Outils de generation d'interfaces : etat de l'art et classification

H. El Mrabet

INRIA

UNITÉ DE RECHERCHE
INRIA ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports Techniques

1 9 9 2

ème anniversaire

N° 126

Programme 3
Intelligence artificielle, Systèmes cognitifs et Interaction homme-machine

OUTILS DE GENERATION D'INTERFACES : ETAT DE L'ART ET CLASSIFICATION

Hamid EL MRABET

Février 1991

Programme 3
Communication Homme-Machine

OUTILS DE GENERATION D'INTERFACES : ETAT DE L'ART ET CLASSIFICATION

USER INTERFACE MANAGEMENT SYSTEMS : STATE OF THE ART AND CLASSIFICATION

Hamid El Mrabet

Novembre 1990

OUTILS DE GENERATION D'INTERFACES : ETAT DE L'ART ET CLASSIFICATION

Résumé

Les outils de génération d'interfaces utilisateurs ont beaucoup évolué ces dernières années. Cette évolution est le fruit des besoins pressants des concepteurs d'interfaces et des utilisateurs finaux des systèmes interactifs. Les systèmes de fenêtrage, les boîtes à outils et les systèmes génériques connaissent un grand succès auprès des concepteurs des interfaces utilisateurs. Cependant, leurs inconvénients posent encore des problèmes pour la génération d'nterfaces, flexibles et réutilisables. Les machines à images abstraites permettent de structurer une application interactive et de séparer les différents niveaux d'abstraction de l'interaction. Les acquis permis par cette première génération d'outils ont favorisé le développement de nouveaux systèmes de génération d'interfaces qui sont les Systèmes de Gestion d'Interfaces Utilisateur (SGIUs)[1] . Cette nouvelle approche en pleine évolution introduit de nouveaux concepts facilitant la spécification des interfaces à un haut niveau d'abstraction, le prototypage rapide, la réutilisation des interfaces et la réduction importante du coût de conception des interfaces utilisateurs, bien que ces systèmes présentent encore un certain nombe d'inconvénients. L'objectif de ce rapport est donc de faire le point sur ces différents outils : leurs architectures, leurs modes de fonctionnement, les mécanismes qu'ils implémentent, leurs niveaux d'abstraction, leurs avantages, leurs inconvénients et leurs apports quant à la génération des interfaces ergonomiques. Une classification des SGIUs, basée sur une liste de critères détaillés, ainsi qu'un tableau récapitulatif de classification de quelques produits sont proposés.

Mots clés

SGIU (Système de Gestion d'Interface Utilisateur), interfaces utilisateurs, application interactive, boîte à outils, systèmes génériques, machine à image abstraite, modèles d'architecture, techniques de spécification d'interfaces, gestion de dialogue.

Abstract

Tools for generating user interfaces have evolved quite a lot in recent years. This evolution results both from the needs of interfaces designers and from the needs of end users of interactive systems. Window systems, tool boxes and generators are quite successful nowadays. Hovewer, they are still characterized by many problems in the generation of flexible and reusable interfaces. Systems based on abstract representations allow the structuration of interactive applications and the distinction of various levels of abstraction. These tools have stimulated the development of new systems allowing the generation of interfaces : User Interface Management Systems. This new approach, still evolutionary, introduces new concepts that help in the specification of interfaces at a high level of abstraction, rapid prototyping, interfaces re-utilisation, and contributes to a drastic decrease in the cost of user interfaces design, eventhough a number of problems are still unsolved. The goal of this report is to present a review of these various tools : their architecture, their operating modes, the mecanisms that are implemented, their abstraction levels, their advantages and drawbacks, and their contributions towards the generation of ergonomical interfaces. A classification of UIMSs based on a list of criteria is presented, as well as a survey of several existing products.

Keywords

UIMSs (User Interface Management Systems), user interfaces, interactive applications, tool boxes, generic systems, abstract image systems, architecture models, interfaces specification techniques, dialogue management.

---

1 Dorénavant nous utiliserons l'abréviation S.G.I.Us. à la place de U.I.M.S. : User Interface Management System. S.G.I.U. désigne aussi un générateur d'interfaces.

SOMMAIRE

1 INTRODUCTION

La création d'interfaces utilisateurs homogènes, cohérentes, adaptables et réutilisables est une tâche très difficile. La difficulté réside dans le manque des méthodologies, de modèles et outils performants d'aide à la conception des ces interfaces. Ces outils couvrent la gestion des périphériques d'entrée d'information et de désignation (sources d'événements), la gestion du dialogue et la gestion de la communication entre l'application (sémantique) et l'interfaces utilisateurs (présentation). Par conséquent, le cycle de développement d'une interfaces utilisateurs doit souvent faire l'objet, d'un prototypage rapide et de modifications afin d'être satisfaisant pour l'utilisateur final. Certains principes comme la séparation modulaire de l'application et de l'interfaces sont établis, et contribuent à une clarification des problèmes inhérents aux interfaces homme-machine. La tâche principale de l'interfaces utilisateurs est de décharger l'application de la gestion du dialogue et de la présentation. Les concepteurs de l'interfaces et des outils de création de l'interfaces ne sont pas nécessairement les mêmes que ceux qui conçoivent l'application. En effet, les fonctionnalités sémantiques de l'application peuvent être développées sans avoir connaissance de l'environnement de présentation graphique et de la syntaxe du dialogue utilisateur.

Un grand interêt donc pour des outils de conception d'interfaces s'exprime dans ce sens. C'est l'objet des travaux actuels dans la recherche et l'industrie des interfaces utilisateurs. Ce rapport essaie donc de faire le point sur les approches, les modèles, les outils et les techniques existantes, pour la génération des interfaces utilisateurs, essentiellement du point de vue utilisation.

La revue de ces outils couvre les boîtes à outils, les systèmes génériques (première génération d'outils) et les SGIUs (ou générateurs d'interfaces) comme nouveaux systèmes pour la génération automatique d'interfaces. Les insuffisances des boîtes à outils ont favorisé l'apparition des systèmes génériques (squelettes d'application). Ces derniers, malgré leurs apports pour la résolution de certains aspects des problèmes posés par les boîtes à outils, introduisent de nouvelles difficultés. Ces difficultés ont mené à réfléchir sur des modèles cohérents d'un système interactif et donc proposer des générateurs d'interfaces bâtis sur des systèmes génériques. L'étude faite dans le cadre de ce travail porte essentiellement sur ces systèmes : leurs définitions, leurs modèles, les différentes approches qu'ils utilisent, les techniques de spécifications d'interfaces, leurs apports, leurs avantages et leurs inconvénients. Ces différents aspects sont traités par une liste de critères de classification de ces systèmes. Un tableau récapitulatif de classification d'un certain nombre de produits est proposé à la fin de ce rapport.

2 LE PROBLEME

Plusieurs études effectuées dans le domaine de la conception des outils de génération d'interfaces utilisateurs indiquent que 40 à 50 % au moins du code total de l'application interactive est dédié à l'interfaces utilisateurs [Myers 88], [Green 87], [Coutaz 90]. La conception des interfaces utilisateurs de haut niveau telles que celles à manipulation graphique directe exige des mécanismes et outils complexes qui tiennent compte de tous les aspects de l'interfaces : la gestion élaborée des objets graphiques complexes, les différentes représentations de ces objets (interne et externe), la gestion du dialogue, la gestion des événements asynchrones des périphériques (pointage, désignation, et entrée d'information), la gestion des événements issus de l'application (sémantique), la rapidité du feedback lexical, syntaxique et sémantique, et le mode de communication avec les fonctionnalités sémantiques de l'application.

Pendant plusieurs années, les concepteurs des systèmes informatiques interactifs consacraient un temps très important à la réalisation de l'interfaces utilisateurs avec des moyens techniques très réduits et de très bas niveau d'abstraction (primitives simples liées à l'environnement physique d'exécution). Tout changement de supports d'affichage obligeait les programmeurs à modifier leurs applications (problème de portabilité et de spécificité). Depuis, pour remédier à ces contraintes, s'est fait sentir le besoin d'outils qui facilitent la tâche du programmeur, et qui permettent de rendre les interfaces utilisateurs plus ergonomiques.

3 BESOINS D'OUTILS DE CONSTRUCTION D'INTERFACES

Concevoir des interfaces utilisateurs à moindre coût et facile à utiliser requiert un certain nombre d'outils performants d'aide au développement et à l'implémentation de l'interfaces. Certains de ces outils ont déjà fait preuve de leur succès comme les standards graphiques (GKS, PHIGS), les systèmes de fenêtrage, les boîtes à outils, les machines à images abstraites, les systèmes génériques et les SGIUs (générateurs d'interfaces) comme, par exemple, Apple MacApp qui permet la réduction du temps de développement de l'interfaces utilisateurs d'un facteur de quatre à cinq [Schmucker 86]. Les paragraphes qui suivent donnent une description de ces différents outils ainsi que leurs relations.

4 OUTILS DE PREMIERE GENERATION

4.1 Boîtes à outils

4.1.1 Définition

Une boîte à outils est une librairie de primitives prêtes à l'utilisation pour la construction d'interfaces utilisateurs. Ces primitives sont organisées en deux catégories : les primitives de gestion de poste de travail et celles de la gestion du dialogue [Coutaz 90].
Les fonctions de gestion de poste de travail définissent un terminal abstrait qui rend transparent les caractéristiques physiques de la station. Les abstractions de ce terminal sont de type :
- abstractions graphiques de base : surface d'affichage ;
- abstractions graphiques construites : icône, curseur, fenêtre ;
- abstractions d'affichage structuré : technique des boîtes ;
- abstractions textuelles : police de caractères ;
- abstractions d'événements : issus des unités logiques ;
- abstractions d'événements synthétisés : issus des programmes clients.

Les fonctions de gestion de dialogue sont basées sur les services de poste de travail et proposent plusieurs objets de présentation :

- les boutons : objets élémentaires,
- les barres de défilement, les menus, et les tableaux de bord : objets complexes.

Les services de gestion du poste de travail et ceux de gestion du dialogue peuvent être réunis dans une seule bibliothèque, c'est le cas du Toolbox du Macintosh [Rose 86]. Ils peuvent aussi être répartis dans des bibliothèques différentes, c'est le cas de l'environnement Xwindows qui regroupe deux niveaux de services : Xlib (niveau serveur X) pour les primitives de base et Xt (niveau intrinsics) pour la gestion du dialogue. Les objets de Xt (widgets) sont extensibles par la définition de nouvelles classes. Des boîtes à outils élaborées ont été bâties sur Xlib et Xt comme Motif d'OSF [OSF 89], HP Toolkit [Young 89], et DEC Toolkit. Les applications clientes peuvent utiliser ces librairies de haut niveau d'abstraction ou directement les services de X (Xlib, Xt).

4.1.2 Avantages

Les avantages d'une boîte à outils peuvent se résumer dans les points suivants :

- l'extensibilité : possibilité d'ajouter de nouvelles primitives à la librairie ;
- la souplesse d'utilisation. Cette souplesse peut amener à des situations non souhaitables (cf. les inconvénients), en particulier la séparation entre la sémantique et la présentation.
- une plus grande portabilité des applications interactives ;
- l'intégration de descripteurs importants du point de vue ergonomique au sein des primitives : les attributs ergonomiques sont spécifiés par le programmeur dans les paramètres des primitives de la boîte à outils, (position des champs, leurs couleurs, taille de la police, ...) ;
- la définition d'un style d'interaction commun à plusieurs systèmes interactifs. Ceci permet une facilité de transfert de connaissances lexicales et syntaxiques entre applications.

4.1.3 Inconvénients

Plusieurs inconvénients sont constatés dans les boîtes à outils :

- une mauvaise décomposition modulaire qui remet en cause l'aspect itératif de la mise au point d'un système interactif (conséquence de la souplesse vue précédemment) ;
- une difficulté d'apprentissage due à l'aspect "en vrac" des primitives de la boîte : c'est au programmeur de choisir les primitives adéquates et les liens entre ces primitives pour répondre à ces besoins. D'après [Coutaz 90], trois mois sont nécessaires pour maîtriser une boîte à outils ;
- spécificité de l'interfaces utilisateurs vis-à-vis des applications et de l'utilisateur ;
- une existence de plusieurs boîtes à outils sur le marché, ce qui gêne la portabilité des logiciels interactifs, le transfert des connaissances lexicales et syntaxiques entre applications, et finalement la normalisation de ces boîtes à outils ;
- une duplication d'efforts : un style de programmation identique pour plusieurs applications.

Vu le bas niveau d'abstraction des boîtes à outils et la redondance de certains aspects des interfaces utilisateurs entre plusieurs applications interactives (cf. le dernier inconvénient), les systèmes génériques ou squelettes d'applications ont beaucoup participé à la résolution de ces inconvénients.

4.2 Systèmes génériques

4.2.1 Définition

Un système générique ou squelette d'application est un assemblage logiciel réutilisable et extensible pour la génération des interfaces utilisateurs [Coutaz 90]. Ce squelette est apparu pour répondre aux nombreuses insuffisances des boîtes à outils décrites précédemment. De plus, ils permettent aux developpeurs d'applications interactives de se consacrer plus à la sémantique (fonctionnalités de l'application) plutôt qu'à la présentation qui est prise en charge en grande partie par le système générique.
Un squelette d'application se compose d'un module *contrôleur réutilisable* et d'un module de *présentation réutilisable*. Le contrôleur est organisé généralement autour d'une boucle de la forme :

tantque pas_fini **faire**
    acquérir_événement ;
    traiter_événement ;
fin_tantque

Par un mécanisme de **surcharges** et d'**extensions**, on peut générer autant d'interfaces utilisateurs que d'applications. On parle dans ce cas de production d'exemplaires de contrôle et de présentation pour chaque application qu'on veut interfacer. Plusieurs systèmes proposent ce style de génération d'interfaces, parmi lesquels on peut citer APEX (APplication EXtensible) [Coutaz 87b], GROW (GRaphical Object Workbench) [Barth 86], un environnement d'objets de présentation réalisé au-dessus de Interlisp-D, Serpent [Bass 88] réalisé en langage C et OPS83 [Forgy 84] dans l'environnement Unix au-dessus de Xwindows, MacApp [Schmucker 86] (considéré aussi comme système de gestion d'interfaces utilisateurs ou générateur d'interfaces par son langage de spécification de dialogue Object Pascal).

4.2.2 Avantages

On peut résumer les avantages des systèmes génériques dans les points suivants :

- une architecture logicielle saine qui facilite la tâche du programmeur et le processus itératif de construction de l'interfaces utilisateurs. On évite ainsi le glissement de la présentation dans la sémantique ;

- une facilité d'assemblage des composants logiciels pour la présentation et le dialogue : certains assemblages sont prêts à l'emploi ;
- un niveau d'abstraction élevé par rapport à celui des boîtes à outils ;
- une possibilité d'intégrer de nouvelles abstractions dans le squelette pour la présentation et la communication avec l'application : extensibilité et souplesse grâce à la programmation objet ;
- réduction du coût de développement ;
- une certaine cohérence entre les systèmes interactifs grâce à la réutilisation du squelette ;
- l'encouragement vers la voie de la génération automatique des interfaces utilisateurs : c'est le cas des Systèmes de Gestion d'Interfaces Utilisateur ou générateurs d'interfaces (SGIUs) (cf. le paragraphe §4).

Malgré ces avantages, un certain nombre d'inconvénients se posent encore dans les systèmes génériques.

4.2.3 Inconvénients

- une difficulté de réutilisation du squelette pour des applications différentes ;
- un niveau d'abstraction encore bas fondé sur celui de la boîte à outils : le programmeur est parfois obligé de manier les primitives de la boîte pour répondre aux besoins spécifiques de présentation d'une application donnée ;
- limitation des applications à interfacer ;
- un protocole de communication (procédural APEX [Coutaz 87b] ou déclaratif Mickey [Olsen 89]) entre l'application et l'interfaces peu performant et limité (surtout pour des aspects dynamiques) : un protocole combinant les aspects procédural et déclaratif pourrait résoudre le problème :
- un manque d'intégration des services ergonomiques tels que les aides, la gestion des erreurs, les fonctions d'historique et les explications à l'utilisateur ;
- un effort important est nécessaire pour réduire la tâche de la programmation de l'interfaces utilisateurs et l'intégration du squelette dans un système de génération automatique de l'interfaces.

Les inconvénients que nous venons d'exposer font l'objet de recherches et de développement de nouveaux systèmes de génération automatique des interfaces utilisateurs à partir des spécifications de hauts niveaux. C'est le cas des SGIUs ou générateurs d'interfaces utilisateurs (cf. le paragraphe §4).

4.3 Machines à images abstraites

4.3.1 Définition

Une machine à images abstraites est une architecture logicielle de système interactif qui permet, par une série de transformations (compositions, affichages), de présenter une structure interne (abstractions) à une application cliente sur un support d'affichage et de restitution indépendamment de ses caractéristiques. Les transformations s'effectuent dans les deux sens (lecture, écriture) : le passage d'une abstraction interne (sémantique) à sa forme de présentation et l'inverse. Cette approche introduit une structure intermédiaire appelée *image abstraite* [Coutaz 90].

4.3.2 Concepts

Il existe deux catégories de machines à images abstraites : la première est spécialisée dans l'expression des relations structurelles, la seconde est fondée sur l'expression des contraintes.

Les machines à images abstraites tournées vers les relations structurelles utilisent une *représentation hiérarchique* : graphe orienté acyclique et arbres décorés d'attributs de présentation. Dans cette classe de machines, on retrouve les structures à boîtes [Knuth 79, Hibbard 84, Nanard 84, Quint 85, Quint 87, Coutaz 84, Coutaz 85a, Borras 87, Marcos 86] et le système graphique universel normalisé PHIGS [ISO 86a, Shuey 86, Shuey 87].

Les machines à images abstraites fondées sur les contraintes permettent des mécanismes d'expression dynamique des contraintes. Un exemple de ces machines est le système ThingLab [Borning 86a, Borning 86b], un environnement de spécifications de contraintes réalisé au-dessus de Smalltalk-80.

Une boîte est un mécanisme de structuration d'informations pour la présentation (images concrètes). C'est une surface avec un certain nombre d'attributs (un contour, un point de réference, un contenu) sur le support de restitution. Le contenu de la boîte est dépendant du programme client sur lequel la boîte n'a aucun regard à la différence des structures PHIGS (ensemble de primitives graphiques) qui peuvent interpréter leurs contenus. La combinaison des deux mécanismes est possible dans un système. Il consiste à remplir les boîtes par des structures PHIGS.

4.3.3 Avantages

Les machines à images abstraites présentent un certain nombre d'avantages qu'on ne trouve pas dans les systèmes de fenêtrage et les boîtes à outils. En effet, le niveau d'abstraction de ces derniers outils est proche de la machine de restitution et ne gère que des objets simples pour la présentation, alors que l'approche des machines à images abstraites permet d'aller plus loin dans la gestion des objets complexes comme ceux manipulés par les constructeurs de documents [Quint 87].

On peut résumer ces avantages dans les points suivants :

- un haut niveau d'abstraction pour la présentation des structures internes des applications clientes ;
- une architecture de mapping qui permet de passer des structures internes aux objets de présentation et inversement ;
- l'application cliente est déchargée des manipulations à caractère non sémantique (transformation des images) : ce rôle est pris en compte dans la machine abstraite ;
- une optimisation des surfaces d'affichage et de restitution ;
- une modification instantanée des objets interactifs et une incrémentabilité de l'interactivité ;
- et enfin, une meilleure structuration logicielle d'un système interactif.

5 ORIENTATION ACTUELLE : SYSTEMES DE GESTION D'INTERFACES UTILISATEUR (SGIUS)

5.1 Introduction

A la différence des outils de construction d'interfaces utilisateurs vus précédemment, les SGIUs introduisent une nouvelle approche et un niveau d'abstraction plus élevé pour la génération des interfaces utilisateurs. Dans les paragraphes qui suivent, nous essayons de faire une synthèse sur ces générateurs d'interfaces. Cette synthèse propose d'une part une vue générale sur les modèles des SGIUs (cf. §5.5., §5.6), les approches et méthodologies (cf. §6.2.1), les différentes techniques de spécification d'interfaces utilisateurs (cf. §6.2.2), et d'autre part une classification générale, basée sur une liste de critères détaillés dans le paragraphe (§6.2), de quelques produits d'origines différentes (industries, laboratoires). Ces produits ont un statut commercial ou encore expérimental.

5.2 Définition

Plusieurs définitions ont été attribuées pour un SGIU. [Hill 87] donne une définition issue de Jim Foley [Foley 84] :

"Un SGIU est un outil utilisé par un administrateur d'IU pour construire des interfaces utilisateurs pour des applications de la même façon qu'un administrateur de bases de données utilise un système de gestion de bases de données (SGBD) pour la gestion des données des applications".

Hill généralise cette définition par :

"Un SGIU est un ensemble d'outils et techniques pour la construction d'interfaces utilisateurs".
Pour être plus précis, un SGIU est un ensemble d'outils logiciels pour la *conception*, l'*implémentation*, l'*exécution*, et l'*évaluation* des interfaces utilisateurs.

[Norman et al. 86] expliquent l'utilisation d'un SGIU et ses fonctionnalités :

Un SGIU permet à un concepteur d'interfaces utilisateurs de spécifier son interface dans un langage de haut niveau. Le SGIU transforme ces spécifications en des objets d'interfaces, la gestion de ces objets et la gestion du dialogue avec l'application .
[Betts et al. 87] propose une définition en termes de fonctionnalités d'un SGIU :

Un SGIU est un outil ou un ensemble d'outils permettant une cooperation interdisciplinaire dans le développement rapide, l'organisation de dialogue, la gestion d'interaction et le contrôle dans un domaine d'application en tenant compte de différents outils d'interaction, de techniques d'interaction, et de styles d'interfaces. Un SGIU permet au concepteur d'application interactive une meilleure production. Dans ce cas, il joue un rôle analogue qu'un langage de quatrième génération dont l'effort est concentré sur la spécification plutôt que sur le codage.

5.3 Caractéristiques

Compte tenu de la définition précédente, un SGIU est utilisé à la fois dans la construction des interfaces utilisateurs et dans la gestion de l'interaction entre l'utilisateur et l'application (la gestion du dialogue). Deux catégories d'individus sont donc impliqués dans le modèle SGIU [Lowgren 88] : les concepteurs et les utilisateurs finaux des

applications interactives. En fait, pour la conception et l'utilisation d'une application interactive finale, quatre individus peuvent y être impliqués [Myers 88] : le concepteur de l'application (les fonctionnalités sémantiques), le concepteur du SGIU, le constructeur de l'interfaces utilisateurs (qui peut être un ergonome, un graphiste) et finalement l'utilisateur final de l'application interactive.

Pour les concepteurs, un SGIU doit générer des interfaces utilisateurs avec les caractéristiques suivantes :

- l'homogénieté ;
- la séparation entre la sémantique et la présentation ;
- des supports et mécanismes pour tenir compte des différents types d'utilisateurs (les experts et les novices) ;
- des supports pour la gestion et le recouvrement des erreurs ;
- le feedback lexical, syntaxique et sémantique ;
- la facilité de modification de l'interface ;
- la réutilisation des composantes de l'interface ;
- la facilité et l'accès à toutes les possibilités de l'application interactive.

Pour l'utilisateur final, un SGIU doit permettre des interfaces :

- homogènes dans différents domaines d'applications interactives ;
- avec des supports d'aide et d'assistance à différents niveaux ;
- avec des supports d'apprentissage et de guidage ;
- et avec des supports et mécanismes d'extension de l'application : accès à des fonctionnalités supplémentaires.

Toutes les caractéristiques vues précédemment sont dues au principe fondamental de la séparation entre la sémantique (l'application) et la présentation (l'interfaces utilisateurs) (cf. le critère de modularité).

5.4 Historique

L'utilisation du terme SGIU (UIMS en anglais) remonte en 1982 par Foley et Thomas [Foley 82] d'après [Hill 87a]. Dès 1968, Newman [Newman 68] décrivait un système appelé "The reaction Handler" qui avait des propriétés analogues à celles que possèdent les SGIUs actuellement. L'article de [Hill 87b] donne plus d'informations sur plusieurs systèmes pionniers et fondateurs du modèle SGIU.

Les concepts proprement dit d'un SGIU ont été introduits et définis par [Kasik 82] dans son article "An User Interface Management System".
Le modèle de Seeheim [Green 85a] introduit ces concepts dans un modèle d'architecture qui structure un SGIU en trois composants communicants : le modèle d'interfaces avec l'application (sémantique), le contrôle du dialogue (définissant la structure du dialogue entre l'utilisateur et l'application) et la présentation. Ce modèle est le premier à avoir mis en évidence le concept de SGIU et en général la structure d'une application interactive en terme d'architecture à trois niveaux. Plusieurs systèmes qui sont apparus depuis sont basés sur le modèle de Seeheim (cf. le paragraphe sur les modèles de SGIU §5.5, §5.6).

5.5 Architecture type

Une architecture type d'un SGIU est proposée Figure 1 [Coutaz 90]. Elle se compose d'un langage de spécification d'interfaces, d'un interpréteur ou compilateur de spécifications (génération d'une forme exécutable), et d'un noyau d'exécution (module central) qui peut être un système générique (squelette d'application). Le mécanisme de génération de l'interface consiste donc à spécifier l'interfaces utilisateurs par le langage proposé par le SGIU. Ensuite, par une analyse et interprétation ou compilation, le système génère une forme interne exécutable de l'interface. A partir de cette forme exécutable (stockée dans un fichier), du noyau de base (squelette), de l'application (fonctionnalités sémantiques), et d'un mécanisme de liaison (link) entre ces différents composants, le système produit l'application interactive finale (prête à l'utilisation).

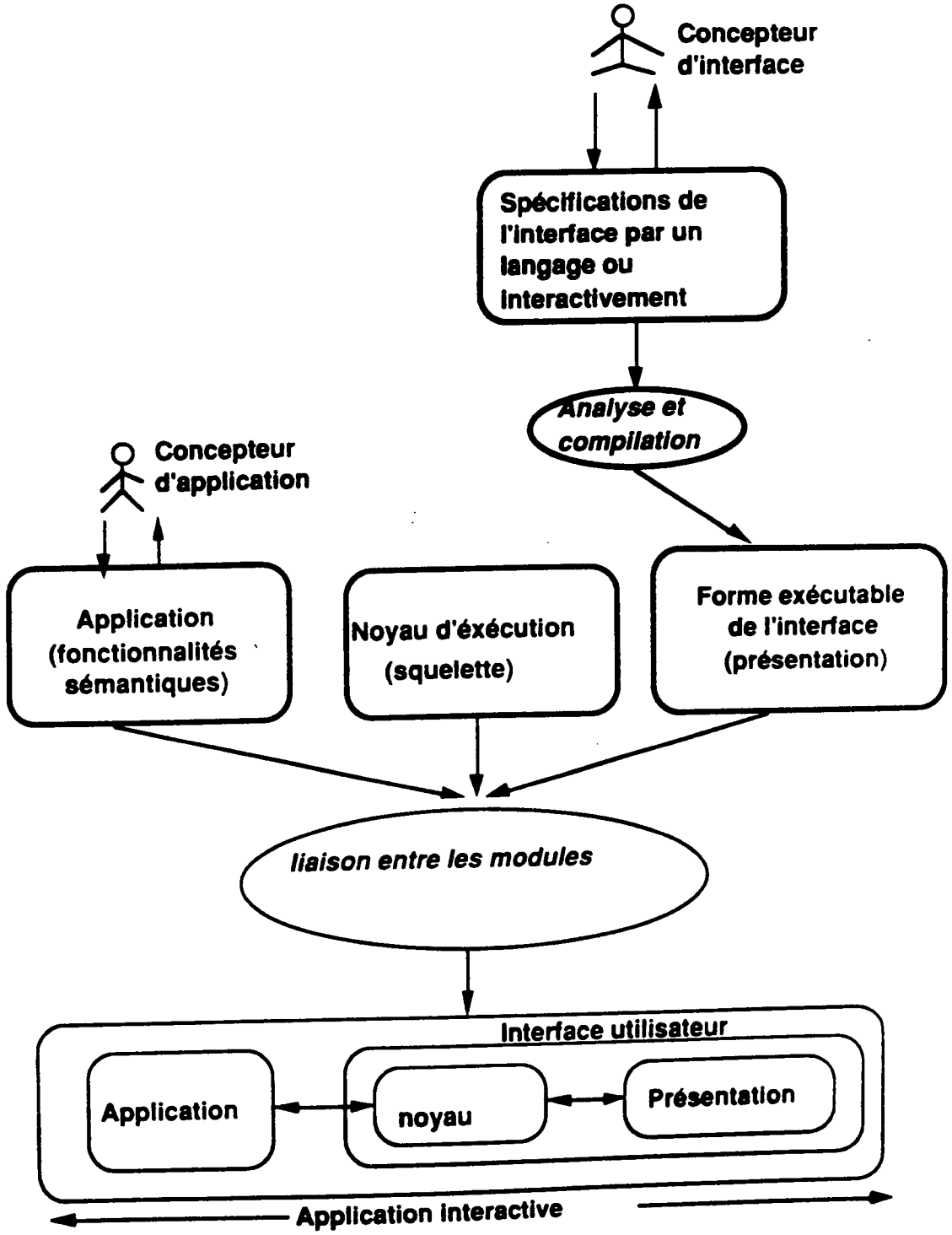

Figure 1. : la génération automatique d'interface utilisateur

5.6 Modèles d'architecture pour application interactive

5.6.1 Le Modèle MVC

Smalltalk [Goldberg 83] est le premier système avoir introduit et mis en oeuvre le concept objet dans les langages de programmation. Ce système présente la particularité d'être à la fois un langage et un environnement interactif de programmation. Les concepts-clés de Smalltalk sont les *objets*, les *classes*, les *instances*, et les *méthodes*.
Parmi les nombreuses classes développées en Smalltalk, trois classes principales distinctes sont regroupées dans le modèle MVC : **Model** (Modèle), **View** (Vue), **Controller** (Contrôleur). La classe Modèle est une représentation abstraite d'un objet avec un ensemble de fonctions, la classe Vue permet d'afficher cet objet à partir du modèle, et la classe Contrôleur permet de gérer les actions de l'utilisateur sur une vue. Le maintien de la cohérence du système est réalisée grâce à la communication de ces trois objets par un mécanisme de messages. Par exemple, si l'utilisateur déplace l'ascenseur d'une barre de défilement, le Contrôleur doit modifier la valeur courante de la barre dans l'objet Modèle qui contient la description abstraite de la barre : borne inférieure, borne supérieure et valeur courante.

Le modèle MVC présente l'avantage de structurer l'application interactive par des composantes distinctes à travers diverses classes d'objets. Cet avantage réside surtout dans la séparation de l'affichage, de la gestion des événements utilisateurs et de la représentation interne des objets manipulés.

COUSIN (COoperative USer INterface) [Hayes 83, Hayes 84, Cousin 84] est une application qui permet de construire des interfaces de type "formulaires" pour des applications développées dans le projet Spice [Spice 84].

5.6.2 Le Modèle COUSIN

Les concepts-clés de COUSIN sont :

- une interface unique pour toutes les applications. L'interface est décrite de manière déclarative et dirigée par les données de l'application. L'avantage de cette technique est de produire des interfaces homogènes.
- des formulaires comme modèle de communication avec l'utilisateur (remplissage des champs, sélection, ...). La description du formulaire est déclarative par un langage spécifique. A partir de cette description, le système génère l'application interactive.

- un système "intelligent" de support pour le remplissage des formulaires : les champs sont typés et ont des valeurs par défaut. Ce système assure le contrôle de la syntaxe ainsi que la validité des arguments saisis.

L'application et COUSIN sont deux processus séparés communicant par un protocole. COUSIN joue le rôle d'un médiateur qui gère les entrées/sorties et le remplissage des champs par différents mécanismes (menus, clés, valeurs par défaut).

COUSIN est un bon outil pour la construction d'interfaces pour des applications textuelles, mais il ne peut pas être étendu à des applications graphiques avec une gestion de dialogue plus complexe. De plus, la spécification de l'interface est non interactive, ce qui est un inconvénient.

5.6.3 Le Modèle de référence

Ce modèle [Lantz et al. 87] est établi selon l'architecture de référence OSI [Zimmermann 80]. Il définit une architecture d'implémentation d'un système interactif dans sa structure globale. Les principales composantes de ce modèle sont (cf. les figures 2a et 2b) :
- les périphériques d'entrée/sortie ;
- un serveur de périphériques d'entrée/sortie ;
- un gestionnaire de périphérique ;
- un gestionnaire de dialogue ;
- la composante application.

Le modèle présente des caractéristiques fondamentales telles que :
- répartition sur des machines différentes ;
- concurrence : la possibilité de communiquer avec plusieurs applications en simultané et la possibilité d'utiliser différents médias de communication en simultané ;
- multimédia : texte, graphique, voix, video tactile avec différents outils de pointage (souris, stylo optique, tablette graphique, clavier, ...) [Bolt 84] ;
- compatibilité : permet la réutilisation des logiciels et systèmes existants sans toucher à leur source, et de minimiser l'échange entre les systèmes existants ;
- collaboration : dialogue multi-utilisateur ;
- portabilité : sur une autre machine, sous un autre système d'exploitation ;
- configurabilité et adaptabilité : adapter l'interfaces utilisateurs aux différents utilisateurs ou classes d'utilisateurs [Lantz 86].

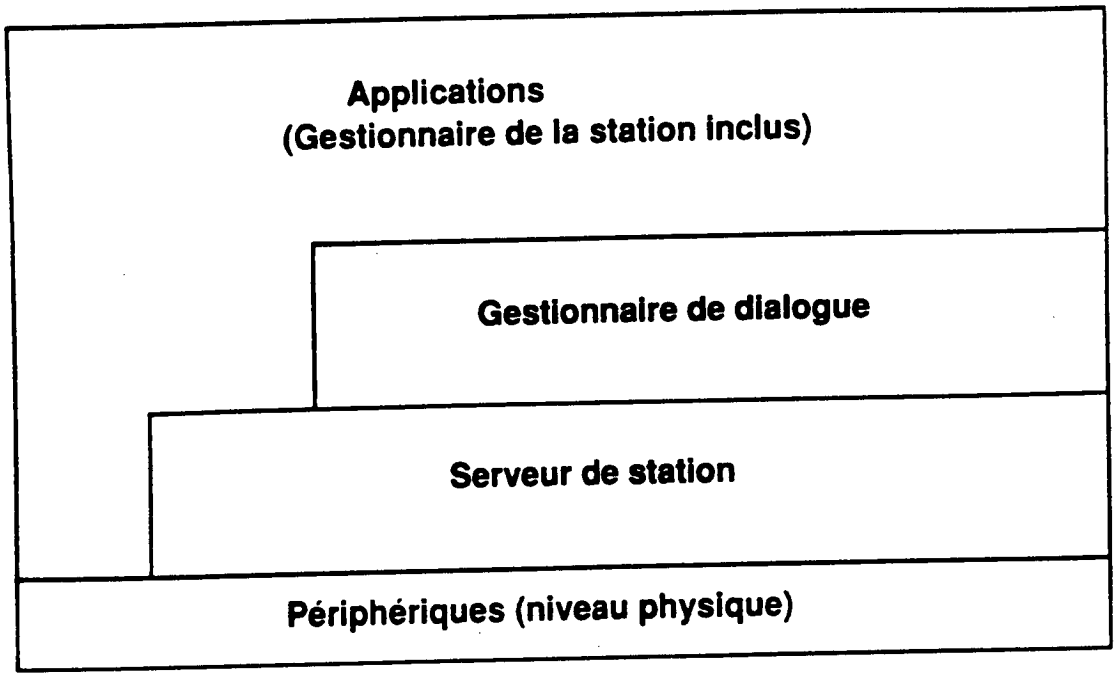

Figure 2a : vue à niveaux du modèle de référence

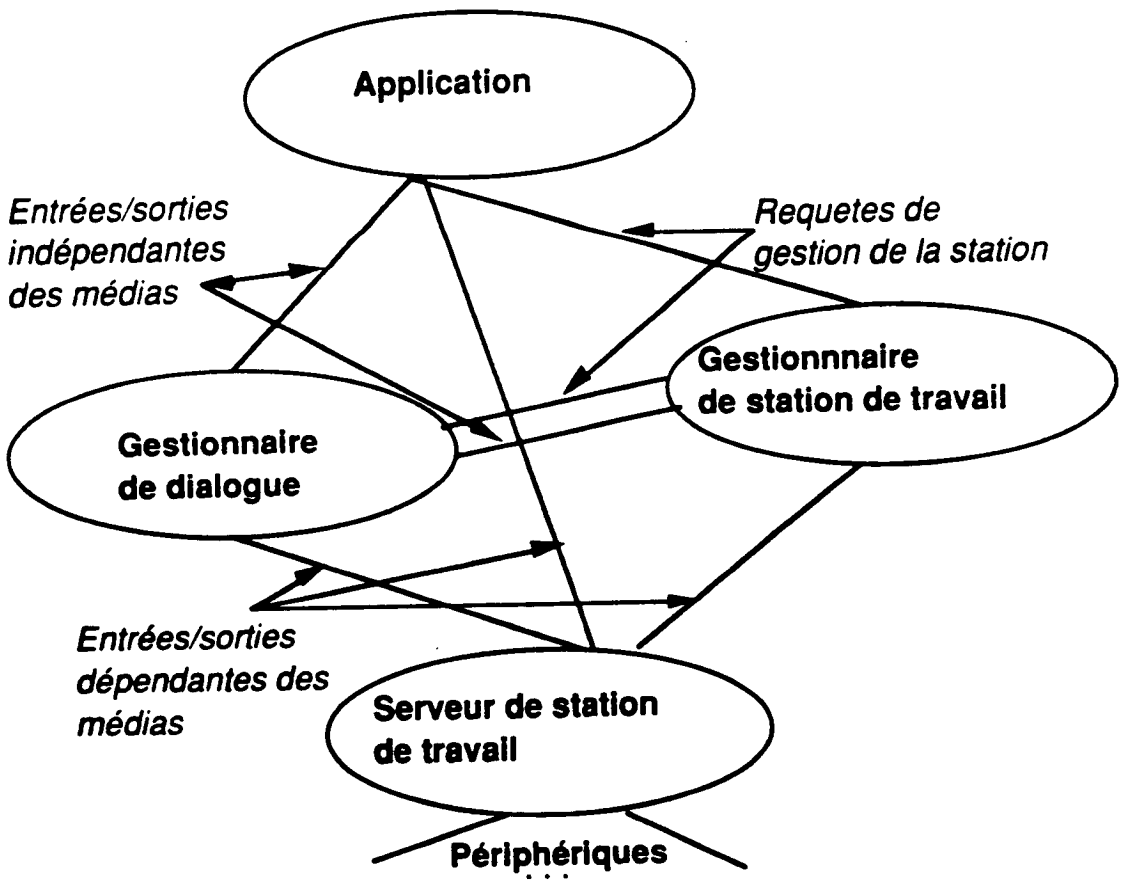

Figure 2b : vue modulaire du modèle de référence

5.6.4 Le Modèle à niveaux

Le modèle de Foley et Van Dam [Foley et Van Dam 82] structure les langages de communication utilisateurs-système et système-utilisateur en quatre niveaux d'abstractions : le niveau conceptuel, le niveau sémantique, le niveau syntaxique et le niveau lexical. Tous les niveaux peuvent être analysés et conçus d'une façon séquentielle en commençant par le niveau conceptuel.
Le niveau conceptuel correspond au modèle utilisateur de l'application. Il définit les classes d'objets, les relations entre ces classes d'objets et les opérations sur ces classes d'objets. Dans le cas d'un éditeur de textes par exemple, les objets sont les lignes de textes et les fichiers ; les relations entre ces types d'objets peuvent être qu'un fichier est une suite de lignes de textes ; les opérations qu'on peut appliquer sur un objet-ligne sont de type insérer-ligne. déplacer-ligne, effacer-ligne, copier-ligne et sélectionner-ligne.

Le niveau sémantique spécifie le détail des fonctionnalités à savoir les informations nécessaires aux opérations sur les objets, la gestion des erreurs sémantiques et les résultats de chaque opération. Au niveau sémantique, on s'intéresse à la signification des objets et les opérations sur ces objets et non la façon comment on manipule les objets (aspect syntaxique).

Le niveau syntaxique définit la séquence des entrées/sorties. Pour les entrées, la séquence est faite en général par une grammaire qui permet les règles de constitution des phrases à partir des mots (jetons). Les jetons sont de type commandes, noms, coordonnées et textes. La syntaxe des sorties inclut l'organisation de l'affichage graphique des objets en deux ou trois dimensions. Les jetons de sortie sont des unités sémantiques non décomposables.

Le niveau lexical spécifie la structure des jetons d'entrée/sortie, c'est-à-dire la façon avec laquelle ces jetons sont composés et formés à partir des primitives de base. Pour les entrées, le lexique définit le vocabulaire avec lequel sont formés les commandes, textes, coordonnées, alors que le lexique des jetons de sortie est défini par les attributs d'affichage des objets : couleur, fontes, taille des objets par exemple.

5.6.5 Le Modèle de Seeheim

Le modèle de Seeheim [Pfaff 85] a été établi pour faire la distinction entre la gestion du dialogue et la sémantique (traitements) dans une application interactive. Ce modèle comporte trois modules distincts (cf. figure 3), chaque module joue un rôle bien

déterminé. La première composante (Présentation) a pour fonction de gérer la présentation et l'affichage des objets ainsi que la gestion des événements physiques d'entrées/sorties. A ce niveau la gestion est purement lexicale. La seconde composante(Contrôle du dialogue) est le médiateur entre l'utilisateur et l'application. Ce module contrôle les commandes d'un point de vue syntaxique. La dernière composante (Modèle d'interface de l'application) assure le lien entre l'application et le module de contrôle de dialogue en faisant appel aux procédures de l'application.
Les trois modules du modèle peuvent être trois processus indépendants communiquant par échange de jetons (tokens), au sens des compilateurs. Ces jetons appartiennent à différents niveaux d'abstractions selon le module (clic souris, sélection d'un objet ...).

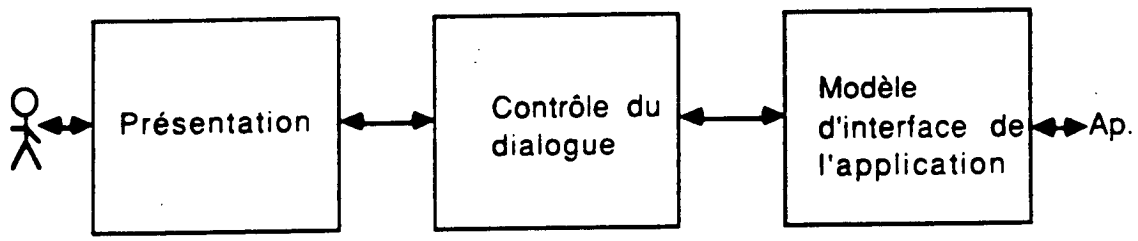

Figure 3 : Le modèle de Seeheim

5.6.6 Le Modèle PAC

Le modèle PAC [Coutaz 87] structure une application interactive en trois composantes distinctes : la Présentation, le Contrôle et l'Abstraction de l'application (sémantique). PAC a été utilisé pour structurer Mouse [Coutaz 86, Coutaz 87], un SGIU par objets et Graffiti [Karsenty 87], un système de gestion d'interfaces utilisateurs. La Présentation définit le comportement des entrées/sorties de l'application par des objets interactifs de présentation (menus, boites de dialogues, boutons, ...), l'Abstraction définit les fonctionnalités sémantiques de l'application et le Contrôle maintient la cohérence entre l'Abstraction et la Présentation (le Contrôle de PAC ne joue pas le même rôle que celui de Smalltalk). PAC est un modèle cohérent et récursif : il s'applique même aux objets interactifs qui sont décomposables en d'autres objets. La Présentation définit le comportement de l'objet vis-à-vis de l'utilisateur, l'Abstraction définit les fonctionnalités de l'objet et le Contrôle maintient la cohérence entre la Présentation de l'objet et son Abstraction. Par exemple (cf. figure 4), la modification d'un objet graphique interactif (camembert) par l'utilisateur modifie automatiquement sa valeur abstraite (valeur) associée, et inversement toute modification de la valeur abstraite d'un objet par l'application met à jour la présentation de l'objet par le Contrôleur.

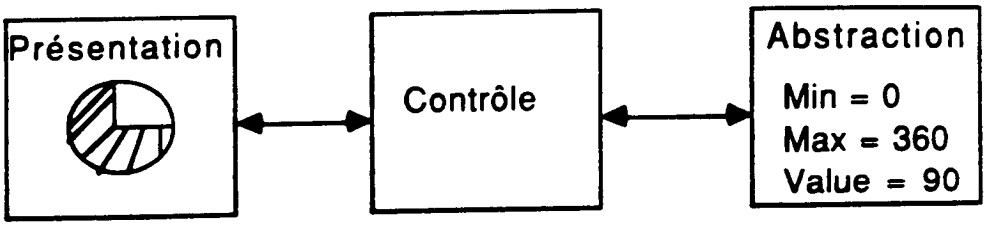

Figure 4 : Le modèle PAC

5.6.7 Le Modèle à canaux liés "linked pipelines"

Ce modèle [Cockton 88] est structuré en cinq composantes principales qui définissent l'architecture d'une application interactive :

- une composante de périphériques d'entrée : "input devices" ;
- une composante pour les périphériques de sortie : "display surfaces" ;
- une composante de gestion d'informations de saisie et d'affichage : "display manager","input manager" ;
- une composante de contrôle de communication qui comprend l'interface avec l'application et accède à la base de données d'objets images et de session : "communication controller" ;
- la composante application (non-interactive core).

Le modèle définit trois formes distinctes de feedback par le domaine de chaque fonction feedback : le feedback de bas niveau (lexical) assuré par les gestionnaires d'entrée et d'affichage d'informations, le feedback syntaxique assuré au niveau du contrôleur de communication et propose des fonctions complexes de description de séquences d'entrée d'informations, et finalement le feedback sémantique assuré au niveau de l'interface avec l'application (réponse sémantique de l'application aux actions de l'utilisateur).

5.6.8 Le Modèle IUAA-NAp

IUAA-NAp (Interface Utilisateur Auto-Adaptative-Noyau d'Application) [Mabrouk 88, EL Mrabet 89] est un modèle cohérent qui structure une application interactive en trois composantes (cf. figure 5) : IHM (Interface Homme/Machine), MAS (Module Adaptateur Superviseur de dialogue), ces deux dernières composantes constituent l'Interface utilisateurs Auto-Adaptative (IUAA) de l'application interactive, et NAp(Noyau de

l'Application). IHM gère la présentation, MAS gère le dialogue, le contrôle et l'adaptation automatique et dynamique de l'interface à l'application connectée et NAp détient les fonctionnalités sémantiques de l'application. L'avantage principal de ce modèle est l'introduction d'une meilleure structuration du feedback dans une application interactive.

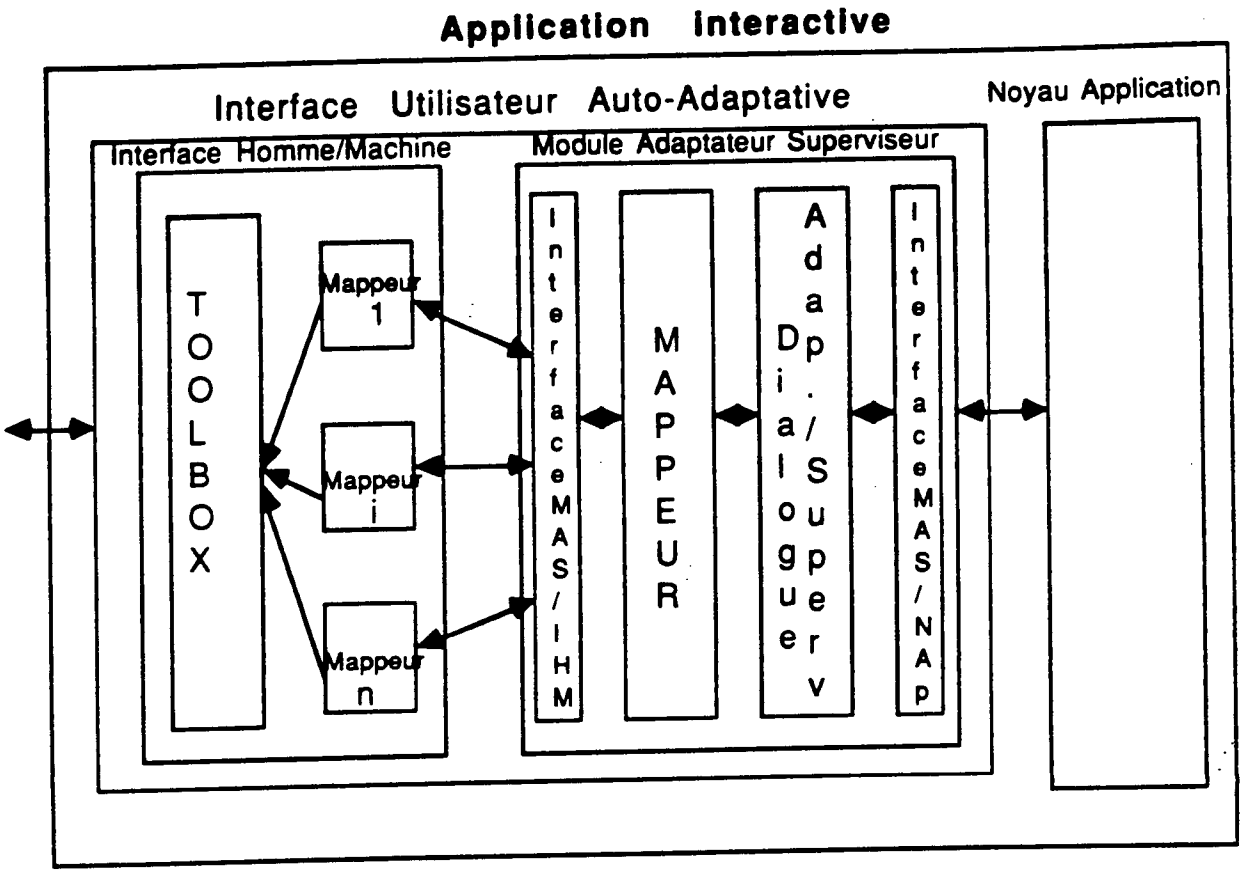

Figure 5 : Le modèle IUAA-NAp

Le modèle de base [Mabrouk 88] ainsi que sa variante par classe d'applications [El Mrabet 89] présentent plusieurs avantages :

- introduction d'un sytème de mapping qui permet de passer des concepts propres à l'application aux concepts abstraits (structures internes), des concepts abstraits aux concepts concrets de présentation et inversement (cf. figure 6) ;

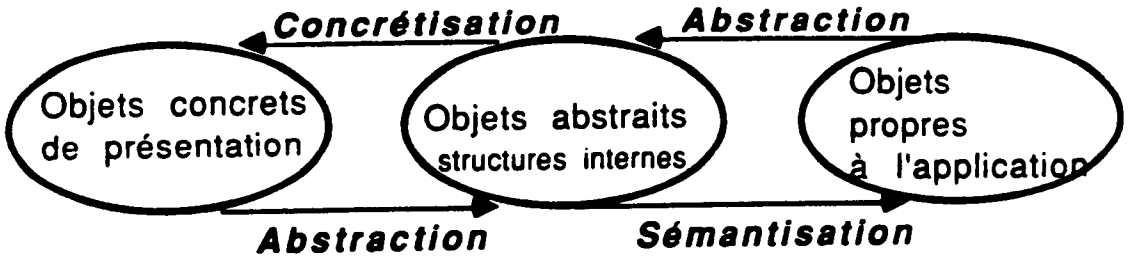

Figure 6 : Le système de mapping

- l'Interface Utilisateur Auto-Adaptative (IUAA) qui présente les caractéristiques principales suivantes :
  - Auto-Adaptativité : l'interface s'auto-adapte automatiquement et dynamiquement à toute application qui lui sera connectée ;
  - Modularité : séparation totale entre la présentation et l'application (sémantique) ;
  - Répartition : l'interface et l'application sont deux processus indépendants s'exécutant sur deux machines différentes. Cette caractéristique est réalisée par un modèle Client/Serveur.

5.7 Types d'interfaces générées par les SGIUs

Les interfaces utilisateurs à manipulation directe deviennent de plus en plus dominantes par rapport aux interfaces utilisateurs utilisant le modèle conversationnel. Le besoin est très grand pour des SGIUs qui permettent la génération des interfaces utilisateurs à manipulation directe. Ce besoin réside dans deux caractéristiques principales : le feedback sémantique et la flexibilité de la présentation. Pour répondre à ces besoins, les SGIUs proposés actuellement sont de plus en plus complexes à concevoir et difficile à l'utilisation. Un choix s'impose entre les SGIUs faciles à l'utilisation mais qui produisent des interfaces et des styles de dialogues simples et les SGIUs complexes et difficiles à utiliser mais qui produisent des interfaces meilleures [Hudson 87].
De plus, le domaine des applications à interfacer par les SGIUs actuellement est limité en particulier pour des applications graphiques complexes.

5.8 Avantages

Les SGIUs présentent plusieurs avantages résumés dans les points suivants :

- l'encouragement pour le développement et la réutilisation des composants de l'interface, ce qui facilite la construction de l'interfaces utilisateurs ;

- le gain de temps qui peut être utilisé dans la correction des erreurs et l'aide automatique ;
- la cohérence entre différentes applications : le niveau d'abstraction des entités du SGIU est le même pour toutes les applications interfacées par ce système ;
- ils favorisent la production de logiciels faciles à utiliser et à apprendre. Le developpeur de l'interface peut se concentrer sur le dialogue, indépendamment de l'application, pour avoir des interfaces utilisateurs homogènes et uniformes ;
- la consistance des interfaces ;
- le prototypage rapide ;
- le test et l'évaluation des interfaces auprès des utilisateurs finaux ;
- et enfin, ils participent à la normalisation des interfaces utilisateurs.

5.9 Inconvénients

Malgré les avantages que présentent les SGIUs, un certain nombre d'inconvénients sont à noter [Myers 88] :

- difficulté d'utilisation : apprentissage des langages de spécification des interfaces en particulier pour les constructeurs d'interfaces non-programmeurs. Bien que la manipulation directe et la génération automatique des interfaces utilisateurs aient apporté des solutions dans ce sens, le champ d'application reste tout de même expérimental ;
- un nombre réduit de fonctionnalités, en particulier pour le contrôle des objets de l'application ;
- restriction du domaine des applications supportées ;
- restriction du parallélisme (multi-applications, multi-utilisateurs, multi-outils de contrôle) ;
- restriction des fonctions dépendantes du contexte (défaire, refaire, aide, documentation) ;
- personnalisation de l'interfaces utilisateurs limitée (adaptativité) ;
- transfert de données inexistant ou limité entre applications (couper-coller) ;
- restriction de la répartition dans un environnement réseau ;
- problèmes de portabilité et de disponibilité des SGIUs dans plusieurs environnements différents (cf. critères de classification) ;
- difficulté de construction des SGIUs produisant des interfaces consistantes et homogènes ;
- manque de supports et outils d'évaluation des interfaces ;

- difficulté de séparer totalement l'interfaces utilisateurs (présentation) de l'application (sémantique) au niveau de l'implémentation ;
- difficulté de compréhension et d'édition des spécifications de l'interface : les langages de spécifications des SGIUs sont peu structurés, donc peu lisibles ;
- peu de SGIUs sont indépendants du style ;
- et enfin l'apport commercial qui est encore faible des SGIUs : les meilleurs SGIUs sont à l'état d'expérimentation.

6 CLASSIFICATION DES SGIUS

6.1 Introduction

Compte tenu d'une abondante littérature sur les SGIUs et les prototypes qui en découlent, une classification de ces produits s'impose. En effet, cette classification basée sur un certain nombre de critères que nous détaillerons plus loin permet de dégager quelques éléments pertinents pour l'étude et l'amélioration des SGIUs actuels. L'objectif principal est d'avoir donc une vision critique sur ces modèles du point de vue de leurs avantages, de leurs inconvénients et de leurs insuffisances en matière de production d'interface consistentes et adaptables à l'application et à l'utilisateur (génération du dialogue, génération de l'aide, sensitivité contextuelle, ...).

Un certain nombre de travaux ont été déjà éffectués dans ce sens [Karsenty 87], [Betts et al. 87], [Myers 88], [Hartson & Hix 89].

La classification de Myers a permis de classifier quelques SGIUs issus des laboratoires américains en fonction des techniques de specifications de dialogues [Myers 88]. Myers suggère aussi de classifier ces systèmes sur le critère de communication (contrôle) entre l'IU et l'Application.

La classification de Karsenty porte essentiellement sur un critère fondamental qui est le contrôle de dialogue (interne, externe, mixte). D'autres critères (du point de vue utilisateur et du point de vue développeur) sont aussi traités dans [Karsenty 87].

La classification proposée par Betts, Green, Foley et al. est basée sur deux catégories de critères : les critères du point de vue utilisateur et les critères du point de vue du concepteur et développeur d'interface [Betts et al. 87].

Une autre façon de classifier les SGIUs est proposée par Hartson & Hix. Elle consiste à proposer un questionnaire, regroupant un certain nombre de rubriques et critères, aux concepteurs de produits SGIUs. Chaque fournisseur de produit SGIU doit remplir ce questionnaire en précisant toutes les caractéristiques de son produit. Cette manière de procéder permet de mettre en comparaison tous les systèmes proposés en matière d'architecture, de fonctionnalités, les apports aux constructeurs d'interfaces, les apports à l'utilisateur, les outils d'implémentation ainsi que d'autres caractéristiques propres et générales. Pour plus de détails, on peut consulter [Hartson & Hix 89].

La classification que nous proposons fait une synthèse des classifications précédentes, i.e., établit une liste exhaustive de critères obtenus à partir de ceux définis dans chaque classification précédente et y ajoute d'autres critères supplémentaires d'adaptabilité des interfaces (côté application et côté utilisateur), de modèle d'application, de test et évaluation d'interface, et d'ergonomie cognitive. Tous ces critères sont détaillés dans le paragraphe qui suit. Un tableau récapitulatif de classification de quelques produits est proposé dans le paragraphe §6.3.

6.2 Critères de classification

Les critères que nous présentons ici sont répartis sur plusieurs familles ou catégories. Nous détaillons dans ce qui suit chaque famille avec des examples de SGIUs satisfaisants à ces critères.

6.2.1 Approches et Méthodologies

Cette famille de critères permet de connaître la ou les méthodologies ou approches utilisées dans la phase de conception d'un SGIU et dans sa phase d'implémentation (architecture d'exécution).

6.2.1.1 Approche Objet

Cette approche présente plusieurs avantages qui permettent la modularité d'un SGIU, l'extensibilité de l'interfaces utilisateurs par ajout de nouveaux objets interactifs de présentation, la réutilisation des composants interactifs pour des applications différentes et le contrôle des objets de l'interfaces par le mécanisme de messages.
Plusieurs systèmes (SGIUs) sont conçus en utilisant cette approche. Parmi ces systèmes, on peut citer Smalltalk [Goldberg83], Graffiti [Karsenty 87], [Beaudouin85], Aida/Masai [ILOG/INRIA], GWUIMS [Sibert 86], MacApp [Schmuker 86a].

6.2.1.2 Approche par événements

Cette approche est détaillée dans le critère de classification des SGIUs : langage à événements pour la spécification d'interfaces utilisateurs (cf. §6.2.2.1 d). A noter que cette technique est la plus performante et la plus utilisée dans les modèles SGIUs existants [Green 86]. Certains systèmes combinent cette technique avec la technique orientée-objet .

6.2.1.3 Approche par inférence

Cette approche présente deux aspects :

- une inférence au sens génération automatique du code à partir d'une spécification graphique directe de l'interface, c'est le cas, par exemple, du système Peridot [Myers 86b] qui utilise la programmation visuelle (cf. le système Peridot, annexes). Cette technique est abordée aussi dans le critère de classification : spécification graphique directe (cf. §6.2.2.2).

- une inférence au sens d'une base de connaissances de conception d'interfaces utilisateurs à un haut niveau d'abstraction. C'est le cas du système UIDE (User Interface Design Environment) de [Foley 88]. Le principe consiste à avoir une base de connaissances de conception d'interfaces utilisateurs en entrée du SGIU qui implémente l'interface. Cette base représente des objets, des actions, des attributs d'objets, des classes de hiérarchie d'objets, des pré-conditions, et des post-conditions portant sur les actions. Une transformation algorithmique peut être appliquée à cette base pour générer différentes interfaces de fonctionnalités équivalentes. La figure suivante décrit les shémas objets et les différentes relations entre ces objets implémentés dans la base de connaissances.

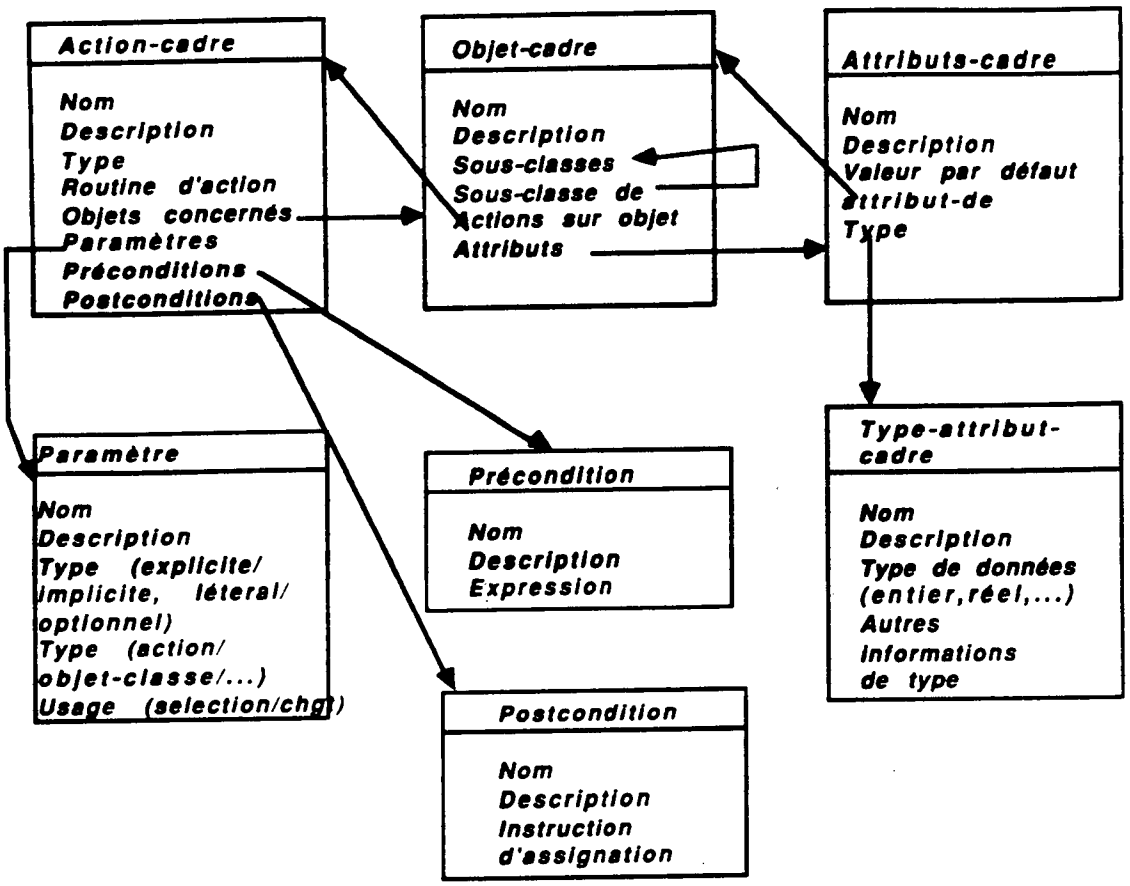

Figure 7 : Objet-cadre et ses relations

6.2.1.4 Approche avec multi-processus

Cette approche est très intéressante dans la mesure où elle structure l'interfaces utilisateurs et l'application (sémantique) sous forme de deux processus communiquants comme dans le système d'exploitation UNIX avec des techniques telles que : IPC (Inter Process Communication) ou RPC (Remote Procedure Call). L'avantage de cette structure est de permettre un aspect fondamental qui est la répartition de l'application interactive dans un environnement réseau. L'interfaces utilisateurs peut être installée sur une machine SUN et l'application sur une machine VAX par exemple. C'est une manière de séparer totalement la sémantique et la présentation. D'autres avantages sont :

- la transparence à l'utilisateur des machines sur lesquelles les applications sont installées, ainsi, l'interfaces utilisateurs lui permet d'accéder à différentes applications possibles,
- le traitement parallèle, éventuellement ;
- la communication entre les applications via l'interfaces utilisateurs ;

- une structure performante de communication entre l'interface et l'application basée sur un modèle client/serveur, c'est le cas du modèle proposé par [El Mrabet 89] ;
- l'indépendance vis-à-vis des machines d'installation.

Le problème qui se pose est de savoir comment cet aspect est tenu en compte dans un SGIU. Une discussion sur ce point est proposée dans [Lantz 87], [Dance et al. 87], [Mabrouk 88], [El Mrabet 89].

6.2.2 Techniques de spécification d'interfaces utilisateurs

Cette famille expose toutes les techniques utilisées dans les SGIUs pour spécifier l'interfaces utilisateurs à générer.

6.2.2.1 Langages de spécification

Avec cette technique, le constructeur ou le concepteur d'interfaces utilisateurs spécifie l'interface à générer en utilisant un langage spécial. Ce langage peut prendre plusieurs formes : menus arborescents, grammaires non contextuelles, réseaux de transitions, langages déclaratifs, langages à événements, langages orientés-objets et d'autres langages spécifiques à chaque SGIU.

Ces langages permettent de spécifier la syntaxe de l'interfaces utilisateurs et le séquencement des actions d'entrée/sortie. Une étude comparative entre réseaux à transition, grammaires et langages à événements a été effectuée par [Green 86].

a). Menus arborescents

La technique est basée sur une hiérarchie de menus.Toute option choisie dans un menu active un sous-menu et ceci d'une façon récursive. Plusieurs systèmes fonctionnent sur ce mode [Kasik 82], [Conklin 87].

b). Réseaux de transitions

Cette technique consiste en un ensemble d'états qui constituent le réseau et un ensemble d'arcs de transitions qui relient les états entre eux. Le mécanisme de transition est effectué par un jeton en entrée. Sur chaque arc du réseau, on peut effectuer des appels aux procédures de l'application et produire des réponses à l'affichage. Le premier SGIU à avoir introduit cette technique est celui de Newman [Newman 68]. L'avantage de ce

mécanisme est de pouvoir décrire des systèmes interactifs complexes [Harel 84]. Cet avantage devient un inconvénient quant à la complexité des réseaux qu'on peut obtenir à la suite de cette description. D'autres inconvénients sont essentiellement la description limitée de types de dialogue (Problème du contexte) et la récupération des erreurs. Des améliorations ont été introduites pour remédier à ces inconvénients. Pour réduire la complexité des réseaux, on utilise des sous-automates qui sont récursifs. Ce type de réseaux est connu sous le nom de RTN (Recursive Transition Network). Une extension de ces réseaux RTN, appelés des réseaux ATN (Augmented Transition Network) ont permis de décrire des langages sensibles aux contextes. Ces réseaux sont suggérés comme base pour la description des systèmes de gestion de dialogue [Kieras 83]. Pour remédier au problème de la récupération des erreurs, on introduit des jokers liés à des diagrammes de traitements d'erreurs. On obtient ainsi un automate fini DPDA (Deterministic Push Down Automaton) [Aho 72] modifié, justifié par l'aspect récursif.

Les réseaux à transitions ont été utilisés dans plusieurs SGIUs [Jacob 85] (cf. figure 8), [Jacob 86], RAPID/USE [Wasserman 82].

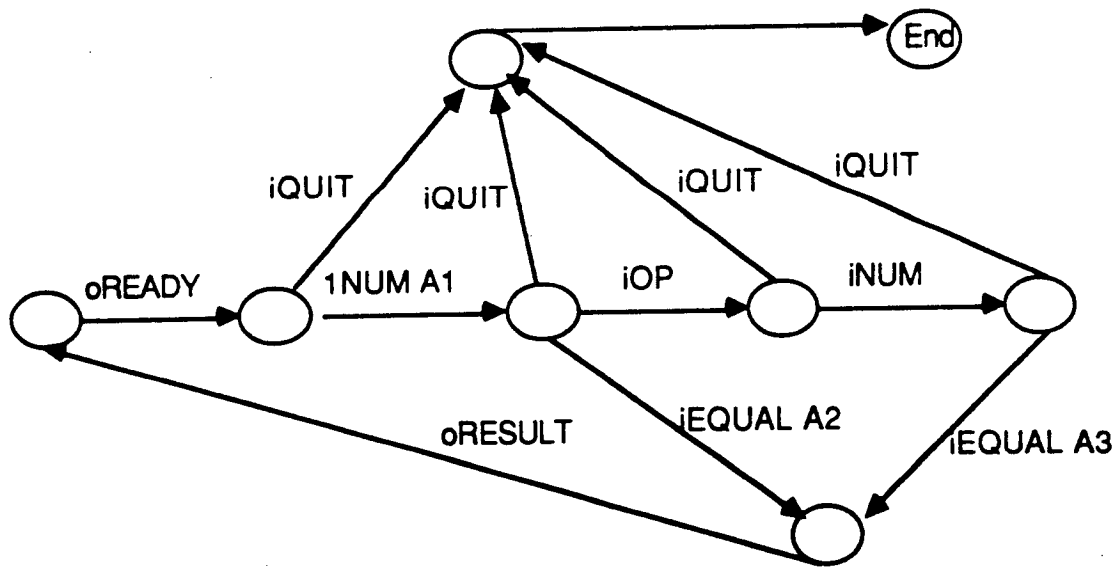

Figure 8 : Une spécification de dialogue basée sur les diagrammes à transition de Jacob : cas d'une calculette de bureau.

c). Grammaires non contextuelles

Cette technique permet de spécifier l'interface dans une forme qui s'approche de la grammaire BNF. Ce mode de spécification est très adapté pour des langages de commandes textuels, la décomposition hiérarchique et la description des objets interactifs (menus, ...). Cependant, il présente beaucoup d'inconvénients quant à la récupération des erreurs, l'annulation en cours de déroulement, en particulier pour des traitements récursifs [Karsenty87], et finalement la description des graphiques [Green86].
Ces grammaires ont été augmentées pour remédier aux inconvénients précédents Syngraph (SYNtax directed GRAPHics) [Olsen83].
Les grammaires attribuées qui sont basées sur les grammaires non-contextuelles permettent de décrire les entités de dialogue à plusieurs niveaux : lexical, syntaxique concret, syntaxique abstrait, sémantique. Ces grammaires manipulent des attributs sémantiques nécessaires à la génération du dialogue. Elles ont été introduites à l'origine par [Knuth 68] pour la description de la sémantique des langages.

Les SGIUs basés sur les grammaires non-contextuelles comme ceux basés sur les réseaux à transitions n'assurent pas d'interfaces à haute interactivité. Ils sont plutôt orientés vers des systèmes de traitements textuels avec une structure syntaxique complexe [Myers88].

d). Langages à événements

Cette technique s'inspire directement des méthodes d'implémentation d'interfaces de type gestionnaires de fenêtres. Les jetons d'entrée sont considérés comme des événements qui sont transmis directement aux handlers (serveurs de traitements des événements). Ces handlers peuvent générer à leur tour des événements en sortie et modifier l'état interne du système en activant d'autres handlers éventuellement ou en appelant les routines de l'application. Les événements sont placés dans une file suivant leur ordre d'arrivée ou bien selon certaines priorités. Les avantages de cette technique est la possibilité de description du parallélisme du dialogue avec diverses applications [Tanner 87], l'adaptation aux gestionnaires de fenêtres et la facilité de communication entre applications (ex : couper/coller) [Karsenty 87]. L'inconvénient des langages à événements réside dans la difficulté de creation de code, l'explosion combinatoire, et la maintenance du code.
La technique à événements est la plus utilisée dans la gestion du dialogue dans les SGIUs et la plus puissante [Green 85] (cf. l'exemple de couper/coller entre applications). Le gestionnaire du dialogue est vu comme une collection de Handlers qui traitent les

événements reçus et produisent d'autres événements qui activent d'autres handlers ou les fonctionnalités de l'application.
Plusieurs SGIUs sont basés sur ce mode de fonctionnement. ALGAE [Flecchia 87] utilise un langage à événement qui est une extension de Pascal. Sassafras [Flecchia 87] [Hill 87a], [Hill 87b] qui implémente un système à événements utilise la même idée mais avec une syntaxe différente. Squeak [Cardelli 85] utilise un langage textuel pour la programmation d'interfaces. Ce langage exploite la programmation concurrente avec un mécanisme de communication par messages qui ressemblent aux événements.
Tous ces systèmes gèrent plusieurs processus différents avec différents modes d'interactions, ce qui est important pour les interfaces utilisateurs.

e). Langages déclaratifs

Cette approche permet de spécifier l'interface par déclaration et non par procédure (le comment pour obtenir l'interface). Cousin [Hayes 85], Domain/Dialogue [Schulert 85] et Open Dialogue d'Apollo fonctionnent sur ce mode. Les interfaces générées par ces systèmes sont basées essentiellement sur des formulaires.
L'avantage de cette technique est de permettre au concepteur de l'interface de se concentrer sur les informations nécessaires à passer d'un état avant à un état suivant sans se préoccuper du séquencement temporel des événements. Mais plusieurs inconvénients découlent de cette approche, en particulier la limitation des types d'interfaces à construire, les types d'interactions et la gestion des objets graphiques supportés.

f). Langages orientés objets

Plusieurs SGIUs utilisent cette approche pour la spécification de l'interfaces utilisateurs. Le constructeur de l'interface a à sa disposition un certain nombre de classes qu'il peut specialiser pour définir un comportement souhaité de l'interface. Ce mode utilise le mécanisme d'héritage de SMALLTALK [Goldberg 83]. Cette approche permet une facilité de création d'interfaces et l'extensibilité des objets de l'interface, mais le domaine est non accessible aux non-programmeurs [Myers 88]. MacApp [Schmucker 86] utilise Object Pascal pour la création des applications Macintosh. GWSGIU [Sibert 86] utilise un langage orienté-objet en Lisp et propose une classification des opérations et objets de l'interface. HIGGENS [Hudson 86] utilise la même approche avec une gestion élaborée d'objets d'interface (modification automatique de la présentation quand il y a modification des structures de données associées).

6.2.3 Mécanismes de communication entre l'IU et l'Application

Ces critères permettent de connaître les mécanismes de communication utilisés par l'interface générée par un SGIU pour faire appel aux fonctionnalités sémantiques de l'application.

Quatre cas de contrôles sont possibles [Tanner 83], [Hartson & Hix 89] : le contrôle interne, le contrôle externe, le contrôle mixte, et le contrôle global.

6.2.3.1 Contrôle interne

Dans ce mode de contrôle, l'application est maître de la situation. L'interface est vue comme un ensemble de primitives que l'application peut appeler quand c'est nécessaire. Ce mode est utilisé en général par les interfaces du type boîtes à outils. L'interface assure la gestion des événements physiques d'entrée/sortie (clics souris, touches clavier) et des événements plus abstraits (sélection dans un menu, ...). L'inconvénient majeur de ce mode de communication est le fait d'avoir la gestion du dialogue dispersée dans le code de l'application. Ceci implique le non respect de la séparation totale entre la présentation et la sémantique. Ce type de contrôle conduit aussi à une incohérence entre diverses applications et à une situation de bloquage pour l'utilisateur [Karsenty 87].

6.2.3.2 Contrôle externe

Dans ce mode de communication, l'application est vue comme un ensemble de procédures que le SGIU peut appeler en fonction des commandes activées par l'utilisateur. Différentes méthodes sont utilisées pour implémenter ce modèle dans les SGIUs [Myers 88]. La première méthode consiste à ce que l'application passe tous les noms des procédures que le SGIU peut appeler si besoin est. La deuxième méthode est d'utiliser une mémoire partagée accessible par les deux composantes concernées (Le SGIU et l'application). Cette mémoire est implémentée sous forme de valeurs actives [Stefik 86], Graffiti [Karsenty 87], qui peuvent être changées automatiquement. La dernière méthode consiste à utiliser des serveurs d'événements pour communiquer avec l'application. Le contrôle externe résout une grande partie des problèmes posés par le contrôle interne, mais ne correspond pas à une solution universelle. De plus l'application peut exprimer le besoin de faire appel à l'interface dans certaines situations pour avertir l'utilisateur ou échanger des informations, d'où la nécéssité d'avoir un contrôle mixte.

g). *Autres langages de spécification*

Un certain nombre de SGIUs utilise un langage spécial de spécifications d'interfaces. FDL (Format Description Language) est le langage de spécification utilisé dans MMIMS [Smart 87]. Panther [Helman 87] permet la spécification des techniques d'interactions d'une manière textuelle par tables. Dans ces tables le concepteur d'interface spécifie la position des objets interactifs, le style de présentation, les procédures à activer et d'autres propriétés de présentation et de dialogue.

6.2.2.2 Spécification graphique directe

La technique consiste à placer les objets interactifs de présentation sur l'écran en utilisant la souris. Ainsi le constructeur de l'interfaces utilisateurs peut dessiner l'interface telle qu'elle sera au moment de l'exécution de l'application interactive. L'avantage de cette technique est de permettre le prototypage rapide et la construction des interfaces par des non-programmeurs. L'inconvénient de cette approche est la difficulté de construire le SGIU lui-même et le domaine limité des interfaces qu'on peut générer, en particulier pour les applications complexes (graphiques, temps réel). Plusieurs SGIUs proposent cette facilité, MENULAY [Buxton 83], TRILLIUM [Henderson 86], RAPID [Freburger 87], GRINS [Olsen 85a], SOS Interface [Hullot 86], PERIDOT [Myers 87a], AIDA/MASAI [ILOG/INRIA], Graffiti [Karsenty 87], TeleUse [TeleLogic].

6.2.2.3 Création automatique

Certains SGIUs fonctionnent sur ce mode. Ils permettent de générer automatiquement l'interfaces utilisateurs à partir d'une spécification des fonctionnalités sémantiques de l'application à interfacer. Une fois l'interface créée, le concepteur peut la modifier à sa guise. La motivation pour ce type de SGIUs réside dans la difficulté des constructeurs d'interfaces à utiliser les autres types de SGIUs [Myers88]. Control Panel Interface [Fisher87] utilise des types de paramètres des procédures de l'application pour créer des objets graphiques interactifs. MIKE (Menu Interaction Kontrol Environment) [Olsen86] génère l'interfaces utilisateurs à partir des procédures de l'application. IDL (Interface Definition Language) [Foley87] génère l'interfaces utilisateurs à partir d'une description sémantique de l'application par un langage type Pascal.

6.2.3.3 Contrôle mixte

Ce modèle permet au SGIU et à l'application de participer au contrôle de dialogue suivant les situations. Dans ce cas l'application est vue comme un processus pouvant communiquer avec l'interface pour échanger des informations et dialoguer avec l'utilisateur. Le contrôle mixte nécessite la définition d'un protocole de communication entre l'application et le SGIU [Karsenty 87].

6.2.3.4 Contrôle global

Ce type de contrôle a été introduit dans le système DMS (Dialogue Management System) [Hix and Hartson 86]. L'application est structurée en trois composantes indépendantes mais communicantes [Hartson et al. 84] : Gestion du dialogue, contrôle global, traitements sémantiques. Le contrôle global détient les mécanismes chargés d'évoquer le gestionnaire de dialogue et les fonctionnalités de l'application.

6.2.3.5 Modèle d'application

Le modèle d'application est un critère très important dans un SGIU. Il permet de connaître le mécanisme de communication du SGIU avec l'application. Plusieurs modèles sont possibles. Graffiti [Karsenty 87] utilise comme modèle d'application une structure basée sur les points d'entrée et les valeurs actives (mémoire partagée avec le SGIU et l'application). D'autres systèmes utilisent le mécanisme de multiprocess : l'application et l'interfaces utilisateurs sont vues comme des processus (au sens UNIX) indépendants mais communiquants (cf. l'exemple du modèle Client/Serveur, [El Mrabet 89]). La communication entre le SGIU et l'application dans ce dernier mode peut être assurée par des fonctions d'encodage et de décodage. Ce mode est très utile pour l'aspect répartition de l'application interactive et la séparation totale entre l'application et l'interfaces utilisateurs.

6.2.4 Critères vus par l'utilisateur

Cette famille de critères expose le point de vue et le jugement d'un utilisateur final vis-à-vis d'un SGIU. Ces critères concernent essentiellement la gestion du dialogue et les objets interactifs de l'interface (présentation).

6.2.4.1 Gestion du dialogue

Les critères groupés dans cette famille permettent de connaître comment un SGIU assure la gestion du dialogue avec l'utilisateur.

a). Organisation du dialogue

L'organisation du dialogue est située sur deux niveaux qu'un SGIU peut supporter. Le premier est le niveau lexical qui regroupe les objets interactifs de présentation (menus, fenêtres, boutons, ...). Le second niveau est la structure du dialogue qui exprime la syntaxe de l'interaction. A ce niveau, l'utilisateur peut modifier les commandes existantes et ajouter de nouvelles commandes à l'interfaces utilisateurs. Les nouvelles commandes peuvent être des combinaisons des commandes existantes ou des commandes qui ajoutent de nouvelles fonctionnalités à l'interfaces utilisateurs.

b). Mécanismes d'organisation du dialogue

Le niveau de sophistication du mécanisme d'organisation du dialogue qu'un SGIU peut offrir permet la flexibilité, la souplesse et l'adaptation de l'interface générée à différents types d'applications. En cas de modification de l'interface, le SGIU doit posséder des mécanismes pour faciliter la tâche de l'utilisateur pour une nouvelle version de l'interface. Plusieurs mécanismes d'organisation du dialogue sont proposés par les SGIUs. Le plus flexible est la programmation par l'exemple [Betts et al. 87], [Myers 88]. D'autres mécanismes sont aussi proposés comme par exemple la spécification textuelle basée sur l'utilisation des macros fonctions.

c). Support d'interaction

Le support d'interaction dans un SGIU est déterminé par les points suivants :

- Aide : acquisition automatique de l'aide à tout instant.

- Apprentissage intégré : celui-ci intégré dans un SGIU permet l'apprentissage rapide de l'utilisateur. Cet apprentissage ne peut être efficace que s'il peut être appelé automatiquement et d'une façon dynamique par l'utilisateur à tout moment.

- Gestion des erreurs utilisateur : une bonne interfaces utilisateurs est celle qui permet le recouvrement des erreurs utilisateur et de remédier aux actions non souhaitables. Cette

gestion d'erreurs intègre la sortie (Escape) d'un contexte, l'arrêt automatique (Stop) de l'execution d'une commande et la fonction d'annulation d'une commande executée (Undo).

- Propositions par défaut : ce point permet à l'utilisateur d'utiliser les propositions par défaut de l'application, ceci épargne l'utilisateur des connaissances nécessaires pour comprendre les capacités et possibilités offertes par l'application.

d). Style de dialogue

Pour répondre aux besoins de différentes catégories d'applications et différents types d'utilisateurs, un SGIU doit intégrer si possible un certain nombre de styles de dialogue. Plusieurs styles sont possibles :

- Interaction par simples commandes,
- Interaction par formulaires,
- Interaction par questions/réponses,
- Interaction par boîtes de dialogue,
- Manipulation graphique avec un feedback lexical,
- Manipulation graphique avec un feedback sémantique,
- Manipulation directe des images graphiques (icônes, ...).

e). Séquencement de dialogue

La souplesse et la flexibilité d'un SGIU réside dans son organisation du dialogue au sens de la liberté de l'utilisateur à naviguer dans la hiérarchie des commandes et fonctionnalités du SGIU. Plusieurs types de séquencements de dialogue peuvent être intégrés dans un SGIU :

- Non hiérarchique : dans cette structure, toutes les commandes sont accessibles à n'importe quel niveau d'organisation du dialogue (il n'y a pas de hiérarchie imposée).

- Hiérarchique : à la différence du type précédent, les commandes sont organisées dans une structure hiérarchique figée. Une commande ne peut être accessible que dans son niveau hiérarchique.

- Hiérarchie non figée : la même que la précédente mais avec une liberté donnée à l'utilisateur d'évoquer des commandes à partir des niveaux différents de hiérarchie.

- Multi-dialogue : ce type de séquencement de dialogue permet à l'utilisateur d'évoquer des commandes au moment d'exécution d'autres commandes, celles-ci peuvent être suspendues. Le multi-dialogue étendu supporté par un SGIU permet à l'utilisateur d'exécuter plusieurs commandes simultanément.

6.2.4.2 Auto-adaptativité vis-à-vis de l'utilisateur

Ce critère permet de connaître si un SGIU génère des interfaces adaptables à l'utilisateur ou à une catégorie d'utilisateur. Cet aspect peut être automatique, c'est-à-dire que l'interface s'auto-adapte automatiquement et dynamiquement à l'utilisateur, ou caractérisé par la personnalisation de l'interface par l'utilisateur si le concepteur de l'interface l'a autorisé lors de la construction. Ce dernier cas est celui implémenté par exemple dans Graffiti [Karsenty 87], les fichiers de ressources du Macintosh (personnalisation de son interfaces utilisateurs).

6.2.4.3 Génération automatique d'aide

Ce critère nous paraît très important pour l'utilisateur qui souhaite avoir à sa disposition un sytème d'aide clair et qui englobe tous les niveaux contextuels du dialogue avec l'application. L'aspect automatique de la génération consiste à génèrer l'aide au moment de la génération des composants de l'interfaces utilisateurs. Peu de systèmes SGIUs permettent cette facilité qui reste tout de même très limitée.

6.2.4.4 Génération automatique de la documentation

De la même façon que la génération automatique d'aide, un bon SGIU est celui qui permet de génèrer une documentation adaptée à l'interfaces utilisateurs qu'on peut produire avec. Dans la plupart des SGIUs, cette caractéristique n'est présente que sous forme d'un tutorial en général. Ce tutorial reste très limité quant aux explications des fonctionnalités des composants de l'interface et la gestion du dialogue avec l'application.

L'aide et la documentation font partie des plus importantes insuffisances des SGIUs proposées actuellement.

6.2.4.5 Simulation d'interaction

Pour faciliter la mise au point d'une interfaces utilisateurs, un SGIU doit permettre au constructeur de l'interface de simuler l'interaction entre l'interface et l'application sans

faire appel aux fonctionnalités réelles de l'application. L'avantage est de pouvoir adapter l'interface à un utilisateur pour une application donnée. C'est une manière efficace de valider l'interface auprès d'un utilisateur final. Une fois que l'interface répond aux besoins après modifications multiples, on peut la sauvegarder et faire le lien avec l'application pour produire une application interactive finale. Le mécanisme de simulation est fondamental pour la mise au point d'une application interactive, test et évaluation d'une interfaces utilisateurs, c'est le cas par exemple de Graffiti [Karsenty 87], Aida/Masai [ILOG/INRIA], Peridot [Myers 86b].

6.2.4.6 Facilité de manipulation et d'apprentissage

Un des aspects des problèmes posés par les SGIUs est par fois la difficulté de leur utilisation en particulier par des non initiés. La réussite d'un SGIU réside dans sa facilité d'utilisation, le niveau d'abstraction qu'il propose et les mécanismes d'apprentissage automatique. Les SGIUs présentant un niveau d'abstraction élevé (manipulation directe des objets graphiques de présentation) sont plus appréciés que les autres, mais ils sont difficiles à construire, Graffiti [Karsenty 87], Aida/Masai [ILOG/INRIA], Peridot [Myers 88] .

6.2.5 Critères vus par le concepteur

6.2.5.1 Auto-adaptativité vis-à-vis de l'application

Ce critère met en évidence une caractéristique intéressante des interfaces utilisateur. Il s'agit de générer une interface qui s'auto-adapte dynamiquement et automatiquement à toute application interactive [Mabrouk 88], [El Mrabet 89]. Certains SGIUs permettent de réutiliser des interfaces pour des applications différentes en les modifiant, mais ceci ne correspond pas à l'aspect auto-adaptatif précisé auparavant.

6.2.5.2 Modularité

Ce critère correspond au principe fondamental de séparation de l'interface (présentation) de l'application (sémantique). Toute application interactive est construite à partir de deux composantes modulaires distinctes : l'interface et les fonctionnalités sémantiques de l'application. L'inconvénient majeur d'introduire la gestion du dialogue dans le code de l'application rend difficile la tâche du concepteur à modifier la présentation et les mécanismes d'interaction entre l'utilisateur et l'application. Un meilleur respect de ce principe de séparation modulaire est d'avoir deux concepteurs différents, celui de

l'interface et celui de l'application. Cette séparation nécessite un protocole de communication entre les deux composantes de l'application interactive, [Karsenty 87].

6.2.5.3 Répartition

Cette caractéristique permet d'avoir l'application interactive répartie sur deux machines différentes dans un environnement réseau : l'interface sur une machine (X) et l'application (sémantique) sur une machine (Y). L'avantage de cette répartition est de profiter de la puissance de calcul de la machine (Y) pour l'application et la puissance graphique de la machine (X) pour l'interface (présentation). D'autres avantages sont la multi-utilisation et l'accès à plusieurs applications d'une façon transparente.

6.2.5.4 Prototypage rapide

Cette caractéristique d'un SGIU permet d'obtenir des interfaces utilisateur conformes aux besoins de l'application et de l'utilisateur final. L'interface est rapidement testée et modifiée si nécessaire. La rapidité du prototypage influe directement sur la qualité de l'application interactive produite.

6.2.5.5 Réutilisation des composantes d'interface

La possibilité offerte par un SGIU de réutiliser des composantes de l'interface augmente énormément la productivité du concepteur de l'interface et réduit d'une façon considérable le temps consacré à cette conception. Cette possibilité concourt également à faciliter le prototypage d'interfaces homogènes qui va dans le sens de la standardisation des interfaces. Plusieurs SGIUs propose cette facilité, Graffiti [Karsenty 87], Aida/Masai [ILOG/INRIA], TeleUse [TeleLOGIC], Peridot [Myers 88], Sos Interface [Hullot 86] (cf. le tableau récapitulatif de classification).

6.2.5.6 Généralité

Ce critère mis en évidence dans MOUSE [Coutaz 87] est la possibilité d'un SGIU de générer des interfaces pour une large variétés d'applications. Cousin [Hayes 85] par exemple ne permet des interfaces que pour des applications type formulaire manipulant essentiellement du texte. Certains travaux, Sos Interface [Hullot 86], Peridot [Myers 88], Graffiti [Karsenty 87], Aida/Masai [ILOG/INRIA], TeleUse [TeleLOGIC] ont permis d'étendre le champ des applications à interfacer, mais le domaine reste tout de même limité en particulier pour des applications graphiques complexes.

6.2.5.7 Architecture ouverte

On entend par ce critère, l'extension des interfaces générées par des SGIUs vers des différents mécanismes d'interaction. Ces mécanismes permettent non seulement d'inclure différents styles d'interaction dans une même application, mais de proposer un style particulier pour une application donnée, ainsi que le passage d'un style à un autre. Un SGIU possédant cet avantage est plus flexible que ceux qui ne l'intégre pas.

6.2.5.8 Test et évaluation

Ce critère ne s'applique que sur des prototypes fonctionnels. Un SGIU doit posséder des outils nécessaires aux tests et évaluations à différentes phases de developement du logiciel (spécification des besoins, conception, implémentation et maintenance).

6.2.5.9 Gestion des périphériques d'entrée : sources d'événements

Plusieurs sources d'événements sont possibles. Ces sources dépendent des périphériques d'entrée dont on dispose (souris, tablette graphique, stylo optique, ...). Le SGIU ordonne ces événements indépendamment des capacités de la machine. La gestion de ces événements est effectuée d'une façon logique, indépendante de l'implémentation physique. L'utilisateur peut activer un seul ou plusieurs périphériques simultanément suivant la vue implémentée dans le SGIU. Certains SGIUs figent l'ordre des événements d'une façon explicite ou implicite, c'est le cas de USE [Wasserman 85], Syngraph [Olsen 83a]. Le fait de fixer l'ordre des événements est un inconvénient majeur pour l'utilisateur [Karsenty 87].

6.2.5.10 Portabilité

La portabilité des SGIUs est un critère important du point de vue commercial en particulier. Un SGIU possédant cette qualité a toutes les chances d'être utilisé par un grand nombre d'utilisateurs. Cette portabilité dépend d'un certain nombre de points à prendre en considération dans un SGIU :

- Création d'un modèle approprié et abstrait pour la gestion des périphériques d'entrée (souris et autres) ;
- Création d'un modèle approprié de la gestion des sorties graphiques ;
- Création et standardisation d'un gestionnaire de fenêre. Ce gestionnaire incorpore les modèles précédents.

En plus des points précédents, les outils et langage d'implémentation d'un SGIU sont les caractéristiques principales de sa portabilité et la portabilité des interfaces qu'il génère. La portabilité est un problème crucial pour tout logiciel, en particulier un SGIU.

6.2.5.11 Intégration des critères ergonomiques

Le plus important pour un utilisateur est d'avoir à sa disposition une interfaces utilisateurs consistente, homogène, cohérente, conviviale et ergonomique. L'aspect ergonomique est essentiel pour tout logiciel, en particulier pour sa commercialisation. Beaucoup de systèmes, moins performants sur le plan fonctionnel mais possédant une bonne interfaces utilisateurs avec les caractéristiques précédentes connaissent le succès. D'autres ayant de mauvaises interfaces utilisateur trouvent d'énormes difficultés pour s'introduire dans le milieu utilisateur bien que leurs fonctionnalités soient performantes. Ce constat met en valeur les critères ergonomiques à tenir en compte dans une architecture d'un SGIU. Le problème est donc de savoir comment peut-on intégrer ces recommandations au sein d'un modèle SGIU. Une possibilté est d'intégrer un guide de style dans le modèle SGIU. Actuellement, il n'y a pas de modèle SGIU qui intègre les critères ergonomiques. En général ce domaine est laissé au soin du concepteur de l'interfaces utilisateurs.

6.3 Tableau récapitulatif de classification

Dans ce tableau (Tableau 3) nous proposons une classification de quelques SGIUs étudiés dans le cadre de ce travail. Cette classification est basée sur des familles de critères fondamentaux (traités auparavant) : les approches, les langages de spécification d'interface, le contrôle, les critères généraux.

Tableau 3

Classification des SGIUs

Critères de classification	Exemples de produits
Méthodes/Approches	
- *Approche Objet*	MacApp [Schmucker 86] GWUIMS [Sibert 86] HIGGENS [Hudson 86] Graffiti [Karsenty 87] MMIMS [Smart 87] Aida/Masai [ILOG/INRIA] TeleUse [TeleLogic]
- *Approche par événements*	Graffiti [Karsenty 87]
- *Approche par inference*	Peridot [Myers 87]
Langages de spécification	
- *Menus arborescents*	Tiger [Kasik 82]
- *Grammaires*	SYNGRAH [Olsen 83]
- *Réseaux de transitions*	RAPID/USE [Wasserman 82] [Jacob 85], [Jacob 86]
- *Langages à événements*	Squeak [Cardelli 85] ALGAE [Flecchia 87] Sassafras [Hill 87a]
- *Langages déclaratifs*	COUSIN [Hayes 85] Domain/Dialogue [Schulert 85]
- *Langages orientés objet*	MacApp [Schmucker 86] GWUIMS [Sibert 86] HIGGENS [Hudson 86]
- *Création automatique*	Control Panel Interface [Fisher 87] MIKE [Olsen 86] IDL [Foley 87]
- *Spécification graphique directe*	Graffiti [Karsenty 87] Aida/Masai [ILOG/INRIA] TeleUse [TeleLogic] Menulay [Buxton 83] Trillium [Henderson 86] RAPID [Freburger 87] Peridot [Myers 87a] Grins [Olsen 85a] SOS Interface [Hullot 86]
- *Autres langages spéciaux*	MIMMS [Smart 87] Panther [Helfman 87]

Tableau 3 (suite)

Critères de classification	Exemples de produits
Contrôles	
- *Contrôle interne*	-
- *Contrôle externe*	Aida/Masai [ILOG/INRIA] Graffiti [Karsenty 87] TeleUse [TeleLogic]
- *Contrôle mixte*	Graffiti [Karsenty 87] (possible)
- *Contrôle global*	DMS [Hix 86]
Critères généraux	
- *Prototypage rapide*	Graffiti [Karsenty 87] Aida/Masai [ILOG/INRIA] Peridot [Myers 87a] SOS Interface [Hullot 86] TeleUse [TeleLogic]
- *Modularité*	Aida/Masai [ILOG/INRIA] Graffiti [Karsenty 87] Peridot [87a]
- *Généralité*	Graffiti [Karsenty 87] (limitée) Aida/Masai [ILOG/INRIA] (limitée)
- *Réutilisation des composantes*	Aida/Masai [ILOG/INRIA] Graffiti [Karsenty 87] TeleUse [TeleLogic]
- *Répartition*	-
- *Architecture ouverte*	TeleUse [TeleLogic]
- *Simulation d'interaction*	Aida/Masai [ILOG/INRIA] Graffiti [Karsenty 87] Peridot [Myers 87a] TeleUse [TeleLogic]
- *Test et évaluation*	Aida/Masai [ILOG/INRIA]
- *Portabilité*	Graffiti [Karsenty 87] (écrit en C) Aida/Masai [ILOG/INRIA]

6.4 Points pertinents

L'étude des critères de classification des SGIUs et le tableau synthétique présenté dans les paragraphes précédents nous amènent à dégager plusieurs remarques sur la génération automatique des interfaces utilisateur. Ces remarques concernent les langages de spécification d'interfaces utilisateurs, les modèles et approches utilisés dans le développement des SGIUs, les mécanismes de communication entre l'interface et l'application, la modularité de l'application interactive, et d'autres critères de caractère général : le prototypage rapide, la gestion du dialogue, et la gestion des événements.

La séparation totale entre l'application et l'interfaces utilisateurs fait l'unanimité de l'ensemble des concepteurs des systèmes interactifs actuellement. C'est un principe fondamental à respecter. Quelques difficultés sont encore à surmonter pour implémenter cette séparation. Ceci dépend en particulier des modèles d'application (modèle à points d'entrée, modèle processus, ...).

Les systèmes SGIUs utilisant l'approche objet et les modèles à événements sont plus modulaires,souples, extensibles et performants [Green 86]. Mais le concepteur se perd dans le code de l'application SGIU une fois que sa taille devient plus importante.

Pour les langages de spécification d'interfaces utilisateurs, une tendance générale se tourne vers la spécification graphique directe et l'utilisation des techniques telles que la programmation visuelle (Peridot [Myers 87a]). Cependant l'extension des objets interactifs de présentation passe inévitablement par la programmation. A noter aussi que les SGIUs utilisant la spécification interactive directe sont complexes et difficiles à réaliser [Myers 88].

Pour les mécanismes de communication entre l'interface et l'application, un contrôle mixte est plus avantageux que les autres types de contrôle. Ceci permet d'avoir un contrôle réparti sur l'application et l'interfaces utilisateurs et une cohérence générale de l'application interactive pour éviter des situations de blocage. L'utilisation des valeurs actives et points d'entrée comme modèle de communication avec l'application a fait ses preuves dans plusieurs SGIUs (Graffiti, Aïda/Masaï, ...). D'autres mécanismes sont aussi possibles : partage des structures de données entre l'application et le SGIU, la structuration de l'application interactive en deux processus communicants (l'interface et l'application) par des fonctions d'encodage et de décodage.

Certains SGIUs (cf. le tableau récapitulatif de classification) proposent des facilités telles que le prototypage rapide, les styles de dialogues, et la simulation de l'interaction. D'autres aspects comme l'aide à la conception, la documentation automatique, la répartition (dans un environnement réseau), l'adaptabilité automatique, les recommandations ergonomiques font encore l'objet des recherches actuelles. La banalisation des SGIUs ne se fera que si on intégre par des outils adéquats ces critères fondamentaux au sein de l'architecture même du SGIU.

7 OUTILS DE CONSTRUCTION D'IU ET ERGONOMIE COGNITIVE

La prise en compte de l'ergonomie cognitive dans le processus de conception d'interfaces utilisateurs devient une nécessité [Scapin 88a], [Scapin 88b], [Scapin 89], [Coutaz 85], [Coutaz 88], [Coutaz 90], [Kieras & Polson 85]. Pour assurer cette prise en compte, les efforts de recherche actuels visent aux développements :

- de formalismes, de méthodes et de modèles théoriques. MAD (Méthode Analytique de Description des Tâches Orientée Conception d'Interfaces Utilisateur) [Scapin 88a], CLG (Command Langage Grammar) [Moran 81], TAG (Task-Action Grammar) [Payne 84], GOMS (Goal, Operator, Method, Selection) [Card 83], [Barthet 86], [Norman 86], Shneiderman 87] ;
- de guides pratiques de conception d'interfaces [Scapin 86] ;
- de stratégies de test [Shneiderman 87].

Une description des points précédents peut être consultée dans [Scapin 88a], [Scapin 88b], [Scapin 89], [Coutaz 88], [Coutaz 90], [Aschehoug 90].

8 AVENIR DES SGIUS ET DIRECTIONS DE RECHERCHE

Compte tenu des insuffisances des SGIUs existants actuellement et les besoins pressants des utilisateurs et concepteurs d'interfaces, on peut projeter l'avenir des SGIUs et les directions de recherche dans ce domaine dans les points qui suivent [Betts et al. 87], [Myers 88], [Coutaz 90] :

- prendre profit des techniques d'intelligence artificielle pour la modélisation des tâches utilisateur, la modélisation conceptuelle de l'interfaces utilisateurs. [Green 87] propose de réfléchir sur une nouvelle génération des SGIUs qui permettent la construction des interfaces utilisateur directement à partir d'un modèle de tâches, des fonctionnalités de l'application, et des propriétés de l'utilisateur ;

- définir et développer des structures de données qui peuvent être des supports adéquats pour les outils de conception d'interfaces utilisateur ;

- développer des outils de conception d'interfaces utilisateurs permettant d'utiliser le haut niveau de connaissances des concepts abstraits de l'interfaces utilisateurs pour des domaines d'applications divers et variés. Ces connaissances incluent les objets, les actions, et les relations pour un problème d'un domaine spécifique ;

- déterminer les caractéristiques d'une bonne interfaces utilisateurs, les outils d'évaluation et de quantification des ces caractéristiques ;

- développer des outils performants à la mise au point des interfaces auprès de l'utilisateur ;

- développer une classification formelle et une catégorisation des fonctionnalités d'un SGIU. Cette classification est très importante aussi pour l'évaluation des SGIUs ;

- mener une recherche en parallèle avec les systèmes distribués (aspect répartition des applications interactives). La plupart des travaux portent sur des SGIUs s'exécutant sur des simples stations de travail monopostes ;

- favoriser la génération automatique de l'interfaces utilisateurs à partir des spécifications de haut niveau (manipulation graphique directe des objets de l'interfaces). Ces spécifications peuvent être définies en termes d'objets, actions sur ces objets et les relations entre ces objets et les actions ;

- proposer des outils et mécanismes de support pour le feedback sémantique : comment concevoir et contrôler des objets dépendants de l'application ;

- encourager l'utilisation des outils et techniques conventionnelles, définis comme supports du processus de développement d'un logiciel. Ces outils incluent les outils de prototypage, de test, de mise au point, de maintenance et de configuration ;

- et enfin intégrer le savoir-faire des ergonomes dans les techniques informatiques, elles-mêmes doivent être intégrées dans une fabrique d'interfaces [Coutaz 90].

Une exigence fondamentale est d'obtenir une architecture SGIU qui permette une meilleure intégration des différents outils d'aide à la conception d'interfaces et qui implémente l'application interactive sans pénaliser la performance de cette dernière [Lowgren 88].

9 NORMALISATION

La normalisation des interfaces utilisateur est un problème qui concerne aussi les travaux de recherche. Un premier pas a été obtenu avec le standard Xwindow pour les stations de travail (comme serveur de fenêtres) ; mais il reste beaucoup d'efforts à faire dans ce sens dans l'objectif de proposer à l'utilisateur un environnement d'interaction uniforme, et cohérent. Une des difficultés majeures dans ce domaine est d'ordre commercial. Chaque constructeur ou développeur de systèmes de gestion d'interfaces tient à garder une certaine identité spécifique qui constitue sa force de vente dans le marché. L'existence de plusieurs boîtes à outils par exemple est déjà un premier obstacle à cette normalisation.

10 CONCLUSION

La génération des boîtes à outils et systèmes génériques

Dans une application interactive, deux composantes fonctionnelles entrent en jeu : les fonctionnalités sémantiques de l'application et l'interfaces utilisateurs. Le coût de développement de l'interface est considérable, environ 50 pour cent et plus du code total de l'application interactive est consacré à l'interfaces utilisateurs. Ce constat implique aussi un temps de développement très important, d'où un besoin urgent d'outils d'aide à la conception d'interface qui a été exprimé dans ce sens. Une première réponse à ces besoins est l'apparition des boîtes à outils comme librairies de primitives prêtes à l'utilisation. Ces librairies ont facilité énormément la tâche des programmeurs d'interfaces utilisateur par leurs avantages (cf. § 4.1.2). Cependant, leurs inconvénients (cf. § 4.1.3) sont des obstacles à une meilleure productivité du logiciel interactif (modularité, réutilisation des composants logiciels, modèle générique de l'interfaces utilisateurs, niveau d'abstraction). Certains aspects de ces inconvénients ont été résolus en partie par les systèmes génériques comme nouveaux outils de génération d'interfaces. Leur apport principal est de permettre au programmeur d'interface un squelette réutilisable prêt à l'utilisation, mais un effort d'adaptation à chaque application est nécessaire. Les systèmes génériques ont permis plusieurs facilités, en particulier la modularité de l'application interactive et le niveau d'abstraction qui est plus élevé que celui fourni par les boîtes à outils. (cf. § 4.2.2). Malgré ces facilités, plusieurs inconvénients sont introduits par les

systèmes génériques (cf. § 4.2.3) pour générer une interfaces utilisateurs de haut niveau. Ces systèmes n'intégrent pas suffisamment d'outils d'aide à la spécification d'interface, d'où la nouvelle génération des SGIUs.

La nouvelle génération des SGIUs

Les SGIUs sont apparus pour répondre aux problèmes posés par les boîtes à outils et les systèmes génériques. Ces générateurs d'interfaces créent une application interactive à partir des spécifications textuelles par un langage spécialisé, des formalismes étendus (grammaires, graphes de transitions), ou interactivement. Ce dernier cas domine la tendance actuelle. En effet, cette caractéristique principale des SGIUs facilite énormément le travail du concepteur d'interface dans la mesure où l'effort est concentré surtout sur les aspects dialogue entre la présentation et la sémantique.

L'expérience montre que les différentes techniques de spécification d'interfaces utilisateurs ne permettent pas d'exprimer tous les aspects d'une application interactive (feedback sémantique, gestion des erreurs utilisateur, asynchronisme des événements, parallélisme, extensions des objets interactifs de présentation). Les grammaires hors contexte sont limitées et ne permettent pas de spécifier l'interface au-delà d'une description lexicale et syntaxique. Les graphes de transition sont mal adaptés à l'expression du parallélisme et à l'aspect opportuniste du comportement de l'utilisateur. Une description de ces derniers aspects d'interaction mène à des graphes plus complexes. Les langages à événements permettent de régler le problème du parallélisme et l'aspect asynchrone de l'interaction entre l'utilisateur et l'application, mais le bas niveau d'interaction et la maintenance du code restent des problèmes inhérents. Les langages déclaratifs libèrent le concepteur de l'enchaînement des événements, mais leur problème essentiel réside dans l'extension des objets interactifs de présentation, qui est limitée voire impossible.

La spécification interactive est une technique qui permet un prototypage rapide et une évaluation immédiate de l'interface. Ce mode épargne le concepteur d'interface des contraintes de programmation. Cependant, le problème qui se pose encore est la généralité et l'extension des objets interactifs de présentation. La programmation visuelle est une technique qui mérite d'être envisagée pour remédier à ce problème.

Malgré les nombreux avantages des SGIUs (cf. § 5.8), en particulier le haut niveau d'abstraction des spécifications de l'interface (accessibles par des non-programmeurs) et la modularité de l'application interactive, certains aspects méritent encore d'être étudiés. Ces aspects concernent la répartition dans un réseau, l'adaptabilité automatique de l'interface à l'utilisateur, l'intégration des critères ergonomiques, et un système performant d'aide à la conception d'interfaces utilisateurs.

L'intégration des connaissances de l'ergonomie cognitive dans les SGIUs

Comme nous l'avons signalé dans le paragraphe précédent, les SGIUs n'intégrent pas encore les recommandations de l'ergonomie cognitive. Un effort doit être entrepris dans ce sens.

Depuis longtemps les sciences cognitives et l'informatique évoluent sur des axes parallèles. La conséquence de ce fait est le manque d'outils de conception de systèmes informatiques adaptés aux informaticiens et l'existence de logiciels mal adaptés aux utilisateurS. A partir de ce constat, il est urgent que les deux communautés collaborent au niveau de toutes les phases du cycle de développement du logiciel (spécification des besoins, analyse préalable, analyse détaillée, codage et évaluation). Cette collaboration ne peut se concrétiser que par l'intégration de la part des informaticiens des outils et formalismes fournis par les cogniticiens dans les architectures même des systèmes informatiques.

Les modèles utilisateur [Gould 88], les formalismes de description de tâches utilisateur et du fonctionnement du système MAD [Scapin 88], TAG [Payne 86], CLG [Moran 81] sont des outils nécessaires, malgré leurs insuffisances, à la définition du fonctionnement sémantique du logiciel interactif (mise en correspondance des variables psychologiques avec celles manipulées par le système) et la définition d'une présentation de haut niveau d'abstraction (cohérence lexicale et syntaxique de l'interface, feedback sémantique et choix d'un modèle d'interaction).

Concevoir un système informatique sans une analyse ergonomique préalable (aspects de présentation et de sémantique) c'est comme programmer le système sans l'appui d'une spécification de base [Coutaz 90].

Le processus d'intégration des connaissances cognitives passe tout d'abord par l'amélioration des outils et techniques de conception des systèmes interactifs en particulier les SGIUs.

Vers des environnements intégrés d'aide à la conception des interfaces

Les outils informatiques d'aide à la conception des interfaces utilisateur, traités dans ce rapport (boîtes à outils, systèmes génériques, machines à images abstraites, SGIUs ou générateurs d'interfaces) répondent partiellement aux exigences pressantes de l'utilisateur. Ces outils méritent d'être améliorés et intégrés dans des environnementS d'aide à la génération d'interfaces. L'intégration de l'ergonomie cognitive dans ces environnements passe avant tout par une modélisation de ses connaissances. Dans ce cas une collaboration étroite entre les informaticiens et les ergonomes s'impose dans l'objectif de permettre à l'utilisateur une fabrique d'interfaces [Coutaz 90]. Le rôle principal de cette

fabrique est d'aider le concepteur à toutes les étapes de génération d'interface depuis la définition des tâches jusqu'à la liaison avec l'application. L'analyse de tâches et la détermination de la présentation sont assurées par les ergonomes. Le choix d'une architecture et la programmation de l'interface sont pris en charge par les informaticiens. Le but dans le moyen ou long terme est d'intégrer toutes ces compétences au sein de l'architecture même de la fabrique par la voie de la modélisation. La figure 9 résume l'historique des outils de génération d'interfaces et met en relief l'idée exprimée dans ce paragraphe quant au besoin d'un environnement intégré d'aide à la conception des interfaces utilisateur.

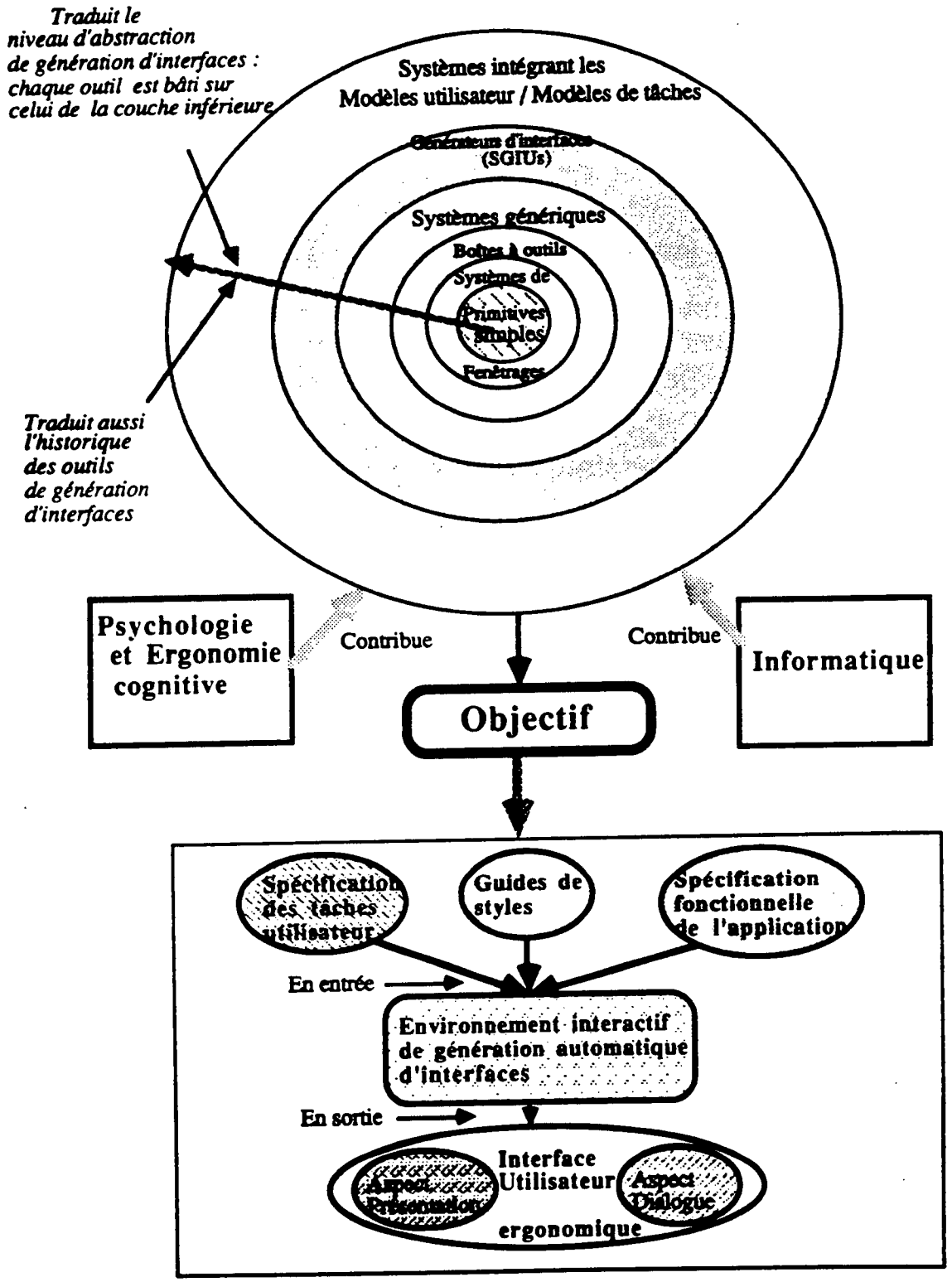

Figure 9 : Synthèse des outils de génération d'interfaces utilisateur

RÉFÉRENCES

[Aschehoug 90]
F. Aschehoug, "*La description des interfaces : une revue de la littérature*", Rapport de Recherche INRIA (à paraître).

[Alcyone]
J. M. Hullot, "*Alcyone, la Boîte à Outils Objets*", Rapport Technique n° 60, Rocquencourt : INRIA.

[Barth 86]
P. S. Barth, "An Object-Oriented Approach to Graphical Interfaces", ACM Transactions on Graphics, 5(2), April 1986.

[Barthet 86]
M.F. Barthet, "*Conception d'applications conversationnelles adaptées à l'utilisateur*", Thèse d'Etat, Institut National Polytechnique de Toulouse, France.

[Bass 88]
L. Bass, E. Hardy, K. Hoyt, R. Little, et R. Seacord, "*The Serpent run time architecture and dialogue model*", Carnegie Mellon Technical Report, CMU/SEI-88-TR-6. January 1988.

[Betts 87]
B. Betts, D. Burlingame, G. Fisher, J. Foley, M. Green, D. Kasik, S. Kerr, D. Olsen, et J. Thomas, "*Goals and Objectives for User Interface Software*", Computer Graphics 21(2), 1987.

[Borning 86a]
A. H. Borning, "*Graphically Defining New Building Blocks in ThinLab*", Human Computer Interaction, 2 (4), 269-295.

[Borning 86b]
A. H. Borning, "*Defining Constraints Graphically*", Computer Human Interaction Conference Proceeding, ACM, New York, April 1986, 137-143.

[Borras 87]
P. Borras, D. Clément, T. Despeyroux, J. Incerpi, G. Khan, B. Lang, et V. Pascual, "*CENTAUR : The system*", Rapport de Recherche n° 777, Rocquencourt : INRIA.

[Buxton 83]
W. Buxton, "*Toward a Comprehensive User Interface Management*", Computer Graphics, 17(3), 35-42, July 1983.

[Card 83]
S. Card, T. Moran et A. Newell, "*The Psychology of Human-Computer Interaction*", New Jersey : Lawrence Erlbaum Associates.

[Cardelli 85]
L. Cardelli et Rob Pike, "*Squeak : A Langage for Communicating with Mice*", Computer Graphics : SIGGRAPH'85 Conference Proceeding. San Francisco, CA. 19(3) July 22-26 1985, 199-204.

[Cockton 88]
G. Cockton, "*Interaction Ergonomics, Control and Separation : Open Problems in User Interface Management Systems*", AMU8811/03H, Scotish HCI Centre, Heriot-Watt University, Chambers Street, Edinburg, EH1 1HX, February 1988.

[Conklin 87]
J. Conklin, "Hypertext : an Introduction and Survey", IEE Computer, 20(9), 17-41, September 1987.

[Cousin 84]
P. Hayes et R. Lerner, "*Cousin User Manual*", Internal Document, 1984, Carnegie Mellon University.

[Coutaz 84]
J. Coutaz et M. Herrmann, "*Adèle et le Médiateur-Compositeur ou Comment rendre une Application Interactive indépendante de l'Interface Usager*", Actes du deuxième colloque de Génie Logiciel, AFCET, Juin 1984, 195-212.

[Coutaz 85a]
J. Coutaz, "*A Layout Abstraction for User System Design*", ACM SIGCHI, January 1985, 18-24. Paru également dans Carnegie-Mellon University Technical Report, CMU-CS-84-167, December 1984.

[Coutaz 85b]
J. Coutaz, "*Abstractions for User Interface Design*", IEEE Computer, 18(9), 21-34, September 1985.

[Coutaz 86a]
J. Coutaz, "*Abstractions pour Interfaces Interactives*", Techniques et Sciences Informatiques, 5(4), 239-250, Juillet 1986.

[Coutaz 86b]
J. Coutaz et F. Berthier, "*The construction of user interfaces*", Office Systems : Methods and Tools, Proceeding. IFIP WG8. 4 Oct. 1986, G. Bracchi et D. Tsichristzis, (Eds.), Amsterdam : North-Holland, 59-66, 1987.

[Coutaz 86c]
J. Coutaz, "*La Construction d'Interfaces Homme-Machine*", Rapport de Recherche 635-I, IMAG Grenoble, 1986.

[Coutaz 87a]
J. Coutaz, "*The Construction of User Interfaces and the Object Paradigm*", Proceedings ECOOP '87, 135-144, June 1987.

[Coutaz 87b]
J. Coutaz, "*PAC, an Implementation Model for Dialog Design*", Proceedings. Interact '87, 431-436, Stuttgart, September 1987.

[Coutaz 88]
J. Coutaz et L. Bass "*Ergonomics and Software Principles for the Construction of Interactive Software*", RR 732-I technical report, September 1988.

[Coutaz 90]
I. Coutaz, "*Interfaces homme-ordinateur : Conception et réalisation*", DUNOD informatique, Avril 1990.

[Dance et al. 87]
J. R. Dance, T. E. Granor, R. D. Hill, S. E. Hudson, J. Meads, B.A. Myers, et A. Shulert, "*The Run-time Structure of UIMS-Supported Applications*", Computer Graphics, 1(2), April 1987.

[El Mrabet 89]
H. El Mrabet ,"*Architecture des Systèmes Interactifs*", Mémoire d'ingénieur INSA-Lyon, BULL-Massy, Juin 1989.

[Fisher 87]
G. Fisher, "*An Object Oriented Construction and Tool Kit for Human Computer Communication*", Computer Graphics, 1(2), April 1987.

[Flecchia 87]
M. A. Flecchia et R. D. Bergeron, "*Specifing Complex Dialogs in ALGE*", Proceedings SIGCHI+GI'87 : Human Factors in Computing Systems, Toronto, Canada, April 5-9 1987, 229-234.

[Foley 88]
J. Foley, W. Chul Kim, S. Kovacevic et K. Murray "*Defining Interfaces at a High Level of Abstraction*", IEEE Software, January 1989.

[Foley 84]
J. Foley et A. Van Dam "*Fundamentals of Interactive Computer Graphics*", Addison Wesley, pul., 1984.

[Foley 87]
J. Foley, "*Transformations on a Formal Specification of User-Computer Interfaces*", Computer Graphics, 21(2), 109-112, April 1987.

[Freburger 87]
K. Freburger, "*RAPID : Prototyping Control Panel Interfaces*", OOPSLA'87 Conference Proceedings October 4-8, 1987 Orlando, Florida SIGPLAN Notices, 22(12) December 1987, 416-422.

[Goldberg 83]
A. Goldberg et D. Robson, "*Smalltalk-80 : The Language and its Implementation*", Reading, MA : Addison-Wesley Publishing Company, 1983.

[Gould 88]
J. D. Gould, "*How to Design Usable Systems*", Handbook of Human-Computer Interaction, M. Helander (Ed.), Amsterdam : North Holand, 1988, 757-789.

[Green 85a]
M. Green, "*Report on Dialogue Specification Tools*", User Interfaces Management Systems, G. E. Pfaff, (Ed.), Berlin : Springer-Verlag, 1985, 9-20.

[Green 85b]
M. Green, "*The University of Alberta user interface management system*", Proceedings of SIGGRAPH'85. In Computer Graphics 19(3) July 1985, 205-213.

[Green 86]
M. Green, "*A Survey of Three Dialog Models*", ACM Transactions on Graphics 5 (3), July 1986, 244-275.

[Green 87]
M. Green, "*Directions for User Interface Management Systems research*", Computer Graphics, 21(2), 1987.

[Harel 84]
D. Harel, "*Statecharts : A Visual Approch to Complex Systems*", Dept. of Applied Mathematics, The Weizman Institute of Science, February 1984.

[Hartson et Hix 89]
H. Rex Hartson et D. Hix, "*Human-Computer Interface Development : Concepts and Systems for its Management*", ACM Computer Srveys, 21(1), March 1989.

[Hayes 83]
P. J. Hayes et P. A. Szekley, "*Graceful Interaction through the Cousin Command Interface*", International Journal of Man-Machine Studies, 19, (3), 285-305, September 1983.

[Hayes 84]
P. J. Hayes, "*Executable Interface Definitions Using Form-Based Interface Abstractions*" Internal Report CMU-CS-84-110, Carnegie Mellon University, March 1984.

[Hayes 85]
P.J. Hayes, P. A. Szekly et R. A. Lerner, "*Design Alternatives for User Interface Management Based on Experience with Cousin*". In Proceeding of the CHI'85 Conference, 169-175, April 1985.

[Helfman 87]
J. Helfman, "*Panther : A Tabular User-Interface Specification System*", Proceedings SIGCHI+GI'87 : Human Factors in Computing Systems. Toronto, Canada, April 5-9 1987, 279-284.

[Henderson 86]
D. A. Henderson, "*The Trillium User Interface Design Environment*", Proceedings SIGCHI'86 : Human Factors in Computing Systems, Boston, April 13-17 1986, 221-227.

[Hibbard 84]
P. Hibbard, "*Mint User's Manual*", Carnegie Mellon University, 1984.

[Hill 87a]
R. D. Hill, "*Event-Response Systems - A Technique for Specifing MultiThreaded Dialogues*", Proceedings SIGCHI+GI'87 : Human Factors in Computing Systems Toronto, Canada, April 5-9 1987, 241-248.

[Hill 87b]
R.D. Hill, "*Supporting Concurrency, Communication and Synchronization in Human-Computer Interaction - The Sassafras SGIU*", ACM Trans. on Graphics 5 (3), July 1986, 179-210.

[Hudson 86]
S.E. Hudson et R. King, "*A Generator of Direct Manipulation Office Systems*", ACM Trans. on Office Systems 4 (2), April 1986, 132-136.

[Hullot 86]
J. M. Hullot, "*SOS Interface, un générateur d'Interfaces Homme-Machine*", Actes des Journées Afcet Informatique Langages Orientés Objet, Bigre+Globule, Janvier 1986, 69-78.

[ILOG/INRIA]
"*Aida/Masai - Manuels de référence*", ILOG, 9 rue Royale 75008 Paris.

[ISO 86]
International Oganization for Standardization, Information Processing Systems - Computer Graphics - "*Programmer's Hierarchical Interface to Graphics (PHIGS) functional description*", ISO DP 9592, October 1986.

[Jacob 85]
R. J. K. Jacob, "*A state Transition Diagram Language for Visual Programming*", IEE Computer, 18 (8), Aug. 1985, 51-59.

[Jacob86]
R. J. K. Jacob, "*A Specification Language for Direct-Manipulation User Interfaces*", ACM Trans. on Graphics, 5 (4) 283-317, October 1986.

[Karsenty 87]
S. Karsenty "*Graffiti : un outil interactif et graphique pour la construction d'interfaces homme-machine adaptables*", Thèse de 3éme cycle, Décembre 1987, Université d'Orsay.

[Kasik 82]
D. J. Kasik, "*A User Interface Management System*", Computer Graphics : SIGGRAPH'82 Conference Proceedings, Boston, MA, 16, (3), July 26-30, 1982 99-106.

[Kieras 85]
D. Kieras et P. Polson, "*An Approch to the Formal Analysis of User Complexity*", International Journal of Man-Machine Studies 22 (4), 1985, 3-50.

[Kieras 83]
D. Kieras et P. Polson, "*A generalized Transition Network Representation for Interactive Systems*", Proceedings CHI'83 Human Factors in Computing Systems, ACM, 103-106, 1983.

[Knuth 79]
D. E. Knuth, "*TEX and Metafont*", New Direction in Typesetting, Digital Press, 1979.

[Lantz 86]
K.A. Lantz, "*On User Interface Reference Models*", ACM SIGCHI Bulletin, 18 (2), 36-44.

[Lantz 87a]
K. Lantz, "*Multi-Process Structuring of User Interface Software*", Computer Graphics 21(2), 1987.

[Lantz 87b]
K. Lantz, P.P. Tanner, C. Binding, K-T. Huang, et A. Dwelly "*Reference Models, Window Systems, and Concurrency*", Computer Graphics, 1(2), April 1987.

[Le-lisp]
"*Le_Lisp, Manuel de reference*", INRIA, J. Chailloux.

[Lowgren 88]
J. Lowgren ,"*History, State and Future of User Interface Management Systems*", SIGCHI'88, ACM /SIGCHI, July 1988.

[Mabrouk 88]
M. Mbark, "*Interface Utilisateur Auto-Adaptative*", Mémoire d'ingénieur, DEA, INSA-Lyon, BULL-Massy, 1988

[Moran 81]
T.P. Moran, "*The Command Language Grammar : a representation for the user interface of interactive computer systems*", International Journal of Man Machine Studies, 15, 3-50.

[Morcos 86]
E. Morcos-Chounet et A. Conchon, *PPML "A general formalism to specify pretty-printing*", Proceedings of IFIP Congress 1986, Dublin : North Holland.

[Myers 86a]
B. A. Myers, "*Visual Programming, Programming by example, and Program Vizualisation ; A taxonomy*", Proceedings SIGCHI'86 : Human Factors in Computing Systems, 59-66, April 86.

[Myers 86b]
B. A. Myers et W. Buxton, "*Creating Highly Interactive and Graphical User Interfaces by Demonstration*", ACM SIGGRAPH 20(4), 249-258, 1986.

[Myers 87a]
B. A. Myers, "*Gaining General Acceptance for SGIU*", Computer Graphics, vol. 1 (2), April 1987.

[Myers 88]
B. A. Myers. "*Tools for Creating User Interfaces : An Introduction and Survey*", CMU-CS-88-107, Carnegie Mellon, January 1988.

[Nanard 84]
J. M. Nanard, "*Manipulation Interactive de Documents*", Techniques et Science Informatiques, 3(6), 1984, 443-451.

[Newman 68]
W. M. Newman, "*A System for Interactive Graphical Programming*", Proceedings of the AFIPS Spring Joint Computer Conference. 1968, 47-54.

[Norman 86]
D. A. Norman et S. W. Draper, "*User Centred System Design*", Amsterdam : Lawrence Erlbaum Associates, 1986.

[Olsen 83a]
D. R. Olsen Jr. et E. P. Dempsey, "*Syntax Directed Graphical Interaction*", ACM Sigplan Notices, 18(6) 112-117, July 1983.

[Olsen 83b]
D. R. Olsen Jr. et E. P. Dempsey, "*SYNGRAPH : A Graphical User Interface Generator*", in Computer Graphics, ACM, July 1983, 43-50.

[Olsen 86]
D.R. Olsen Jr,. "*Mike : The Menu Interaction Control Environment*", ACM Trans. on Graphics 5(4), October 1986, 318-344.

[Olsen 89]
D. R. Olsen, "*A Programming Language Basis for User Interface Management*", CHI'89 Conference Proceedings, special issue of the SIGCHI Bulletin, ACM Press, 1989, 171-176.

[OSF 89]
OSF/Mtif, "*Programmers's Reference Manual*", Revision 1.0, Open Software Foundation, Eleven Cambridge Center, Cambridge, MA02142, 1989.

[Payne 84]
S.J. Payne et T.R.G. Green, "*Task-Action Grammars*", Proceedings of Interact'84, London, U.K., 139-144.

[Pfaff 85]
G. R. Pfaff, *User Interface Management Systems*. Berlin : Springer-Verlag, 1985, 224 pages.

[Quint 85]
V. Quint et I. Vatton, "*Grif : un Editeur Interactif Structuré*", Rapport de Recherche TIGRE 27, IMAG-Laboratoire de Génie Informatique, BP 53 X, 38042 Grenoble Cedex, 1985.

[Quint 87]
V. Quint, "*Une Approche de l'Edition Structurée des Documents*", Thèse de doctorat d'Etat, Université Scientifique Technologique et Médicale de Grenoble, 1987.

[Reiss 87]
S. P. Reiss, "*A Conceptual Programming Environment*", Proceedings ICSE, 225-235, April 1987.

[Rose 86]
C. Rose et al., "*Inside Macintosh*", Addison Wesley, 1986.

[Scapin 87]
D. L. Scapin, "*Guide ergonomique de conception des interfaces homme-machine*", Rapport de Recherche n° 77, Rocquencourt : INRIA.

[Scapin 88a]
D. L. Scapin, "*Vers des Outils Formels de Description des Tâches orientés Conception d'Interfaces*", Rapport de Recherche n° 893, Rocquencourt : INRIA.

[Scapin 88b]
D. L. Scapin, P. Reynard, et A. Pollier"*La Conception ergonomique d'Interfaces : Problèmes de Méthodes*", Rapport de Recherche n° 957, Rocquencourt : INRIA.

[Scapin 89]
D. L. Scapin, C. Pierret-Golbreich, et I. Delouis, "*Un Outil d'Acquisition et de Représentation des Tâches Orienté-Objet*", Rapport de Recherche n° 1063, Rocquencourt : INRIA.

[Schmucker 86]
K. J. Schmucker, "*MacApp : An Application Framework*", Byte, 189-192, August 1986.

[Schulert 85]
A.J. Schulert, G. T. Rogers et J. A. Hamilton, "*ADM : A Dialog Manager*", In CHI'85 Proceedings, The Association for Computer Machinery Publ., April 1985.

[Shneiderman 83]
B. Shneiderman, "*Direct Manipulation : A Step Beyond Programming Languages*", IEEE Computer, 16(8) Aug. 1983, 57-69.

[Shneiderman 87]
Ben Shneiderman, "*Designing the User Interface : Strategies for Effective Human-Computer Interaction*, Addison Wesley Publishing Company, 1987.

[Shuey 86]
D. Shuey, D. Bailey, et T.P. Morrissey, "*PHIGS, A Standard, Dynamic, Interactive Graphics Interface*", IEEE Computer Graphics and Application, August 1986, 50-57.

[Shuey 87]
D. Shuey, "*PHIGS, a graphics platform for CAD application development*", Computer-Aided Design, 19(8), 1987, 410-417.

[Sibert 86]
J.L. Sibert, W. D. Hurley, et T. W. Bleser, "*An Object-Oriented User Interface Management System*", Computer Graphics : SIGRAPH'86 Conference Proceedings, 20(4), Aug. 18-22, 1986, Dallas Texas, 259-268.

[Smart 87]
J. Smart, "*MMIMS : The Applications Interface*", MMIMS/AI/FS, March 1987, CAP Group : UK.

[Spice 84]
S. E. Fahlman et S. P. Harbison, "*The Spice Project*". In Interactive Programming Environments, D. R. Barstow, H. E. Shrobe, et E. Sandewall, (Eds.), 546-557, 1984.

[Stefik 86]
M. Stefik, D.G. Bobrow, et K. M. Kahn, "*Integrating Access-Oriented Programming into a Multi-Paradigm Environment*", IEEE Software, 3(1), Jan. 1986, 10-18.

[Tanner 83]
P. P. Tanner et W. Buxton, "*Some issues in Future User Interface Management System (SGIU) Development*", Proceedings IFIP WG 5.2 Workshop on SGIU, seeheim, November 1983.

[Tanner 87]
P. P. Tanner, "*Multi-Thread Input*", Computer Graphics, 1(2), April 1987.

[TeleLogic]
TeleLogic, Swedish Telecom "*TeleUse Slides*", Linkoping.

[Wasserman 85]
A. I. Wasserman et D. Shewmake, "*The Role of Prototypes in the User Software Engineering (USE) Methodology*". In Hartston H (Ed.), Advances in Human-Computer Interaction, New Jersey : Ablex Publishing Corp, 1985.

[Wasserman 82]
A. I. Wasserman et D.T. Shewmake, "*Rapid Prototyping of Interactive Information Systems*", ACM Software Engigneering Notes, 7(5), 171-180.

[Williams 87]
G. Williams, "*HyperCard*", Byte, 109-117, December 1987.

[Young 89]
D. A. Young, "*X Window Systems Programming and Applications with Xt*", Prentice-Hill, 1989.

Annexe 1

COUSIN

COUSIN (COoperative USer INterface) [Hayes 83, Hayes 84, Cousin 84] est une application qui permet de construire des interfaces de type "formulaires" pour des applications développées dans le projet Spice [Spice 84].

Les concepts-clés de COUSIN sont :

- une interface unique pour toutes les applications : l'interface est décrite de manière déclarative et dirigée par les données de l'application.
- des formulaires comme modèles de communication avec l'utilisateur (remplissage des champs et sélection).
- un système "intelligent" de support pour le remplissage des formulaires : les champs sont typés et ont des valeurs par défaut. Ce système permet aussi le contrôle, la vérification syntaxique et la validité des arguments saisis.

L'application et COUSIN sont supposés être deux processus séparés communiquant par un protocole. COUSIN joue le rôle de médiateur entre l'application et l'utilisateur.

Annexes

Exemples de SGIUs

Nous présentons dans ce paragraphe quelques SGIUs récents utilisant diverses approches et méthodes qui nous paraissent intéressantes.

Annexe 2

GRAFFITI

a). Introduction

Graffiti [Karsenty 87] est un SGIU basé sur l'approche objets. C'est un outil interactif et graphique pour la construction d'interfaces homme-machine. Il permet de décrire d'une manière graphique directe des interfaces adaptables dynamiquement par l'utilisateur à l'utilisateur. L'interface construite peut être éditée, modifiée et personnalisée pour l'utilisateur final (renommer les commandes d'un menu, réorganisation des fonctions de clavier, repositionnement des objets). Graffiti est constitué d'un noyau d'interface exécutable et d'une bibliothèque de fonctions qui gèrent et manipulent des objets interactifs graphiques. Ces objets peuvent être associés à des données de l'application. Graffiti assure une cohérence et une représentation correcte de ces objets vis-à-vis des manipulations effectuées par l'utilisateur et les modifications éventuelles de l'application sur les données associées. La définition du comportement sémantique des objets en fonction des événements reçus de l'utilisateur est assurée par un mécanisme d'associations de ces objets à des points d'entrée de l'application. Graffiti assure la gestion du dialogue entre l'utilisateur, l'interface et l'application. Il est bâti sur un système par objets UFO (User Friendly Objects) [Beaudoin 85] et sur un certain nombre de classes d'objets prédéfinis (cf. figure a).

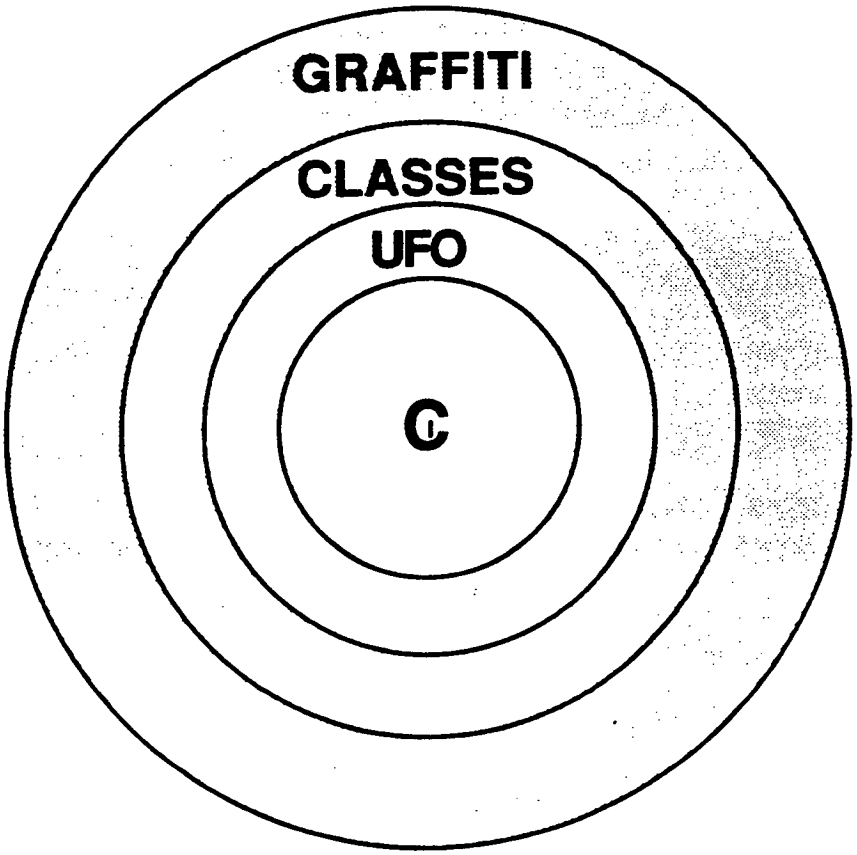

Figure a : Graffiti à partir de C , de UFO (User Friendly Objects), et de classes d'objets

b). Architecture

Graffiti utilise l'approche objet pour la réalisation des composantes de base de l'interface. Ces objets sont réalisés par UFO (cf. paragraphe précédent). La gestion du dialogue dans Graffiti est basée sur le modèle d'événements. Les événements provoquent l'envoi des messages aux objets désignés.

Le modèle d'implémentation de Graffiti est réalisé suivant le modèle PAC (Présentation, Abstraction, Contrôle) [Coutaz 87].

L'application interactive construite à partir de Graffiti fonctionne par un mécanisme basé sur la définition des points d'entrée de l'application et les valeurs actives : variables simples ou structurées, partagées entre l'application et l'interface. Les opérations de lecture et d'écriture sur ces variables sont contrôlées et peuvent déclencher l'envoi des messages. La figure suivante met en évidence l'application interactive produite par Graffiti.

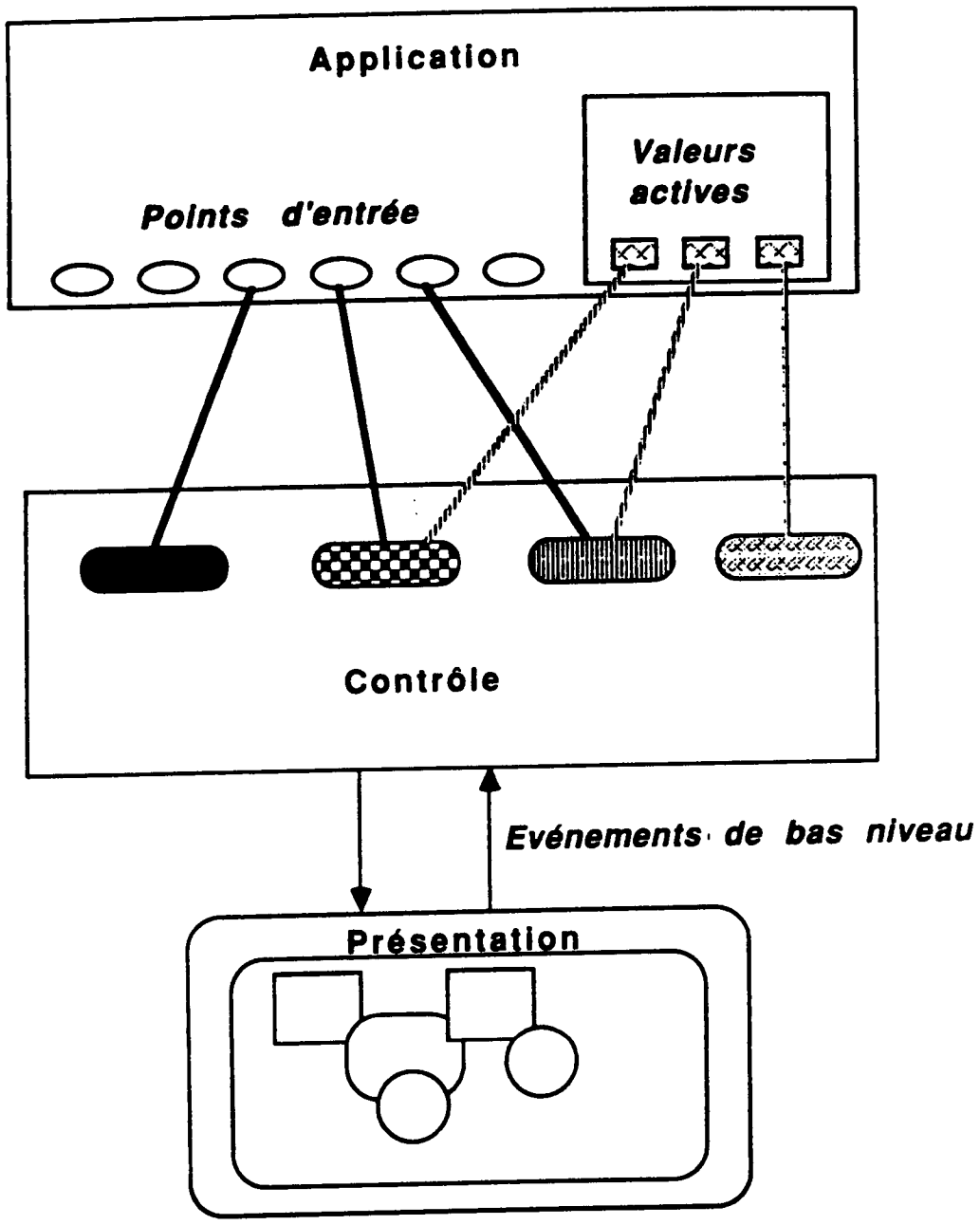

Figure b : Mise en évidence d'une application interactive

Tableau 1 : Objets interactifs de base de Graffiti

c). Objets interactifs de base de Graffiti

Graffiti propose un ensemble d'objets interactifs de base. A partir de ces objets, on peut composer des objets plus complexes (boîtes de dialogue, formulaires, ...). Le tableau 1 décrit ces objets.

Objet	Attributs
Menu	titre (avec alignement et police de caractères)
	sens horizontal ou vertical
	type (permanent, déroulant, fugitif)
	nombre maximum de commandes visibles
	alignement des commandes
	espacement entre les commandes
	mode de mise en évidence des commandes
Commande	nom
	clé
	police de caractères
	icône
	attributs d'inhibition et de visibilité
Entier : barre de défilement	valeur inférieure
	valeur supérieure
	valeur courante
	taille de l'ascenseur
Icône et bouton	rectangle délimiteur
	nom
	pictogramme
	fonction d'activation
Objet texte	chaîne de caractères
	police de caractères
	zone délimitant la chaîne
	fonction d'activation
bouton de radio : énuméré	chaîne de caractères
	bouton de sélection
	alignement horizontal ou vertical
boîte à options : booléen	chaîne de caractères
	rectangle à cocher
	alignement horizontal ou vertical
Tableau	...
Fenêtre	titre
	barres de défilement
	sous-fenêtres

d). Composants et étapes de construction de l'interface

La figure suivante décrit les composants et les étapes de construction de l'interfaces utilisateurs par Graffiti. Au début le concepteur de l'interface construit son interface d'une façon interactive avec l'éditeur de Graffiti, il peut simuler l'interaction sans faire appel à l'application pour tester et valider cet interface. Une fois l'interface mise au point, il peut la sauvegarder dans une base de données. La production de l'application interactive finale se fait par édition de lien entre les points d'entrée de l'application et la bibliothèque de Graffiti qui charge la structure de l'interface sauvegardée et initialise les points d'entrée et les valeurs actives. La réutilisation de l'interface sauvegardée pour des applications différentes est possible grace à la redéfinition des objets de l'interface, des valeurs actives et les points d'entrée de l'application.

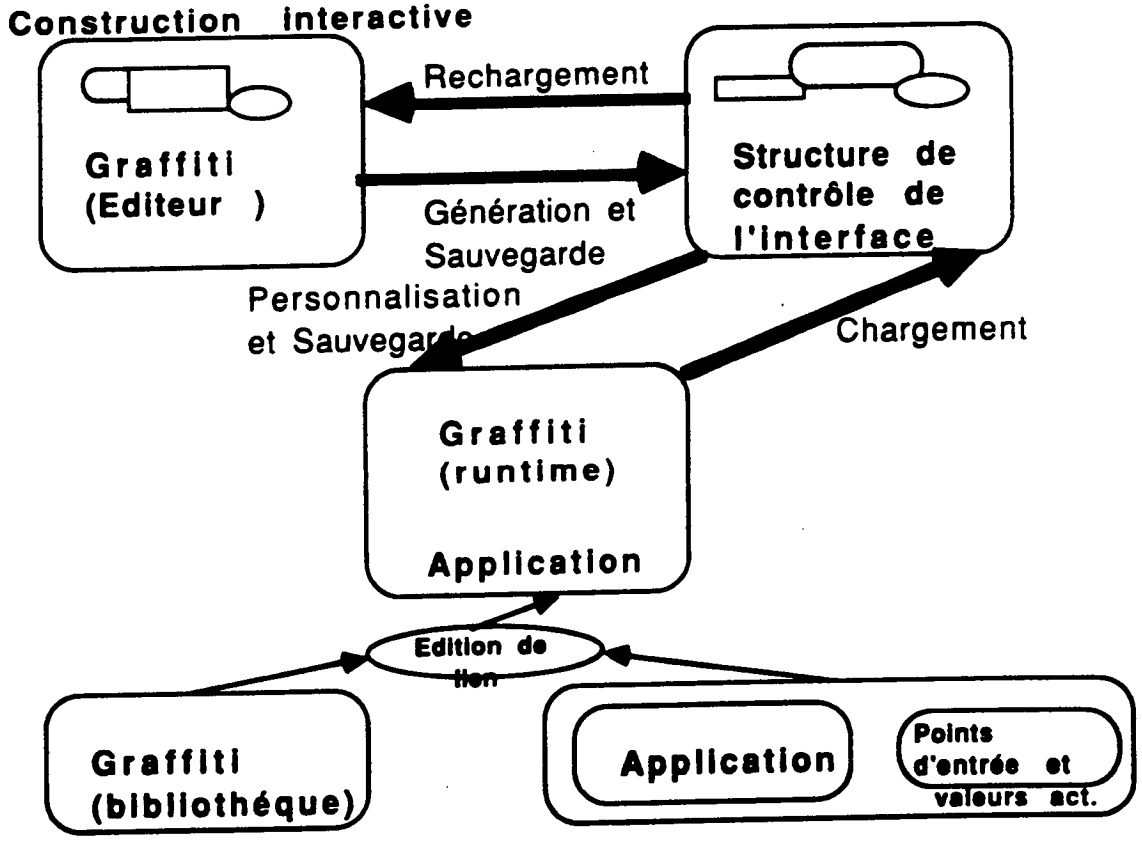

Figure c : Spécification interactive et génération automatique d'une interface utilisateur

Annexe 3

Aïda/Masaï

Aida [ILOG/INRIA] est un environnement de développement d'applications graphiques interactives, développé au dessus d'une couche objets en Le_Lisp [Le_Lisp]. Aida utilise un ensemble d'objets graphiques de base appelés images qui peuvent être assemblés pour former des objets plus complexes. Les objets sont de type barre de défilement, bouton, ligne de texte éditable. A chaque objet est attachée une fonction qui est appelée lorsque l'utilisateur désigne cet objet. Des contraintes graphiques d'alignement, appelées constructeurs, peuvent être définies pour un ensemble d'objets. Aida est un environnement complet de primitives disponibles au programmeur et qui permettent la construction des interfaces graphiques avec simplicité et progressivement. Les objets graphiques de l'interfaces utilisateurs réagissent aux événements utilisateur et l'aspect extensible de l'ensemble des objets permet au concepteur de l'interface de se construire ses propres objets réutilisables pour d'autres applications interactives. Aida est particulièrement intéressant grâce aux possibilités suivantes :

- un nombre important de composants prédéfinis pour construire des tableaux de bord ;
- la possibilité de construire des bibliothèques de composants ;
- la facilité d'extension grâce à l'utilisation de la programmation orienté-objet ;
- les outils interactifs de développement, de mise au point et d'extension ;
- la documentation en ligne pouvant être consultée et mise à jour à tout moment.

Masai [ILOG/INRIA] est un outil de développement interactif fondé sur la méthodologie de programmation des interfaces graphiques qui accompagne Aida. Au lieu de programmer l'image de l'interface graphique en Aida, l'utilisateur la dessine directement à la souris (manipulation graphique directe). L'interfaces utilisateurs construite par Masai peut être à tout moment testée en cours d'édition. Masai engendre ensuite le code Aida qui construit l'image de l'interface graphique. Plusieurs centaines de lignes de code Aida commentées peuvent ainsi être écrites en quelques heures grâce à Masai. L'utilisation de Masai permet de séparer la construction de la partie graphique de l'interfaces utilisateurs du programme qui est appelé par les différentes actions sur les objets interactifs de l'interfaces utilisateurs. Cependant, il faut écrire les programmes qui sont déclenchés par les composants de l'interface graphique. Les conditions de déclenchement de ces programmes peuvent être la pression d'un bouton-poussoir, le choix dans un menu, ... Ces programmes récupèrent les valeurs saisies dans l'interface et les transmettent au

programme interfacé qui peut être écrit en Lisp, C ou Fortran. Masai possède des éditeurs de texte et des éditeurs spécialisés pour la construction de l'interfaces utilisateurs. Masai permet aussi une grande facilité de construction d'interface par des non informaticiens (des ergonomes par exemple) grâce à des outils de dessin tel Aidapaint qui est intégré dans l'environnement Masai. Il est possible de personnaliser Masai en précisant dans un fichier de paramétrisation les options de la génération du code, la configuration initiale de l'interfaces utilisateurs de Masai. Chaque utilisateur peut construire très facilement un Masai adapté à ses besoins spécifiques (gestion, scientifique, interface statique ou animation). La conception de Masai tire profit du modèle MVC de Smalltalk, de la génération automatique de code source Aida à l'aide des techniques orientées objet et de la facilité de compréhension et d'extension par l'utilisateur, puisqu'une grande partie de Masai a été réalisée avec Masai. Aida et Masai sont utilisés pour le maquettage, le prototypage et pour des réalisations industrielles dans des domaines variés. Les figures suivantes (figure d et figure e) montrent un exemple de réalisation d'interfaces utilisateurs avec Masai.

programme interfacé qui peut être écrit en Lisp, C ou Fortran. Masai possède des éditeurs de texte et des éditeurs spécialisés pour la construction de l'interfaces utilisateurs. Masai permet aussi une grande facilité de construction d'interface par des non informaticiens (des ergonomes par exemple) grâce à des outils de dessin tel Aidapaint qui est intégré dans l'environnement Masai. Il est possible de personnaliser Masai en précisant dans un fichier de paramétrisation les options de la génération du code, la configuration initiale de l'interfaces utilisateurs de Masai. Chaque utilisateur peut construire très facilement un Masai adapté à ses besoins spécifiques (gestion, scientifique, interface statique ou animation). La conception de Masai tire profit du modèle MVC de Smalltalk, de la génération automatique de code source Aida à l'aide des techniques orientées objet et de la facilité de compréhension et d'extension par l'utilisateur, puisqu'une grande partie de Masai a été réalisée avec Masai. Aida et Masai sont utilisés pour le maquettage, le prototypage et pour des réalisations industrielles dans des domaines variés. Les figures suivantes (figure d et figure e) montrent un exemple de réalisation d'interfaces utilisateurs avec Masai.

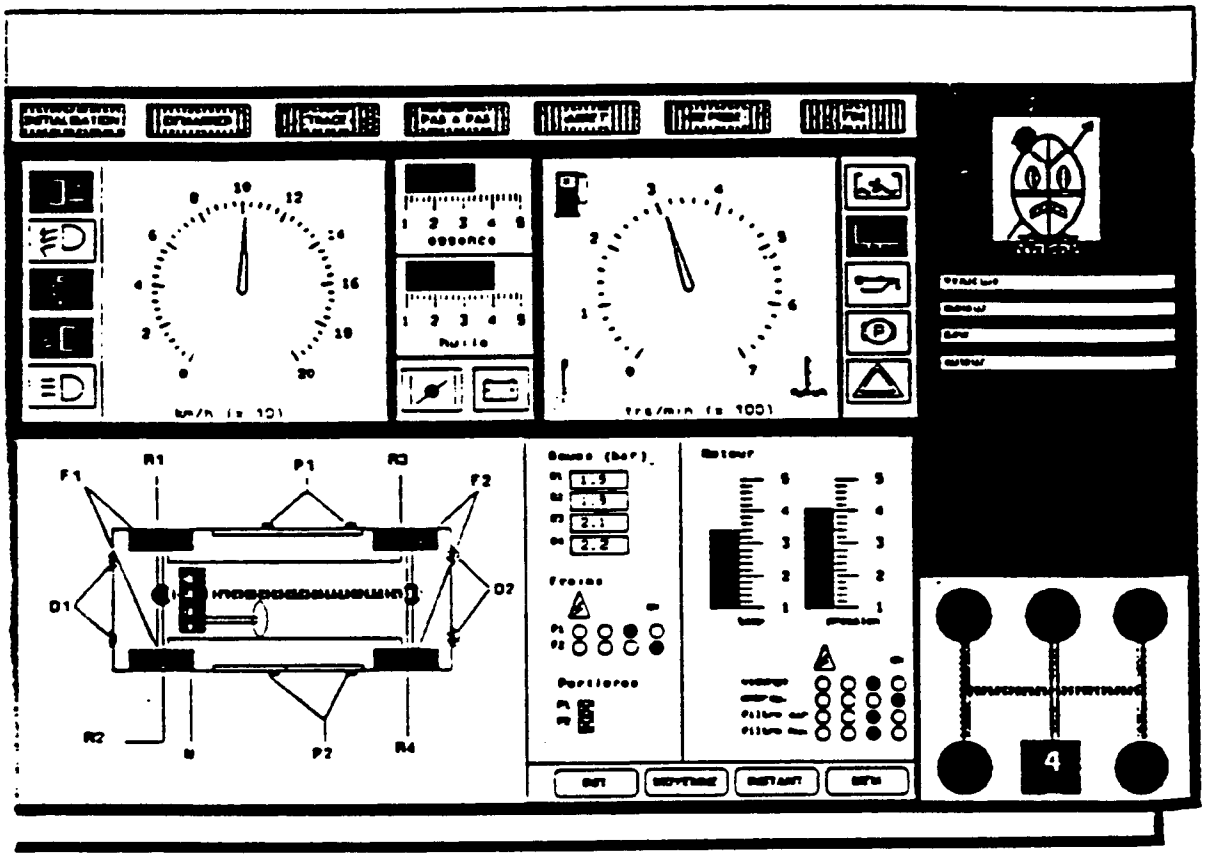

Figure d : Simulation d'un tableau de bord d'une voiture par Aïda/Masaï

Annexe 4

TeleUse

TeleUse [TeleLogic] est un SGIU développé par TeleLogic à LinKoping (Suède Telecom). Ce système se compose d'un certain nombre de composants logiciels (cf. figure f) :

- VIP : un éditeur graphique d'interfaces utilisateur. L'utilisateur peut d'une façon interactive créer, déplacer et modifier les objets interactifs de présentation (boutons, menus, ...),
- un langage D : c'est un langage à événements basé sur les règles qui permettent d'implémenter le dialogue entre l'utilisateur et l'application. Ce langage supporte les dialogues parallèles. Un compilateur spécifique permet de générer du C à partir des spécifications en D. Ce langage est muni aussi d'un interpréteur et un deboggueur,
- AIM : une interface entre le dialogue et l'application,
- une librairie runtime et un constructeur d'interfaces utilisateurs (UIbuilder).

Ce système présente certaines caractéristiques telles que :

- le contrôle mixte ;
- la généralité (limitée) ;
- le prototypage rapide ;
- la construction interactive de l'interface ;
- la portabilité ;
- la réduction du coût de maintenance ;
- la réduction du temps de développement ;
- il est basé sur le modèle de références ;
- il suit le standard Xwindow ;
- l'extensibilité des objets interactifs (widgets) ;
- la structure de l'interfaces utilisateurs est orientée objet ;
- la gestion performante du dialogue grâce au langage D et son compilateur ;
- les programmes écrits en Aida peuvent utiliser le X-window.

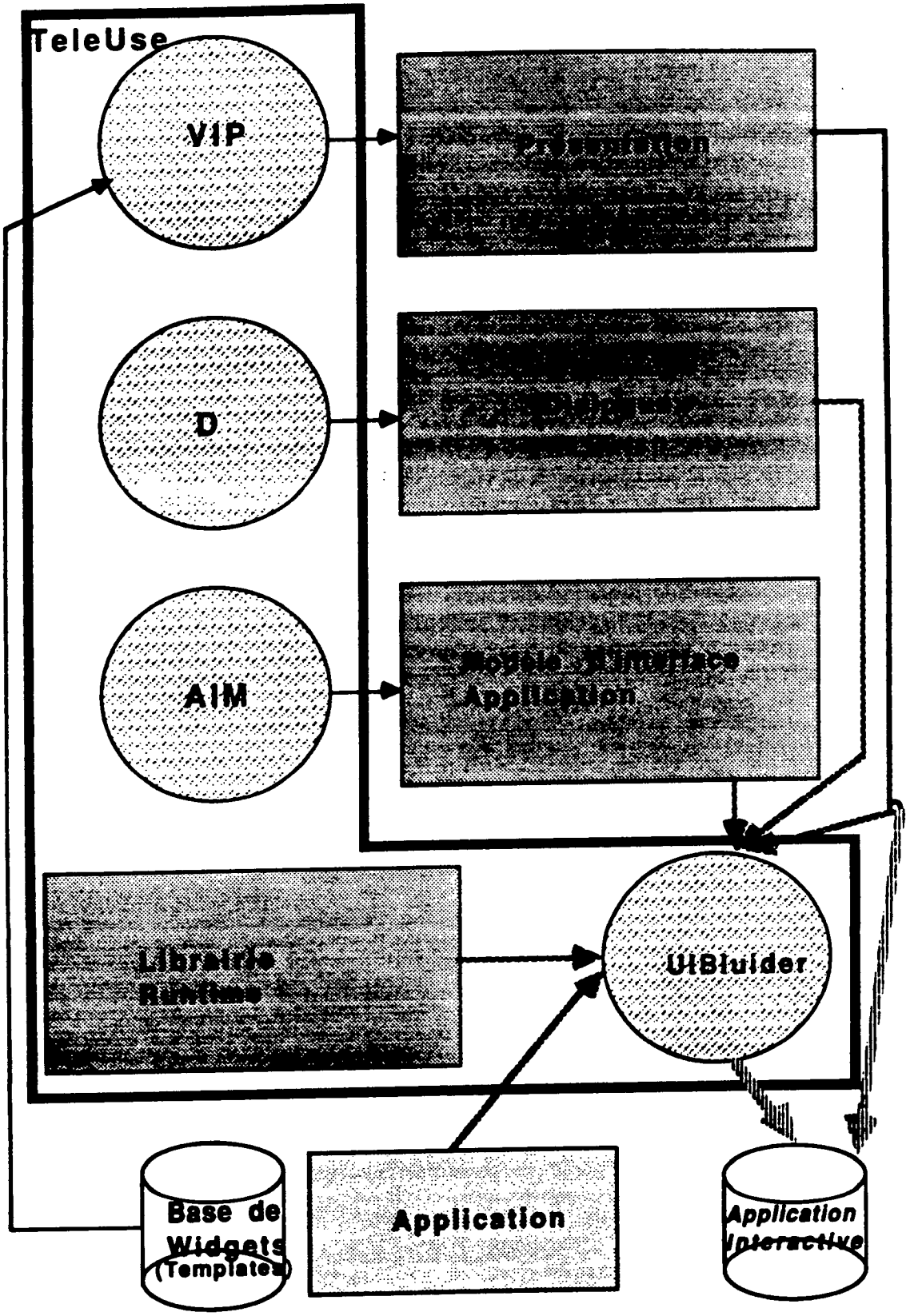

Figure f : Architecture de TeleUse

Annexe 5

MMIMS

MMIMS [Smart 87] est un SGIU développé par CAP Group UK dont le langage de spécification de l'interfaces utilisateurs, FDL (Format Description Langage) repose sur le concept d'objet et classes d'objets. Le constructeur de l'interfaces utilisateurs spécifie son interface en la décrivant par une suite d'instructions FDL. Ces spécifications sont ensuite transformées en des objets interactifs de présentation qui constituent l'interfaces utilisateurs de l'application interactive. Une spécification interactive directe est possible avec MMIMS en le couplant avec Metradoc II (un environnement interactif d'éditeurs graphiques développé par SEMA Group France). Metradoc II génère à partir des spécifications graphiques, un fichier FDL qui sera en entrée de MMIMS. Le tableau 2 résume les objets interactifs prédéfinis de MMIMS et manipulés par FDL.

GWUIMS

GWUIMS (George Washington University UIMS) [Sibert 86] repose sur l'approche orientée objet. Plusieurs types d'objets se partagent les fonctionnalités de l'application interactive : deux types d'objets différents pour les entrées et les sorties, un type d'objet pour la représentation graphique, un type d'objet application, et un type d'objet qui fait le lien entre l'objet application et l'objet présentation. Les objets présentation assurent le passage du niveau lexical au niveau syntaxique et inversement, alors que les objets intermédiaires assurent le passage entre le niveau sémantique et le niveau syntaxique et inversement (cf. figure g). Avec une telle structure l'indépendance entre les différents niveaux d'abstraction de l'application interactive est assurée. La communication entre les trois composantes (présentation, dialogue/contrôle, sémantique) est faite par le mécanisme d'envoi de messages entre objets. L'avantage principal de ce système est de prouver l'adéquation et la puissance de l'approche objet pour la conception des SGIUs souples, flexibles et extensibles.

Tableau 2

Objets interactifs prédéfinis de MMIMS

Type-objet / *Attributs*	SCREEN	WINDOW	F. FRAME	GF RR AA PM HE IC	MF ER SA SM AE GE	TF TR YA ME	MESSAGE	PROMPT	PANE	CHOICE	COMMAND	INPUT	OUTPUT
Item_class	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui
Item_Id	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui
Label		1						Oui		Oui	Oui	Oui	Oui
Image		2	3	4	3	3		Oui		Oui	Oui	Oui	Oui
Value		5						Oui		Oui		Oui	Oui
Pick_list												Oui	
Value_of_choice										Oui			
Choice_availab										Oui			
Visible	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui
LabeL_visib		6								Oui	Oui	Oui	Oui
Value_visib		7								Oui		Oui	
Selectable									Oui	Oui	Oui	Oui	
Sizing_font	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui
Live_font	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui
Dead_font	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui
Value_font	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui
Device		Oui	Oui	Oui	Oui	Oui							
Applic_data	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui
Update	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	
No_of_cont	Oui	Oui	Oui						Oui	Oui			
Parent		Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui
First_child	Oui	Oui	Oui						Oui				
Child_by_Id	Oui	Oui	Oui						Oui				
Next_item		Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui
Previous_item		Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui

Légende

1 : Information textuelle dans l'icône

2 : Icône

3 : Curseur

5 : nom des rayures de la fenêtre (window)

6 : AI_ON = label visible sur l'icône, AI_OFF = label non visible sur l'icône

7 : AI_ON = ouvert (open), AI_OFF = icônisé.

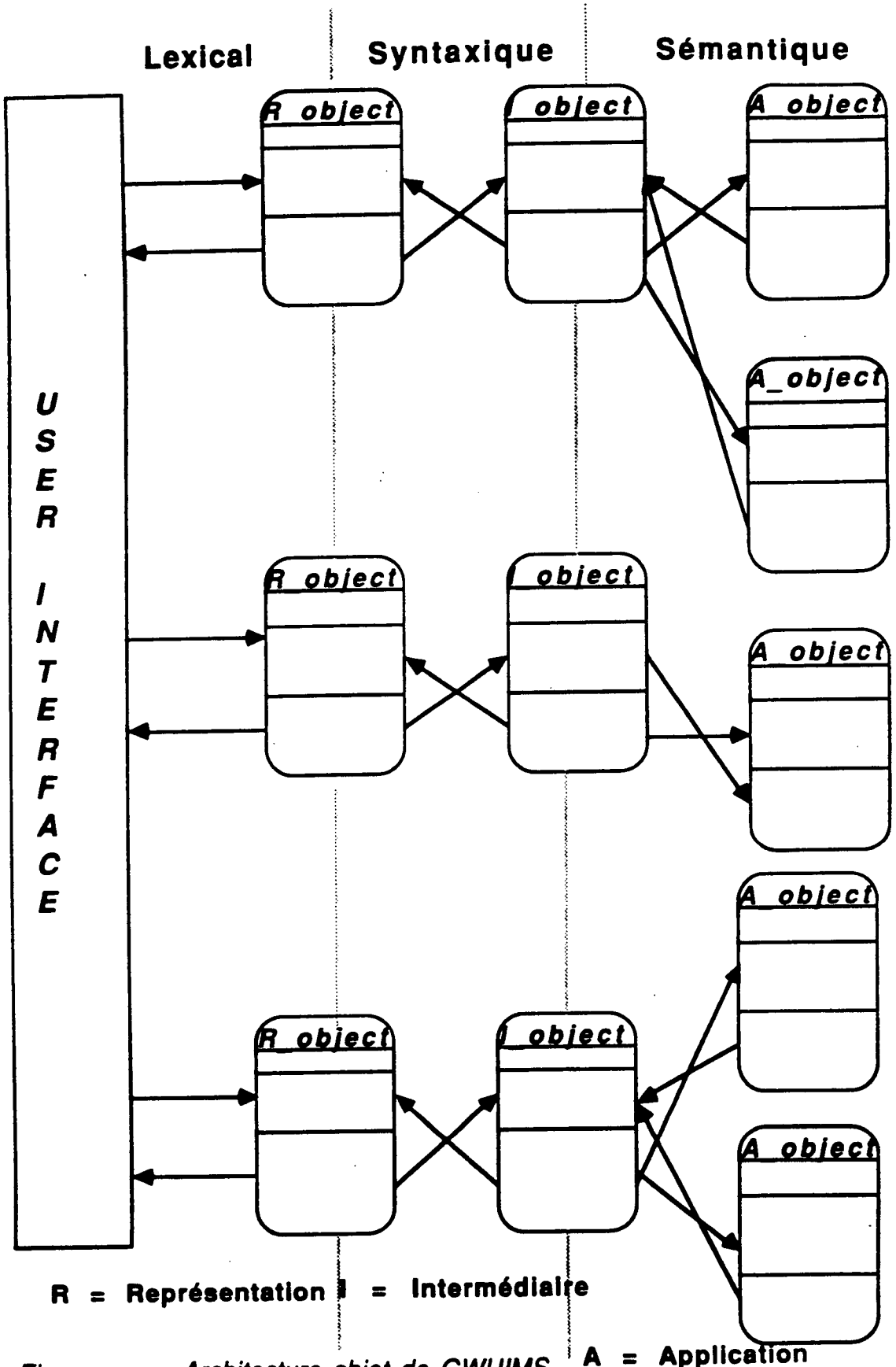

Figure g : Architecture objet de GWUIMS

Tous les objets de GWUIMS sont définis par des classes et chaque objet est une instance d'une classe d'objet. Toute instance d'une classe possède les mêmes méthodes que les autres instances de la classe; elle diffère des autres seulement par les valeurs de ses attributs et ses relations avec les autres objets ou instances. Par exemple, une jauge température est une instance de la classe objet graphique dont la définition est ainsi :

Class : graphic_object;
Attributs : color, font, linestyle, position, visibility, hot_spot_extent, pickability,
vector_list, text_list, pix_rect_list;
Relationships : graphic_object_list;
Methods : display, make_invisible, change_position;
Message : none;

Annexe 6

Peridot

Peridot (Programming by Example for Reel-time Interface Design Obviating Typing) [Myers 86b] est un SGIU développé à l'université de Toronto (Canada). Ce système utilise une approche basée sur la programmation par l'exemple [Myers 86a]. L'utilisateur peut construire son interface d'une façon graphique interactive (Manipulation graphique directe). La particularité de ce système est qu'il permet au concepteur de l'interface de montrer ce qu'il veut construire et comment l'interface doit se dérouler avec l'utilisateur final.

Le principe général de génération d'interfaces par Pridot est de permettre au constructeur d'interfaces de construire différents objets graphiques visuellement et d'agir sur ces objets de la même manière qu'un utilisateur final le ferait en manipulant l'application interactive finale (déplacement des objets, clics souris, ...). Le système essaie alors d'inférer sur les relations du concepteur et les différents objets manipulés (programmation par l'exemple). Par un système de dialogue avec le concepteur d'interface, Peridot essaie de valider l'interface et générer le code correspondant au dialogue avec l'utilisateur. Le concepteur a toujours la possibilité de spécifier explicitement les relations entre les actions et les objets concernés.

Pour mettre en évidence les fonctionnalités de Peridot, la figure h montre un exemple simple qui consiste à construire un menu. Les différentes sous-figures (a, b, c, d) correspondent à des étapes de construction du Menu (spécification des paramètres, modification, inférence).

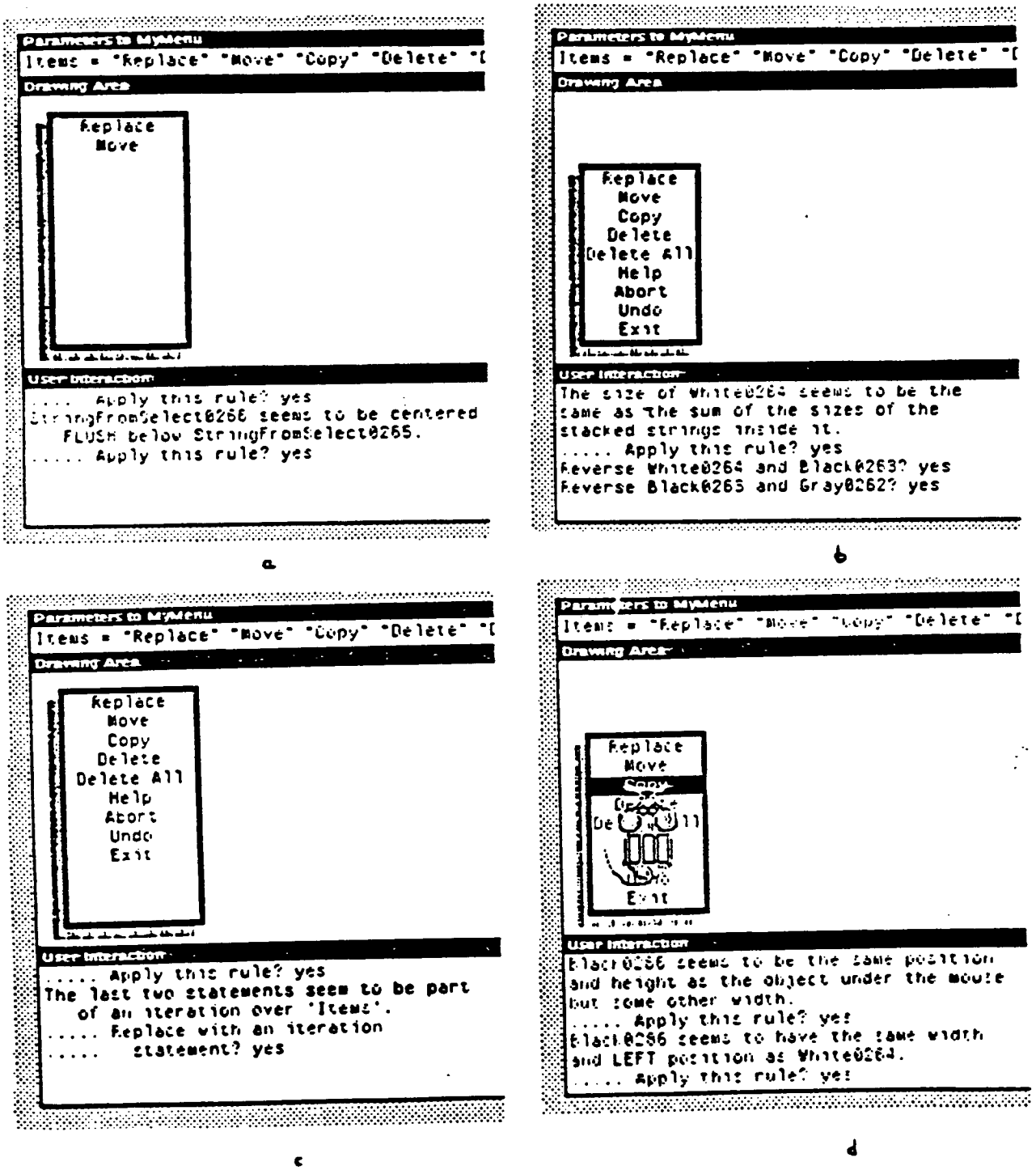

Figure h : Exemple de construction d'un menu avec Peridot

Annexe 7

SOS Interface

SOS Interface [Hullot 86] est développée en Le-lisp [Le-lisp] sur Macintosh et utilise la boîte à outils par objets, Alcyone [Alcyone]. SOS Interface est un éditeur pour la construction d'interfaces utilisateur. L'application à interfacer est vue comme un ensemble de points d'entrée pour l'interface, l'application et l'interface étant deux modules distincts. SOS Interface permet de manipuler des objets graphiques appelés images. L'utilisateur peut créer ces objets, les éditer et les rendre activables. Ceci est rendu possible par un mécanisme d'association des images et les fonctions points d'entrée de l'application. L'activation de tout objet graphique déclenche automatiquement l'appel d'une fonction de l'application. SOS Interface permet d'éditer quatre types d'entités :

- les liaisons clavier-fonctions ;
- les menus ;
- le code d'application ;
- les tableaux de bord.

Les figure i montre la construction interactive de l'interfaces utilisateurs d'une application par SOS Interface. Chaque objet graphique interactif est associé à une fonction de l'application. Le déclenchement des fonctions est causé par la réception des événements utilisateur (clic souris, clavier, ...).

Annexe 7

SOS Interface

SOS Interface [Hullot 86] est développée en Le-lisp [Le-lisp] sur Macintosh et utilise la boîte à outils par objets, Alcyone [Alcyone]. SOS Interface est un éditeur pour la construction d'interfaces utilisateur. L'application à interfacer est vue comme un ensemble de points d'entrée pour l'interface, l'application et l'interface étant deux modules distincts. SOS Interface permet de manipuler des objets graphiques appelés images. L'utilisateur peut créer ces objets, les éditer et les rendre activables. Ceci est rendu possible par un mécanisme d'association des images et les fonctions points d'entrée de l'application. L'activation de tout objet graphique déclenche automatiquement l'appel d'une fonction de l'application. SOS Interface permet d'éditer quatre types d'entités :

- les liaisons clavier-fonctions ;
- les menus ;
- le code d'application ;
- les tableaux de bord.

Les figure i montre la construction interactive de l'interfaces utilisateurs d'une application par SOS Interface. Chaque objet graphique interactif est associé à une fonction de l'application. Le déclenchement des fonctions est causé par la réception des événements utilisateur (clic souris, clavier, ...).

source par un langage de haut niveau d'abstractions (par une spécification interactive). Cet environnement intégré (spécification de la présentation, du dialogue) permettra d'étendre le domaine des applications à interfaces et d'assurer une meilleure productivité d'interfaces utilisateurs.

source par un langage de haut niveau d'abstractions (par une spécification interactive). Cet environnement intégré (spécification de la présentation, du dialogue) permettra d'étendre le domaine des applications à interfaces et d'assurer une meilleure productivité d'interfaces utilisateurs.