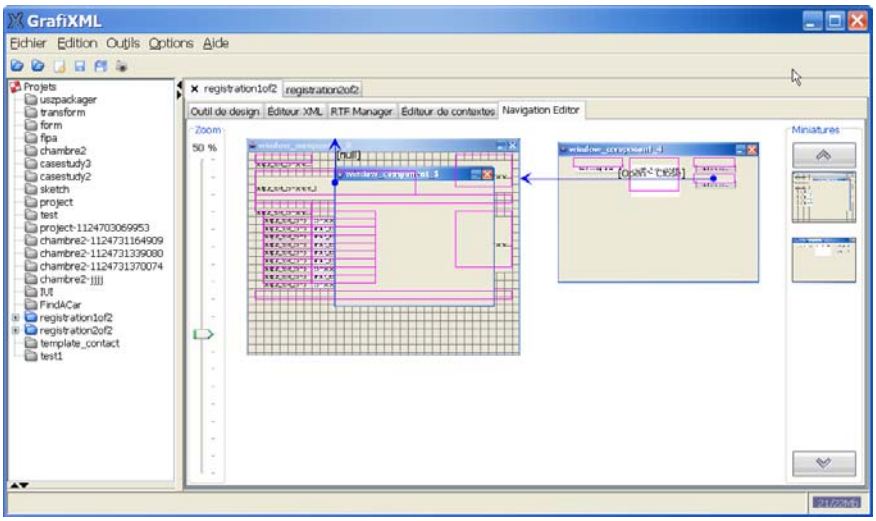


Figure 10. *Interface de GrafiXML en mode de spécification de la navigation (prototypage navigationnel)*

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 PC, un PDA, pour (dé)composer des interfaces entre elles, pour évaluer leur qualité ergonomique en cours de conception, etc.

Enfin, pour supporter le prototypage navigationnel, GrafiXML offre un mode de spécification de la navigation (figure 10). 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 appartenant à un conteneur vers un autre conteneur ou un de ses objets 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 être gouverné par le comportement d'un objet interactif en particulier (p. ex. un bouton de commande) ou du conteneur lui-même, signifiant par là un comportement attaché à ce conteneur. Ces comportements sont actuellement prédéfinis.

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. Actuellement, GrafiXML ne supporte que la navigation globale. Nous sommes en train de développer la partie relative à la spécification de la navigation locale. Tâche plus complexe étant données 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 prévoyons de nous limiter à une stylistique simple.

GrafiXML ne dispose pas actuellement de la capacité à incorporer un objet interactif non prédéfini dans UsiXML. Tout objet interactif considéré comme une instance ou une agrégation d'instances des classes présentées à la figure 2, même partielles, peut être spécifié dans GrafiXML. S'il faut spécifier un objet interactif qui n'existe pas dans cette modélisation (p. ex. un histogramme, un objet interactif dépendant d'un domaine d'activité particulier), on peut définir librement un objet informel en le dessinant et lui adjoignant des propriétés personnalisables. Mais ce dessin ne sera pas reconnu et les spécifications UsiXML résultant ne seront que le reflet des propriétés spécifiées. Par conséquent, GrafiXML souffre de la même lacune que les éditeurs concernant les objets métier.

4.2 Prototypage à un niveau de fidélité moyenne

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 programmation générative).

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 moyenne.

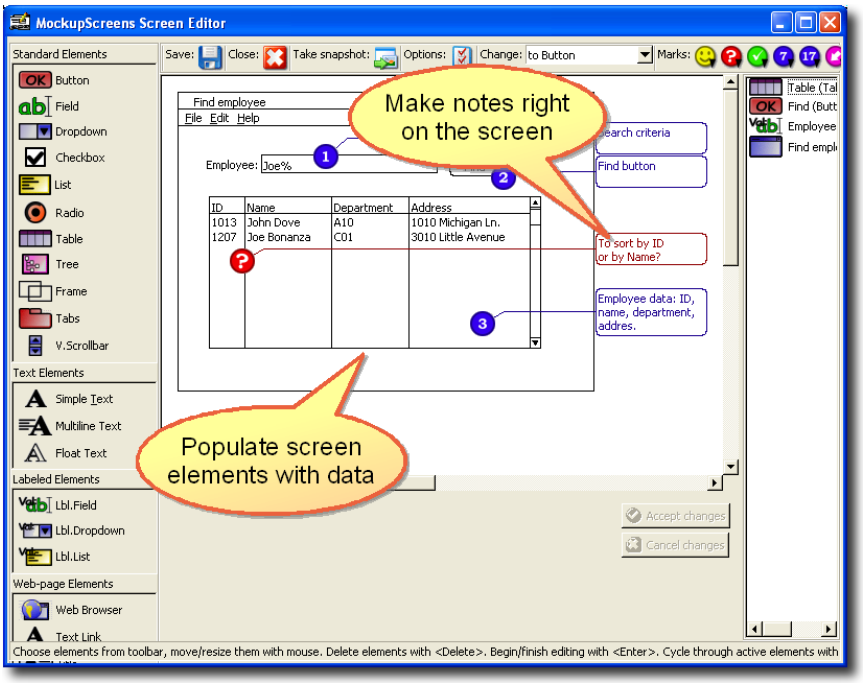


Figure 11. *Interface de MockupScreens en mode de spécification de la présentation (prototypage présentationnel)*

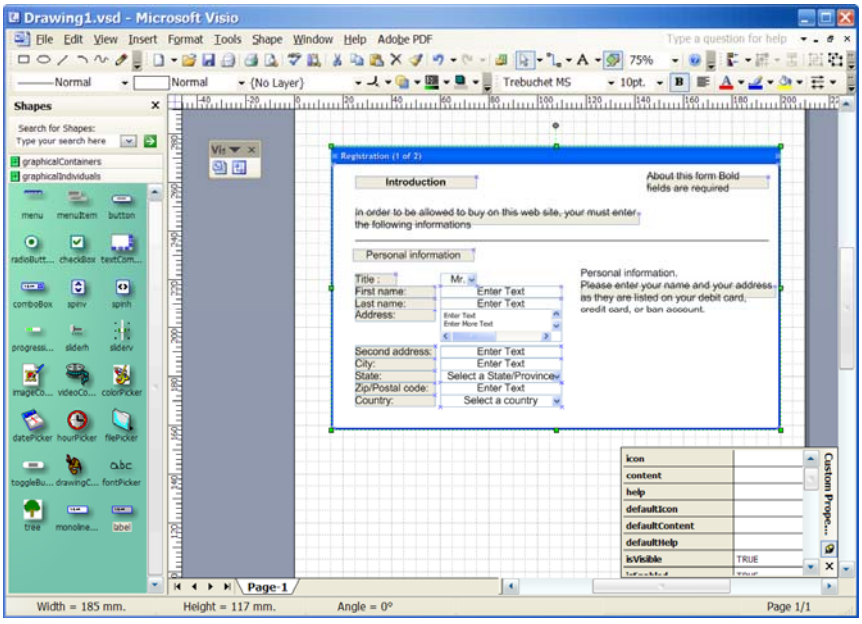


Figure 12. *Interface de VisiXML en mode de spécification de la présentation (prototypage présentationnel)*

Il existe divers éditeurs d'interface à niveau de fidélité moyenne tels que MockupScreens (http://www.mockupscreens.com/), iRise (http://www.irise.com), ExcelProto (Berger, 2006). Mais ces éditeurs sont tous liés à une plate-forme donnée (principalement, une interface graphique dans l'environnement Ms-Windows ou une interface Web). Dès qu'un prototypage navigationnel survient, une programmation est demandée au concepteur. Ils ne génèrent pas de spécifications fonctionnelles et ne peuvent pas générer de code pour différents environnements en raison de cette lacune.

Pour y remédier, nous avons développé VisiXML (figure 12), un éditeur d'interface à niveau de fidélité moyenne 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 12). 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 moyenne) 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 moyenne, 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 12. Ces propriétés sont retenues dans la génération des spécifications en UsiXML. La capacité d'expression de VisiXML s'arrête 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 (Tohidi *et al.*, 2006) et de créativité (Xiaogang *et al.*, 2002). Certes, VisiXML comporte la faculté de dessin, mais sa représentation liée à une plate-forme unique ne permet pas de faire penser qu'il s'agit d'une interface en pleine évolution : son niveau formel de détails (Hong *et al.*, 2001) procure l'impression qu'il s'agit déjà d'une interface presque finale, en tout cas aux 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 (2006) a imaginé un outil de prototypage d'interface à un niveau de fidélité moyenne 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.

4.3 Prototypage à un niveau de fidélité basse

Pour ne plus véhiculer l'impression qu'une interface est quasi finale lorsque sa représentation se situe à un niveau de fidélité élevée, il est judicieux de passer à un niveau de fidélité basse. A ce niveau, la représentation graphique de l'interface que l'on manipule doit être la plus naturelle possible pour ne pas entraver le processus de création, de conception exploratoire. Cette représentation manipulée au sein de l'éditeur est donc fondamentale pour donner à l'utilisateur l'impression qu'il s'agit d'une interface en cours de conception et non d'une interface finale. Le caractère informel (Hong *et al.*, 2001) des spécifications est à préserver, de même que la capacité à concevoir une interface sans contrainte.

Au bas de l'échelle des représentations graphiques se trouve GUILayout (Blankenhorn, 2004) dont l'interface est reproduite à la figure 13. 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 9, le concepteur a dessiné un conteneur reprenant les zones de texte (destinées à contenir les informations de guidage) et la zone de formulaire.

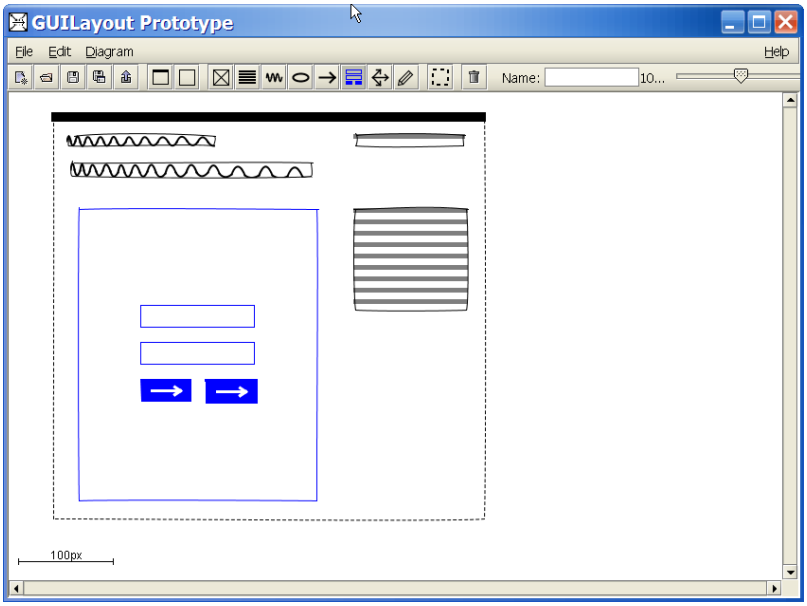


Figure 13. *Interface de GUILayout en mode de spécification de la présentation (prototypage présentationnel)*

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 (1996) avec son outil EtchaPad. Meyer estime que même une représentation graphique de l'interface finale qui serait indépendante de la plate-forme (p. ex. la partie gauche de la figure 14) 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 (Meyer, 2005) permettant d'affecter automatiquement des contours flous aux représentations graphiques des objets interactifs représentés (la partie droite de la figure 14).

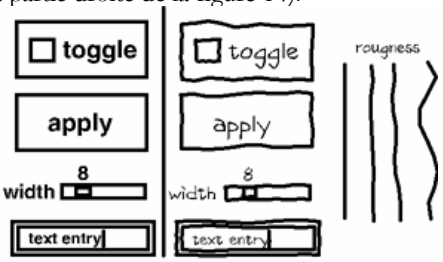


Figure 14. *Deux représentations graphiques d'une interface en prototypage à niveau de fidélité basse*

A un niveau plus élevé qu'EtchaPad (Meyer, 1996) réside une famille d'outils d'esquisse d'interface manipulant une représentation dessinée ou esquisse, en anglais, "sketching" (Buxton, 2005) de l'interface en cours de prototypage. Les représentants les plus significatifs de cette famille sont dans l'ordre alphabétique : Denim (Hong *et al.*, 2001), EtchaPad (Meyer, 1996), FreeForms (Plimmer et Apperley, 2003, Plimmer et Apperley, 2004), JavaSketchit (Caetano *et al.*, 2002) basé sur la bibliothèque de reconnaissance de formes CALI (Fonseca *et al.*, 2002), SILK (Landay et Myers, 1995, Landay et Myers, 2001), SketchRead (Alvarado et Randall, 2004). Ces outils partagent un mode de fonctionnement presque commun : au lieu de manipuler une représentation graphique de l'interface, proche ou éloignée comme dans Meyer (2005), 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 l'étude préliminaire (Coyette et Vanderdonckt, 2005), 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. Il est intéressant d'étudier le binôme (concepteur, utilisateur final) en situation collaborative de conception. 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. La multi-fidélité est *mixte* (Petrie et Schneider, 2006) 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 diminuent. 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 (Virzi *et al.*, 1996).

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 moins contraint 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 en général aux trois premiers principes, mais pas au dernier. C'est pourquoi, nous avons développé SketchiXML (Coyette *et al.*, 2004, Coyette *et al.*, 2005, Coyette et Vanderdonckt, 2005), un éditeur d'interface à niveau de fidélité basse fondé sur le langage UsiXML, dont l'interface est reproduite à la figure 15. Dans cet outil, le concepteur peut spécifier 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 naturelle sur des plates-formes de type tablette : TabletPC ou Wacom Cintiq (figure 16) 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 et restaurant l'interaction en manipulation directe. 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 par un ingénieur du système SketchiXML dans une grammaire graphique sous la forme de trois représentations préférées des objets d'interaction. Ces trois représentations sont issues de tests utilisateur qui ont permis de déterminer quelles 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. Il est possible d'ajouter une nouvelle représentation pour un même objet à tout moment.

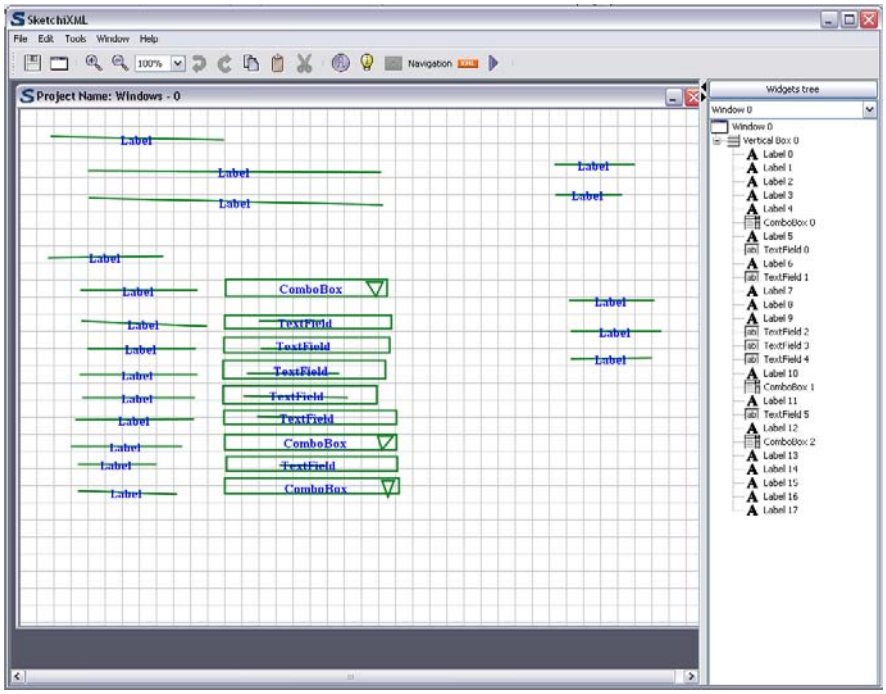


Figure 15. *Interface de SketchiXML en mode de spécification de la présentation (prototypage présentationnel)*

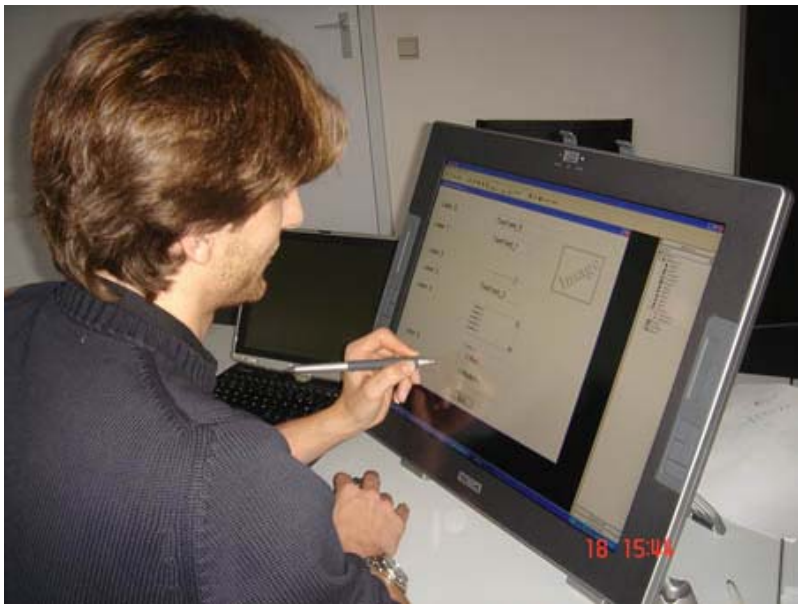


Figure 16. *Utilisation de SketchiXML sur une tablette graphique*

SketchiXML tente 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 un processus de conception, hautement créatif par nature (Xiaogang *et al.*, 2002).
- **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 13 montre que certaines formes ont été reconnues et enrichies 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 niveau de fidélité. Il ne doit pas passer d'un niveau 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 (Fonseca *et al.*, 2002), 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 (Hong *et al.*, 2001) qui ne supporte aucune reconnaissance et plus loin que FreeForms (Plimmer et Apperley, 2004) qui reconnaît moins d'objets, SketchiXML reconnaît environ une trentaine d'objets interactifs, c'est-à-dire tous ceux qui sont définis dans l'interface concrète en UsiXML (Limbourg et Vanderdonckt, 2004a). Il peut donc exporter le résultat de l'esquisse dans un fichier UsiXML qui est alors récupéré dans l'outil GrafiXML (figures 8, 10) afin d'être affiné jusqu'au niveau de détails demandé (Vanderdonckt, 2005).

Au terme de ce parcours d'outils, la figure 4 peut être revisitée à la lumière des avancées effectuées et permet de couvrir à présent les trajectoires de la figure 17.

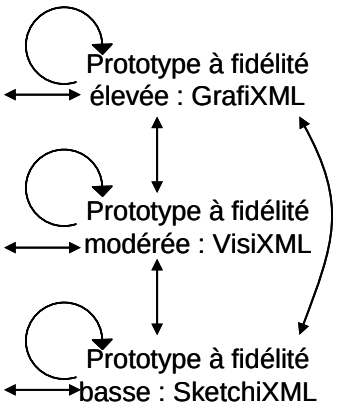


Figure 17. *Trajectoires possibles de prototypages en UsiXML*

5 Conclusion

Dans cet article, 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, moyenne, basse) et qu'il est possible de passer de n'importe quel niveau de fidélité à un autre en principe. Dans cette trilogie, chaque outil est associé à un niveau de fidélité seulement (p. ex. VisiXML est associé au niveau de fidélité moyenne) ou à deux niveaux (p. ex. SketchiXML permet de dessiner en fidélité basse et de reconnaître l'interface dessinée pour en obtenir une représentation en fidélité élevée). Nous pensons davantage poursuivre vers un outil intégrant vraiment le concept de multi-fidélité (Coyette *et al.*, 2007) en n'obligeant plus le concepteur à passer d'un outil de support à l'autre lorsqu'il change de niveau de fidélité. Au contraire, cette faculté devrait être offerte au sein d'un même outil de support, ce qui sera le cas de SketchiXML.

Cette faculté devrait encourager davantage une évolution progressive du cycle de vie de développement de l'interface en accord avec son caractère ouvert, itératif et incomplet (Sumner *et al.*, 1997) : toutes les trajectoires de prototypage sont couvertes (figure 16). 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 son caractère naturel et non contraint est attractif : le concepteur, l'utilisateur final lui-même ou bien les deux ensemble peuvent pour la première fois collaborer directement au prototype en cours de conception, partageant chacun leurs impressions ou les répercutant directement sur l'interface en cours de prototypage. Gageons que ces nouvelles formes de prototypage procureront à l'utilisateur final un réel sentiment d'implication.

Espérons aussi que ce prototypage puisse s'étendre à d'autres modalités que la modalité graphique, notamment le prototypage rapide d'interfaces vocales, tactiles, haptiques, multimodales (Stanciulescu et Vanderdonckt, 2006). Par exemple, Topiary (Li *et al.*, 2004) exploite l'expérience acquise dans le développement d'applications sensibles au contexte d'utilisation pour proposer des représentations de natures différentes utiles au prototypage de telles applications. Plus centré sur une modalité en particulier, SUEDE (Klemmer *et al.*, 2000) permet de prototyper rapidement une application vocale en manipulant une abstraction de questions et réponses. Moins centré sur une modalité en particulier, Crossweaver (Sinha et Landay, 2001) permet de manipuler une représentation graphique d'une interface multimodale dont le comportement est également dessiné à l'aide de transitions, à l'instar de DEMAIS (Bailey et Konstan, 2003). Dans tous ces cas, le problème de la représentation à manipuler s'avère plus complexe que la question de la stylistique d'objets interactifs graphiques. La représentation dominante peut rester graphique, mais ne peut se borner à ne rester que graphique. Par exemple, prototyper une interface vocale sans recourir à cette modalité, ne fût-ce qu'en mode de simulation, enfreindrait le principe de naturalité. Si on ne peut pas exprimer naturellement le contenu d'une interface, quelle que soit sa modalité, cela constituera une sérieuse entrave. Il y a donc ici tout un champ d'investigation qui s'ouvre : comment, par exemple, prototyper une interface tactile en mettant en œuvre un rendu tactile de celle-ci tout au long du cycle de vie de son développement ? Ou bien simplement un rendu graphique de celle-ci pour autant que cela s'avère être utilisable. Ce n'est pas parce qu'une IHM est prédominante pour une ou plusieurs modalités qu'il faut nécessairement les représenter avec une stylistique empruntée à ces mêmes modalités.

Dans un proche avenir, nous tenons à poursuivre l'intégration du concept de prototypage à fidélité multiple au sein d'un même éditeur, tel que SketchiXML (Coyette *et al.*, 2007) de façon à expérimenter quels sont les niveaux de fidélité préférés en fonction des types d'utilisateurs (p. ex. un concepteur, un utilisateur final).

Remerciements

Les auteurs tiennent à remercier le projet ReQuest (Rapid prototyping of e-commerce applications) financé par la Région Wallonne sous la convention n°315592 dans le cadre du programme «WIST» (Wallonie Information Science & Technology). Les études réalisées dans cet article n'auraient pas été rendues possibles sans l'existence du réseau d'excellence européen SIMILAR (The European research taskforce creating human-machine interfaces SIMILAR to human-human communication – http://www.similar.cc) sous la convention FP6-IST1-2003-507609. Enfin, nous remercions vivement Suzanne Kieffer et Mickaël Nicolay pour leur participation dans les études expérimentales menées avec SketchiXML et les relecteurs anonymes pour leurs conseils constructifs, précieux et précis, qui ont réellement permis d'améliorer cet article.

Références

Alvarado, C., Randall, D. (2004). SketchREAD: A Multi-domain Sketch Recognition Engine. In *Proceedings of 17th Annual ACM Symposium on User Interface Software and Technology*, UIST'2004, Santa Fe, 24-27 octobre 2004, ACM Press, New York, 23-32.

Bailey, B.P., Konstan, J.A. (2003). Are Informal Tools Better? Comparing DEMAIS, Pencil and Paper, and Authorware for Early Multimedia Design. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, CHI'2003, Fort Lauderdale, 5-10 avril 2003, ACM Press, New York, 313-320.

Baümer, D., Bischofberger, W.R., Lichter, H., Züllighoven, H. (1996). User Interface Prototyping - Concepts, Tools, and Experience. In *Proceedings of 18th Int. Conf. on Software Engineering*, ICSE'1996, Berlin, 25-29 mars 1996, IEEE Computer Society Press, Los Alamitos, 532-541.

Berger, N. (2006). The Excel Story. *Interactions*, vol. 13, num. 1, 14-17.

Berkun, S. (2000). *The Art of User Interface Prototyping*. Disponible à http://www.scottberkun.com/essays/essay12.htm

Blankenhorn, K. (2004). *A UML Profile for GUI Layout*. Thèse de Maîtrise, University of Applied Sciences Furtwangen, Furtwangen. Disponible à http://www.bitfolge.de/pubs/thesis/index.html

Boar, B.H. (1984). *Application prototyping: a requirements definition strategy for the 80s*. John Wiley & Sons, New York.

Buxton, W. (2005). Sketching and Experience Design. In *Proceedings of 10th IFIP TC 13 International Conference on Human-Computer Interaction*, INTERACT'2005, Rome, 12-16 septembre 2005, Lecture Notes in Computer Science, Vol. 3585, Springer-Verlag, Berlin, 1.

Caetano, A., Goulart, N., Fonseca, M., Jorge, J. (2002). JavaSketchIt: Issues in Sketching the Look of User Interfaces. In *Proceedings of the 2002 AAAI Spring Symposium on Sketch Understanding*, Palo Alto, mars 2002, AAAI Press, Menlo Park, 9-14.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J. (2003). A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers*, vol. 15, num. 3, 289-308.

Coyette, A., Faulkner, S., Kolp, M., Limbourg, Q., Vanderdonckt, J. (2004). SketchiXML: Towards a Multi-Agent Design Tool for Sketching User Interfaces Based on UsiXML. In *Proceedings of 3rd International Workshop on Task Models and Diagrams for user interface design*, TAMODIA'2004, Prague, 15-16 novembre 2004, Palanque, P., Slavik, P., Winckler, M. (Eds.), ACM Press, New York, 75-82.

Coyette, A., Vanderdonckt, J., Faulkner, S., Kolp, M. (2005). Generating Abstract User Interfaces from an Informal Design. In *Proceedings of 17th International Conference on Software Engineering and Knowledge Engineering*, SEKE'2005, Taipei, 14-16 juillet 2005, Ijseke Press, Taiwan.

Coyette, A., Vanderdonckt, J. (2005). A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces. In *Proceedings of 10th IFIP TC 13 International Conference on Human-Computer Interaction*, INTERACT'2005, Rome, 12-16 septembre 2005, Lecture Notes in Computer Science, Vol. 3585, Springer-Verlag, Berlin, 550-564.

Coyette, A., Kieffer, S., Vanderdonckt, J. (2007) Multi-Fidelity Prototyping of User Interfaces. In *Proceedings of 11th IFIP TC 13 International Conference on Human-Computer Interaction*, INTERACT'07, Rio de Janeiro, 10-14 septembre 2007, Lecture Notes in Computer Science, Vol. 4662, Springer-Verlag, Berlin, 149-162.

Engelberg, D., Seffah, A. (2002). A Framework for Rapid Mid-Fidelity Prototyping of Web Sites. In *Proceedings of the IFIP 17th World Computer Congress - TC13 Stream on Usability: Gaining a Competitive Edge*, WC'2002, 25-29 août 2002, Kluwer Academic Press, Dordrecht, 203-215.

Fonseca, M.J., Pimentel, C., Jorge, J.A. (2002). CALI: An Online Scribble Recognizer for Calligraphic Interfaces. In *Proceedings of the 2002 AAAI Spring Symposium on Sketch Understanding*, Palo Alto, mars 2002, AAAI Press, Menlo Park, 51-58.

Hong, J.I., Li, F.C., Lin, J., Landay, J.A. (2001). End-User Perceptions of Formal and Informal Representations of Web Sites. In *Extended Abstracts of Proceedings of ACM Conference on Human Factors in Computing Systems*, CHI'2001, Seattle, 31 mars – 5 avril 2001, ACM Press, New York, 385-386.

Klemmer, S.R., Sinha, A.K., Chen, J., Landay, J.A., Aboobaker, N., Wang, A. (2000). Suede: a Wizard of Oz Prototyping Tool for Speech User Interfaces. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, UIST'2000, San Diego, 6-8 novembre 2000, ACM Press, New York, 1-10.

Landay, J.A., Myers, B.A. (1995) Interactive Sketching for the Early Stages of User Interface Design. In *Proceedings of ACM Conference on Human Factors in Computing Systems*, CHI'1995, Denver, 7-11 mai 1995, ACM Press, New York, 43-50.

Landay, J.A., Myers, B.A. (2001) Sketching Interfaces: Toward More Human Interface Design. *IEEE Computer*, vol. 34, num. 3, 56-64.

Li, Y., Hong, J.I., Landay, J.A. (2004). Topiary: a Tool for Prototyping Location-enhanced Applications. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, UIST'2004, Santa Fe, 24-27 octobre 2004, ACM Press, New York, 217-226.

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez, V. (2004). UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proceedings of 9th IFIP Working Conference on Engineering for HCI jointly with 11th International Workshop on Design, Specification, and Verification of Interactive Systems*, EHCI-DSVIS'2004, Hamburg, 11-13 juillet 2004, Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 200-220.

Limbourg, Q., Vanderdonckt, J. (2004a). UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. In *Engineering Advanced Web Applications*, Matera, M., Comai, S. (Eds.), Rinton Press, Paramus, 325-338.

Limbourg, Q., Vanderdonckt, J. (2004b). Addressing the Mapping Problem in User Interface Design with UsiXML. In *Proceedings of 3rd International Workshop on Task Models and Diagrams for user interface design*, TAMODIA'2004, Prague, 15-16 novembre 2004, Ph. Palanque, P. Slavik, M. Winckler (eds.), ACM Press, New York, 155-163.

McCurdy, M., Connors, C., Pyrzak, G., Kanefsky, B., Vera, A. (2006). Breaking the Fidelity Barrier: An Examination of our Current Characterization of Prototypes and an Example of a Mixed-Fidelity Success. In *Proceedings of ACM Conference on Human Factors in Computing Systems*, CHI'2006, Montreal, 22-27 avril 2006. ACM Press, New York, 1233-1242.

Meyer, J. (2005). Creating Informal Looking Interfaces. Disponible à http://www.cybergrain.com/tech/pubs/lines_technote.html

Meyer, J. (1996). EtchaPad – Disposable Sketch Based Interfaces. In *Conference Companion of Proceedings of ACM Conference on Human Factors in Computing Systems*, CHI'1996, Vancouver, 13-18 avril 1996, ACM Press, New York, 195-198.

Newman, M.W., Lin, J., Hong, J.I., Landay, J.A. (2003). DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice. *Human-Computer Interaction*, vol. 18, 259-324.

Nielsen, J. (1993). Prototyping. Chapter 4. In *Usability Engineering*, Nielsen, J. (Ed.), Academic Press, 93-101.

Petrie, J.N., Schneider, K.A. (2006). Mixed-Fidelity Prototyping of User Interfaces. In *Proceedings of 13th International Workshop on Design, Specification, and Verification of Interactive Systems*, DSV-IS'2006, Dublin, 26-28 juillet 2006, Lecture Notes in Computer Science, Springer-Verlag, Berlin.

Plimmer, B.E., Apperley, M. (2003). Software for Students to Sketch Interface Designs. In *Proceedings of 9th IFIP TC 13 Int. Conference on Human-Computer Interaction*, INTERACT'2003, Zurich, 1-5 septembre 2003, Rauterberg, M., Menozzi, M., Wesson, J. (Eds.), IOS Press, Amsterdam, 73-80.

Plimmer B., Apperley, M. (2004). Interacting with Sketched Interface Designs: An Evaluation Study. In *Proceedings of ACM Conference on Human Factors in Computing Systems*, CHI'2004, Vienne, 24-29 avril 2004, ACM Press, New York, 1337-1340.

Rettig, M. (1994). Prototyping for tiny fingers. *Communications of the ACM*, vol. 37, num. 4, 21-27.

Rudd, J., Stern, K., Isensee, S. (1996). Low vs. High-Fidelity Prototyping Debate. *Interactions*, vol. 3, num. 1, 76-85.

Sefelin, R., Tscheligi, M., and Giller, V. (2003). Paper prototyping – what is it good for? A comparison of paper-and computer-based prototyping. In *Extended Proceedings of ACM Conference on Human Aspects in Computing Systems*, CHI'2003, Ft. Lauderdale, 5-10 avril 2003, ACM Press, New York, 778-779.

Sinha, A.K., Landay, J.A. (2001). Visually Prototyping Perceptual User Interfaces Through Multimodal Storyboarding. In *Proceedings of IEEE Workshop on Perceptive User Interfaces*, PUI'2001, Orlando, novembre 2001.

Snyder, C. (2002). *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces*. Series in Interactive Technologies, Morgan Kaufmann, San Francisco.

Souchon, N., Vanderdonckt, J. (2003) A Review of XML-Compliant User Interface Description Languages. In *Proceedings of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems*, DSV-IS'2003, Madeira, 4-6 juin 2003, Jorge, J., Nunes, N.J., Cunha, J. (Eds.), Lecture Notes in Computer Science, Vol. 2844, Springer-Verlag, Berlin, 2003 377-391.

Stanciulescu, A., Vanderdonckt, J. (2006). Design Options for Multimodal Web Applications. In *Proceedings of 6th International Conference on Computer-Aided Design of User Interfaces*, CADUI'2006, Bucarest, 6-8 juin 2006, Springer-Verlag, Berlin, 41-56.

Sumner, T., Bonnardel, N., Harstad Kallak, B. (1997). The Cognitive Ergonomics of Knowledge-Based Design Support Systems. In *Proceedings of ACM Conference on Human Aspects in Computing Systems*, CHI'1997, Atlanta, 22-27 mars 1997, ACM Press, New York, 83-90.

Tohidi, M., Buxton, W., Baecker, R., Sellen, A. (2006). User Sketches: A Quick, Inexpensive, and Effective way to Elicit More Reflective User Feedback. In *Proceedings of 4th Nordic Conference on Human-Computer Interaction*, NordiCHI'2006, Oslo, 14-16 octobre 2006, ACM Press, New York, 105-114.

Vanderdonckt, J. (2005). A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In *Proceedings of 17th Conference on Advanced Information Systems Engineering*, CAiSE'2005, Porto, 13-17 juin 2005, Pastor, O., Falcão e Cunha, J. (Eds.), Lecture Notes in Computer Science, Vol. 3520, Springer-Verlag, Berlin, 16-31.

Virzi, R.A., Sokolov, J.L., Karis, D. (1996). Usability problem identification using both Low- and High-Fidelity Prototypes. In *Proceedings of ACM Conference on Human Aspects in Computing Systems*, CHI'1996, Vancouver, 13-18 avril 1996, ACM Press, New York, 236-243.

Walker, M., Takayama, L., Landay, J. (2002). High-fidelity or low-fidelity, paper or computer medium? In *Proceedings of the 46th Annual Meeting of the Human Factors and Ergonomics Society*, HFES'2002, Baltimore, 30 septembre – 4 octobre 2002, Human Factors and Ergonomics Society, Santa Monica, 661-665.

Xiaogang, X., Liu, W., Xiangyu, J., Zhengxing S. (2002). Sketch-based User Interface for Creative Tasks. In *Proceedings of 5th Asia Pacific Conference on Computer-Human Interaction*, APCHI'2002, Beijing, novembre 2002, 560-570.