UNIVERSITÉ DU QUÉBEC À MONTRÉAL

INTEGRATION DES DIRECTIVES DE L'UTILISABILITE DANS LES COMPOSANTS ET LES ENVIRONNEMENTS DE DEVELOPPEMENT D'INTERFACES USAGER GRAPHIQUES

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR

MOULOUD AIT-IKHELEF

DÉCEMBRE 2010

UNIVERSITÉ DU QUÉBEC À MONTRÉAL
Service des bibliothèques

Avertissement

La diffusion de ce mémoire se fait dans le respect des droits de son auteur, qui a signé le formulaire *Autorisation de reproduire et de diffuser un travail de recherche de cycles supérieurs* (SDU-522 – Rév.01-2006). Cette autorisation stipule que «conformément à l'article 11 du Règlement no 8 des études de cycles supérieurs, [l'auteur] concède à l'Université du Québec à Montréal une licence non exclusive d'utilisation et de publication de la totalité ou d'une partie importante de [son] travail de recherche pour des fins pédagogiques et non commerciales. Plus précisément, [l'auteur] autorise l'Université du Québec à Montréal à reproduire, diffuser, prêter, distribuer ou vendre des copies de [son] travail de recherche à des fins non commerciales sur quelque support que ce soit, y compris l'Internet. Cette licence et cette autorisation n'entraînent pas une renonciation de [la] part [de l'auteur] à [ses] droits moraux ni à [ses] droits de propriété intellectuelle. Sauf entente contraire, [l'auteur] conserve la liberté de diffuser et de commercialiser ou non ce travail dont [il] possède un exemplaire.»

REMERCIEMENTS

Je tiens à remercier mon directeur de recherche, Monsieur Hakim Lounis, professeur au département d'informatique de l'UQÀM, pour son soutien, son support, sa patience et son dévouement qui m'ont aidé tout au long de la réalisation de mon mémoire. Je remercie également les réviseurs de ce mémoire de maîtrise.

Je dois aussi remercier ma femme qui n'a pas cessé de me soutenir par sa patience continue et son attention constante tout au long de ce travail.

Enfin, tous mes remerciements à Dieu, ma défunte mère, mon père, mes frères, mes sœurs, ma grande famille et tous mes amis qui m'ont toujours soutenu et encouragé.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

GLOSSAIRE DES TERMES ET ACRONYMES UTILISÉS

IUG: Interface Utilisateur Graphique

HCI: Human Computer Interaction

UCD: User Centered Design

IU: Interface utilisateur

XML: eXtensible Markup Language

HCD: Human-Centered Design

UE: Usability Engineering

SE: Software Engineering

RUP: Rational Unified Process

JAVA : Langage de programmation de Sun Microsystems

Toolkits: Trousse à outils

Builder Tools : Outil de développement

Beans: Classe du langage de programmation orientée objet JAVA

Browse: Naviguer

Design: Conception

IEEE: Institute of Electrical and Electronics Engineers

ISO: Organisation internationale de normalisation

Framework: Cadre de travail

Widgets: composant d'interface

SOF: Open Software Foundation

Motif: désigne l'interface utilisateur graphique permettant de créer des applications pour les systèmes X Window sous Unix

Windows: Système d'exploitation de Microsoft.

CASE : Computer Aided Software Engineering

IDC : International Data Corporation

Mac : Macintosh

JFC : Java Fundation Class

Swing: Trousse d'outil d'interfaces graphiques du langage de programmation Java

Toolbox : Boite à outil

Visual Basic : Langage de programmation de Microsoft

VBX: Composent de boite a outil de Visual Basic

.mak: Extension de fichiers de Visual Basic

JBuilder: Outil de développement d'application

EDI: Environnement de développement intégré

PowerBuilder: Outil de développement d'application

API: Application programming interface

SWT: Standard Widget Toolkit

AWT: Abstract Windowing Toolkit

ActiveX: Composent de boite à outil de Visual Basic

JFace: Outil d'interfaces utilisateur en utilisant SWT

ON/OFF: Allumer/éteindre

Eclipse: Environnement de développement intégré

Repository Tools : Outil d'entreposage

C++ : Langage de programmation

HP-UX : système d'exploitation propriétaire de type Unix développé par Hewlett-Packard

Add-ons : mécanisme utilisé pour améliorer ou étendre le fonctionnement d'un logiciel.

RÉSUMÉ

Concevoir un logiciel facilement utilisable est une mission difficile et complexe. Pour accomplir cette tâche, les concepteurs ont besoin d'outils efficaces dédiés à cela. Un outil efficace doit être basé sur une expertise de conception reconnue. Canaliser le savoir de conception de systèmes utilisables réussis est important pour les concepteurs novices et expérimentés. Les directives de l'utilisabilité sont une technique utilisée pour transmettre ce savoir.

Présentement, il n'y a pas de mécanisme efficace permettant de guider et d'assister les développeurs à créer des interfaces graphiques conformes aux directives de l'utilisabilité. Cette problématique devient plus critique pour les développeurs inexpérimentés et peut même compromettre la raison d'être des directives.

Ce projet consiste à définir une approche qui adresse l'intégration de directives de l'utilisabilité dans les constructeurs d'interfaces graphiques usager.

.

ABSTRACT

Designing usable software is tricky and it is not an easy mission. To accomplish this task designers need efficient tools that are dedicated for. Effective design tools should be based on proven design knowledge. Capturing knowledge about the successful design of usable systems is important for both novice and experienced designers. Usability guidelines are one technique that is used to convey this knowledge.

Currently, there are no mechanisms that guide and assist developers to build a UI that complies with existing guidelines. This issue becomes more critical for inexperienced developers while compromising the expected benefits of the guidelines.

This project proposes a novel mechanism for incorporating usability design guidelines into UI development tools and toolkits..

CHAPITRE I

INTRODUCTION

Ce chapitre d'introduction présente la problématique d'incorporer les directives de l'utilisabilité dans les composants d'interfaces utilisateurs (IU) et le cycle de développement du logiciel. Aussi, il énumère les objectifs et la portée de ce mémoire. Enfin, ce chapitre présente la méthodologie de recherche et l'organisation de ce manuscrit.

1.1 La pertinence d'intégrer l'utilisabilité dans le cycle de développement du logiciel

L'utilisabilité a été longtemps considérée et perçue différemment. Pour beaucoup d'ingénieurs de logiciels, cela signifie simplement « la facilité d'utilisation » ou « facile à utiliser » ("ease-of-use" or "user friendly"). Le terme utilisabilité a été introduit au début de l'apparition de la science de l'interaction personne-machine. Aussi, nous trouvons ce terme dans la définition des spécifications du projet, isolé parmi d'autres spécifications non fonctionnelles. La norme de « IEEE » Std.610.12-1990 reflète cette perception de l'utilisabilité: " La facilité avec laquelle un utilisateur peut apprendre à opérer, à préparer des entrées, et à interpréter des sorties d'un système ou d'un composant" ("The ease with which a user can learn to operate, to prepare inputs for, and to interpret outputs of a system or component").

Les chercheurs de l'approche centrée sur l'utilisateur (HCD : *Human Centered Design*) et les professionnels de l'utilisabilité ont emprunté des concepts issus de la psychologie et de la cognition. L'utilisabilité y est définie comme un ensemble d'attributs, tels que la satisfaction, l'apprentissage et la performance de l'utilisateur (temps d'accomplissement et d'exécution de tâche, taux d'erreur). Ce point de vue est illustré par la définition suivante: "La capacité du

logiciel a être appris, compris, manipulé et attrayant à l'utilisateur, quand utilisé sous certains conditions" ("The capability of the software product to be understood learned, used and attractive to the user, when used under specified conditions") [SO/IEC FCD 9126-1, 1998].

L'importance de considérer l'utilisabilité comme un facteur et un attribut de la qualité du logiciel dans le cycle de développement, augmente constamment dans les organisations de développement de logiciels. De plus en plus, l'utilisabilité et les tests d'acceptation des utilisateurs sont sur le point d'être reconnus comme un test de mesure de la qualité pour les applications interactives. Ce test ne concerne pas seulement les applications traditionnelles d'ordinateurs de bureau, mais également les sites web de commerce électronique et des services mobiles. Beaucoup d'études ont démontré les avantages d'un engagement fort à considérer le test de l'utilisabilité dans le cycle de développement du logiciel. Une discussion détaillée justifiant le coût de l'effort associé au test de l'utilisabilité, indépendamment des méthodes spécifiques de l'utilisabilité, est donnée en exemple [Bias & Mayhew, 1994; Landauaer, 1995; Karat, 1997; Donahue, 2001; Marcus, 2002]

Cependant, malgré l'importance croissante des méthodes de l'utilisabilité dans le domaine du génie logiciel, et l'évidence de leur avantages, il semble que les pratiques en matière de génie logiciel sont encore déficientes quant aux méthodes de l'utilisabilité. Par exemple, dans un sondage sur 8.000 projets, le groupe Standish a constaté que le manque d'implication des utilisateurs et l'analyse des besoins inachevée, représentent les deux motifs les plus importants de l'échec du projet [Standish, 1995].

Comme d'autres, nous disons que ces problèmes et résultats sont principalement dus au fait que dans le développement du logiciel interactif avec une interface utilisateur importante, la plupart des méthodologies de génie logiciel ne fournissent aucun mécanisme pour:

- Identifier et définir les besoins de l'utilisateur et les exigences de l'utilisabilité explicitement et empiriquement;
- Tester et valider avec des utilisateurs les besoins, les prototypes préliminaires de conception, aussi bien que les systèmes entièrement fonctionnels avant, pendant et après le développement, et le déploiement du système.

La philosophie de Human-Centered-Design et les méthodes relatives à l'utilisabilité ont été présentées par les professionnels de l'utilisabilité pour fournir des solutions puissantes à de tels problèmes [Norman, 1986, Mayhew, 1999; Vredenburg, 2003]. Cependant, les méthodes les plus répandues de génie logiciel, comme RUP (Rational Unified Process) ou les plus récentes approches de développement, restent défaillantes au niveau de l'intégration explicite de ces méthodes de HCI/UE [Kazman, 2003; Seffah, 2004]. Ce vide actuel entre les communautés de génie logiciel et de l'utilisabilité a empêché le transfert des techniques entre ces communautés. D'une certaine façon, ce vide est également un obstacle pour la validation et l'amélioration du HCI et des techniques de l'utilisabilité.

Actuellement, tous les développeurs de logiciels reconnaissent l'importance et la puissance de l'utilisabilité. Cependant, l'utilisabilité demeure le domaine des visionnaires, des praticiens éclairés et des grandes organisations, mais elle est absente dans la pratique quotidienne chez le développeur de logiciels typiques. La pratique et la théorie restent encore rares sur la manière d'incorporer efficacement des méthodes de l'utilisabilité au cycle de développement des logiciels existants. Alors que les normes comme ISO 15529 (Usability Maturity Model) et ISO 13407 (Human-Centered-Centered Design Processes for Interactive Systems) fournissent des moyens pour évaluer les possibilités d'une organisation à adopter des pratiques en matière de HCD. Toutefois, la défaillance réside au niveau de la façon à mettre réellement en application l'amélioration des processus et des stratégies d'organisation qui mènent à une intégration efficace. Souvent, ça reste ambigu aux professionnels de l'utilisabilité et du génie logiciel. C'est pourquoi, certains outils d'UE et méthodes sont plus adéquats que d'autres dans un certain contexte de développement.

D'ailleurs, historiquement l'approche centrée sur l'utilisateur (HCD) a été présentée comme l'opposé, et parfois comme remplacement, à la philosophie « system-functionality-driven » généralement utilisée dans les boîtes à outils de génie logiciel [Norman, 1986]. La réalité est que l'utilisabilité et les techniques de génie logiciel diffèrent par leurs forces et faiblesses. Aussi, même si leurs objectifs se chevauchent dans quelques secteurs, elles diffèrent dans d'autres. Les méthodes de l'utilisabilité devraient être vues comme partie intégrante de chaque activité de développement du logiciel. Toutefois, malgré leurs avantages bien documentés, il reste beaucoup à faire pour les adopter. Il est évident qu'un framework intégré

qui incorpore les principes de conception, de développement et d'évaluation des deux domaines apportera une utilisation plus efficace de l'utilisabilité dans le développement.

Le manque d'intégration des processus de l'utilisabilité, de génie logiciel et des pratiques en matière de génie logiciel était le principal sujet de nombreuses recherches et ateliers organisés pendant la dernière décennie. Ces ateliers ont mis en évidence l'écart entre les méthodes de l'utilisabilité et de génie logiciel, ainsi que l'importance d'y remédier [Artim, 1997; Artim, 1998; Seffah and Hayne, 1999; Nuno, 1999; Gulliksen, 1998; Kazman et all., 2003]. La liste complète de ces ateliers et la publication relative à ce domaine est disponible à https://securedoc.gi.polymtl.ca/mdesmarais/CHISE/index.Shtml

La conclusion que nous pouvons tirer de ces ateliers est qu'il est clair que l'intégration de l'utilisabilité et les méthodes relatives au HCI/HCD dans le domaine du génie logiciel devrait être considérée à différents niveaux. Il faut aussi prendre en considération les perspectives organisationnelles et culturelles du produit, du processus et des outils.

Au niveau des processus et outils, qui sont notre principal intérêt dans ce mémoire, plusieurs d'entre eux ont été proposés pour s'assurer que les techniques de l'utilisabilité sont employées pendant le développement du logiciel. Alors que quelques méthodes proposent des processus complets qui entourent le cycle de développement entier, un grand nombre d'approches visent à étendre des artéfacts individuels de génie logiciel, comme les cas d'utilisations, à des spécifications de l'utilisabilité. L'introduction de l'analyse des tâches comme un artéfact pré-requis dans la phase d'analyse est un exemple typique d'approche d'intégration à travers des artéfacts individuels [Artim, 1997; Artim, 1998]. Les problèmes peuvent surgir parce que simplement la présentation des artéfacts n'assure pas leur utilisation appropriée. Par exemple, en raison des similitudes entre "use cases" et l'analyse de tâches [Artim, 1998; Seffah, 1999; Rosson, 1999], les ingénieurs de logiciels et de l'utilisabilité essayent souvent de substituer l'un à l'autre, sans se rendre compte que les deux outils sont conçus différemment afin d'atteindre différents buts.

1.2 Approche d'intégration proposée: Directives de l'Utilisabilité dans les Environnements de Développement d'Interfaces Usagers

Dans notre travail, nous étudions les mécanismes spécifiques d'intégration consistant à incorporer des directives de l'utilisabilité dans les composants d'interface utilisateur et les environnements de développement d'interface usager, généralement connus sous le nom de constructeur d'interface graphique usager (GUI builders) (Figure 1.1). Par des mécanismes d'intégration, nous voulons dire une solution logicielle qui peut être implémentée et facilement incorporable dans le code du constructeur d'interface graphique (GUI builders) lui-même.

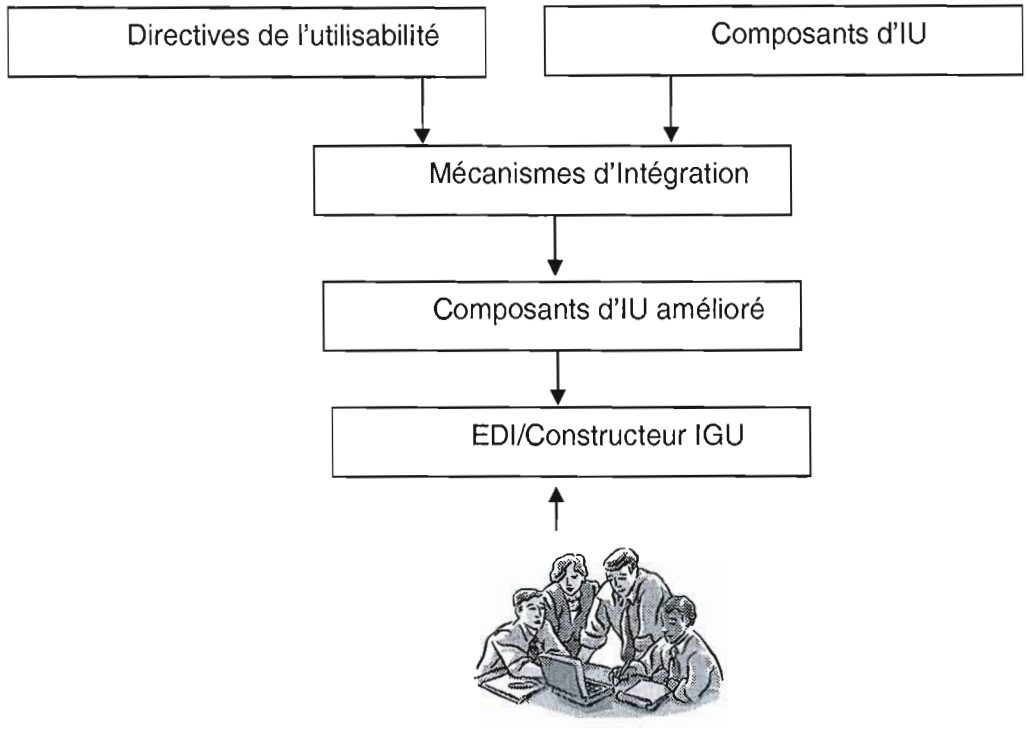


Figure 1.1 approche d'intégration proposée via directives et constructeur d'interface (GUI builders)

Dans ce mémoire, le terme composant d'IU, également connu comme "widgets", "controls" ou élément, est employé pour référer aux composants de logiciel utilisés dans la création d'interface usager. Dans le contexte des interfaces utilisateurs graphiques (GUIs), ceci inclut les fenêtres, les boites de dialogue, les boutons, les listes, les menus, les tables, les arbres, les étiquettes, les champs de texte, les barres d'outils, etc. Généralement ces composants sont organisés dans des trousses à outils (toolkits), par type d'application de développement d'interface. Ils sont aussi incorporés dans les packages de développement de logiciels. Afin de créer une interface usager, les développeurs combinent ces composants en utilisant les constructeurs d'interface graphique.

Les directives de l'utilisabilité visent à encapsuler (capture) l'expertise (la connaissance) de conception sous forme de petites règles, qui peuvent être utilisées lors de la création de nouvelles interfaces. Les directives représentent le niveau intermédiaire de guide de conception en progression à partir des principes d'abstraction aux conventions spécifiques.

En générale, les directives de l'utilisabilité sont décrites dans un document qui contient la description de l'usage et le style de ces composants d'interface graphique. Les exemples de guides de style commerciaux incluent les guides de style de OSF/Motif [opengroup.org], Windows user expérience de Microsoft [microsoft.com] ou Java Look and Feel de Sun Microsystems [oracle.com/us/sun/index.html]. Ces directives sont souvent associées à une trousse d'outils (toolkit) commerciale et à un environnement de développement. Elles peuvent agir comme base pour des guides de style personnalisés qui sont adaptés pour un produit d'une organisation.

Les directives sont principalement utilisées durant les phases de développement et de test de l'utilisabilité d'interface usager pour assurer la consistance de la conception d'interface. Le développement de guides de styles est une phase importante pour les équipes de projets. Les guides de style sont une façon utile pour capter et documenter les décisions de conception, et pour prévenir l'imperfection de l'utilisabilité du logiciel.

Bien que les règles soient certainement utiles dés le début du cycle de développement pour assurer l'utilisabilité des logiciels, elles souffrent des mêmes problèmes, à savoir conflits, inconsistances et ambiguïté. De plus les règles de style sont très limitées à un type d'applications très particulières ou aux plateformes (système d'exploitation). Ainsi, leur capacité à étendre les règles de l'utilisabilité à une large audience est limitée.

Plus fondamentalement, il est souvent rapporté dans la littérature scientifique que les directives de conception ont plusieurs problèmes lors de leur utilisation [Martijn et al, 2000]. Il est difficile de choisir les directives appropriées à un problème particulier de conception. D'ailleurs, des fois, les directives peuvent même sembler se contredire et par conséquent le concepteur peut ne pas résoudre le problème. De plus, Tidwell a indiqué qu'il est même difficile pour un concepteur novice, laissé seul, de mémoriser et d'utiliser efficacement toutes ces nombreuses directives [Tidwell, 1999]. Des fois, il est difficile de choisir parmi ces principes quand ils arrivent en conflits. Lors de l'utilisation d'un composant spécifique, le concepteur doit déterminer la meilleure solution par supposition, ou en ayant recours à d'autres moyens.

1.3 Objectifs spécifiques du mémoire

Ce travail de recherche vise à remédier au problème de la séparation physique entre les directives de l'utilisabilité et les constructeurs d'interface graphique usager (GUI Builder). Les directives sont des techniques qui captent la connaissance (expertise) de l'utilisabilité. Les trousses à outils (toolkits) et les constructeurs d'interface graphique existants fournissent une liste de composants que les concepteurs combinent pour créer une IU. Cependant, les développeurs doivent maîtriser les directives de l'utilisabilité. Il ou elle doit aussi savoir comment et quand les appliquer lors de la phase de création de l'IU. Actuellement, il n'existe pas de mécanisme qui guide les développeurs et leur fournit la possibilité de construire une IU conforme aux directives existantes. En général, ce problème est plus critique pour les développeurs inexpérimentés.

Notre objectif est de définir un mécanisme sous la forme d'un logiciel qui permet l'intégration de directives de l'utilisabilité dans les constructeurs d'interface graphique usager. Les contributions majeures de la recherche qui sont présentés dans ce mémoire sont:

1. Analyse des environnements de développement d'interface existant et évaluer comment la problématique d'intégration de l'utilisabilité est prise en considération afin d'étudier comment les directives peuvent être incorporées dans ces outils.

2. Proposition d'une architecture conceptuelle et d'un mécanisme permettant d'incorporer, au niveau du code, les directives de l'utilisabilité dans les constructeurs d'interface graphique usager.

3. Implémentation d'une telle architecture dans le contexte de l'environnement de Java Eclipse et les directives de « Java look and feel ». Cette implémentation vise à démontrer la faisabilité technique de l'architecture proposée et le mécanisme d'intégration.

Afin de valider et de démontrer l'applicabilité de l'approche, nous allons développer une étude de cas. Aussi, nous allons illustrer étape par étape comment une directive spécifique peut être incorporée dans un constructeur d'interface au moyen de composants d'interface usager. Plus spécifiquement, nous allons introduire un Wizard que nous avons développé afin d'appliquer les directives via un environnement de développement d'interface usager.

1.4 Organisation du mémoire et table des matières

Le mémoire est organisé comme suit:

Dans ce premier chapitre, nous avons mis en évidence le besoin pour intégrer l'utilisabilité dans le cycle de développement du logiciel. Aussi, nous avons discuté l'approche spécifique proposée tout en détaillant les points majeurs de ce mémoire.

Chapitre 2 "Directives de l'utilisabilité" présente les différentes techniques employées pour encapsuler (capturer) le savoir faire (l'expertise) de conception. Aussi, nous discuterons des limitations et des avantages dans l'application de patrons et directives dans le processus de développement.

Chapitre 3 "Outils de développement d'Interface personne-machine" répertorie les outils de développement d'interface usager existants. Ces outils se classent dans différentes catégories incluant les trousses à outils (Toolkits), les créateurs d'interfaces usagers, Java Beans et les environnements de développements intégrés. Nous allons énumérer les forces et les faiblesses de ces outils en terme d'intégration des directives de l'utilisabilité. Finalement, ce chapitre montrera une façon pour améliorer ces outils existants et pour faciliter l'intégration des directives de l'utilisabilité.

Chapitre 4 "Proposition d'architecture d'Intégration " introduit une approche permettant d'incorporer les directives l'utilisabilité dans les composants et les outils de développement d'interface usager. En particulier, nous allons présenter les détails pour créer chaque composant d'interface et l'influence de différents types de directives. De plus, l'outil que nous avons proposé, "Wizard d'aide à la demande pour la conception" est présenté.

Chapitre 5 "Exemple illustratif et Application" l'application pratique de notre approche sera illustrée par l'introduction d'un exemple. Nous allons introduire les différents niveaux de guide de conception que notre outil fournit. Nous allons montrer comment appliquer différentes directives relatives à une boite de dialogue. Aussi, le chapitre fournit certains détails techniques relatifs à notre framework expérimental: Java Eclipse et « Java Look and Fell » directives.

Finalement, le chapitre 6 résume notre travail et présente des avenues futures pour la recherche dans ce domaine.

CHAPITRE II

DIRECTIVES DE L'UTILISABILITE

Concevoir un système facilement utilisable est une mission difficile et complexe. Pour accomplir cette tâche, les concepteurs ont par conséquence besoin d'outils dédiés et efficaces. Un outil efficace doit être basé sur une expertise de conception éprouvée. Capturer le savoir de conception de systèmes utilisables réussis est important pour les concepteurs novices aussi bien qu'expérimentés [Martijn, 2000]. Les directives de l'utilisabilité sont une technique utilisée pour transmettre ce savoir. Ce chapitre présente, défini et détaille trois exemples de directives de l'utilisabilité les plus connues, tout en mettant en lumière certaines de leurs limites. De même, il met en évidence les motivations de notre travail de recherche.

2.1 Définition

Les directives pour les interfaces personne-machine sont conçues pour assister les concepteurs d'interfaces et les développeurs de logiciels dans la création d'interfaces faciles à apprendre et à utiliser. Généralement, elles fournissent de l'information sur la théorie derrière les éléments d'interfaces personne-machine "look and feel" et la pratique d'utilisation d'éléments de base d'interfaces (exemples: Boutons, Boites de Dialogue, etc). Aussi, elles décrivent comment concevoir une interface "look and feel" pour un environnement particulier, et améliorer la consistance et la standardisation entre les applications d'une plateforme.

En général, les directives définissent les éléments ainsi que le comportement de l'interface utilisateur tels que:

- Menus
- Fenêtres
- Boites de dialogues
- Contrôle
- Couleur
- Langage

De plus, il est important de savoir que, concernant les applications, les directives présentent des exemples de bonne et mauvaise utilisation des éléments et des comportements d'interfaces. Elles montrent comment combiner, les composants d'interface avec les comportements, l'esthétique et le langage, en vue de créer un produit de qualité.

Par ailleurs, les directives aident les concepteurs dans plusieurs directions, notamment [Maskery, 2000] :

- Fournir une expérience consistante et prédictible pour l'utilisateur final, réduisant ainsi le besoin de formation et de support.
- Réduire le travail de l'équipe de développement en permettant la réutilisation de la conception des composants.
- Permettre un transfert de connaissance et de compétence entre les équipes de développement.

2.2 Classification des directives

En général, les directives prennent une des deux formes suivantes:

- Les directives d'interfaces personne-machine qui fournissent un ensemble de règles indépendamment des plateformes, et un ensemble complet de directives dans différents aspects des interfaces, incluant les techniques de conception centrées sur l'utilisateur (user-centered-design) [Brown 1988; Heckel 1991; Smith, Mosier 1986].

- Les directives de style qui adressent le quand et le comment utiliser les widgets d'interfaces et les problèmes généraux "look and feel". Souvent, elles dépendent d'une plateforme particulière comme Macintosh [Apple Computer Inc. 1992], Windows [Microsoft Corporation 1992] ou [Sun Microsystems 2001].

2.2.1 Directives universelles

CASE (Computer Aided Software Engineering) est un terme utilisé pour décrire n'importe quel outil de développement de logiciels. Toutefois, il est souvent utilisé pour référer aux outils de développement qui utilisent des diagrammes lors de la conception de logiciels.

Les concepteurs de logiciels ont utilisé des diagrammes pour représenter leurs schémas de conception. Au fil du temps, la nature de ces diagrammes et les outils utilisés pour les produire ont beaucoup changé. Au début, tout comme les traitements de texte ont remplacé les machines à écrire, les outils ont remplacé le papier et les stylos. Cependant, beaucoup de ces outils CASE sont devenus inutiles parce qu'ils ne fournissent pas une valeur ajoutée aux concepteurs de logiciels [Livari, 1996]. Plus tard, les outils CASE ont ajouté la génération de code sophistiqué, la rétro-ingénierie, et des fonctions de contrôle de version. Ces fonctions ont ajouté de la valeur grâce à l'automatisation de certaines tâches de conception, par exemple, la conversion d'une conception en un squelette de code source. Toutefois, les outils CASE actuels présentent des lacunes pour faire face aux défis des concepteurs de logiciels [Jason Elliot Robbins, 1999].

IDC (International Data Corporation), une société d'études de marché qui recueille des données sur tous les aspects des équipements informatiques et des logiciels, a publié une série de rapports sur les outils CASE. IDC a prévu l'augmentation du taux de croissance annuel des revenus des outils CASE sur le marché mondial, composé de 54,6% par rapport à 127,4 millions de dollars en 1995 à 1,125.2 millions de dollars en l'an 2000 [IDC 1996].

La nécessité d'assister les concepteurs dans leur taches ne cesse d'augmenter. Cependant, la conception de logiciels n'est pas une simple tâche d'automatisation du processus de transformation de spécifications. Elle inclut aussi des tâches complexes de prise de décisions qui requièrent l'attention de concepteurs qualifiés. Les outils qui assistent les concepteurs dans la prise de décision constituent un bon moyen d'améliorer la productivité des concepteurs et la qualité de leur travail.

Aider les concepteurs à prendre des décisions est important parce que leurs décisions de conception vont fortement influencer les moyens de mise en œuvre et l'effort de maintenance nécessaires plus tard. Le soutien des concepteurs est également important car ils sont surchargés de travail et ont parfois la responsabilité de réaliser des tâches de conception dans des domaines pour lesquelles ils n'ont pas de formation appropriée.

En 1987, Guindon, Krasner, et Curtis ont identifié plusieurs difficultés rencontrées par les concepteurs de logiciels:

Les principales observations retenues étaient: (1) le manque de plans de conception spécialisée, (2) l'absence d'un méta-schéma du processus de conception conduisant à une mauvaise allocation des ressources aux différentes activités de conception, (3) la carence de la priorisation de problèmes conduisant à une mauvaise sélection de solutions alternatives, (4) la difficulté à considérer toutes les contraintes dans la définition d'une solution, (5) la difficulté à effectuer des simulations théoriques avec de nombreuses étapes ou cas de tests, (6) la difficulté d'assurer le suivi et le retour au sous-problème dont la solution a été reportée et (7) la difficulté d'étendre ou de fusionner des solutions de sous-problèmes individuels pour former une solution complète [Guindon, Krasner, et Curtis, 1987.

L'année d'après, Curtis, Krasner, et Iscoe (1988) ont conclu dans leur étude que la "faible connaissance du domaine d'application" était l'une des principales difficultés de développement des systèmes logiciels de grande taille. Cette faiblesse est une autre façon de dire qu'il est difficile pour un concepteur de posséder toutes les compétences dont il aura besoin pour réaliser une conception complexe. Même les experts dans un domaine restreint, devront regarder à l'extérieur de ce domaine pour parvenir à une conception complète.

Le simple fait que le niveau d'expertise peut varier d'une personne à une autre implique que les outils d'aide à la conception peuvent contribuer à pallier aux lacunes que les concepteurs peuvent avoir. Cette contribution peut prendre la forme d'explications, de règles, d'exemples, de modèles ou de suggestions. Si l'outil n'intègre pas des connaissances dans un domaine donné, il peut offrir des suggestions.

En plus de la complexité du domaine d'application, les outils de conception sont eux-mêmes difficile à manipuler. Les concepteurs doivent donc se familiariser avec ces outils afin de pouvoir les utiliser efficacement. Même si un outil donné est utilisé fréquemment, il peut contenir des fonctionnalités qui sont rarement exploitées.

Les directives d'interfaces utilisateur sont ce que beaucoup de gens pensent lorsqu'ils entendent cette phrase : "les facteurs humains pour les interfaces utilisateurs". Aussi, les directives sont souvent appelées le "bon sens " de la conception d'interfaces, l'utilisateur. Bien que cela soit vrai en théorie, les concepteurs semblent souvent perdre leur bon sens quand ils travaillent sur les interfaces utilisateur. Si les directives sont le bon sens, alors pourquoi ce type de bon sens fait défaut dans de nombreuses interfaces utilisateurs? [Deborah Hix and H. Rex Hartson John Wiley,1993]

La réponse à cette question réside dans le fait que les directives ne sont pas seulement le "bon sens " en question. En effet, l'application de ces directives à des situations spécifiques est beaucoup plus que du bon sens, car les directives sont souvent contradictoires. C'est pourquoi déterminer celle à appliquer et comment l'appliquer dans une situation particulière, nécessite une connaissance et une expertise beaucoup plus approfondies que celles de nombreux concepteurs d'interfaces utilisateur. Cependant, il reste que les directives fournissent une

grande partie des fondements pour la création des guides de styles (comme les sections 2.2.2).

En fait, les directives sont publiées dans des livres, des rapports et des articles qui sont accessibles au public. Elles ne sont pas spécifiques à une seule organisation, mais plutôt applicables à tous les types de conception d'interfaces utilisateur. Même si elles sont formulées de manière générale, une compréhension claire de leur conception fournit une aide précieuse. Les directives sont parfois empiriquement dérivées et / ou validées, mais souvent elles sont simplement une opinion d'expert fondée sur l'expérience. La différence majeure entre les directives et les normes est que ces dernières sont, du moins théoriquement, contraignantes dans une interface utilisateur, tandis que les directives servent plus comme suggestions sur la façon de concevoir une interface facilement utilisable [Deborah Hix and H. Rex Hartson John Wiley, 1993].

Un des meilleurs exemples, et d'ailleurs des plus connus, de recueil de directives est la compilation de Smith et Mosier [1986]. Cette collection de 944 directives, très étendue dans son champ d'application, a évolué sur plusieurs décennies. Elle est organisée autour de divers axes, tels que la saisie, l'affichage, la transmission et la protection des données et le contrôle de séquences. Chaque directive est soigneusement documentée, puis suivi par un exemple, les exceptions notables, des commentaires appropriés sur l'application de la directive et, enfin, une liste de références (livres, articles) et des renvois aux sections du document de Smith et Mosier.

Voici un exemple de ce document présentant des conseils sur la manière de concevoir le format d'affichage d'interfaces [Deborah Hix and H. Rex Hartson John Wiley, 1993]:

Consistent format

Adopt a consistent organization for the location of various display features from one display to another.

— *Example*: One location might be used consistently for a display title, another area might be reserved for data output by the computer, and other areas dedicated to display of control options, instructions, error messages, and user command entry.

— *Exception*: It might be desirable to change display formats in some distinctive way to help a user distinguish one task or activity from another, but the displays of any particular typeshould still be formatted consistently among themselves.

— *Comment*: The objective is to develop display formats that are consistent with accepted usage and existing user habits. Consistent display formats will help establish and preserve user orientation. There is no fixed display format that is optimum for all data handling applications, since applications will vary in their requirements. However, once a suitable format has been devised, it should be maintained as a patern to ensure consistent design of other displays.

— *Reference*: BB 1.1, 1.8.5, EG 2.2.5, 2.3, 2.3.3, MS 5.15.3.2.1, 5.15.3.3.4; Foley and Van Dam, 1982; Stewart, 1980; Taylor, McCan and Tuori, 1984. See also 4.0-6. [Smith & Mosier,1986, p. 170]

Malgré la rigueur des exemples, des explications et des références de Smith et Mosier, le document a néanmoins quelques limites. La plupart des directives sont fortement orientées vers les interfaces textuelles et alphanumériques, sans beaucoup d'attention consacrée aux interfaces graphiques. Aussi, même si une directive spécifique applicable à une situation de conception est disponible, il peut être difficile de la localiser. Le volume du document a conduit à cette difficulté, et ce, en dépit de l'existence de plusieurs logiciels interactifs qui

auraient pu faciliter la recherche d'information pour le genre de directives de Smith et Mosier.

D'un autre côté, les directives sont d'ordre général dans leur applicabilité et nécessitent beaucoup d'interprétation pour être utiles. Leur principal avantage c'est d'offrir un guide flexible et aider à établir des objectifs et des décisions de conception. Mais elles doivent être adaptées afin de produire des règles de conception spécifiques. Ces directives restent néanmoins une base importante pour la conception d'interfaces utilisateur.

De 1984 à 1986, l'armée américaine a compilé pour ces concepteurs d'interfaces utilisateur, l'expertise de l'utilisabilité existante en un seul ensemble de directives et organisé dans un livre de 478 pages. Cela semble beaucoup, mais ne l'est pas comparativement aux 1277 directives de l'utilisabilité identifiées pour le web et l'intranet. Il faut dire que cela n'est pas une compilation exhaustive [Jakob Nielsen 2005].

2.2.2 Directives de style

La création de directives de style peut servir d'activité centrale de coordination pour une équipe de conception intégrée impliquant les équipes de documentation, de développement, d'utilisabilité, de conception, de marketing, ainsi que d'autres groupes. De plus, les directives de style peuvent être un outil de formation très utile pour les nouveaux membres d'équipes de conception. Elles peuvent servir comme un outil pour assurer la cohérence et la consistance dans la conception de logiciels.

2.2.2.1 Directives d'Apple

Les directives d'Apple sont conçues pour assister les concepteurs dans le développement de produits qui fournissent aux utilisateurs du système d'exploitation Mac une consistance à travers les applications. Le système d'exploitation Mac combine un puissant noyau avec une interface conviviale appelée « Aqua ». En suivant ces directives, les développeurs peuvent assurer que [apple.com] :

- Les utilisateurs apprendront l'application plus rapidement, si l'interface ressemble et se comporte comme des applications qui leurs sont déjà familières.

- Les utilisateurs peuvent accomplir leurs tâches plus rapidement, car les applications bien conçues facilitent la tâche de l'utilisateur.
- Les utilisateurs avec des besoins spéciaux trouveront l'application plus accessible.
- L'application aura la même apparence que toutes les autres applications du système d'exploitation Mac.
- L'application sera plus facile à documenter, car les interfaces intuitives et les comportements standards ne nécessitent pas beaucoup d'explication.

Ces directives fournissent des détails spécifiques au sujet de la conception pour la conformité à Aqua dans le système d'exploitation. Aqua est l'ensemble des apparences et comportements du système d'exploitation Mac. Aqua définit l'apparence standard des composants spécifiques d'interface comme les fenêtres, les menus et les contrôles.

Le tableau suivant présente un ensemble d'exemples de directives de Mac :

Manipulation directe : permettre aux utilisateurs de se sentir en contrôle des objets présentés par l'ordinateur.
Voir-et-Pointer: les utilisateurs interagissent directement avec l'écran, en sélectionnant des objets et en exécutant des tâches à l'aide d'un pointeur.
Consistance : utiliser des éléments standards d'interface de Macintosh pour assurer la consistance dans l'application et bénéficier de la consistance à travers les applications.
Ce que vous voyez est ce que vous recevez "WYSIWYG (What You See Is What You Get)": assurer qu'il n'y a pas de différence significative pour l'utilisateur, par exemple, entre l'information affichée à l'écran et celle imprimée.
Contrôle de l'utilisateur : permettre à l'utilisateur, non à l'ordinateur, d'initier et de contrôler les actions.
Feedback et dialogue : quand un utilisateur initie une action, fournir des indications que l'application a reçu les entrées de l'utilisateur et qu'elles sont en cours de traitement.
Tolérance: les utilisateurs ont besoin d'avoir la possibilité de pouvoir essayer des actions ou fonctions sans causer des dommages au système.
Percevoir la stabilité : fournir un ensemble d'objets clair et fini et un ensemble d'actions à exécuter sur ces objets. Quand les actions ne sont pas disponibles, elles ne sont pas éliminées mais sont simplement cachées.
Intégrité de l'esthétique : concevoir les produits pour être plaisants et être vus à l'écran pour longtemps.

Tableau 2.1 Exemples de directives de Mac

2.2.2.2 Directives de Microsoft

Cette section présente les principes de base de conception d'interfaces pour les applications de Microsoft Windows. Aussi, elle explique les techniques et les méthodologies utilisées dans le processus de conception d'interfaces personne-machine. Les directives proposées sont classées en différentes catégories. Ce qui suit, introduit brièvement ces catégories en présentant certaines directives.

Utilisateur en contrôle

Un important principe de conception d'interfaces personne-machine est de permettre à l'utilisateur de se sentir en contrôle du logiciel plutôt que de se sentir être contrôlé par le logiciel. Ce principe a de nombreuses directives de conception :

- L'hypothèse opérationnelle est que les actions sont initiées par l'utilisateur non par la machine ou le logiciel. L'utilisateur joue un rôle actif et non réactif. Les tâches peuvent être automatisées. On doit implémenter l'automatisme de façon à permettre à l'utilisateur de choisir ou de contrôler.

- En raison de différences dans leurs compétences et préférences, les utilisateurs doivent avoir la possibilité de personnaliser les aspects de l'interface. Le logiciel doit refléter les paramètres des utilisateurs pour les différentes propriétés du système, comme les couleurs, les fontes ou autres options.

- Séparer les processus comme imprimer. Ainsi, pas besoin de charger l'application au complet pour exécuter une seule opération.

- Exécuter les processus longs en arrière plan pour garder ainsi le premier plan interactif. Par exemple, lors de l'impression d'un document, l'utilisateur doit avoir la possibilité de minimiser la fenêtre même si le document ne peut pas être altéré. Le multitâche dans Windows permet de définir des processus ou des threads distincts dans l'arrière plan.

Manipulation directe

Concevoir le logiciel de façon à permettre aux utilisateurs de manipuler directement l'information présentée par le logiciel. Les utilisateurs doivent voir comment leurs actions affectent les objets présents à l'écran, lorsqu'ils glissent un objet pour le relocaliser ou naviguent d'un emplacement à un autre dans un document. Aussi, l'information visible et les choix réduisent considérablement l'effort mental de l'utilisateur. Pour les utilisateurs, il est plus facile de reconnaître une commande que de se souvenir de sa syntaxe.

Les métaphores courantes fournissent un chemin intuitif pour l'accès direct. En permettant aux utilisateurs de transférer leurs connaissances et expériences, les métaphores rendent plus facile de prévoir et d'apprendre le comportement du logiciel. Lors de l'utilisation des métaphores, il n'est pas nécessaire de limiter l'implémentation d'un objet à son équivalent du monde réal. Par exemple, un classeur dans le bureau de Windows n'est pas comme son équivalent en papier; il peut être utilisé pour organiser une variété d'objets tels que les imprimantes, les calculateurs et d'autres classeurs. De plus, un classeur de Windows ne peut pas être trié de la même façon que son équivalent du monde réel.

Consistance

La consistance permet aux utilisateurs d'appliquer les connaissances déjà acquises à des nouvelles tâches et apprendre de nouvelles fonctions plus rapidement. Ceci est dû au fait que les utilisateurs n'ont pas besoin de consacrer du temps à essayer de mémoriser la diversité des interactions. En fournissant la stabilité et la consistance, on rend l'interface familière et prédictible.

La consistance est importante à travers tous les aspects de l'interface, incluant les noms des commandes, la présentation visuelle de l'information, le comportement opérationnel et le placement des éléments à l'écran.

Pour concevoir la consistance dans le logiciel, il est nécessaire de prendre en considération les principes et directives suivants :

- La consistance dans l'application. Représenter les fonctions courantes en utilisant des interfaces et un ensemble de commandes consistants. Par exemple, éviter d'implémenter la commande « Copie » qui exécute immédiatement une opération dans une situation, mais dans une autre, affiche une boite de dialogue nécessitant qu'un utilisateur spécifie une destination. Comme une conséquence de cet exemple, la même commande doit être utilisée pour accomplir les fonctions qui semblent similaires à l'utilisateur.

- La consistance dans l'environnement opérationnel. En maintenant un haut niveau de consistance entre l'interaction et les standards d'interfaces fournies par Windows, les utilisateurs pourront appliquer aux nouveaux logiciels les compétences interactives précédemment acquises.

Tolérance

Les utilisateurs aiment explorer une interface et souvent ils apprennent par essais et erreurs. Une interface efficace fournie un ensemble approprié de choix et avertit les utilisateurs sur les situations potentielles qui peuvent endommager le système ou les données. Plus encore, les actions offertes doivent être réversibles.

Même dans une interface bien conçue, les utilisateurs peuvent faire des erreurs. Les erreurs peuvent être physiques (accidentellement pointer une mauvaise commande ou donnée) ou mentales (sélectionner une mauvaise commande ou donnée). Une conception efficace évite les situations susceptibles de générer des erreurs.

Feedback

Il s'agit de fournir du feedback sur les actions de l'utilisateur le plus souvent possible. Un bon feedback aide à confirmer que le logiciel répond aux entrées et communique les détails qui distinguent la nature de l'action. Un feedback efficace est opportun et est présenté de manière la plus cohérente avec le contexte de l'interaction de l'utilisateur. Même pendant le

traitement d'une tâche particulière, la machine informe l'utilisateur sur l'état du processus et comment l'annuler si nécessaire. Rien n'est plus troublant pour l'utilisateur qu'un écran statique ne répondant pas aux entrées. Un utilisateur typique ne tolérera que quelques secondes une interface qui ne répond pas.

Aussi, il est important que le type de feedback utilisé soit approprié à la tâche exécutée. Il est possible de communiquer une simple information à travers les changements de pointeur ou une barre d'état avec message. Pour un feedback complexe, il peut être nécessaire d'afficher un contrôle de progression ou une boite de messages.

Esthétique

La conception visuelle est une partie importante de l'interface de l'application. Les attributs visuels fournissent des caractéristiques valorisantes et communiquent des signaux importants sur le comportement interactif des objets particuliers de l'interface. Aussi, il est important de se rappeler que chaque élément visuel apparaissant à l'écran, potentiellement attire l'attention des utilisateurs. Il s'agit de fournir un environnement cohérent qui contribue clairement à aider l'utilisateur à assimiler l'information présentée. Les compétences d'un infographiste ou d'un concepteur visuel peuvent être d'une utilité inestimable pour cet aspect de la conception.

Simplicité

Une interface doit être simple (non simpliste), facile à apprendre et à utiliser. Elle doit aussi fournir toutes les fonctionnalités de l'application. Une conception efficace assure l'équilibre de ces objectifs.

Une façon de supporter la simplicité est de réduire la présentation de l'information au minimum requis pour une communication adéquate. Par exemple, éviter les descriptions trop longues pour les noms de commandes ou de messages. Les phrases non pertinentes ou imprécises encombrent la conception et rendent difficile la tâche d'extraire l'information essentielle. Une autre façon de concevoir une interface simple et utile, est d'utiliser des

messages naturels. Le rangement et la présentation d'éléments affectent leurs significations et associations

Aussi, la simplicité est en corrélation avec la familiarité; souvent les choses connues semblent simples. Chaque fois que possible, il faut essayer de construire des liens s'approchant des connaissances et expérience des utilisateurs.

Le dévoilement progressif de l'information implique une organisation soigneuse ainsi qu'un affichage au moment approprié. La réduction de la quantité d'information présentée à l'utilisateur, réduit de facto la quantité d'information qu'il doit traiter. Par exemple, il faut utiliser les menus pour afficher les listes d'actions ou de choix, et utiliser des boites de dialogue pour afficher les ensembles d'options.

Le tableau suivant présente un ensemble d'exemples de directives pour les interfaces utilisateur de Microsoft.

Quand une commande n'est pas supportée par une fenêtre, n'afficher pas son bouton de commande
Le bouton de fermeture de fenêtre doit toujours apparaître comme le bouton le plus à droite. Aussi, laisser un espace entre celui-ci et les autres boutons.
Les boutons pour Minimiser et Maximiser une fenêtre doivent être toujours affichés respectivement l'un après l'autre.
Le bouton de restitution doit toujours remplacer les boutons utilisés pour Maximiser ou Minimiser la fenêtre quand cette commande est achevée.
Placer les commandes de transfert sur un côté et les ordonner comme suit: Couper, Copier, Coller et autre commandes spécialisées de Coller.
Quand la commande de propriétés est présente, il faut la placer comme la dernière commande dans le menu.
Utiliser un titre unique pour le menu à travers la barre et des items uniques pour le menu dans un menu individuel. Les noms d'items peuvent être répétés dans les différents menus pour présenter des actions similaires.
Garder bref le texte de menu d'une commande, mais présenter clairement sa fonctionnalité. Utiliser des mots uniques pour les barres et les titres de menus. Ne pas créer de mots composés artificiels.

Tableau 2.2 Exemples de directives pour les interfaces utilisateur de Microsoft.

2.2.2.3 Directives de "Java Look and Feel"

La plateforme de JAVA atteint sa maturité et les concepteurs ainsi que les développeurs reconnaissent le besoin en consistance, compatibilité et facilité d'utilisation des applications JAVA. «Java Look and Feel » remplit ce besoin en fournissant une apparence distinctive, indépendamment de la plateforme, et un comportement standard. L'utilisation de ce « Look and Feel » réduit la phase de conception, le temps de développement, le coût de la formation et de la documentation pour tous les utilisateurs.

« Java Look and Feel » est l'interface par défaut pour les applications construites avec la librairie JFC (Java Foundation Class) et le framework de Swing. « Java look and Feel » est conçue pour être utilisé sur différentes plateformes et elle fournit:

- Consistance dans l'apparence et le comportement des éléments courant de conception.
- Compatibilité des composants et des styles d'interactions personne-machine avec les standards de l'industrie.

Organisation Logique

Les applications qui utilisent « Java Look and Feel » se composent d'éléments d'interfaces personne-machine affichés dans des fenêtres. L'organisation d'une application en fenêtres et composants doit être consistante avec la division logique que les utilisateurs perçoivent de leurs tâches. L'organisation logique est importante, spécialement dans des applications qui affichent plusieurs objets dans différentes fenêtres. Par exemple, une application pour la gestion d'un grand réseau peut contenir:

- Des fenêtres qui affichent un ensemble de domaines de réseau
- Des vues (comme les icônes ou les entrées de tables) de chacun des nœuds de domaine
- Des vues de propriétés de chaque nœud (par exemple, son adresse de réseau)

La plateforme JAVA fournit plusieurs types de fenêtres conçues pour différents genres d'interactions. L'information dans une fenêtre se compose d'éléments (et leurs propriétés) qui permettent aux utilisateurs d'accomplir des actions ou de rapporter l'information sur des actions. Les fenêtres principales, secondaires et utilitaires fournissent le conteneur de plus haut niveau d'une application. Une fenêtre principale est une fenêtre dans laquelle l'interaction principale de l'utilisateur avec les données ou le document prend place. Une application peut utiliser un nombre illimité de fenêtres principales, qui peuvent indépendamment être ouvertes, fermées ou redimensionnées. Une fenêtre secondaire est une fenêtre de support qui dépend d'une fenêtre principale (ou une autre fenêtre secondaire). Une fenêtre utilitaire est une fenêtre dont le contenu affecte une fenêtre principale active. De plus, les fenêtres secondaires et utilitaires restent ouvertes quand les fenêtres principales sont fermées ou minimisées.

Pour organiser l'information dans une application, le développeur doit répondre aux questions suivantes:

- Est-ce que l'information doit être affichée dans la fenêtre principale ou secondaire ?
- Dans quel type de fenêtre va chaque information?
- Comment les différents types de fenêtres sont intitulés?

Dans une fenêtre, l'utilisabilité dépend souvent de l'organisation logique des menus. Un menu affiche une liste d'options (items du menu) permettant aux utilisateurs de choisir ou de naviguer. Typiquement, les menus sont logiquement regroupés et affichés par l'application afin d'éviter à l'utilisateur de mémoriser toutes les commandes disponibles. Les menus dans « Java Look and Feel » peuvent apparaître statiques, et restent affichés sur l'écran après être sectionné par l'utilisateur. En général, le principal rôle des menus est de permettre d'accéder aux fonctionnalités de l'application, mais aussi de fournir aux utilisateurs un moyen rapide de voir ce que ces fonctionnalités font.

Il y a trois types de menus dans les applications « Java Look and Feel »: les menus déroulants, les sous-menus et les menus contextuels. Un menu déroulant est un menu dont le

titre s'affiche dans la barre de menu. Un sous-menu apparait adjacent à un item de menu dans un menu déroulant; sa présence est indiquée par une flèche à coté de l'item. Un menu contextuel affiche les listes de commandes, propriétés ou attributs qui s'appliquent à l'item sélectionné ou les items sous le pointeur. Les menus contextuels peuvent aussi avoir des sous-menus.

Dans la plupart des applications, les éléments de menu doivent avoir ces caractéristiques et le but de chacune est de promouvoir l'utilisabilité.

- Quand une fenêtre est à sa taille par défaut, assurer que tous les titres de ses menus déroulants s'affichent sur une seule ligne dans la barre du menu sans être déformés.
- S'assurer que le titre d'un menu déroulant se compose exactement d'un seul mot.
- Dans les menus déroulants, assurer que le libellé de chaque item diffère du titre du menu.

Adaptabilité

Des fois, les applications ont besoin d'afficher un nombre varié d'objets d'interfaces utilisateur. Par exemple, dans une application qui contrôle les ordinateurs d'une corporation, le nombre d'objets représentant les ordinateurs appartenant à un site particulier peut augmenter rapidement. Lors de la recherche d'un objet particulier dans une fenêtre représentant ce site, un utilisateur peut avoir besoin de voir 15 objets dans un mois et 1500 dans le prochain. L'interface utilisateur d'une telle application doit être adaptable. En d'autres mots, l'utilisateur doit avoir la possibilité de trouver, voir et manipuler un grand nombre d'objets.

Le concepteur d'interfaces personne-machine, fait plus qu'assembler les composants de l'interface dans une fenêtre. Il les présente pour aider les utilisateurs à comprendre les tâches à accomplir et à adopter un enchainement naturel à travers les actions. Une bonne conception d'interfaces, passe souvent inaperçu car tout fonctionne comme prévu. Les utilisateurs remarquent les problèmes de conception de l'application si des obstacles les empêchent

d'accomplir leurs tâches. En prêtant une attention particulière aux détails, et aux tests avec des utilisateurs réels on peut aider à éliminer ces difficultés.

Dans certaines applications, l'effet des actions des utilisateurs diffère suivant les situations ou les modes définies dans l'application. Souvent, un mode laisse les utilisateurs accomplir seulement certaines actions. Par exemple, dans un mode d'une application de dessin, cliquer sur un objet après un autre peut sélectionner ces objets. Refaire la même action dans un autre mode de la même application, peut dessiner une ligne entre les objets. En d'autres mots, la même action peut avoir différents effets dans différents modes.

En limitant les actions potentielles des utilisateurs, les modes facilitent aux développeurs la traduction de ces actions en code. Les applications avec modes peuvent être difficiles à utiliser car :

- Les utilisateurs doivent mémoriser le mode en vigueur. Si les utilisateurs ignorent le mode en cours ou s'ils oublient de le changer les effets de leurs actions vont être incorrects

- Les utilisateurs doivent passer d'un mode à un autre, ce qui nécessite des actions supplémentaires sur la souris ou le clavier

D'un autre côté, les modes peuvent aider les utilisateurs, en les prévenant d'exécuter accidentellement des actions non voulues, par exemple, l'activation d'un bouton de commande lors d'un changement de fenêtres.

Prédictibilité

Pour apprendre les nouvelles fonctionnalités d'une application, les utilisateurs comptent souvent sur leurs expériences précédentes avec d'autres fonctions de l'application. Même une légère inconsistance entre l'apparence et le comportement des différentes fonctionnalités peut frustrer les utilisateurs et réduire leur productivité.

L'une des principales caractéristiques d'une interface personne-machine bien conçue est la consistance entre ses modules. Cette caractéristique aide les utilisateurs à apprendre plus

facilement des fonctionnalités similaires de l'interface. Cela peut aider les utilisateurs à se familiariser plus rapidement avec l'interface de l'application. Ceci peut se faire en créant les modules similaires à partir du même ensemble de composants de JFC, tout en implémentant les mêmes patrons ou idiomes.

Dans les interfaces utilisateurs, un idiome est un ensemble de composants configurés d'une façon standard pour fournir une apparence et un comportement particulier.

Si les idiomes sont présentés d'une façon consistante dans toute l'application, les utilisateurs arrivent à reconnaitre chaque idiome, même dans de nouveaux contextes, et peuvent correctement prédire ce que chaque idiome permet de faire. Un exemple d'idiome très utilisé est l'idiome pour naviguer présenté à la figure 2.1 :

Figure 2.1 Exemple d'un idiome: Idiome pour naviguer

L'idiome de navigation permet aux utilisateurs de taper ou de choisir le nom d'un objet existant, comme un fichier. Cet idiome se compose toujours d'un libellé, un champ de texte éditable et un bouton de commande. Chaque fois que les utilisateurs voient l'idiome de navigation, ils savent que le champ de texte peut être rempli en tapant un nom ou en cliquant le bouton de commande, pour choisir le texte à partir d'une liste.

Efficacité

L'application doit être efficace pour optimiser son utilisabilité. L'organisation logique, l'adaptabilité et la prédictibilité d'une application contribuent à son efficacité. L'efficacité est importante, spécialement si les tâches des utilisateurs sont complexes et longues. Un outil d'aide comme un wizard peut assister les nouveaux utilisateurs inexpérimentés afin de pouvoir travailler efficacement.

Dans les applications qui contrôlent et gèrent des systèmes en temps réel, comme les gros systèmes d'ordinateurs et des réseaux, la capacité de l'utilisateur à répondre efficacement aux alertes peut prévenir des pannes majeures.

Même dans un logiciel bien conçu, les tâches complexes ou inhabituelles peuvent être difficiles. L'accomplissement de ce type de tâches peut devenir facile et rapide en fournissant un type d'interface utilisateur connu comme un wizard.

Un wizard est une fenêtre qui guide un utilisateur à travers une tâche, étape par étape en demandant une série de réponses à l'utilisateur, et en accomplissant la tâche basée sur ces réponses. Mise à part pour les réponses d'utilisateur, un wizard fournit toutes les informations nécessaires à la réalisation de la tâche. Typiquement, les wizards sont proposés pour simplifier la réalisation des tâches, donc même les utilisateurs novices peuvent accomplir facilement une tâche complexe. Souvent, les wizards simplifient et accélèrent la réalisation de tâches. Un wizard se compose d'une série de pages dans une fenêtre. Chaque page représente une étape ou la portion d'une étape de la tâche d'un utilisateur. La figure ci-dessous présente une page typique.

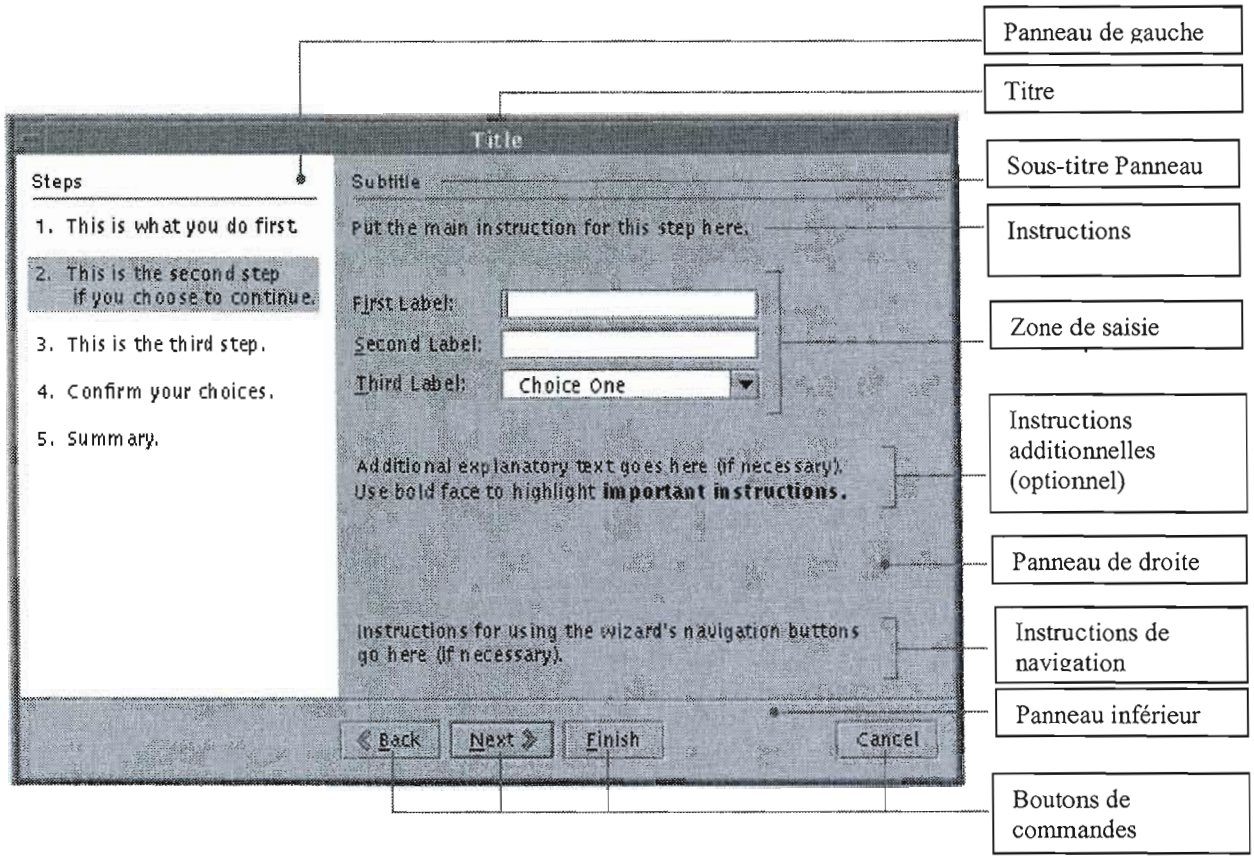


Figure 2.2 Anatomie d'une fenêtre d'un wizard

Chaque page se compose d'une barre de titre et de trois sections, droite, gauche et inférieure comme le montre la figure ci-dessus. Un wizard peut avoir plusieurs types de pages, tels qu'une page de recueil des données de l'utilisateur et une page qui résume les résultats. Cependant, seulement les pages d'entrées de données de l'utilisateur sont nécessaires dans chaque wizard. Les autres types de pages sont optionnels ou sont requises seulement sous certaines conditions.

La table suivante présente un ensemble d'exemples de directives de « Java Look and Feel ».

Puisque les fenêtres utilitaires ne dépendent pas de la fenêtre principale, éviter de fermer automatiquement les fenêtres utilitaires quand les fenêtres principales sont fermées.
Afficher une liste déroulante horizontale si les utilisateurs ne peuvent pas voir toute l'information dans le panneau de la fenêtre. Par exemple, dans une application de traitement de texte qui prépare les pages imprimées, les utilisateurs doivent pouvoir voir les marges et le texte aussi.
Fournir des raccourcis permettant aux utilisateurs de naviguer d'un onglet à un autre, et des onglets aux panneaux en utilisant les opérations du clavier.
Si un panneau d'onglets nécessite de multiples lignes d'onglets, considérer la répartition du contenu entre plusieurs boites de dialogues ou autre composants. Aussi, considérer l'affichage vertical des onglets pour que plusieurs lignes puissent être affichées dans une seule colonne.
Afficher une boite de dialogue de progression (ou une barre de progression ailleurs dans l'application) si une opération prend plus de 6 secondes.
Utiliser des titres de menu facilitant aux utilisateurs de déterminer le menu qui contient les items qui les intéressent. Par exemple, le menu de format contient les commandes permettant aux utilisateurs de changer le format de leurs documents ou données.
Afficher des raccourcis dans le texte du bouton, à l'exception des boutons de commandes «Défaut » et « Annuler » dans les boites de dialogue.
Afin de rendre les boutons de commandes sans texte plus accessibles, créer des messages d'aide qui décrivent ou nomment les fonctions des boutons.
Utiliser des phrases majuscules dans le texte d'un libellé qui communique le statut. Ne pas fournir le point de ponctuation de fin sauf si le texte est une phrase complète.

Tableau 2.3 Exemples de directives de « Java Look and Feel »

2.3 Avantages et limites des directives

Les directives de l'utilisabilité sont devenues une approche reconnue pour supporter le processus de développement d'interfaces personne-machine. Cependant, cela ne veut pas dire que les directives peuvent remplacer les règles d'or de conception d'interfaces comme l'implication et le feedback de l'utilisateur à partir du début du prototype, ainsi que le développement itératif [Gould 1985]. En effet, les directives peuvent améliorer considérablement la qualité des étapes itératives, participant à une amélioration de la qualité et à une réduction du nombre d'itérations impliquées dans le cycle du développement [Strong 1994].

Néanmoins, il est souvent rapporté que l'utilisation des directives implique certains problèmes [Martijn et al., 2000]. Il est difficile de sélectionner les directives appropriées à un problème de conception spécifique. Aussi, d'autres questions sont restées sans réponses comme, quand un composant (widget) particulier doit être utilisé ou comment les éléments de l'interface s'intègrent à l'ensemble. De plus, Tidwell a souligné qu'il était difficile de mémoriser toutes ces directives pour un concepteur novice [Tidwell, 1999]. Des fois, il est difficile de faire la distinction parmi ces principes quand ils rentrent en conflit. Alors, le concepteur peut ne pas pourvoir résoudre le problème de conception ou bien, il doit trouver la meilleure solution par supposition, ou recours à d'autres moyens.

Tous les travaux de recherche précédents sont centrés sur le contenu et la structure des directives de conception. Des versions de directives en ligne sont aussi créées mais utilisent un système simple basé sur des hypertextes pour l'accès aux directives. Peu de réflexions ont été avancées pour définir quand les directives sont applicables ou comment peuvent-elles être redéfinies pour répondre aux exigences de l'utilisabilité, pour un type d'application spécifique. De plus, le nombre élevé de directives de l'utilisabilité a compliqué la tâche du concepteur. C'est difficile de décider quelles directives sont appropriées à un problème de conception donné. Comme d'autres, nous indiquons que ces problèmes sont dus principalement au manque d'outils d'aide pour le développement de logiciel avec une interface utilisateur complexe. La plupart des méthodes de génie logiciel ne fournissent aucun mécanisme pour incorporer explicitement les directives d'interfaces utilisateurs dans les méthodes de conception et de validation de la conformité de l'utilisabilité au grand

ensemble de directives qui peuvent s'appliquer. Peu de travaux de recherche ont été fait pour automatiser la propagation de l'expertise de conception, par l'incorporation des propriétés de l'utilisabilité et des directives dans les composants d'interfaces personne-machine et d'environnement de développement. Dans ce mémoire, nous explorons cette avenue.

CHAPITRE III

OUTILS DE DEVELOPPEMENT D'INTERFACES PERSONNE-MACHINE

Dans ce chapitre, nous analysons les outils de développement d'interfaces graphiques personne-machine existants et étudions leurs limites en terme d'intégration et de support de l'utilisabilité. Aussi, nous étudions d'autres types d'outils incluant des dépotoirs de directives en ligne, ainsi que les beans de Java et la technologie basée sur les composants. Ensuite, nous allons discuter le besoin pour la création d'un nouvel outil pour améliorer et faciliter l'utilisation des directives de l'utilisabilité. Ceci peut-être réalisé en incorporant les directives de conception de l'utilisabilité dans les composantes et l'environnement de développement d'interfaces usagers, ce qui est le principal objectif de ce mémoire. Spécifiquement, nous allons explorer les possibilités d'intégration de l'utilisabilité en utilisant les Plug-Ins dans un environnement spécifique de Java: l'environnement Eclipse.

3.1 Introduction aux outils de développement d'interfaces personne-machine

Actuellement, une grande variété d'outils de développement d'interfaces usagers est disponibles. Ces outils visent à simplifier le codage des interfaces utilisateurs d'applications complexes en fournissant aux développeurs des blocs préconçus nommés « widgets », composants ou contrôles. Ces blocs sont manipulés visuellement par le développeur dans une interface usager consistante, permettant ainsi à une petite équipe de développer un grand nombre d'interfaces en une courte période. Les outils de développement d'interfaces usagers améliorent l'utilisabilité en fournissant à l'équipe de développement la possibilité de prototypage; les changements proposés peuvent être rapidement montrés à l'utilisateur final pour assurer la validation et l'acceptation des spécifications. Cet aspect peut diminuer le temps requis par les changements de l'interface utilisateur (IU) lors de la phase d'opération et

de maintenance. Cependant, les pratiques quotidiennes démontrent que cet objectif n'est pas totalement atteignable.

Ainsi, plusieurs organisations de développement proposent de personnaliser le style et les directives générales de l'utilisabilité dans le but de les intégrer dans des widgets et des constructeurs d'interfaces utilisateurs graphiques (IUG) [Billingsley, 1995; Rosenweig 1996]. Cependant, même avec ces approches personnalisées, il y a un certain nombre de préoccupations quant à la nature des directives qui peuvent réduire l'impact sur l'utilisabilité des interfaces utilisateurs [Gould et al. 1991; Souza, Bevan, 1990]. En effet, les outils de développement actuels présentent un manque dans l'intégration des directives de l'utilisabilité en tant que partie intégrante du processus de développement. En général, le problème de l'utilisabilité est relégué à un rôle secondaire au cours du développement, et dans plusieurs situations, un simple avis humain intervient dans le processus de certification.

Actuellement, les outils de développement d'interfaces disponibles sur le marché peuvent être classés selon différentes catégories en termes de fonctionnalités brutes, comme les trousses à outils « Toolkits », les constructeurs IU, les Beans de Java et les environnements de développement intégrés. Chaque catégorie d'outils est détaillée dans les sections suivantes.

3.2 Trousse à outils « Toolkits » pour IU

La plupart des logiciels actuels sont développés en utilisant le modèle d'interfaces utilisateurs graphiques (IUG). Ce modèle fournit un ensemble de contrôles aussi connus sous le nom de "widgets", qui composent l'interface utilisateur. Les contrôles incluent les champs de texte modifiables, des cases à cocher, des boutons radio, des curseurs, des barres de défilement, des boutons, des compteurs et divers types de menus.

Certains outils de développement d'interfaces utilisateurs sont appelés des toolkits; ils sont essentiellement des bibliothèques de contrôles d'interfaces graphiques. D'autres sont appelés "systèmes de gestion d'interface utilisateur" (SGIUs). Ce sont des infrastructures d'applications préconstruites qui fournissent non seulement des composants d'interfaces utilisateurs, mais aussi du support lors de l'exécution pour la communication entre ces composants et les modules de l'application. Certains toolkits d'IU et SGIUs fournissent des

programmes "constructeurs" qui permettent de créer des interfaces visuellement et interactivement, en glissant des éléments dans la position désirée au moyen de la souris ou du clavier. D'autres nécessitent que les développeurs décrivent l'interface en utilisant un langage de programmation (Zarmer et Johnson, 1990).

L'idée de créer des applications par la combinaison dynamique des composants (contrôles) écrits et compilés séparément dans le système a été démontrée (Palay, AJ, et al. 1988) au centre de technologie de l'information de l'université Carnegie Mellon. Chaque composant contrôle son rectangle sur l'écran où d'autres composants peuvent être incorporés. La principale raison du succès du modèle de composants est qu'il traite l'aspect de l'importance et de l'utilité du développement d'applications: comment partitionner le logiciel en petits modules, tout en offrant d'importantes capacités d'intégration pour les utilisateurs.

Exemple de Toolkits : Swing de Java

Swing est un toolkit d'interfaces graphiques du langage de programmation Java et une partie de « Java Fondation Classes » (JFC). Il inclut des widgets d'interfaces graphiques tels que les boîtes de texte, les boutons et les tableaux.

Les Widgets de Swing fournissent de meilleurs écrans d'affichage que le Toolkit de fenêtrage précédent. Puisqu'ils sont écrits en langage Java pur; ils s'exécutent de la même façon sur toutes les plateformes. Ils assurent l'uniformité au «Look and Feel » non pas en utilisant les installations de la plateforme native, mais par émulation de leur environnement. Cela signifie qu'il est possible d'obtenir un « Look and Feel » standard sur différentes plateformes. L'inconvénient de ces composants est la lenteur de leur exécution. L'avantage est l'uniformité de leur comportement sur toutes les plateformes.

3.3 Outil de création d'interfaces graphiques « UI Builder Tools »

Une autre contribution importante de la recherche dans le domaine des interfaces personne-machine a été la création des constructeurs d'interfaces utilisateurs graphiques (IUG). Ces constructeurs sont des outils mis au point pour améliorer la productivité des équipes de développement, et pour réduire le coût du codage de l'interface dans les phases de développement et de maintenance. Une étude a révélé qu'en moyenne 48% du code d'une

application est consacré à l'interface utilisateur, et 50% du temps de développement nécessaire à l'application entière est consacré à la portion de l'interface utilisateur [Myers 95]. L'utilisation des constructeurs d'IUG peut réduire considérablement ces chiffres. Par exemple, il a été rapporté que le système d'Apple a réduit le temps de développement par un facteur de quatre. Une autre étude a révélé qu'une application utilisant un outil IUG populaire peut réduire jusqu'à 83% les lignes de code écrites et prend la moitié du temps par rapport à des applications écrites sans outil d'interfaces graphiques [Myers 95]. L'étude originale a été effectuée à Stanford, Xerox Palo Alto et MIT, dans les années 1970 [Myers 95].

Visual Basic et "les éditeurs de ressources" qui viennent avec Microsoft Visual C++ sont parmi les outils les plus populaires. Trillium de Xerox PARC (Henderson) et Menu Lay de l'Université de Toronto (Buxton) sont parmi les premiers outils étudiés dans cette catégorie d'outils. Par la suite l'idée a été raffinée par Jean-Marie Hullot et un chercheur de l'INRIA puis Hullot, a ramené l'idée avec lui chez NeXT.

Une raison importante du succès du constructeur d'interfaces utilisateurs a été l'utilisation des moyens graphiques pour exprimer des concepts graphiques (par exemple: la mise en page interface). En déplaçant certains aspects de l'implémentation d'interfaces utilisateurs, du code conventionnel vers un système interactif de spécification, on a grandement facilité la tâche des programmeurs moins expérimentés.

Ceci a permis à de nombreux experts d'un domaine de créer des prototypes et d'implémenter des interfaces très adaptées à leurs tâches. Cela a permis aussi aux professionnels de la conception visuelle de s'impliquer davantage dans la création de l'apparence des interfaces.

Exemple d'outil de création d'interfaces utilisateurs: Visual Basic

Visual Basic est un outil graphique. Il permet de créer directement des fenêtres, des menus, des boutons, etc. Reste à écrire le code pour le traitement des messages et des événements du système. L'inconvénient majeur de Visual Basic est qu'il faut disposer des add-ons pour implémenter des fonctions avancées. Les add-ons sont appelés fichiers VBX.

Les composants de base de l'interface sont présentés dans la figure suivante :

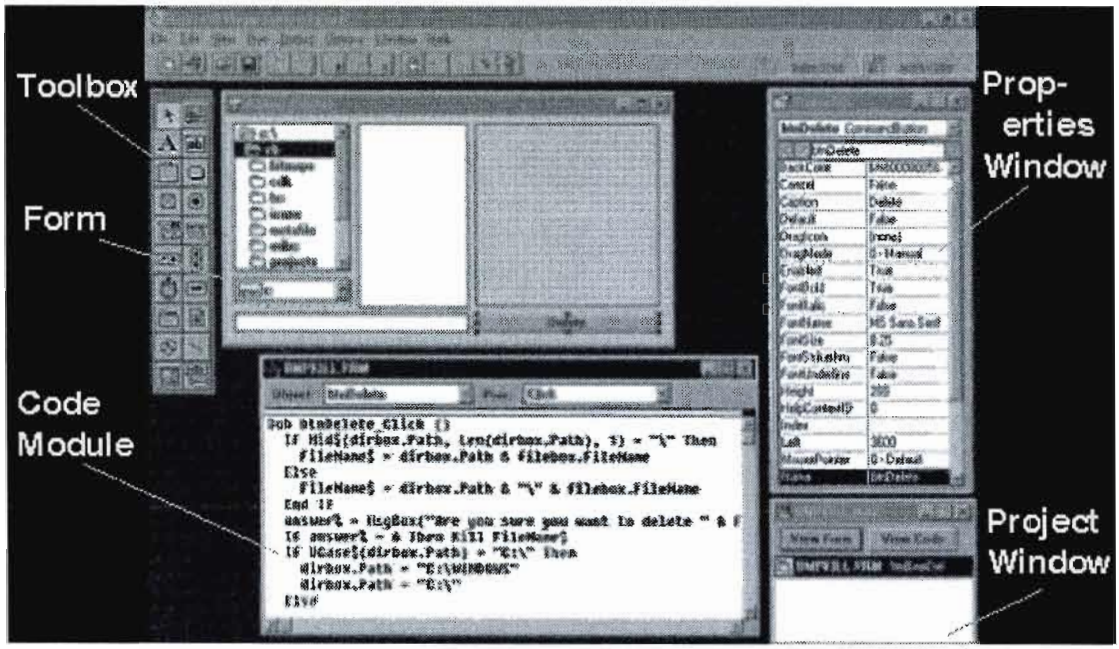


Figure 3.1 Fenêtre d'une interface de base [http://www.cc.gatech.edu/classes/cs6751_97 _winter/ Topics/gui-builder/]

- Boîte à outils « Toolbox » : tous les contrôles VBX sont contenus dans la barre d'outils. Les boutons de la barre d'outils sont utilisés pour dessiner des contrôles sur le formulaire.

- Formulaire « Form » : ceci est ce que l'utilisateur voit. Il contient des contrôles, et ses actions sont définies par le code.

- Module de code : le code des contrôles est défini dans le module de code. Notez les deux zones au sommet du module de code. Celle de gauche montre l'objet dont le code est en cours de définition. Celle de droite montre la procédure, ou l'événement du contrôle qui est en cours de définition.

- Fenêtre des propriétés : les propriétés d'un contrôle sont définies dans cette fenêtre, comme la couleur, les dimensions et les opérations.

- Fenêtre de projet : ceci est une représentation du fichier .mak qui définit quels sont les fichiers à utiliser dans le programme. Chaque formulaire, base de module de code et le VBX utilisé avec le fichier .mak sont affichés dans cette fenêtre

3.4 Beans de Java

Qu'est-ce que Beans de Java?

Beans de Java est un composant logiciel qui définit comment les objets seront développés et la façon dont ils peuvent communiquer les uns avec les autres. Cela permet aux développeurs de construire rapidement et de personnaliser une application en reliant ensemble différents objets, de la même manière que des Legos sont empilés ensemble pour construire une maison.

Outre le gain de temps et d'argent, cela permet de développer des applications plus cohérentes et plus fiables. Comme d'autres outils de développement d'interfaces utilisateur tels que Visual Basic et ActiveX, Beans de Java repose largement sur le modèle d'interfaces utilisateurs graphiques qui fournissent les caractéristiques des objets et comment ils sont liés. De plus, Beans de Java permet au développeur d'ajouter, de supprimer et de modifier des objets pour personnaliser une application. Les liens entre les objets dans l'application se rétablissent automatiquement.

Qu'est-ce qu'un Bean?

Selon l'équipe de Sun, Beans sont des composants logiciels réutilisables qui peuvent être manipulés visuellement au moyen d'un outil [http://java.sun.com]. Par exemple, pour construire un ordinateur, il ne faut pas concevoir un contrôleur de disque, un contrôleur graphique et un clavier. Au lieu de cela, il suffit d'acheter les composants d'un fabriquant et les assembler. Maintenant, dans le développement de logiciels, le même principe peut s'appliquer avec des composants Bean de Java.

Ainsi, au lieu de créer des applications monolithiques qui prennent des années à être développées et ne sont plus à jour, même avant d'être déployées, l'interface utilisateur peut être créée à partir de Beans.

Que contient un Bean?

Un Bean est défini par sa structure qui doit se conformer à un ensemble de règles de conception. Un outil de développement comme JBuilder offre tout ce qu'il faut savoir sur un Bean et permet son utilisation. Les Beans ont besoin de savoir comment gérer les événements. Lorsque quelque chose se passe, les Beans doivent être en mesure de réagir et faire en sorte que les événements se produisent. Un Bean doit être configurable visuellement, sinon un outil de développement ne peut pas fonctionner correctement. Les Beans doivent également être en mesure de sauvegarder et de restaurer leurs états. La première chose à examiner à l'intérieur d'un Bean concerne ses propriétés. Chaque propriété est appelée attribut du Bean. Lorsqu'un outil de développement est utilisé, un éditeur de propriétés permet au développeur d'applications de modifier les propriétés. La deuxième caractéristique d'un Bean est ses méthodes. Elles offrent des moyens pour manipuler un Bean et l'accès aux propriétés. Un outil de développement aura accès à toutes ou à un sous-ensemble de méthodes publiques d'un Bean. Finalement, il y a les événements qui servent de mécanisme de notification entre les Beans.

Personnalisation, introspection et persistance

Quand un Bean est créé, ses capacités sont publiées. Ceci expose les noms de tous les événements et propriétés supportés. Le processus de publication est appelé personnalisation. Quand un Bean est réutilisé dans un outil de développement, l'outil a besoin de découvrir ce qui a été publié. Ce processus de découverte est appelé introspection.

Pour que les Beans soient utiles, ils ont besoin d'un moyen de conserver leurs états. Si le même Bean est réutilisé dans 20.000 applications sans persistance, le Bean sera le même dans les 20.000 applications. Cependant, avec la persistance, il est possible de personnaliser chaque Bean à être différent pour chaque application. Ensuite, après que le Bean ait été configuré (à l'extérieur de l'application), il peut être sauvegardé séparément et réutilisé pour chaque instance de l'application.

Exemple: un Bean de Java qui affiche une boîte de dialogue.

```
// This example is from the book "Java in a Nutshell, Second Edition".
 public class YesNoDialog {
 protected String message, title;
 protected String yesLabel, noLabel, cancelLabel;
 protected int alignment;

 // Methods to query all of the bean properties.
 public String getMessage() { return message; }
 public String getTitle() { return title; }
 public String getYesLabel() { return yesLabel; }
 public String getNoLabel() { return noLabel; }
 public String getCancelLabel() { return cancelLabel; }

 // Methods to set all of the bean properties.
 public void setMessage(String m) { message = m; }
 public void setTitle(String t) { title=t; }
 public void setYesLabel(String l) { yesLabel = l; }
 public void setNoLabel(String l) { noLabel = l; }
 public void setCancelLabel(String l) { cancelLabel = l; }
 protected Vector listeners = new Vector();
 public void addAnswerListener(AnswerListener l) {
  listeners.addElement(l);}
  public void removeAnswerListener(AnswerListener l) {
  listeners.removeElement(l);}
 /** Send an event to all registered listeners */
 public void fireEvent(AnswerEvent e) {
```

```
// Make a copy of the list and fire the events using that copy.
  Vector list = (Vector) listeners.clone();
  for(int i = 0; i < list.size(); i++) {
    AnswerListener listener = (AnswerListener)list.elementAt(i);
    switch(e.getID()) {
    case AnswerEvent.YES: listener.yes(e); break;
    case AnswerEvent.NO:  listener.no(e); break;
    case AnswerEvent.CANCEL: listener.cancel(e); break; }}}
  public YesNoDialog() {
  this("Question", "Your\nMessage\nHere", "Yes", "No", "Cancel",
  LEFT);}
 /** A constructor for programmers using this class "by hand" */
 public YesNoDialog(String title, String message,
                    String yesLabel, String noLabel, String cancelLabel,
                    int alignment) {
  this.title = title;
  this.message = message;
  this.yesLabel = yesLabel;
  this.noLabel = noLabel;
  this.cancelLabel = cancelLabel;
  this.alignment = alignment;}
 /** This method makes the bean display the dialog box */
 public void display() {

// Create a frame with the specified title.
  final Frame frame = new Frame(title);
  frame.setLayout(new BorderLayout(15, 15));


  // Put the message label in the middle of the window.

  frame.add("Center", new MultiLineLabel(message, 20, 20, alignment));
```

```
// Create an action listener for use by the buttons of the dialog.
ActionListener listener = new ActionListener() {
 public void actionPerformed(ActionEvent e) {
  frame.dispose();    // pop down window
  if (listeners != null) {     // notify any registered listeners
   String cmd = e.getActionCommand();
   int type;
   if (cmd.equals("yes")) type = AnswerEvent.YES;
   else if (cmd.equals("no")) type = AnswerEvent.NO;
   else type = AnswerEvent.CANCEL;
   fireEvent(new AnswerEvent(YesNoDialog.this, type));}}};

// Create a panel for the dialog buttons
Panel buttonbox = new Panel();
buttonbox.setLayout(new FlowLayout(FlowLayout.CENTER, 25, 15));
frame.add("South", buttonbox);

// Create each specified button, specifying the action listener
if ((yesLabel != null) && (yesLabel.length() > 0)) {
 Button yes = new Button(yesLabel);       // Create button.
 yes.setActionCommand("yes");             // Set action command.
 yes.addActionListener(listener);      // Set listener.
 buttonbox.add(yes); }                 // Add button to the panel.
  if ((cancelLabel != null) && (cancelLabel.length() > 0)) {
 Button cancel = new Button(cancelLabel);
 cancel.setActionCommand("cancel");
 cancel.addActionListener(listener);
 buttonbox.add(cancel);}
 frame.show();}
```

```
/*** A main method that demonstrates how to use this class*/
  public static void main(String[] args) {
    YesNoDialog d =
      new YesNoDialog("YesNoDialog Test",
                      "There are unsaved files.\n" +
                      "Do you want to save them before quitting?",
                      "Yes, save and quit",
                      "No, quit without saving",
                      "Cancel; don't quit",
                      YesNoDialog.CENTER);

    // Register an action listener for the dialog.
    d.addAnswerListener(new AnswerListener() {
      public void yes(AnswerEvent e) { System.out.println("Yes"); }
      public void no(AnswerEvent e) { System.out.println("No"); }
      public void cancel(AnswerEvent e) { System.out.println("Cancel"); }});

    // Now pop the dialog up.  It will pop itself down automatically.
    d.display();}}
```

3.5 Environnement de développement intégré (EDI)

Un environnement de développement intégré est un environnement de programmation qui a été empaqueté. En général, il se compose d'un éditeur de code, un compilateur, un débogueur, une trousse à outils, un outil de gestion de configuration, un composant de navigation et un constructeur d'interfaces utilisateurs graphiques (Figure 3.2).

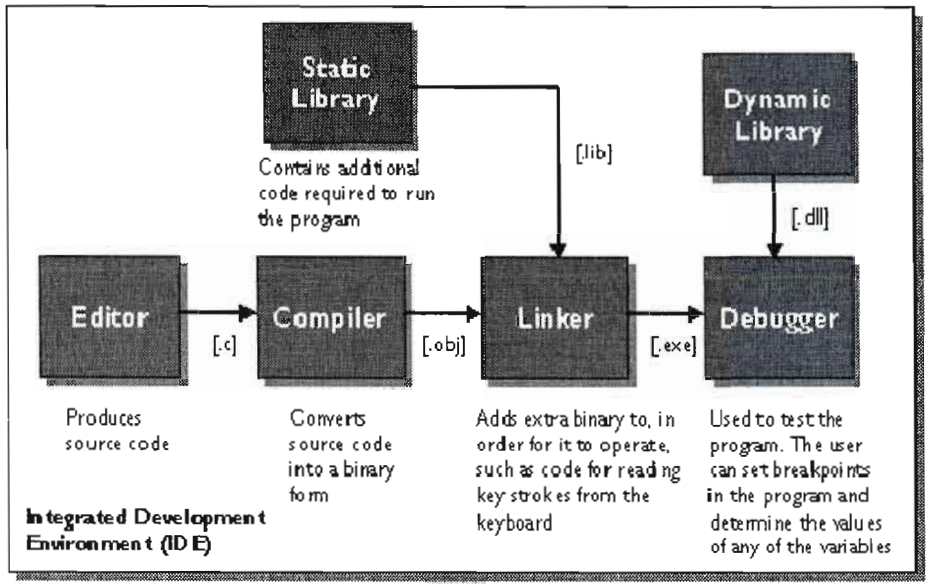


Figure 3.2 Architecture générale d'un environnement de développement intégré [William Buchanan – 2002]

Les EDIs fournissent un cadre convivial pour de nombreux langages de programmation modernes, tels que Visual Basic, Java et PowerBuilder.

L'environnement de développement intégré implémente l'outil d'interfaces utilisateur et définit des points d'extensions permettant à d'autres widgets d'interfaces utilisateur (composants) de contribuer aux actions de menu et barre d'outils, aux opérations de glisser-déposer et aux boîtes de dialogue et assistants. Aussi, un EDI définit des frameworks qui sont généralement utiles pour le développement d'interfaces utilisateur. Ces frameworks sont utilisés pour développer des composants d'interfaces utilisateur. L'utilisation des frameworks, non seulement facilite le développement, mais aussi assure la standardisation de l'apparence

et du niveau d''intégration logique. Enfin, il fournit un haut niveau de construction d'applications pour supporter les boîtes de dialogue, les actions, les préférences des utilisateurs et la gestion des widgets. De plus, il comprend une API portable et assure une intégration étroite avec les interfaces graphiques natives du système d'exploitation.

Exemple d'EDI: Eclipse

Eclipse est un EDI de Java qui a été conçu pour l'intégration d'outils de développement du web et d'applications. La force de la plateforme réside dans en ce qu'elle encourage : le développement rapide de fonctionnalités intégrées sur la base d'un modèle de plug-ins.

Eclipse fournit un modèle d'interfaces utilisateur (IU) courant. Il est conçu pour fonctionner sur plusieurs systèmes d'exploitation tout en assurant une intégration harmonieuse. Les Plug-ins peuvent être programmés avec les APIs portables d'Eclipse et s'exécuter sans changement sur n'importe quel système d'exploitation supporté.

Aussi, Eclipse offre une architecture permettant le chargement et l'exécution dynamique de plug-ins. La plateforme gère la logistique et le fonctionnement pour trouver le bon code. L'interface utilisateur de la plateforme fournit un modèle de navigation standard. Chaque plug-in peut alors se concentrer sur la façon de bien réaliser un petit nombre de tâches.

Architecture ouverte

La plateforme Eclipse définit une architecture ouverte; chaque équipe de développement de plug-ins peut se concentrer sur son domaine d'expertise. On laisse les experts du domaine développer le module de traitement et les experts de l'utilisabilité développer l'outil final de l'utilisateur. Si la plateforme est bien conçue, de nouvelles fonctionnalités et des niveaux d'intégration peuvent être ajoutés sans aucun impact négatif [eclipse.org].

La plateforme Eclipse utilise une infrastructure qui facilite l'intégration d'outils du point de vue de l'utilisateur final. De nouveaux outils peuvent être incorporés à Eclipse en utilisant des crochets « hooks » appelés des points d'extensions établis.

La plateforme elle-même est construite sous la forme de couches de plug-ins. Chacune définie des extensions à d'autre points d'extensions de bas niveau de plug-ins, et qui à leur tour définissent leurs propres points d'extension pour plus de personnalisation. Ce modèle d'extension permet aux développeurs de plug-ins de rajouter une variété de fonctions aux outils de base de la plateforme. Les artefacts de chaque outil, tels que les fichiers et autres données, sont coordonnés par un modèle de ressources de plateforme commun.

Les développeurs de plug-ins bénéficient aussi de cette architecture. La plateforme gère la complexité des différents environnements d'exécution, tels que les différents systèmes d'exploitation ou les environnements de serveurs de groupes de travail. Les développeurs de plug-ins peuvent se concentrer sur leur tâche spécifique au lieu de se préoccuper des problèmes d'intégration.

Infrastructure et les Toolkits d'IU

L'interface utilisateur de la plateforme d'Eclipse est construite autour d'une infrastructure qui fournit la structure d'ensemble et présente une interface extensible pour l'utilisateur. L'API et l'implémentation de l'infrastructure sont construites à partir de deux outils :

- SWT - un ensemble de widgets et une librairie graphique intégrés avec le système natif de fenêtres mais indépendants de l'API du système.
- JFace – un toolkit d'interfaces utilisateur en utilisant SWT qui simplifie les tâches courantes de programmation d'interfaces utilisateur.

SWT

Standard **W**idget **T**oolkit (SWT) fournit une API indépendante du système d'exploitation pour les widgets et les graphiques implémentés, de manière à permettre une intégration harmonieuse avec le système natif de fenêtres. L'interface de l'environnement d'Eclipse et les outils qui s'y intègrent utilisent SWT pour la présentation de l'information à l'utilisateur.

Un problème constant dans la conception de toolkits de widgets est la difficulté d'intégrer des toolkits portatifs avec le système natif de fenêtres. AWT (Abstract Windowing Toolkit) de

Java fournit des widgets de bas niveau tels que les listes, les champs de texte, les boutons, mais pas de widgets de haut niveau tels que des arbres. Les widgets d'AWT sont implémentés directement avec les widgets natives sur tous les sous-systèmes de fenêtres. La construction d'une interface utilisateur en utilisant seulement l'outil AWT signifie la programmation des composants élémentaires communs à tous les systèmes de fenêtres de systèmes d'exploitation. Le toolkit de Java Swing adresse ce problème en émulant des widgets comme les arbres et les tables. Swing fournit également des couches d'émulation de « Look and Feel » qui essaye de faire ressembler les applications aux fenêtres du système natif. Cependant, l'émulation de widgets invariablement reléguée derrière le « look and feel » de widgets natives, et l'interaction de l'utilisateur avec le widget simulé sont en général assez différentes pour être perceptibles.

SWT a adressé cette problématique par la définition d'une API qui est disponible à travers un nombre de systèmes de fenêtres supportés. Pour chaque système natif de fenêtres, l'implémentation de SWT utilise des widgets natives la où c'est possible, et là où aucune des widgets est disponible, l'implémentation de SWT fournit une émulation appropriée. Les widgets de bas niveau tels que les listes, les champs de texte et les boutons sont implémentés partout sous leurs formes natives. Généralement, certains widgets de haut niveau peuvent avoir besoin d'être émulées sur des systèmes de fenêtres. Par exemple, la barre d'outils des widgets de SWT est implémentée comme la barre d'outils widgets natifs sur Windows et comme un widget émulé sur Motif. Cette stratégie permet à SWT de maintenir un modèle de programmation consistant dans tous les environnements.

Une intégration harmonieuse avec le système de fenêtres natif n'est pas strictement une question de « look and feel » (apparence et comportement). SWT interagit aussi avec des fonctionnalités natives de bureau telles que glisser-déposer, et peut utiliser des composants développés avec les modèles de composantes de systèmes d'exploitation, comme les contrôles ActiveX de Windows.

L'implémentation interne de SWT fournit une implémentation séparée et distincte pour chaque fenêtre natif du système. Les librairies natives de Java sont complètement différentes, avec des API spécifiques au système de fenêtres.

Le fait que la librairie SWT fournit une API indépendante du système d'exploitation pour les composants d'interfaces utilisateur et simplifie les tâches courantes de programmation d'interfaces utilisateur, l'a rendu un bon candidat pour constituer la base de notre librairie améliorée qui est l'un des principaux éléments de la solution que nous avons proposé.

JFace

JFace est un toolkit d'interfaces utilisateur avec des classes pour la manipulation de nombreuses tâches de programmation d'interfaces. JFace est indépendante du système d'exploitation et est conçu pour fonctionner avec SWT.

JFace comprend un toolkit, les composants habituels d'interfaces utilisateur et les registres de polices, de dialogues, de préférence et d'assistant de framework. Il inclut aussi des rapports d'avancement pour les opérations de longue durée. Deux de ses plus intéressantes caractéristiques sont les actions et les visionneurs « viewer ».

Le mécanisme d'actions permet à des commandes de l'utilisateur d'être définies indépendamment de leur localisation exacte sur l'interface utilisateur. Une action représente une commande qui peut être déclenchée par l'utilisateur via un bouton, un menu ou un élément dans une barre d'outils. Chaque action clé connaît ses propres propriétés de l'interface (label, icône, etc) qui sont utilisées pour construire le widget pour la présentation de l'action. Cette séparation permet à la même action d'être utilisée à plusieurs endroits dans l'interface, et signifie qu'il est facile de changer où l'action est présentée sans avoir à modifier le code de l'action elle-même.

Les visionneurs sont des adaptateurs fondés sur des modèles pour certains widgets de SWT. Les visionneurs gèrent le comportement et fournissent un plus haut niveau de sémantique que ceux des widgets de SWT. Les visionneurs standards des listes, des arbres et des tables supportent le remplissage du visionneur avec des éléments du domaine du client et maintiennent la synchronisation des widgets avec les changements du domaine. Ces visionneurs sont configurés avec un contenu et un label du fournisseur. Le fournisseur de contenu sait comment faire la correspondance entre les éléments d'entrée du visionneur et le contenu attendu de lui, et comment assurer l'application appropriée des changements du

domaine lors des mises à jour des visionneurs correspondants. Le fournisseur de label sait comment produire le label spécifique et l'icône nécessaires à l'affichage de n'importe quel élément du domaine dans le widget. Les visionneurs peuvent également être configurés avec des filtres conçus sur des éléments. Les clients sont informés des sélections et des événements en termes d'éléments du domaine qu'ils fournissent au visionneur. L'implémentation du visionneur gère la correspondance entre les éléments du domaine et les widgets de SWT. Le visionneur standard pour le texte supporte les opérations courantes, telles que le comportement du double clique, l'annulation, le coloriage et la navigation par index du caractère ou le numéro de ligne. Le visionneur de texte fournit un modèle de document pour le client et gère la conversion du document à l'information requise par le widget de texte de style SWT. Plusieurs visionneurs peuvent être ouverts sur le même modèle ou document; tous sont mis à jour automatiquement lorsque le modèle ou le document change dans l'un d'eux.

Comme souligné, JFace inclut des classes pour manipuler des composants d'interfaces utilisateur et fournit deux caractéristiques intéressantes : un mécanisme d'actions et des visionneurs. En raison de ces caractéristiques, JFace a été utilisé comme base pour le composant intégrateur de notre approche.

3.6 Conclusion

Les outils d'interfaces sont censés faciliter la tâche de développement d'interfaces utilisateur utilisables, ce qui devrait permettre aux développeurs d'être plus productifs, indépendamment de leur niveau d'expérience [Jeff J, 2000]. Cependant, comme discuté dans le présent chapitre, la plupart des outils existant ne réussissent pas à remplir cette promesse, car ils ne fournissent pas un support suffisant à l'utilisabilité. La section suivante énumère les limites de ces outils:

- Les composants sont de très bas niveau. Leur utilisation dans le développement d'applications est comme construire une maison en bois, à partir de blocs de bois au lieu de panneaux, de planches ou de bûches [Jeff J. 2000]; cela peut être fait, mais cela va prendre beaucoup plus de temps et d'effort. Parfois, les composants des toolkits sont de bas niveau dans une tentative malencontreuse de permettre un maximum de souplesse

dans l'interface utilisateur développée. Cependant, la flexibilité sans directives (non guidée) est le plus souvent défaillante en termes d'amélioration de l'utilisabilité. Les toolkits sont de bas niveau pour plusieurs raisons: par exemple, les développeurs de toolkits manquent de temps ou ne sont pas suffisamment motivés ou qualifiés pour concevoir des blocs de développement de haut niveau, qui couvrent tous les aspects de l'utilisabilité désirée.

- La majorité des toolkits permettent aux concepteurs de créer des interfaces utilisateurs qui violent les directives de l'utilisabilité et donnent aux développeurs peu ou pas de directives. La plupart des toolkits affirment supporter un ou plusieurs standards d'interfaces utilisateur graphiques, tels que Windows et Mac, mais la réalité est tout autre. Aussi, Ils permettent aux concepteurs de faire de mauvais choix, comme l'utilisation de faux contrôles pour un paramètre. Le contrôle peut apparaitre familier, mais ceci est un détail mineur s'il est le mauvais contrôle ou si le comportement est inadéquat, comme une case à cocher utilisée pour un paramètre non conçu pour la fonction ON/OFF

Le bas niveau des composants fournis par les toolkits cause problème car il ne permet pas aux professionnels de concevoir des interfaces de qualité. Ils sont facilement et rapidement assemblables par les développeurs sans une expérience de l'utilisabilité nécessaire pour compenser le manque de guide au niveau des toolkits. Le résultat est que les interfaces utilisateurs développées à partir de ces outils présentent des lacunes en termes d'utilisabilité.

De plus, aucune de ces technologies n'adresse les vrais problèmes des directives de l'utilisabilité. Le support est nécessaire pour définir quand les directives sont applicables ou comment peuvent-elles être raffinées pour répondre au besoin de la tâche de l'utilisateur et d'un type spécifique d'applications.

Peu de travaux de recherche ont été fait pour intégrer l'utilisation des directives dans les outils d'interfaces utilisateur et accumuler les connaissances sur la conception d'interfaces utilisateur sous une forme qui peut capter les relations entre les composants de l'interface et les directives de conception de l'utilisabilité.

Dans le prochain chapitre, nous proposons une approche qui adresse cette problématique par la combinaison d'Eclipse, Java beans et des directives de l'utilisabilité. Le résultat élargit la notion traditionnelle des directives de l'utilisabilité pour fournir des directives détaillées qui sont intégrées dans l'environnement de développement dans un outil conforme. L'outil va également assurer la correspondance entre le contexte de conception et les directives, dans le but d'assister les développeurs, en leur permettant d'utiliser les directives de conception existantes.

CHAPITRE IV

PROPOSITION D'UNE ARCHITECTURE D'INTEGRATION

Dans ce chapitre, nous détaillons la solution proposée pour incorporer les directives de conception pour l'utilisabilité dans un outil de développement. Notre solution intègre les directives de conception concernant un composant d'interface utilisateur dans le code de l'interface utilisateur, qui sont représentés comme des beans. Ces beans sont ensuite incorporés dans Eclipse via le concept de plug-ins.

4.1 Architecture globale de l'outil proposé

La figure 4.1 présente l'architecture proposée pour l'incorporation des directives de conception dans des composants d'interfaces utilisateur, les toolkits de développement d'interfaces et les environnements de développement (Figure 4.1). L'outil comprend les composants suivants:

1. Une librairie améliorée de composants d'interfaces utilisateur où chaque élément inclut comme partie de son implémentation les directives de conception qui lui sont associées.

2. Un intégrateur qui sélectionne les directives de conception applicables au contexte du composant d'interface utilisateur en cours de développement.

3. Un assistant qui fournit aux concepteurs de l'aide tels que des messages d'avertissement ou de rappel. Il offre aussi de l'assistance aux concepteurs novices en les guidant à travers le processus de développement de l'interface utilisateur. Cet outil offre de multiples niveaux d'aide selon les besoins du concepteur. Chaque niveau prend en considération l'expérience du concepteur en matière d'utilisabilité et la complexité de l'interface utilisateur en cours de développement.

Note: les éléments en pointillés montrent les outils existants et réutilisés. Les autres sont les composants que nous avons développés.

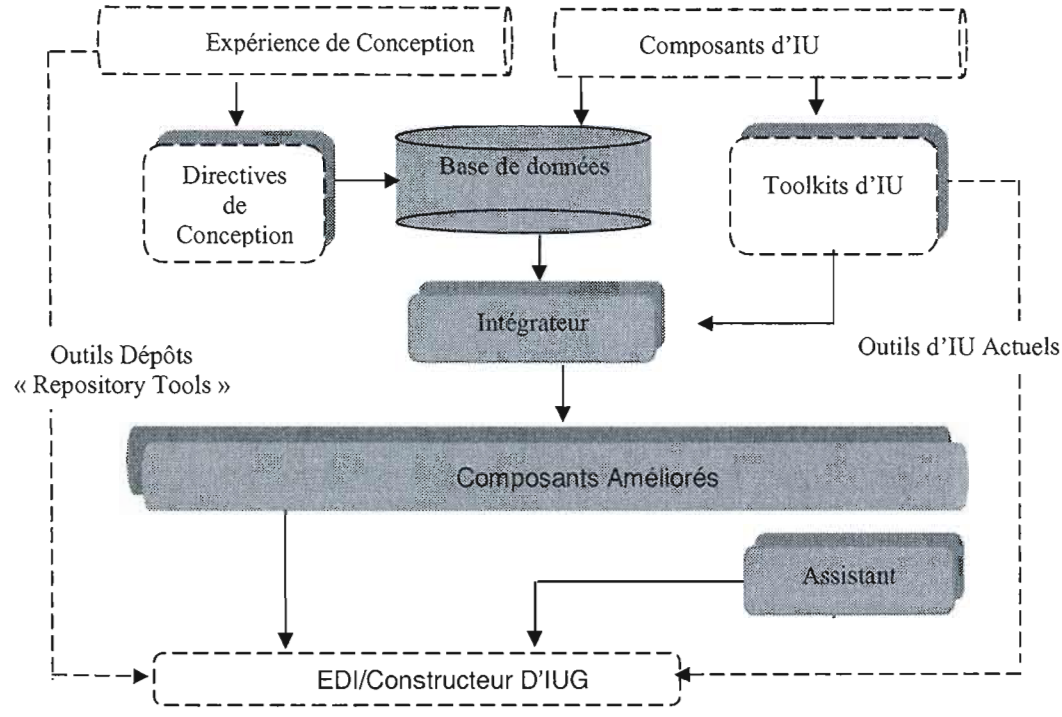


Figure 4.1 Vue de l'architecture proposée

4.2 Eclipse, Java et Beans comme infrastructure d'implémentation

L'architecture proposée a été implémentée en utilisant l'EDI Eclipse Java et Java beans. Dans cette section, nous justifions ce choix tout en mettant en évidence certains concepts de Java qui ont facilité l'implémentation de notre approche. Cela inclut la sérialisation des beans, l'introspection et les plug-ins.

4.2.1 Pourquoi les Beans de Java?

Dans le chapitre précédent, nous avons introduit le bean de Java. Deux de ses caractéristiques qui ont facilité l'implémentation de notre outil sont la sérialisation et l'introspection.

Le concept de sérialisation permet la conversion d'un objet du format interne utilisé dans une application à un format adapté pour la persistance ou l'interopérabilité (John Sharp, Andy Longshaw & Peter Roxburgh, 2003). Par exemple, dans le Framework .NET de Microsoft, la sérialisation est employée lors de l'utilisation de l'architecture à distance de .NET ou de l'accès aux services Web. Le langage de programmation Java supporte la sérialisation en tant que mécanisme de collecte de propriétés des composants d'IU durant le processus de développement. Notre approche nécessite un tel mécanisme qui permet de sauvegarder les propriétés des composants d'interface avec la capacité de les retrouver dynamiquement en cas de besoin. La sérialisation de Java permet de réduire la complexité nécessaire pour accomplir cette tâche. Typiquement, le concept de sérialisation de bean de Java est utilisé pour la manipulation et la validation des propriétés de composants d'interface utilisateur. La sérialisation permet à notre outil de conception d'obtenir et de stocker l'information concernant les propriétés de l'utilisabilité des composants d''interface, au cours du processus de développement assisté.

L'introspection est le processus par lequel un environnement de développement découvre les propriétés, les événements et les méthodes qui sont supportés un bean (Ron Ben-Natan&Ori Sasson, 2002). Le principal but du mécanisme de l'introspection des beans de Java est d'offrir un moyen par lequel les capacités d'un bean peuvent être inspectées. Cette caractéristique

essentielle des beans de Java a été employée pour permettre à l'assistant de notre outil de valider les propriétés des composants d'interface selon les critères de l'utilisabilité. Pour se conformer à ce mécanisme, il faut définir et déclarer le composant amélioré comme un bean. Les variables du bean correspondent aux propriétés de l'utilisabilité du composant. Le bean inclut aussi des méthodes de validation des valeurs de ces variables selon les directives de conception de l'utilisabilité.

Cela permet à notre outil de conception de déduire les informations concernant les propriétés de l'interface utilisateur qui sont en cours de changement. En connaissant les propriétés de l'utilisabilité d'un composant, l'outil est en mesure de fournir la propriété adéquate et de valider les directives de conception qui doivent être prises en considération.

4.2.2 Pourquoi Eclipse?

Eclipse est un environnement de développement intégré (EDI) qui est utilisé pour créer diverses applications multiplateformes, y compris les sites Web, Java, C++, Enterprise Java Beans et les applications distribuées. Eclipse est un logiciel libre « open source ». Il fournit un framework basé sur les plug-ins ; le framework facilite la création, l'intégration et l'utilisation des outils logiciels. La plateforme Eclipse est écrite en langage Java et comprend des toolkits de construction des plug-ins d'extension et des exemples. Il a déjà été déployé sur plusieurs plateformes de développement, incluant Linux, HP-UX, AIX, Solaris, Mac et les systèmes basés sur Windows.

Basé principalement sur des caractéristiques de Java, Eclipse définit une architecture ouverte pour la découverte dynamique, le chargement et la gestion de plug-ins. La plateforme Eclipse utilise les plug-ins pour intégrer de nouveaux outils ou programmes dans l'environnement Eclipse. Les outils développés peuvent se brancher sur la surface de travail en utilisant des crochets appelés points d'extension (plug-ins). La figure ci-dessous montre une vue simplifiée (Figure 4.2) de cet environnement. Du au fait qu'Eclipse est basé sur une architecture ouverte et un modèle de plug-ins, nous le considérons comme le meilleur candidat pour implémenter l'assistant et l'intégrateur, qui sont les composants essentiels de l'architecture proposée.

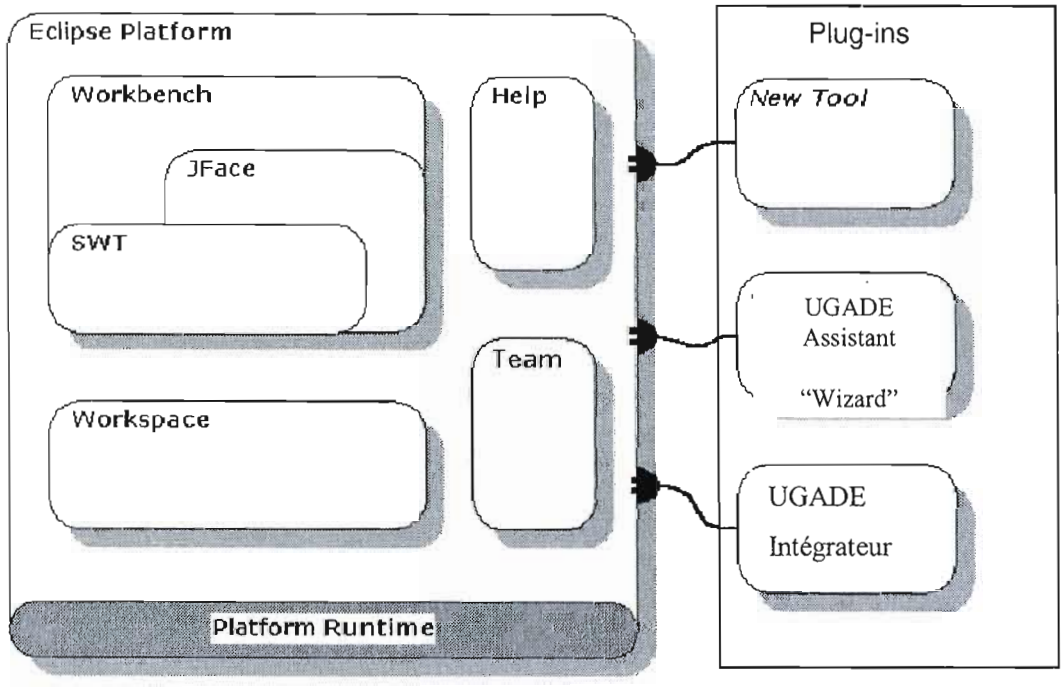


Figure 4.2 Architecture de la plate-forme Eclipse.

4.3 Description détaillée

UGADE (Usability Guideline Assisted Development Environment) est l'implémentation de l'architecture proposée et détaillée à la section 4.1. Elle peut être considérée comme une extension de l'environnement d'Eclipse auquel on rajoute les trois composantes de base de notre architecture. L'intégrateur et l'assistant sont intégrés à Eclipse via les Plug-ins; la librairie améliorée est implémentée avec les principes des Beans de Java.

Cet outil fournit une boîte contenant des directives de l'utilisabilité des interfaces. Le concepteur peut consulter la description détaillée de chaque directive avec des exemples pratiques. Aussi, il peut obtenir de l'aide et des instructions pour appliquer les directives appropriées à la création d'interfaces utilisables.

Dans cet outil de conception, diverses directives sont nécessaires pour inclure les aspects pertinents de l'utilisabilité de l'interface utilisateur. Comme illustré dans les chapitres

précédents, il existe de nombreux aspects, ainsi que des outils connexes. Dans le cadre de notre travail, nous avons développé et implémenté en Java la première version de l'outil

4.3.1 Fonctions principales

La première version (1.0) de l'outil qui implémente notre approche est conçue pour fournir aux concepteurs deux principaux modes de fonctionnement.

Mode Consultation (Browse)

L'outil peut fournir aux concepteurs d'interfaces, sur demande, des détails et des informations sur les directives de conception. L'information est présentée dans un format spécial contenant le nom et la description des directives, ainsi que des exemples pratiques.

Le concepteur peut obtenir de l'information sur les directives à tout moment, peu importe si l'outil est utilisé en mode "Browse" ou "Design". Dans le mode "Browse", le concepteur a juste besoin de choisir les directives spécifiques dans la boîte des directives de conception; les informations associées seront affichées. Aussi, dans le mode "Design", le concepteur peut afficher les informations en cliquant sur le bouton droit de la souris pointant le label de la directive en cours d'utilisation.

Enfin, l'outil peut démontrer la corrélation entre les directives et le processus de conception. Au fur et à mesure que le concepteur avance dans le processus de conception, les directives pertinentes au contexte seront affichées.

Mode Conception (design)

L'outil peut supporter le processus de conception d'un niveau plus général vers un niveau plus spécifique. Trois de ces niveaux sont fournis au concepteur:

a) Niveau de l'application : le concepteur peut créer le prototype d'une application, qui caractérise une certaine architecture, contenant des fenêtres (écrans) et des composants spécifiques dans chaque fenêtre.

b) Niveau de la fenêtre : le concepteur peut choisir des directives appropriées pour la conception d'écrans et la structure de la fenêtre, puis intègre les éléments requis dans la fenêtre.

c) Niveau des composants : le concepteur peut sélectionner dans la boîte à outils des composants qui sont conformes aux directives de conception, incluant la navigation, l'apparence et le comportement « Look and Feel », et l'information des conteneurs.

Aussi, l'outil peut illustrer des directives de conceptions appropriées en fonction des circonstances.

Enfin, l'outil supporte la combinaison et l'organisation des directives existantes. Il fournit un mécanisme permettant de vérifier l'utilisation des directives. Ainsi, il peut automatiquement examiner la compatibilité de certaines directives et donner en conséquence, les instructions au concepteur. Le concepteur peut ajouter, déplacer, voire supprimer des directives

4.3.2 L'interface utilisateur

L'interface utilisateur UGADE est composée de trois zones (figure 4.3):

1. Boîte de directives : toutes les directives de conception sont présentées dans cette zone et regroupées en fonction des catégories des composants.

2. Espace de travail : il est la principale surface d'opération pour les concepteurs. Ils peuvent effectuer des activités de navigation et de conception dans cette zone.

3. Barre de Boutons : elle est composée de 5 boutons, appelés "Browse", "Design", "Add", "Delete" et "Quit" ils représentent les fonctionnalités d'UGADE.

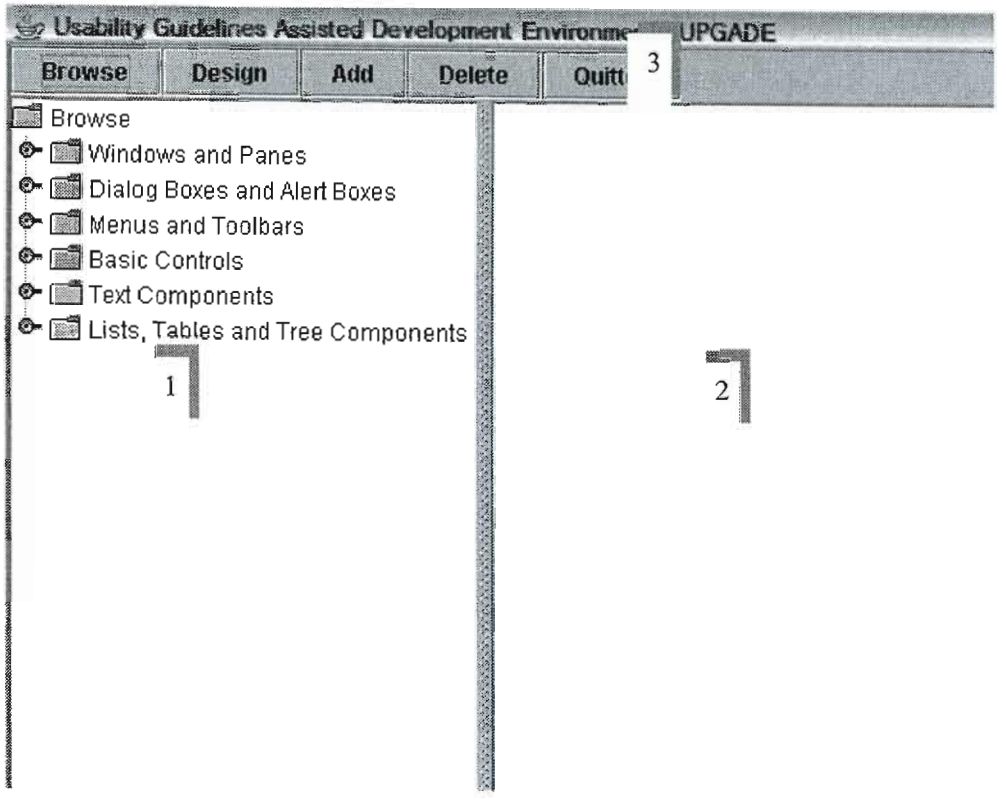


Figure 4.3 L'interface utilisateur d'UGADE 1.0.

4.3.3 Librairie améliorée d'interfaces utilisateur

Comme indiqué ci-dessus dans la figure 4.1, la librairie améliorée d'interfaces utilisateurs d'UGADE comprend un ensemble de directives de conception, un outil d'interface utilisateur et des widgets. Cette librairie est écrite en Java pour accommoder les concepteurs afin de développer des applications conformes aux exigences de l'utilisabilité, en utilisant des composants d'interface tels que les boîtes de dialogue ou d'alerte. De plus, la librairie est l'implémentation de notre approche qui intègre les directives de l'utilisabilité dans les composants et les environnements de développement d''interfaces utilisateur.

La librairie améliorée d''interfaces utilisateur de UGADE fournit des éléments d'interfaces utilisateur visuels et interactifs, utilisables dans le développement d'applications avec l'ajout de quelques lignes de code. Tous les composants de la librairie ont été mis au point sur la base du modèle de plug-in et sont fondés sur Eclipse en raison de son architecture ouverte.

Cette section fournit une vue d'ensemble (Figure 4.4) de la librairie et illustre l'amélioration et le renforcement de l'utilisation des directives de conception dans le processus de développement d'interfaces utilisateurs.

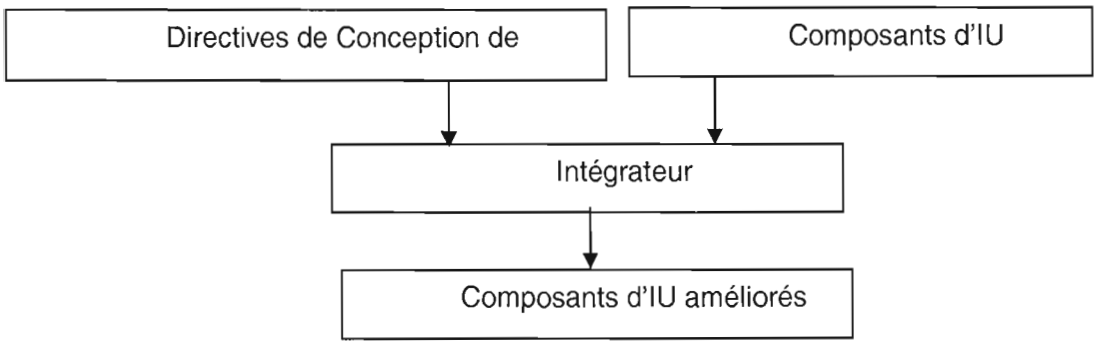


Figure 4.4 Librairie de composants d'interfaces utilisateur améliorée

4.3.4 Support : de l'assistant à l'alerte

Durant le mode conception, l'outil présente de multiples niveaux d'assistance basés sur les besoins des concepteurs. Ces niveaux sont :

- L'assistance étape par étape, à travers l'ensemble du processus de développement/conception.
- Des conseils personnalisés.
- L'identification de directives critiques pour prévenir des omissions accidentelles.
- La validation et la confirmation automatique de la conformité aux exigences de l'utilisabilité.
- Des messages de mise en garde ou de rappel

4.3.5 Stockages des données

4.3.5.1 Description

Les directives de conceptions et les composants d'interfaces sont garants du bon fonctionnement du système. Il faut donc décrire le lieu de stockage de ces données, et il est important d'éviter leurs duplications. La solution proposée utilise une base de données qui sera gérée par un système de gestion de base de données (SGBD) permettant un accès par composants et par contexte. Pour cela, une base de données relationnelle tel que SQL Server 2008 peut être considérée. Elle facilitera la gestion des directives, les composants d'interfaces en occurrence, les opérations d'ajout, de modification, de suppression et de recherche

4.3.5.2 Tables

La base de données est composée principalement des tables suivantes :

Nom	Type	Nombre de caractères
Identifiant	Alphanumérique	10
Catégorie	Alphabétique	50
Titre	Alphabétique	200
Description	Alphabétique	1024

Tableau 4.1 Table de directives

Nom	Type	Nombre de caractères
Identifiant	Alphanumérique	10
Titre	Alphabétique	100
Description	Alphabétique	1024

Tableau 4.2 Table de composants

Nom	Type	Nombre de caractères
Identifiant Directive	Alphanumérique	10
Identifiant Composant	Alphanumérique	10

Tableau 4.2 Table de relation

4.4 Conclusion

Dans ce chapitre, nous avons présenté une vue d'ensemble de l'architecture proposée pour concevoir notre solution qui permet d'intégrer les directives de conception dans le code des composants d'interfaces utilisateur. Aussi, nous avons montré l'avantage de réutiliser l'environnement de développement intégré Eclipse Java et Java beans comme infrastructure pour la réalisation de la solution. Nous avons justifié ce choix et mis en évidence les concepts de sérialisation de beans, l'introspection et les plug-ins de Java qui ont facilité l'implémentation de notre solution. De plus, nous avons présenté l'outil UGADE (Usability Guideline Assisted Development Environment) qui représente l'implémentation de l'architecture proposée. La première version (1.0) de cet outil est conçue pour fournir aux concepteurs deux principaux modes de fonctionnement à savoir le mode conception et le mode consultation (browse).

CHAPITRE V

EXEMPLES ILLUSTRATIFS ET APPLICATION

***NOTE** : Les messages et les libellés utilisés dans ce prototype sont en anglais c'est principalement pour une éventuelle utilisation chez IBM. Cependant si nécessaire, je ferai la traduction.*

Dans ce chapitre, nous présentons des exemples qui démontre l'applicabilité de notre approche. Nous allons introduire les différents niveaux de conception qu'UGADE supporte. À titre d'exemple, nous montrons comment appliquer différentes directives concernant une boîte de dialogue.

5.1 Naviguer « Browse »

Naviguer est le mode de fonctionnement par défaut lorsque l'outil UGADE est lancé. Comme le montre la figure 5.1, la boîte des directives de conception affiche toutes les catégories de directives. Chaque catégorie est symbolisée par un répertoire qui regroupe toutes les directives applicables.

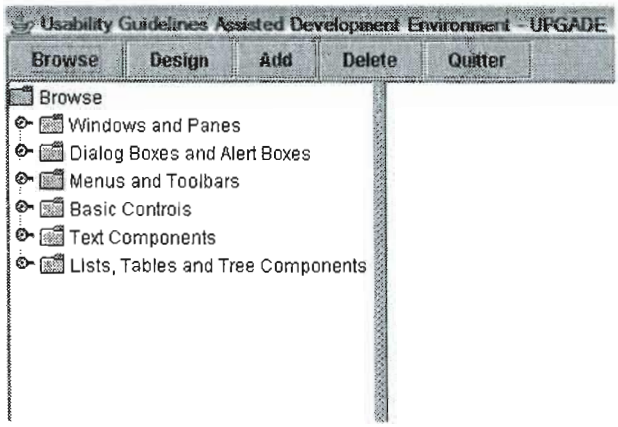


Figure 5.1 Le mode naviguer est activé lorsque l'outil UGADE est lancé.

Étape 1: Sélectionner une catégorie appropriée de directives de conception

Supposons que nous choisissions la catégorie des directives de conception de fenêtres et panneaux "Windows and Panes". La sélection de cette catégorie affiche le contenu organisé en une structure arborescente qui comprend toutes les directives de conception liées aux fenêtres et aux panneaux (figure 5.2).

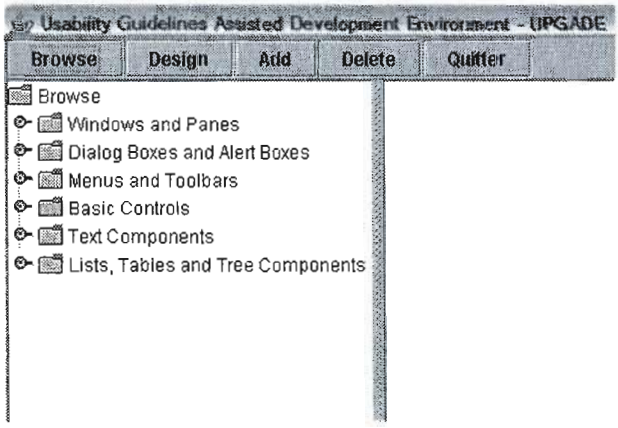


Figure 5.2 Ensemble de directives de conception de fenêtres et panneaux"Windows and Panes"

Étape 2: Parcourir les informations sur les directives de conception

Lorsque le concepteur choisit "Constructing Windows" dans la boîte de directives de conception, les directives correspondantes sont affichées dans la zone de travail. Si le concepteur, en particulier le novice, veut découvrir l'interprétation de certaines directives et comprendre comment les utiliser correctement, il peut consulter les informations détaillées de chacune d'elles dans la zone de travail. Celle-ci contient toutes les informations concernant les directives de conception pour concevoir une fenêtre "Constructing Windows". Aussi, elle aide le concepteur à réduire la charge de mémorisation et à améliorer la compréhension des directives de conception (Figure 5.3).

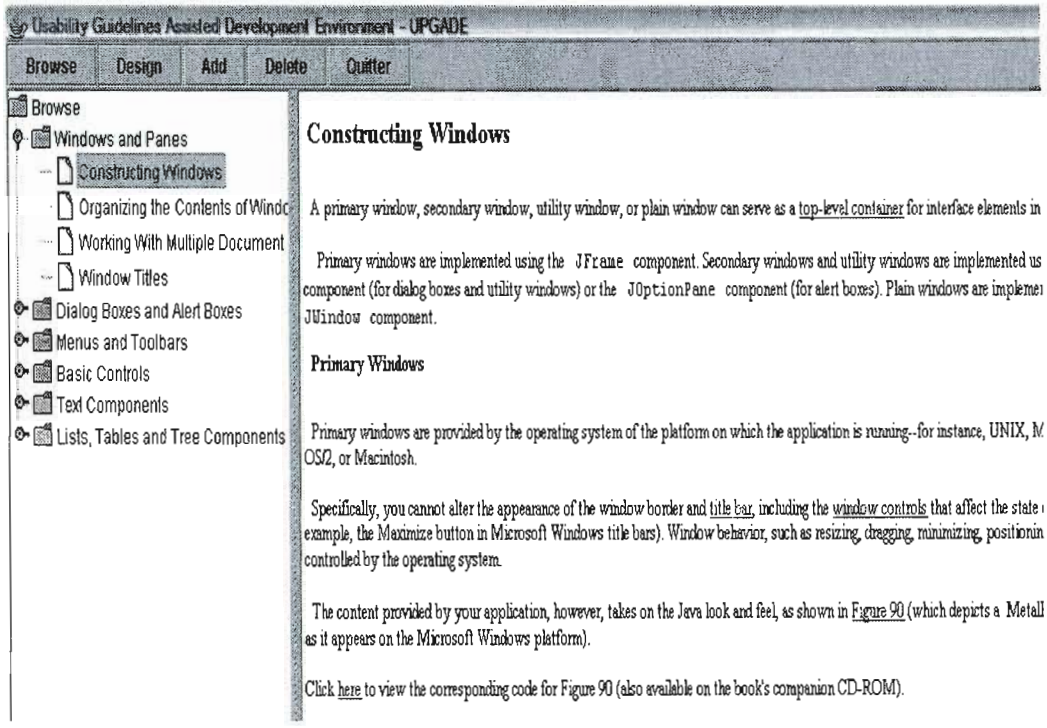


Figure 5.3 Un exemple de description des directives pour le développement de fenêtres.

5.2 Conception « Design »

En utilisant notre outil, le processus de conception est divisé en six étapes.

Étape 1: Activation du mode conception

Pour activer le mode conception, le concepteur doit cliquer sur le bouton "Design" dans la barre d'outils. Comme le montre la figure 5.4, le nom de la racine des catégories symbolisées par les répertoires devient Design. Ainsi, une surface d'opération vierge apparaît dans la zone de travail.

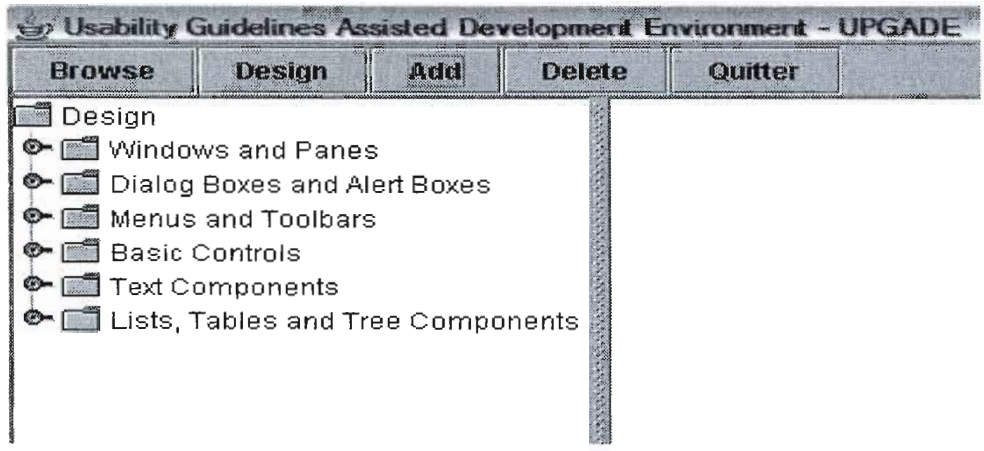


Figure 5.4 Un exemple d'activation de la fonction de conception

Étape 2: Sélectionner une catégorie de composants d'interfaces utilisateur

Le processus de conception débute par la sélection de la catégorie de composants. Nous avons défini des catégories de composants de l'interface utilisateur, telles que les fenêtres et les panneaux, les menus et les barres d'outils, les contrôles de base, etc. Chaque catégorie comprend les directives associées au type de composant. Le concepteur doit choisir une catégorie de directives qui convient au composant désiré. La figure 5.5 montre la catégorie des boîtes de dialogue et d'alerte " Dialog Boxes and Alert Boxes" qui est le plus populaire dans les applications.

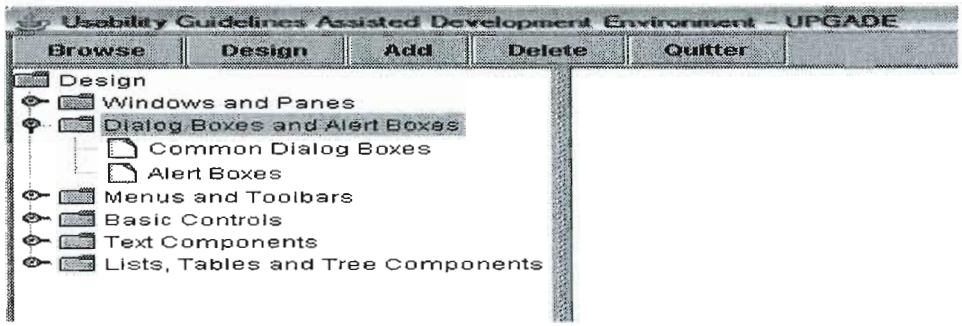


Figure 5.5 Un exemple de catégorie de composants

Étape 3: Acquérir la liste de composants de l'interface utilisateur

Le point de départ de la prochaine étape du processus de conception est de sélectionner la liste des composants selon les besoins. Cela énumérera les composants de l'interface utilisateur correspondants dans la zone de travail. Comme le montre la figure 5.6, la liste des boites de dialogue courantes "Common Dialog Boxes" est sélectionnée.

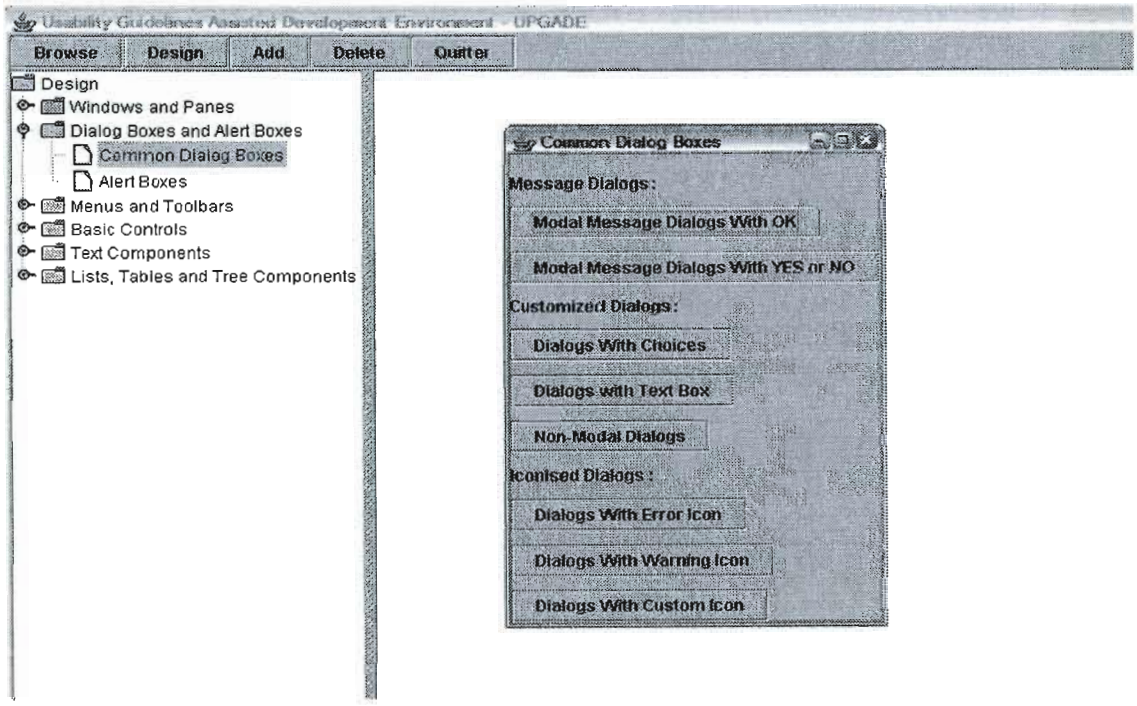


Figure 5.6 Un exemple de liste des composants d'interface utilisateur.

Étape 4: Sélectionner un composant approprié d'IU

Le choix de type de boite de dialogue "Modal Message Dialogs with OK" (voir Figure 5.6) affichera le composant comme indiqué dans la Figure 5.7.

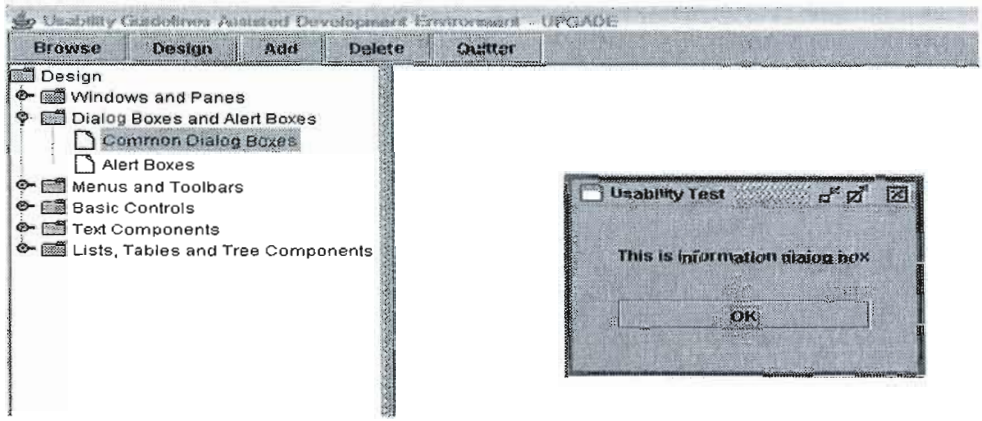


Figure 5.7 Un exemple de composant d'interface utilisateur.

Étape 5: Afficher la fenêtre des propriétés du bouton "OK"

En pointant le bouton OK de la boîte de dialogue et en cliquant sur le bouton droit de la souris, cela permet l'affichage de la fenêtre des propriétés du bouton. Comme le montre la figure 5.8, les propriétés nom et taille du bouton OK peuvent être changées.

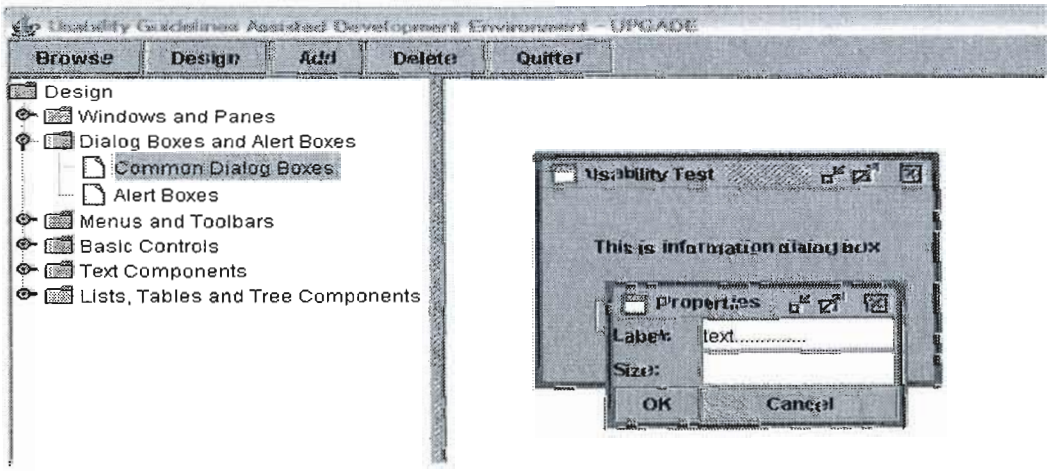


Figure 5.8 Fenêtre de propriétés du bouton «OK

Étape 6: Changer le nom du bouton

Lorsque le concepteur saisi un nouveau nom pour le bouton OK dans la fenêtre des propriétés, une validation est effectuée. Le concept de sérialisation de Bean de JAVA permet de sauvegarder les valeurs des propriétés des composants de l'interface, et de les reconstruire en un objet réel pour être utilisé par UGADE.

Ensuite, le mécanisme vérifie l'utilisation des directives de conception de la langue. Il vérifie aussi la conformité du texte « nom » saisi en fonction du contexte d'utilisation. Ceci est accompli par le mécanisme d'introspection de Bean de JAVA qui fournit un moyen par lequel un environnement de développement découvre quelles sont les propriétés supportées. De cette situation résulte les possibilités suivantes:

Comme le montre la Figure 5.9, "YES" est saisi dans le champ nom de la fenêtre des propriétés, puis en cliquant sur OK, il est accepté par l'outil UGADE. Cela permet de changer le nom du bouton de "OK" à "YES".

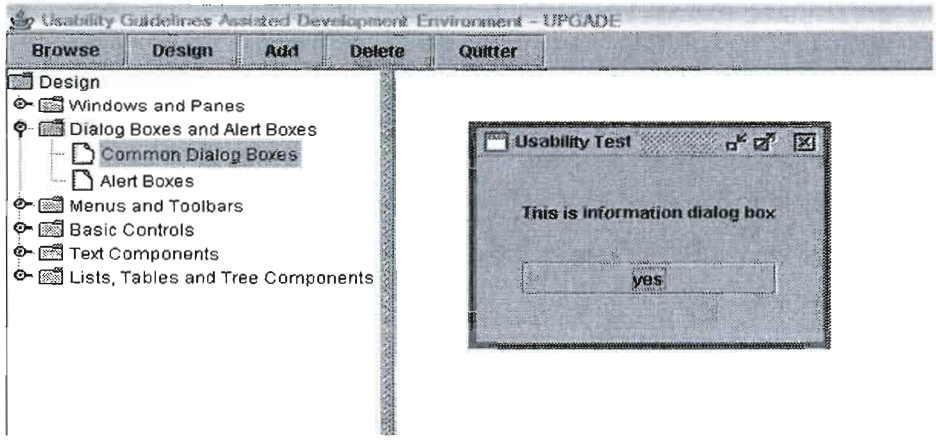


Figure 5.9 Changer le nom du bouton

Cependant, si le concepteur n'est pas familier avec les attributs du bouton, il pourrait entrer une valeur inappropriée. Avec notre mécanisme de vérification automatisé, l'outil UGADE informera le concepteur que le nom saisi n'est pas conforme. Si "OUI" est saisi ce qui n'est pas conforme au contexte (mot anglais), l'outil UGADE affichera le message montré dans la

figure 5.10. Plus encore, si le texte entré est inconnu, l'outil UGADE affichera le message montré dans la Figure 5.11.

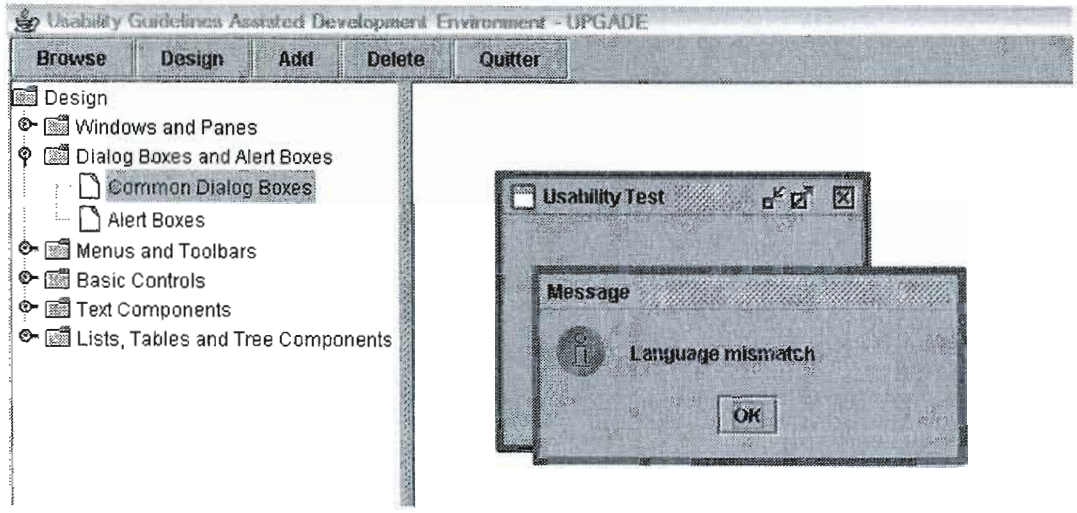


Figure 5.10 Exemple de directives de conception de vérification de la langue.

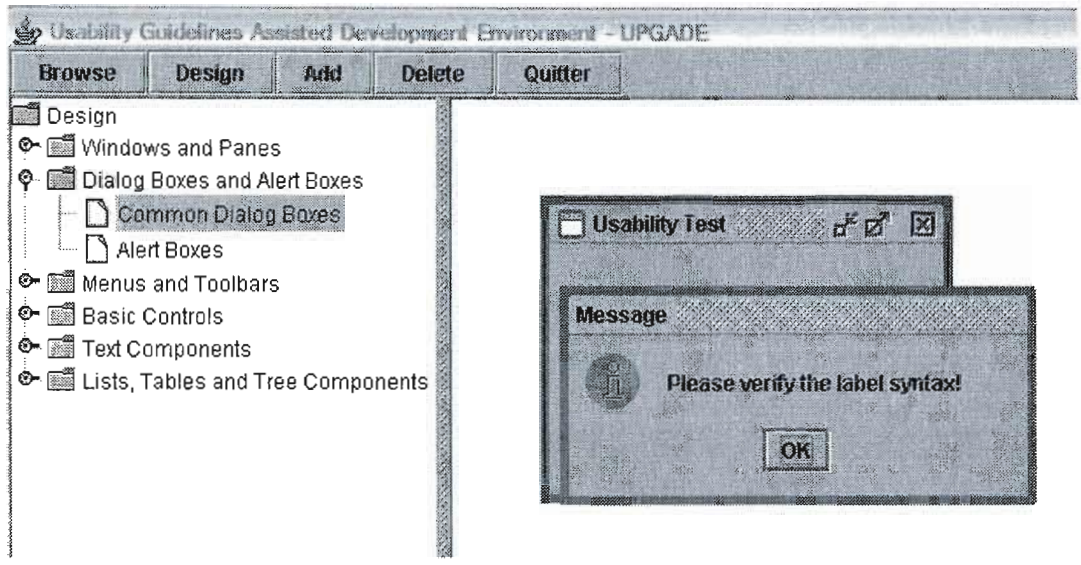


Figure 5.11 Exemple de directives de conception pour la vérification de syntaxe.

Ces étapes sont applicables à toutes les catégories de concepteurs quel que soit leur niveau d'expertise. Cependant, la différence est remarquable après la sixième étape et deux cas

peuvent se présenter. Le premier, le cas d'un concepteur novice, l'outil est utilisé en mode assisté alors après cette étape l'utilisateur sera retourné à la fenêtre de propriétés figure 5.8. Cela pour forcer le concepteur à effectuer les modifications requises tel que suggérées par l'outil. Le deuxième, cas si le concepteur est un expert et utilise l'outil en mode conseil, alors après la sixième étape c'est le composant en cours de développement qui va être affiché comme figure 5.9. Dans ce cas, deux possibilités se présentent au concepteur. La première est d'ignorer le conseil, de passer à une autre étape et de créer un nouveau composant. La deuxième, est de prendre en considération les suggestions de l'outil et d'accomplir les modifications nécessaires.

5.3 Conclusion

Avec cette exemple illustratif, nous avons non seulement montré que notre outil fonctionne, mais également que l'approche proposée est techniquement réalisable. En utilisant le mécanisme de plug-in d'Eclipse ainsi que l'introspection et la sérialisation de Bean de JAVA, nous avons réussi à implémenter nos idées. Cependant, ce ne sont pas tous les types de directives de conception qui peuvent être supportées par cette approche. Par exemple, les directives de conception relatives à des industries spécifiques, comme les interfaces graphiques d'un tableau de bord ou d'un simulateur d'avions sont difficiles à implémenter dans notre outil UGADE. Ceci est dû à la spécificité et la quantité d'information gérées et affichées par ce genre d'interfaces. De ce fait, les directives présentent des particularités afin de répondre aux exigences de ce type d'interfaces telles que la couleur, la taille des caractères et des icones, etc.

CHAPITRE VI

CONCLUSION ET PERSPECTIVES

6.1 Conclusion

Dans ce mémoire, nous avons étudié la problématique d'intégration des directives de conception de l'utilisabilité dans les outils de développement d'interfaces utilisateur. Les précédents travaux de recherche et les pratiques quotidiennes ont démontré que les outils de développement d'interfaces utilisateur, tels que les trousses à outils «toolkits », les constructeurs d'IU, les Beans de JAVA et les environnements de développement intégrés, n'ont pas vraiment pris en considération la question de l'utilisabilité [Kazman, 2003; Seffah, 2004]. La plupart de ces outils ne fournissent pas suffisamment d'aide sur l'utilisation des principes et des directives de conception d'interfaces utilisateur. Certains ont été proposés pour apporter une aide et faciliter l'utilisation des directives, tels que les outils d'entrepôt « Repository » et de documentation en ligne. Cependant, ces outils présentent des faiblesses, comme discuté dans ce mémoire. De plus, aucun de ces outils de développement d'IU, n'est conçu pour assister les développeurs novices, surtout lorsqu'il s'agit de faire un choix parmi une vaste liste de directives ou quand celles-ci sont en conflit.

L'approche que nous avons proposée vise à pallier à l'absence de définitions quand les directives sont applicables et comment peuvent-elles être affinées pour répondre aux critères de l'utilisabilité, ce qui est l'une des principales limites des outils de développement existants. Le but de notre travail de recherche était de développer une approche pour incorporer les directives de conception au niveau du code des composants d'interfaces utilisateur et des environnements de développement d'interfaces.

Une des principales contributions de notre travail de recherche est la conception d'une librairie améliorée qui supporte pleinement l'intégration des directives de conception et des composants d'interfaces utilisateur. En général, les directives de conception sont destinées à capter ou encapsuler les connaissances de conception. Dans ce mémoire, nous avons étendu la capacité de l'environnement de l'outil de développement en fournissant une librairie améliorée pour supporter l'intégration des directives de conception et des composants d''interfaces utilisateurs. Dans cette optique, nous avons proposé un cadre conceptuel général dans lequel les directives peuvent être adaptées au contexte particulier d'utilisation. Cela a été réalisé en utilisant la trousse d'outils d'Eclipse: SWT et JFace. La librairie « Standard Widget Toolkit » (SWT), fournit une API indépendante du système d'exploitation pour l'implémentation des « widgets ». L'outil JFace fourni des classes pour la manipulation des composants d'IU et un mécanisme pour les actions et les visionneurs. Aussi, nous avons adapté l'environnement de l'outil de développement pour qu'il supporte les directives axées sur la conception assistée. Dans le cadre de ce travail de recherche, nous avons analysé les problèmes de l'intégration des directives de conception de l'utilisabilité et les composants d'interfaces des environnements de développement.

Une autre contribution importante de ce travail de recherche est l'implémentation de l'architecture proposée en utilisant le langage Java. À titre d'exemple, nous avons examiné le cas d'un prototype d'une boîte de dialogue d'IU. Dans le cadre de cet exemple, différentes directives relatives aux composants d'interfaces d'une boîte de dialogue ont été découvertes et appliquées, tout en fournissant une assistance de conception. Le principal but du prototype est de démontrer l'intégration harmonieuse des directives de conception dans les composants de l'interface utilisateur et l'environnement de développement. Cependant, nous n'avons pas adapté le prototype et les directives pour toutes les plateformes.

Notre dernière contribution a été le développement d'un outil complet, UGADE (Usability Guideline Assisted Development Environment) qui, en plus des fonctions de navigation et de recherche, fournit de l'assistance étape par étape, tout au long du processus de conception d'interfaces utilisateur. Cela permet aux développeurs de tirer pleinement profit des directives

de conception qui sont disponibles dans le monde réel. De plus, UGADE inclut le concept de conception assistée à la demande, en fonction du contexte. L'outil peut supporter des concepteurs expérimentés en leur fournissant des messages d'avertissements ou de rappels. Aussi, il fournit aux concepteurs novices la capacité de développer des interfaces utilisateur efficaces et efficientes. La sérialisation et l'introspection sont les deux caractéristiques de Beans de Java qui ont facilité l'implémentation de cela. Enfin, la notion de greffon « plug-in » et le concept d'architecture ouverte d'Eclipse ont facilité l'intégration d'UGADE dans l'EDI.

6.2 Recherche future

L'approche et l'outil UGADE présentés dans ce mémoire font partie d'un projet de recherche à long terme. Notre travail de recherche a répondu à quelques questions sur l'incorporation des directives de conception de l'utilisabilité dans les composants d'interfaces utilisateur, les outils et les environnements de développement. Le principal objectif est qu'UGADE va développer et implémenter une approche pour supporter pleinement l'intégration des directives de conception de l'utilisabilité dans les composants d'interfaces utilisateur. De plus, UGADE vise à supporter cette approche, en permettant aux développeurs débutants d'être plus productifs et d'opérer à un haut niveau d'abstraction lors de la création d'interfaces utilisateur.

Comme étape future, nous suggérons l'implémentation des patrons de l'utilisabilité en utilisant le langage « markup language » basé sur XML. Le language XML facilite la génération complète et opérationnelle d'une interface utilisateur. Pendant ce temps, afin de rendre les résultats de notre travail de recherche bénéfique pour le monde industriel, les concepteurs de sites Web, les développeurs de logiciels et de directives de conception et les chercheurs doivent tester et évaluer UGADE. C'est aussi le cas de mes collègues travaillant au laboratoire d'IBM à Toronto.

ANNEXE A

EXEMPLES DE DIRECTIVES DE CONCEPTION D'IBM

About IBM Style [IBM Style Sixth Edition 2004]

These guidelines are developed and maintained by a committee of technical editors from each IBM7 Division around the world who ensure that the guidelines address a global audience that uses a wide range of IBM products.

The guidelines are arranged alphabetically by topic title.

Abbreviations, list of

Overview: Do not use a list of abbreviations separate from the glossary

Details: Except for IBM maintenance manuals, do not provide a separate list of abbreviations. Include individual abbreviations among the entries in the glossary, which defines both terms and abbreviations.

Dashes

Overview: Avoid using the em dash and en dash

Details: Avoid using em dashes to set off parenthetical phrases, because such a construction can be ambiguous to translators. Use parentheses instead, or rewrite the sentence.

Example (incorrect): If the results of the user's actions cannot be made obvious immediately--for example, if a network delay interferes--tell the user that the actions are still being processed.

Example (correct): If the results of the user's actions cannot be made obvious immediately, tell the user that the actions are still being processed. This might be necessary when, for example, a network delay interferes. Do not use an em dash where a period or colon will work.

Example (incorrect): Move mode--When you issue a GET macro in this mode, ...

Example (correct): Move mode: When you issue a GET macro in this mode, ...

Because the en dash is for the most part indistinguishable from a hyphen, it is not necessary to use special coding to create an en dash. You can use a hyphen instead.

Spacing: Do not put blanks around em dashes.

Data areas

Overview: Use lowercase for spelled-out names; use all-caps abbreviations

Details: Use lowercase for the spelled-out name of a data area (control block, table, table entry, list entry, queue, queue entry, work area, buffer, parameter list, and others).

Examples:

A data control block

The page table entry

The image construction area

For treatment of an all-caps abbreviation used as the name of a data area (such as DCB, PTE, or CDE),

see Abbreviations, use and treatment and All-caps abbreviations, meanings.

Symbolic names of data areas If the programming language is not case sensitive, use all caps for the symbolic name of a data area in a

program.

Example:

OUTBUF

INAREA

Dates

Overview: Guidelines for presenting dates

Details: Follow these guidelines when presenting dates.

General guidelines: To avoid confusion as to the meaning of a date, do not use an all-numeric representation in text. In the United States, 12/1/98 means 1 December 1998; in many other countries, it means 12 January 1998.

Use the one- or two-digit figure for the day first, spell out the name of the month, and then use the four-digit figure for the year (for example, 6 June 2001); put one space between the day and the month and between the month and the year. Do not use the / symbol or hyphen in dates.

If an abbreviated form must be used (for example, in a table) use one of these forms: 1Dec2000 or 1DEC2000. For dates after the year 1999, use a four-

digit figure for the year; for dates before the year 2000, you can use a two-digit figure for the year. Do not mix two-digit and four-digit figures for the year.

Example (incorrect):

review dates: 31Dec99 and 16Jan2000

Example (correct):

review dates: 31Dec1999 and 16Jan2000

Do not use a leading zero with a single-digit date unless you need to include the zero to align dates in a list or table. In text, do not use a leading zero with a date unless you must include it to correctly show a date format (for example, in a date field) or an example of such.

Example:

In the Date field, type 01jan00.

When you refer to a month and year, do not use a comma between the month and the year.

Example (incorrect):

June, 2000

Example (correct):

June 2000 Date ranges

To express a date range in text, use through to be clear that the second date is included in the range. Do not use to or between.

To express a date range in an abbreviated form (for example, in a table) use an en dash with no spaces.

If you are using a tool that does not include the en dash, use a hyphen with no spaces. Do not use an abbreviated or hyphenated form in text, and do not mix the text form and the abbreviated or hyphenated form.

Examples (incorrect):

between June 2001 and January 2002

6-12 June 2001

June-September 2001

from 6Jun2001-12Jun2001

Examples (correct):

from June 2001 through January 2002

from 6 June through 12 June 2001

from June through September 2001

6Jun2001-12Jun2001

Equality expressions

Overview: Express equality with text instead of the equal sign

Details: In running text, indicate equality by using words such as is, equals, or is equal to in preference to using an equal sign.

Example (incorrect):

If bit 1 = 1, the routine returns control to the caller.

Examples (correct):

If bit 1 is on, the routine returns control to the caller.

If bit 1 equals 1, the routine returns control to the caller.

ANNEXE B

EXEMPLES DE DIRECTIVES DE CONCEPTION DE MICROSOFT

These sections comprise the detailed user experience guidelines for Windows. These principles were used to design Windows 7.

The goals for this official Windows User Experience Interaction Guidelines for Windows 7 are to:

- Establish a high quality and consistency baseline for all Windows- based applications.

- Answer your specific user experience questions.

- Make your job easier!

The Windows common controls are familiar, consistent, flexible, and accessible. They require no learning from experienced Windows users and they are the right choice in almost all situations.

That said, common controls work best for the usage patterns they were designed for.

Buttons, combo boxes, and sliders are examples of controls--interface elements users can manipulate to perform an action, select an option, or set a value. Here are examples of controls design guidelines:

Check Boxes

With a *check box*, users make a decision between two clearly opposite choices. The check box label indicates the selected state, whereas the meaning of the cleared state must be the unambiguous opposite of the selected state. Consequently, check boxes should be used only to toggle an option on or off or to select or deselect an item.

A typical group of check boxes.

Note: Guidelines related to layout are presented in a separate article.

Is this the right control?

Is this the right control?

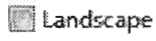
To decide, consider these questions:

- Is the check box used to toggle an option on or off or to select or deselect an item? If not, use another control.
- Are the selected and cleared states clear and unambiguous opposites? If not, use radio buttons or a drop-down list so that you can label the states independently.
- When used in a group, does the group comprise independent choices, from which users may choose zero or more? If not, consider controls for dependent choices, such as radio buttons and

check box tree views.

- When used in a group, does the group comprise dependent choices, from which users must choose one or more? If so, use a group of check boxes and handle the error when none of the options are selected.
- Is the number of options in a group 10 or fewer? Since the screen space used is proportional to the number of options, keep the number of check boxes to 10 or fewer. For more than 10 options, use a check box list.
- Would a radio button be a better choice? Where check boxes are suitable only for turning an option on or off, radio buttons can be used for completely different options. If both solutions are possible:
- Use radio buttons if the meaning of the cleared check box isn't completely obvious.

Incorrect:



In this example, the opposite choice from Landscape isn't clear, so the check box isn't a good choice.

Correct:

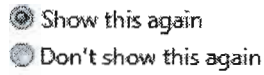
In this example, the choices are not opposites so radio buttons are the better choice.

- Use radio buttons on wizard pages to make the alternatives clear, even if a check box is otherwise acceptable.

- Use radio buttons if you have enough screen space and the options are important enough to be a good use of that screen

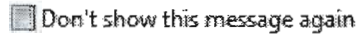
space. Otherwise, use a check box or a drop-down list.

Incorrect:



In this example, the options aren't important enough to use radio buttons.

Correct:



In this example, a check box is an efficient use of screen space for this peripheral

option.

- Use a check box if there other check boxes on the window.

- Does the option present a program option, rather than data? The option's values shouldn't be based on context or other data.

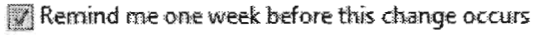
For data, use a check box list or multiple-selection list.

Usage patterns

Check boxes have several usage patterns:

An individual Choice

A single check box is used to select an individual choice.

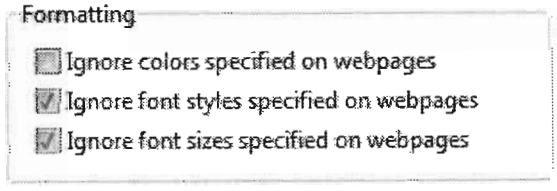


A single check box is used for an individual choice.

Independent choices (zero or more)

Unlike single-selection controls such as radio buttons, users can select any combination of options in a group of check boxes.

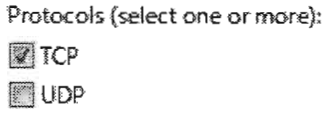
A group of check boxes is used to select from a set of zero or more choices.



A group of check boxes is used for independent choices.

Dependent choices (one or more)

You may need to represent a selection of one or more dependent choices. Because Microsoft® Windows® doesn't have a control that directly supports this type of input, the best solution is to use a group of check boxes and handle the error when none of the options are selected. A group of check boxes can also be used to select from a set of one or more choices.



A group of check boxes is used where at least one protocol must be selected.

Mixed choice

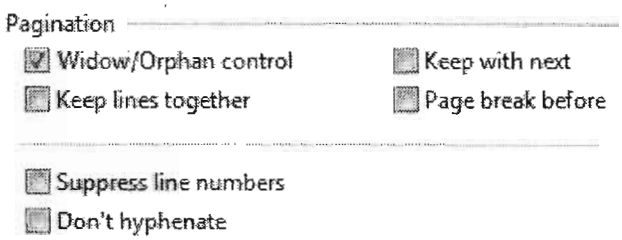
In addition to their selected and cleared states, check boxes also have a mixed state for multiple selection to indicate that the option is set for some, but not all, objects.

A mixed-state check box.

Guidelines

General

- **Group related check boxes**. Combine related options and separate unrelated options into groups of 10 or fewer, using multiple groups if necessary.



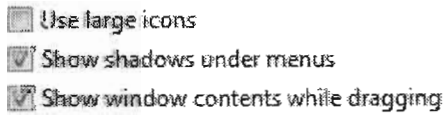
An example of groups of related, independent options.

- Reconsider using group boxes to organize groups of check boxes—this often results in unnecessary screen clutter.

- List check boxes in a logical order, such as grouping highly related options together or placing most common options first, or following some other natural progression. Alphabetical ordering isn't recommended because it is language dependent, and therefore not localizable.

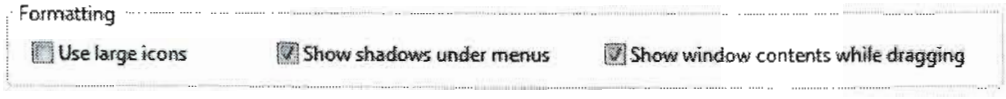
- Align check boxes vertically, not horizontally. Horizontal alignment is harder to read.

Correct:



In this example, the check boxes are correctly aligned.

Incorrect:

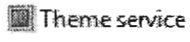


In this example, the horizontal alignment is harder to read.

- Don't use the mixed state to represent a third state. The mixed state is used to indicate that an option is set for some, but not all,

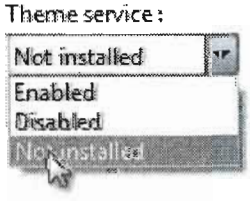
child objects. Users shouldn't be able to set a mixed state directly—rather the mixed state is a reflection of the child objects. The mixed state isn't used as a third state for an individual item. To represent a third state, use radio buttons or a drop-down list instead.

Incorrect:



In this example, the mixed state is supposed to indicate that the Theme service isn't installed.

Correct:

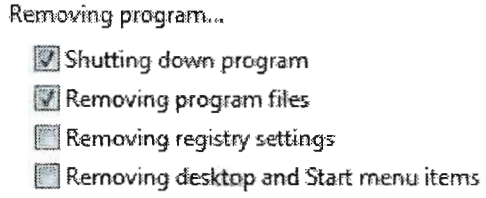


In this example, users can choose from a list of three clear options.

- Clicking a mixed state check box should cycle through all selected, all cleared, and the original mixed states. For forgiveness, it's important to be able to restore the original mixed state because the settings might be complex or unknown to the user. Otherwise, the only way to restore the mixed state with confidence would be to cancel the task and start over.

- Don't use check boxes as a progress indicator. Use a progress indicator control instead.

Incorrect:



In this example, check boxes are used incorrectly as a progress indicator.

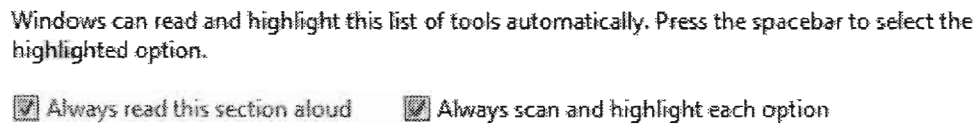
Correct:

Example of a typical progress bar.

- Show disabled check boxes using the correct selection state. Even though users can't change them, disabled check boxes convey

information so they should be consistent with results.

Incorrect:



In this example, the "Always read this section aloud" option should be cleared because the section isn't read when the option is disabled.

- Don't use the selection of a check box to:

- Perform commands.

- Display other windows, such as a dialog box to gather more input.

- Dynamically display other controls related to the selected control (screen readers cannot detect such events).

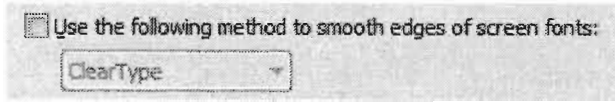
Don't show this <item> again

- Consider using a *Don't show this <item> again* option to allow users to suppress a recurring dialog box only if there isn't a better alternative. Try to determine beforehand if users really need the dialog; if they do, always show the dialog, and if they don't, eliminate the dialog.

For more guidelines and examples, see Dialog Boxes.

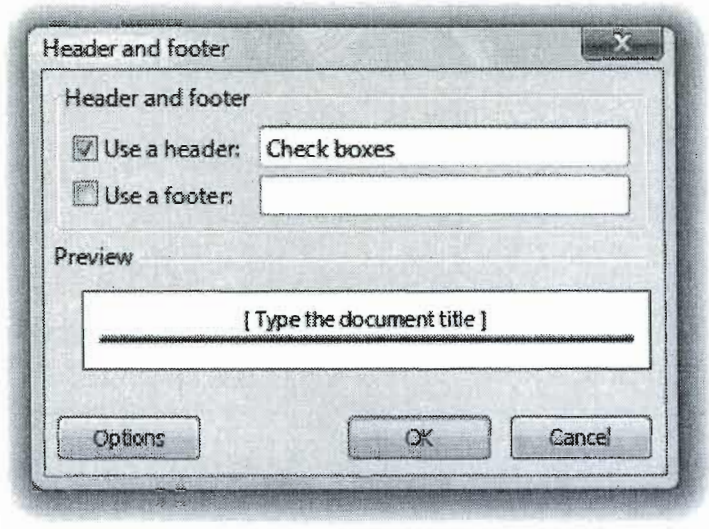
Subordinate controls

- Place subordinate controls to the right of or below (indented, flush With the check box label) the check box and its label. End the check box label with a colon.



In this example, the check box and its subordinate control share the check box label and its access key.

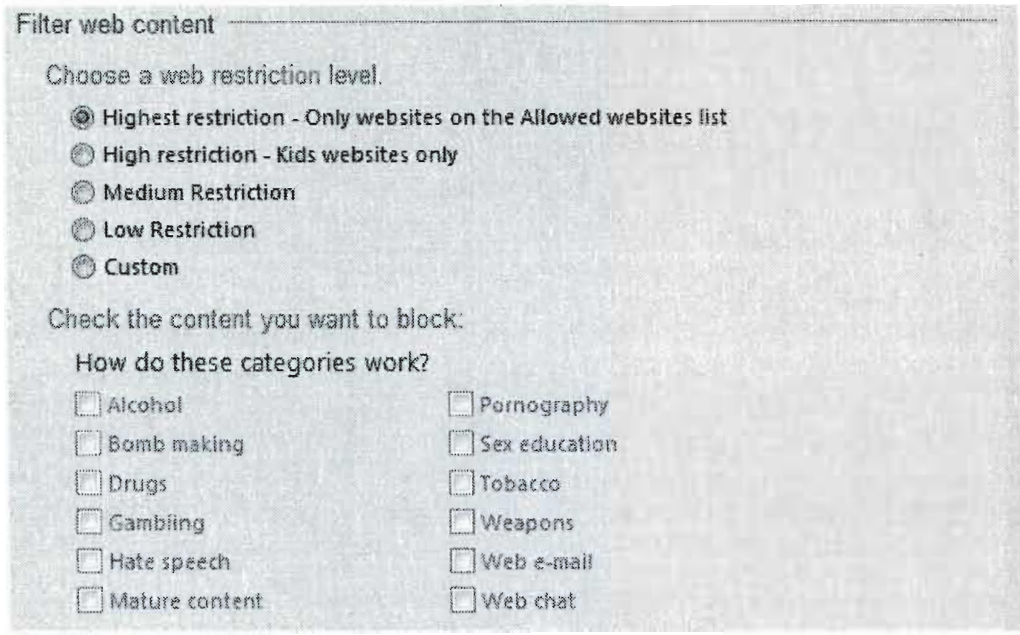
- Leave dependent editable text boxes and drop-down lists enabled if they share the check box's label. When users type or paste anything into the box, select the corresponding option automatically. Doing so simplifies the interaction.



In this example, entering a header or footer automatically selects the option.

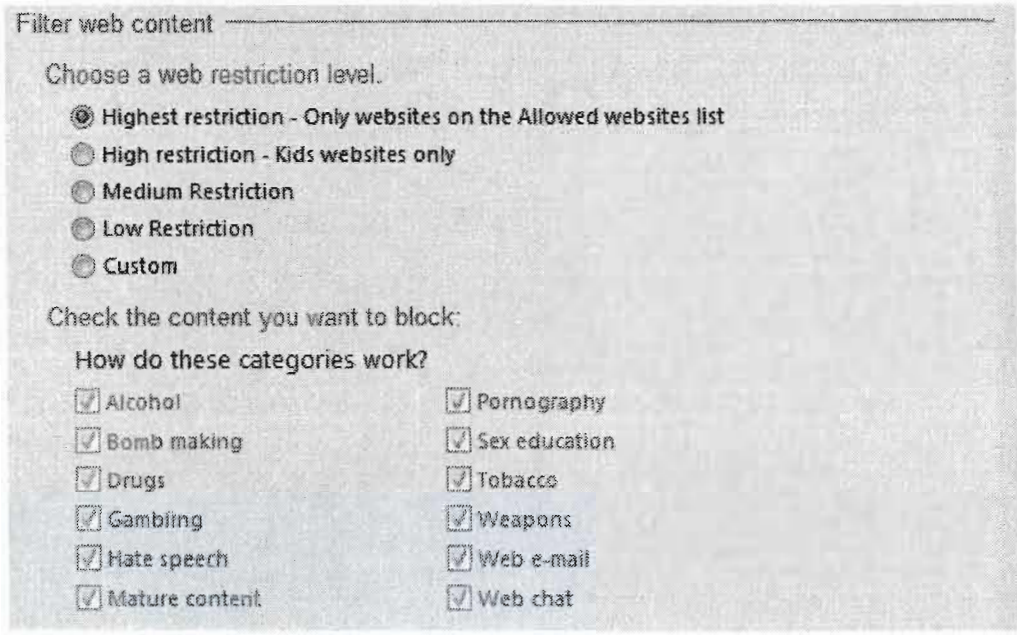
- If you nest check boxes with radio buttons or other check boxes, disable these subordinate controls until the high-level option is selected. Doing so avoids confusion about the meaning of the subordinate controls.
- Make subordinate controls to a check box contiguous with the check box in the tab order.
- If selecting an option implies selecting subordinate check boxes, explicitly select those check boxes to make the relationship clear.

Incorrect:



In this example, the subordinate check boxes aren't selected.

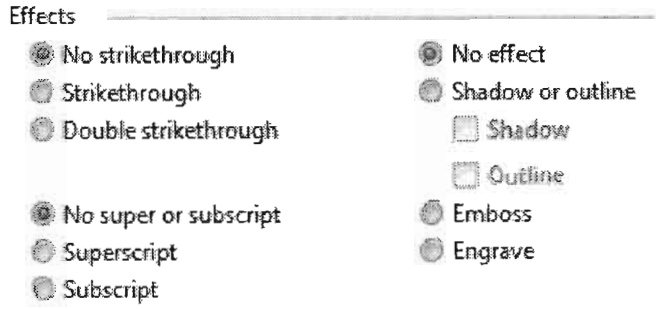
Correct:



In this example, the subordinate check boxes are selected, making their relationship to the selected option clear.

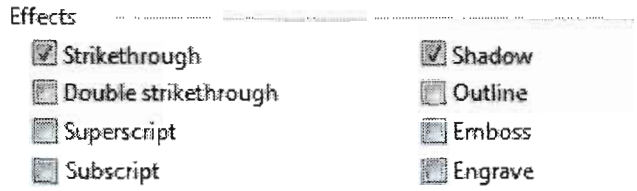
- Use dependent check boxes if the alternatives add unnecessary complexity. While check boxes should be independent options, sometimes alternatives such as radio buttons add unnecessary complexity.

Correct:



In this example, the use of radio buttons is accurate, but creates unnecessary complexity.

Better:



In this example, the use of check boxes is simpler and allows users to focus on selecting the desired options instead of on their complex relationship.

Important: Apply this guideline only in extremely rare circumstances, when showing the dependencies adds significant

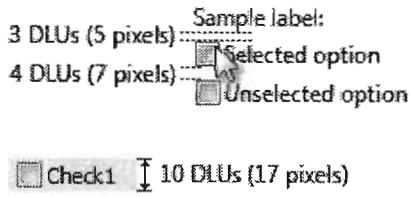
complexity without adding clarity. In the previous example, it is unlikely that users would attempt to choose both superscript and

subscript, and if they did, it would be easy to understand that they were exclusive options.

Default values

- If a check box is for a user option, set the safest (to prevent loss of data or system access), most secure and private state by default. If safety and security aren't factors, select the most likely or convenient value.

Recommended sizing and spacing



Recommended sizing and spacing for check boxes.

Labels

Check box labels

- Label every check box.
- Assign a unique access key to each label. For guidelines, see Keyboard.
- Use sentence-style capitalization.
- Write the label as a phrase or an imperative sentence, and use no ending punctuation.
- Exception: If a check box label also labels a subordinate control that follows it, end the label with a colon.
- Write the label so that it describes the selected state of the check box.
- For a group of check boxes, use parallel phrasing and try to keep the length about the same for all labels.
- For a group of check boxes, focus the label text on the differences

among the options. If all the options have the same introductory text, move that text to the group label.

- Use positive phrasing. Don't phrase a label so that selecting a check box means not to perform an action.

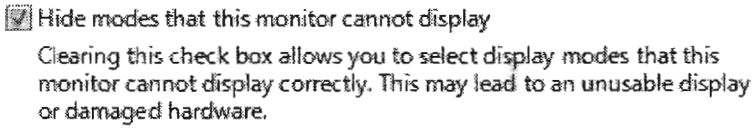
- Exception: Don't show this <item> again check boxes.

Incorrect:

In this example, the option doesn't use positive phrasing.

- Describe just the option with the label. Keep labels brief so it's easy to refer to them in messages and documentation. If the option requires further explanation, provide the explanation in a static text control using complete sentences and ending punctuation.

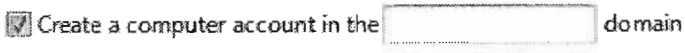
  Note: Adding an explanation to one check box in a group doesn't mean that you have to provide explanations for all check boxes in the group. Provide the relevant information in the label if you can, and use explanations only when necessary. Don't merely restate the label for consistency.



In this example, a check box label has additional explanatory text beneath it.

- If an option is strongly recommended, consider adding "(recommended)" to the label. Be sure to add to the control label, not The supplemental notes.
- If you must use multi-line labels, align the top of the label with the check box.
- Don't use a subordinate control, the values it contains, or its units label to create a sentence or phrase. Such a design isn't localizable because sentence structure varies with language.

Incorrect:



In this example, the text box is incorrectly placed inside the check box label.

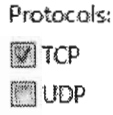
Check box group labels

- Use the group label to explain the purpose of the group, not how to

make the selection. Assume that users know how to use check boxes. For example, don't say "Select any of the following choices".

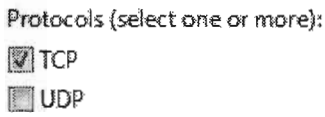
- End each label with a colon.
- Don't assign an access key to the label. Doing so isn't necessary and it makes the other access keys harder to assign.
- For a selection of one or more dependent choices, explain the requirement on the label.

Correct:



In this example, users might think that they can only make one selection.

Better:



In this example, it's clear that users can make more than one selection.

Documentation

When referring to check boxes:

- Use the exact label text, including its capitalization, but don't include the access key underscore or colon. Include the word *check box*.

- Refer to a check box as a *check box*, not *option*, *checkbox*, or just *box*, because *box* alone is ambiguous for localizers.
- To describe user interaction, use *select* and *clear*.
- When possible, format the label using bold text. Otherwise, put the label in quotation marks only if required to prevent confusion.

Example: Select the Underline check box.

BIBLIOGRAPHIE

Artim J. 1998. Incorporating Work, Process And Task Analysis Into Commercial And Industrial Object-Oriented Systems Development, http://gate.cruzio.com/~artim/CHI98

Artim J. 1997. Integrating User Interface Design through Task Analysis and Use Cases, http://www.cutsys.com/CHI97/Artim.html.

Addison W. 1992. Apple Computer Inc. Macintosh Human Interface Guidelines.

Apple ttp://developer.apple.com/documentation/UserExperience/Conceptual/OSXHI Guidelines/index.html

Buxton W. et al. 1983. "Towards a Comprehensive User Interface Management System," in Proceedings SIGGRAPH'83: Computer Graphics.

Bias & Mayhew DJ. 1994. Cost-justifying usability.

Brown C. 1988. Human-Computer Interface Design Guidelines, Ablex, New Jersey.

Donahue G. M. 2001. Usability and the bottom line. IEEE Software, 18, 1, 31-37.

Gulliksen J, Lantz A. & Boivie I. 1998. USER CENTERED DESIGN IN PRACTICE – PROBLEMS AND POSSIBILITIES. // Proceedings of the CSCW '98 conference.

Heckel P. 1991. The Elements of Friendly Software Design, Sybex, San Francisco.

ISO/IEC FCD 9126 1998. Information Technology, Spftware Product Evaluation, Quality, Characteristics and guidelines for their Use.

Gould J. D. and C. H. Lewis. 1985. Designing for Usability - Key Principles and What Designers Think, Communications of the ACM, vol. 28.

Jeff J. 2000. GUI Bloopers: Don'ts and Do's for Software Developers and Web Designers.

Jakob N. 2005. Alertbox: Durability of Usability Guidelines

Sharp J, Longshaw A and Roxburgh R. 2003. Microsoft Visual J# .NET (Core Reference) by John Sharp, Andy Longshaw and Peter Roxburgh Microsoft Press.

Karat. 1997. Evolving the scope of user-centered design. Communications of the ACM, 40, 7, 33-38. 1997

Kazman 2003. Workshop overview: Bridging the gaps between software engineering and human-computer interaction.

Landauer T. K. 1995. The trouble with computers: Usefulness, usability, and productivity.

Marcus A. 2002. Return on investment for usable UI design.

Mayhew D. J. 1999. The usability engineering lifecycle: A practitioner's handbook for user interface design.

Microsoft Corporation. 1992. The Windows Interface: An Application Design Guide, Microsoft Press, Redmond, WA.

Maskery 200. http://www.maskery.ca

Myers B.A. 1998. "A Brief History of Human Computer Interaction Technology." *ACM* interactions.

Martijn V W, Gerrit C and Anton E. 2000. Patterns as Tools for User Interface Design.

Martijn W. 2000. A structure for usability based patterns. Pattern collection and position paper at the CHI 2000 Workshop on Pattern Languages for Interaction Design: Building Momentum (The Hague, The Netherlands).

Norman D. A. & Draper S. W. 1986. User centered system design: New perspectives on human-computer interaction, 32-65. Hillsdale, NJ.

Palay, A.J., et al. 1988. "The Andrew Toolkit - An Overview," in Proceedings Winter Usenix Technical Conference.

Rosson. 1999. Integrating Development of Task and Object Models. Communications of the ACM, Vol. 42(1).

Ron B. and Ori S. 2002. IBM WebSphere Application Server: The Complete Reference.

Strong G. W. 1994. New Directions in Human-Computer Interaction Education, Research, and Practice.

Standish Group. 1995. The CHAOS report. Technical report.

Smith S. L., Mosier J. N. 1986. Guidelines for Designing User Interface Software, ESD-TR-86-278, Technical Report, The MITRE Corporation.

Seffah, A. 1999. Integrating Human factors analysis techniques with use cases and OO methods, Vol. 1743.

Seffah A. and Hayne. 1999. Integrating Human Factors in Use Case and OO Methods, 12th European Conference on Object-Oriented Programming Workshop Reader, Lecture Notes in Computer Science 1743.

Sun Microsystems 2001. Java Look and Feel Design Guideline

Sun Microsystems 1996. JavaSoft, *JavaBeans.* Sun Microsystems, JavaBeans V1.0,

Tidwell J.1999. Common Ground: A Pattern Language for Human-Computer Interface Design.

Vredenburg, K. 2003. Building ease of use into the IBM user experience.

William B. 2002. The complete handbook of the Internet

ZARMER C.L. AND JOHNSON.A 1990. User-interface tools: Past, present, and future trends. Hewlett-Packard Laboratories Tech. Rep. HPL-90-20, Hewlett-Packard, Palo Alto, Calif.