Le modèle SIROCO : de la spécification conceptuelle des interfaces utilisateur à leur réalisation

Véronique Normand

THESE

Présentée par

Véronique NORMAND

pour obtenir le titre de docteur
de l'Université Joseph Fourier - Grenoble I
(Arrêté ministériel du 23 novembre 1988)

spécialité INFORMATIQUE

Le modèle SIROCO : de la spécification conceptuelle des interfaces utilisateur à leur réalisation

---

Thèse soutenue le 17 avril 1992 devant la commission d'examen.

Sacha Krakowiak	Président
Joëlle Coutaz	
Jocelyne Nanard	
Michel Beaudoin-Lafon	Examinateurs
Dominique Scapin	
Roland Balter	

Thèse préparée au sein de l'Unité Mixte de Recherche Bull-IMAG

Remerciements

Je tiens à remercier l'ensemble des personnes qui, par leurs conseils, leurs encouragements ou leur aide, ont contribué à l'aboutissement de ce travail :

Joëlle Coutaz, Professeur à l'Université Joseph Fourier de Grenoble, qui m'a fait découvrir un domaine de recherche passionnant et m'a encadrée tout au long de ce travail. Je la remercie de la confiance qu'elle m'a accordée, de son écoute attentive et, plus simplement, de son amitié.

Sacha Krakowiak, Professeur à l'Université Joseph Fourier, Directeur adjoint de l'Unité Mixte Bull-IMAG, qui m'a accueillie au sein du projet Guide et m'a donné la liberté de mener cette recherche, et qui a accepté de présider le jury,

Jocelyne Nanard, Maître de Conférence à l'Université des Sciences et Techniques du Languedoc, qui a accepté d'être rapporteur de cette thèse et, par sa lecture scrupuleuse et ses remarques pertinentes, m'a permis de l'améliorer,

Michel Beaudoin-Lafon, Maître de Conférence à l'Université de Paris XI, qui a bien voulu être rapporteur de ce document, et le faire bénéficier de ses critiques constructives,

Dominique Scapin, Directeur de Recherche à l'INRIA, qui a manifesté son intérêt pour ce travail et a accepté de faire partie du jury,

Roland Balter, Directeur de l'Unité Mixte Bull-IMAG, qui m'a manifesté un soutien amical tout au long de ces années de recherches,

Alexander Rudnicky, "senior scientist" à Carnegie Mellon University aux Etats-Unis, qui m'a accueillie au sein de son équipe pendant quelques mois, et m'a permis d'élargir ma vision du domaine des IHMs,

Malcolm Bennett, qui a mis en place l'environnement graphique dans le projet Guide, sans lequel je n'aurais pu travailler,

L'ensemble des personnes de l'Unité Mixte Bull-IMAG, qui, par leur gentillesse et leur dynamisme, m'ont fait bénéficier d'un cadre de travail incomparable,

L'ensemble des membres de l'équipe LGI-IHM, qui m'ont manifesté un soutien amical.

Une pensée particulière pour Jean-Michel, ses patientes relectures de ce document, ses critiques pertinentes, ses encouragements constants. Son soutien quotidien m'a été précieux tout au long de ces travaux.

Plan de la thèse

Introduction générale

Dans un monde que l'avancée des techniques de transmission et de traitement de l'information transforme un peu plus chaque jour, ce sont non seulement nos habitudes de travail et de vie quotidienne qui sont affectées, mais aussi, plus profondément, nos manières de percevoir, de raisonner et de communiquer.

> ... la plupart des logiciels contemporains jouent un rôle de *technologie intellectuelle* : ils réorganisent peu ou prou la vision du monde de leurs utilisateurs et modifient leurs réflexes mentaux.
>
> Pierre Levy, Les technologies de l'intelligence,
> Editions La Découverte, 1990.

L'évolution des interfaces homme/ordinateur est partie prenante de ce phénomène. Une interface homme/ordinateur, ou, par abus de langage, *interface homme/machine* (IHM), désigne l'ensemble de logiciels et d'appareils matériels permettant la communication entre un système informatique et ses utilisateurs humains. Cette vision du monde dont est porteur un logiciel, c'est l'interface homme/ordinateur qui la détient et la communique à l'usager. Point de rencontre entre l'homme et l'ordinateur, l'interface constitue aujourd'hui l'objet des soins les plus attentifs ; il n'en a pas toujours été ainsi. La longue et profonde évolution des interfaces se reflète dans l'évolution du vocabulaire employé pour les désigner : le passage des notions d'entrées et de sorties au terme d'interface homme-ordinateur, de plus en plus souvent ramené au simple vocable d'interface, sans autre précision.

> La dactylocodeuse ou l'opératrice de saisie alimentait la machine, d'autres opérateurs recueillaient et traitaient les résultats du calcul. Le vocabulaire témoignait de la position que l'automate occupait au cœur du dispositif sociotechnique. L'"entrée" et la "sortie" étaient situées de part et d'autre d'une machine centrale. Cette époque est révolue. Moyennant un véritable pliage logique, les deux extrémités se sont rejointes et, tournées du même côté, elles forment aujourd'hui l'"interface". Au moment où la majorité des utilisateurs ne sont décidément plus des informaticiens de profession, quand les problèmes subtils de la communication et de la signification supplantent ceux de la gestion lourde et du calcul brut qui furent ceux de la première informatique, l'interface devient le point nodal de l'agencement sociotechnique.
>
> Pierre Levy, op. cité.

Tel est le contexte général à l'origine du courant de recherche sur les IHMs, recherche à laquelle cette thèse s'efforce d'apporter contribution.

Le sujet

C'est à la conception et à la réalisation des systèmes interactifs que nous nous intéressons. Issus de la recherche sur l'ingénierie des Interfaces Homme-Machine (IHM), nos travaux ont plus précisément pour thème le support à ces activités de conception et de réalisation.

Les résultats actuels de la recherche offrent d'une part des méthodes et modèles, et d'autre part des outils informatiques. L'exploration des contributions des deux types nous conduit aux observations suivantes, point de départ et motivation de cette thèse :

1. Les modèles de conception sont coupés des modèles de réalisation. Par nature, les premiers ne concernent que les activités de spécification conceptuelle et/ou externe d'une interface ; les derniers offrent des modèles d'architecture d'un système sans lien avec les résultats de la spécification.
2. Les modèles d'architecture proposés comme guides de la réalisation sont des modèles généraux et théoriques, dont l'emploi est souvent rendu malaisé par l'absence de références pratiques.
3. La majorité des outils d'aide au développement offre des services du niveau de la spécification externe d'une interface : la description de sa présentation, et dans certains cas d'une partie de la dynamique du dialogue ; les aspects conceptuels de l'interface sont totalement ignorés. Les outils de spécification fonctionnelle et de génération automatique d'interface, encore peu représentés, offrent des perspectives intéressantes qui ne sont jusqu'à présent qu'effleurées.

Cette thèse met l'accent sur la notion de représentation conceptuelle d'une interface. Nous montrons que la transition entre conception et réalisation peut être traitée à partir de la représentation conceptuelle des interfaces, dans le cadre d'un modèle d'architecture tout comme dans la perspective d'un outil d'aide au développement. Nous proposons l'utilisation d'un modèle de représentation conceptuelle d'une interface utilisateur à la fois comme cadre de référence d'un modèle d'architecture étendu, et comme entrée d'un générateur de code réalisant l'interface.

Nous avons conçu un modèle de représentation conceptuelle d'interface caractérisé par la distinction entre dimension de fonctionnement et dimension d'utilisation d'une part, et par une modélisation à objets d'autre part. Ce modèle de représentation est muni d'un langage de spécification, l'ensemble répondant au nom de SIROCO. SIROCO donne lieu à la définition d'un modèle d'architecture étendu utilisant le contenu d'une spécification conceptuelle comme argument, apportant ainsi un support à la transition entre conception et réalisation. Ce support théorique est concrétisé par le développement d'un prototype de générateur de code : SIROCO-Guide prend en entrée une description en langage SIROCO, et produit en langage Guide le code nécessaire à la réalisation de l'interface spécifiée.

Organisation de la thèse :

Cette thèse est composée de six chapitres dont l'organisation reflète notre propre démarche de recherche.

Le Chapitre I n'a d'autre but que d'introduire le cadre à l'intérieur duquel se placent nos travaux. Le regard que nous posons sur un système interactif retient avant tout ses dimensions humaines : l'utilisateur, le concepteur et le développeur. Le Chapitre I considère chacune de ces dimensions, et en explicite les principaux aspects. L'objectif est de décrire un ensemble de notions à la base de nos travaux, tout en précisant la terminologie que nous choisissons d'adopter dans ce document.

Les Chapitres II et III rapportent notre exploration de l'existant, et la réflexion personnelle issue de nos observations. C'est ici que se dessine la motivation de cette thèse, et que sont définis les objectifs de nos travaux.

Le Chapitre II analyse les méthodes et modèles de développement des systèmes interactifs décrits dans la littérature. Nous distinguons les méthodes de conception et les méthodes de développement, en accordant attention au problème de la transition entre ces deux ensembles de méthodes. Les méthodes de conception donnent lieu à un éventail de solutions spécialisées ; les méthodes de développement, plus uniformes, offrent des modèles d'architecture guidant la structuration du code d'un système interactif. Nous constatons dans un premier temps l'insuffisance des modèles d'architecture théoriques ; dans un second temps, nous observons l'absence de lien entre modèles de conception et modèles de réalisation. Cette double constatation est à la source des travaux décrits dans cette thèse.

Le Chapitre III explore l'espace des outils logiciels d'aide au développement d'un système interactif. C'est une analyse du niveau d'abstraction des concepts d'un outil qui est effectuée. A partir du résultat de cette analyse, nous nous interrogeons sur la place d'un outil dans le processus de conception d'un système interactif, et, en référençant les conclusions du Chapitre II, nous précisons nos axes de recherche : le support à la conception, le pont entre modèle de conception et modèle de réalisation, ainsi que la génération automatique d'interface. Notre objectif est un outil d'aide au développement axé sur une représentation conceptuelle de l'interface.

Les Chapitres IV et V présentent le résultat de nos travaux de thèse.

Le Chapitre IV introduit le langage de spécification SIROCO, en mettant en lumière le modèle de représentation conceptuelle sous-jacent à ce langage. Nous décrivons tout d'abord les principes dont est issu SIROCO - plus précisément, les principes régissant le contenu et la forme du langage : la perspective de la logique d'utilisation d'un système et le modèle de structuration à objets. Le modèle de représentation sur lequel repose le langage SIROCO est ensuite défini. Le formalisme même fait l'objet d'une présentation rapide, sans entrer dans les détails de la syntaxe du langage SIROCO. Enfin, nous proposons d'éclairer notre exposé par une section offrant un exemple complet de spécification d'un système interactif selon ce langage.

Le Chapitre V décrit notre approche de la réalisation d'un système interactif à partir d'une spécification en langage SIROCO : la passerelle jetée en direction des modèles de réalisation, et les principes du générateur de code réalisé. Nous décrivons l'utilisation de SIROCO comme cadre de référence d'un modèle d'architecture étendu, puis nous montrons comment SIROCO fournit la matière d'un outil de génération automatique du code de

réalisation d'un système interactif. Le chapitre décrit la démarche adoptée dans la conception de cet outil de génération de code, et présente les principales caractéristiques de la maquette de générateur réalisée pour le langage Guide, sous le nom de SIROCO-Guide.

Le Chapitre VI donne une évaluation des travaux réalisés autour de SIROCO, relativement à l'existant. Dans un premier temps, nous évaluons l'originalité et l'intérêt de SIROCO-Guide par rapport aux outils de nature voisine que décrit la littérature (les références choisies sont UIDE et MacIDA UIMS). Dans un second temps, notre réflexion porte sur nos travaux de modélisation ; nous faisons le point sur l'intérêt et les potentialités de SIROCO relativement au processus général de conception et de développement des interfaces utilisateur.

En conclusion de cette thèse, nous proposons un résumé des apports des travaux présentés, et nous décrivons les principales perspectives de développement de nos travaux.

Trois annexes sont fournies en fin de ce document. L'Annexe A présente la grammaire du langage de spécification SIROCO ; l'Annexe B est une description technique du prototype de générateur de code SIROCO-Guide ; l'Annexe C offre un exemple de code généré par SIROCO-Guide.

Chapitre I

Un système interactif et ses dimensions humaines

1 Introduction

Un système logiciel interactif peut être perçu de multiples façons. Au-delà du programme informatique dans la mémoire de l'ordinateur, au-delà des formes et couleurs sur l'écran, nous souhaitons mettre l'accent sur la *dimension humaine* qui, à notre sens, caractérise un système interactif. Utilisateur, concepteur et développeur[1] sont les trois composantes de cette dimension humaine, illustrée en Figure I-1.

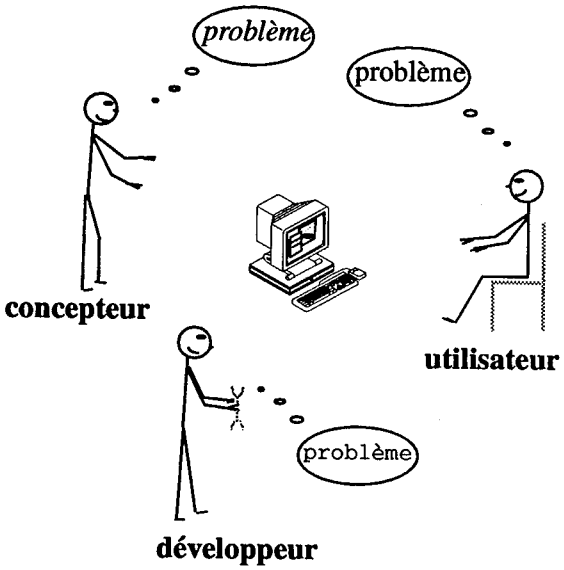


Figure I-1 *Les dimensions humaines.*

Un système logiciel a pour rôle de permettre le traitement par son utilisateur d'un problème donné. L'*utilisateur* constitue naturellement le principal élément de la dimension humaine attachée au système interactif. Le système est pour ce premier un outil ; l'interaction homme-ordinateur dénote l'utilisation de cet outil en vue du traitement du problème, et de la réalisation des objectifs de l'utilisateur.

Tout outil porte l'empreinte de son créateur ; le système interactif, en raison de la complexité des problèmes auxquels il répond, est, plus que tout autre, marqué par les idées de son *concepteur*. En effet, le système interactif est porteur d'une vision du monde qui détermine les termes dans lesquels vont être menées les tâches de l'utilisateur. Cette vision du monde, c'est celle du concepteur du système. L'élaboration de cette vision du monde s'effectue lors du travail de conception, à partir de l'analyse de l'existant et des besoins de l'utilisateur. Le concepteur est ainsi le deuxième acteur de la dimension humaine d'un système interactif. Le système n'apparaît plus maintenant comme un simple outil dont se sert l'utilisateur pour

1. Les termes d'utilisateur, concepteur et développeur d'un système sont utilisés ici dans un sens générique : ces singuliers dénotent des *rôles* qui, selon les cas, sont pris en charge par des ensembles d'individus. Un système est le plus souvent conçu et développé pour plusieurs utilisateurs, par une équipe de concepteurs couvrant l'étendue des analyses et spécifications inhérentes au travail de conception, et par un ensemble de développeurs spécialisés.

réaliser un objectif ; c'est également le lieu de communication entre deux individus, le concepteur et l'utilisateur. Il s'agit pour le concepteur de transmettre à l'utilisateur sa vision du monde ; il s'agit pour l'utilisateur de percevoir la vision du concepteur, et d'exprimer ses besoins dans les termes de cette vision.

Enfin, un troisième acteur vient compléter la dimension humaine d'un système interactif : le *développeur* du système. C'est lui qui, de façon effective, crée sur support informatique le système imaginé par le concepteur.

Les sections suivantes reprennent cette triple perspective humaine afin d'en expliciter les principaux aspects. L'objectif est d'introduire un ensemble de notions qui fournissent le cadre de développement de nos travaux, tout en précisant la terminologie que nous choisissons d'adopter dans ce document.

Nous proposons dans un premier temps une définition d'un système interactif selon cette perspective humaine, définition qui servira par la suite de référence. Sont ensuite successivement décrites les notions de tâche, d'interaction, d'utilisateur, et de concepteur et de développeur.

Mise au point préliminaire

La multiplicité des interprétations possibles des termes informatiques les plus élémentaires nécessite de préciser le vocabulaire que nous employons.

Les termes de *système interactif* et d'*application interactive* sont utilisés indifféremment dans ce document. Tous deux dénotent un logiciel manipulable par l'utilisateur en vue du traitement d'un problème donné. Afin d'alléger notre discours, l'adjectif "interactif" ou "interactive" pourra être omis. Le terme "application" est ainsi équivalent à "application interactive", contrairement à une interprétation parfois rencontrée, qui assimile l'application à la partie non interactive du logiciel.

2 Définition référence

Nous proposons ici une définition d'un système interactif qui servira de référence tout au long de ce document. Reflet de la triple dimension humaine évoquée en introduction de ce chapitre, cette définition est inspirée de celle donnée par Norman dans [Norman 83] (cité dans [Tauber 90]) et dans [Norman 86] ; sa caractéristique est d'introduire les notions de modèles conceptuel, fonctionnel, et mental d'un système interactif.

Soit σ un système interactif. On définit σ comme un quintuplet $< P_\sigma, I_\sigma, C_\sigma, R_\sigma, M_\sigma >$, dont les éléments résultent de l'adoption de différentes perspectives sur σ, et que nous définissons ci-après. La Figure I-2 illustre les cinq éléments de σ, et les relations existant entre eux.

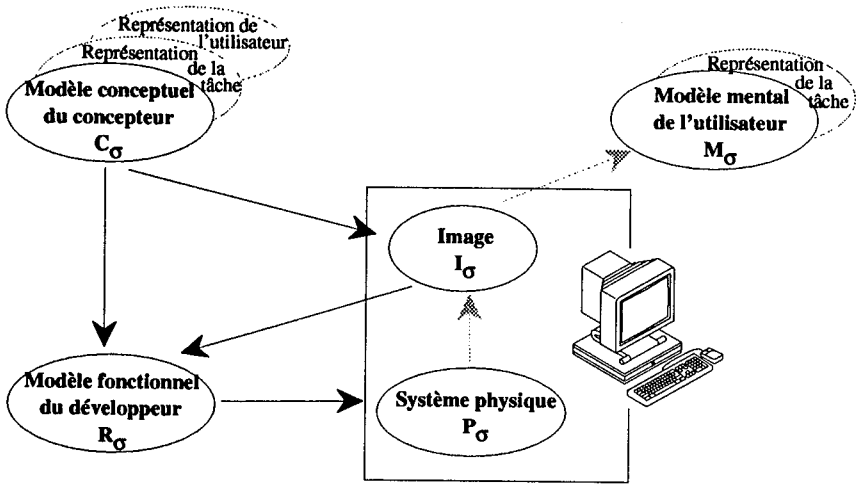


Figure I-2 *Modélisation d'un système interactif.*

Une première vue de σ porte sur son existence strictement physique : P_σ représente le système physique. Il s'agit là du résultat effectif de la tâche de développement de σ.

Définition : Le **système physique** P_σ est l'ensemble des modules de code dénotant l'existence physique du système interactif σ, éventuellement accompagnés de documents explicatifs.

I_σ décrit une vision externe du système : l'image qu'offre le système à l'utilisateur.

Définition : L'**image** I_σ représente la présentation perceptible du système σ, et les caractéristiques de l'interaction que permet le système ; entrent également en compte les éventuels manuels d'utilisation fournis avec le système.

Au-delà des représentations concrètes qu'offrent P_σ et I_σ, un système se caractérise par trois grands types de représentations abstraites : C_σ, le modèle conceptuel du concepteur, R_σ, le modèle fonctionnel du développeur, et M_σ, le modèle conceptuel de l'utilisateur.

Définition : Le **modèle conceptuel** C_σ est la représentation du système élaborée par le concepteur. C_σ décrit la vision du monde proposée par le concepteur : les concepts (entités et opérations) dont le système est porteur, ainsi que les relations entre ces concepts.

C_σ est le résultat de la conception du système ; c'est autour de ce modèle que s'organisent les tâches de mise en œuvre de σ. La définition de C_σ repose sur un ensemble de contraintes, d'hypothèses et de choix quant à l'objectif recherché pour le système σ, et quant aux caractéristiques de l'utilisateur et de son activité. Ces contraintes, hypothèses, et choix se traduisent notamment en un modèle de tâche (cf section 3) décrivant les tâches auxquelles σ apporte un support.

Définition : Le **modèle du développeur** R_σ est une représentation fonctionnelle décrivant la façon dont C_σ est réalisé sur le support informatique, et dont l'image I_σ est obtenue.

R_σ décrit le système en termes informatiques (fonctions, objets, modules, etc.). R_σ constitue un guide de réalisation de P_σ.

Il est aujourd'hui admis que l'activité de l'utilisateur d'un système s'organise autour de la construction et l'application d'un *modèle mental* du système. Un modèle mental est défini par Carroll dans [Carroll 84] comme "les structures et processus que l'on impute à une personne afin de justifier le comportement et la pratique de cette personne". M_σ est le modèle mental développé par l'utilisateur de σ.

Définition : Le **modèle mental** M_σ est la représentation conceptuelle du système σ détenue par l'utilisateur de ce système.

Nous évoquerons en section 5 le contenu de ce modèle mental. Notons dès à présent que ce modèle mental repose sur une représentation des tâches dont le système permet la réalisation, et des procédures de mise en œuvre de ces tâches.

P_σ, I_σ, C_σ, R_σ et M_σ dénotent cinq facettes du système σ, distinctes mais non indépendantes. I_σ est bien entendu le résultat perceptible de la mise en œuvre de P_σ. P_σ est la réalisation concrète des spécifications définies par C_σ et de la description de I_σ, selon les termes de R_σ. Enfin, M_σ est construit par l'utilisateur à partir de sa perception de l'image du système I_σ.

3 La tâche

L'objectif d'un système logiciel est de permettre le traitement par l'utilisateur d'un problème donné. Ce traitement passe par la mise en œuvre d'un ensemble de tâches.

Définition : Une **tâche** définit un objectif à atteindre par l'utilisateur, à l'aide du système interactif.

Par exemple, "consulter une boîte-à-lettres", "lire un message", "répondre à un message" sont des tâches, dans le contexte d'un système de messagerie électronique. On distingue les *tâches composées* et les *tâches élémentaires*. Une tâche composée est une tâche dont réalisation fait intervenir la mise en œuvre d'autres tâches. L'analyse d'une tâche composée donne lieu à une **décomposition hiérarchique** en sous-tâches liées par des relations temporelles (séquentialité, parallélisme, etc.) ou bien logiques (options, choix d'une sous-tâche parmi plusieurs, etc.). La décomposition en sous-tâches se poursuit récursivement jusqu'à ce que l'on parvienne à un niveau de sous-tâches dites élémentaires, que l'on ne peut décomposer plus avant sans faire d'hypothèses sur le système à disposition. La Figure I-3 illustre le concept de décomposition hiérarchique d'une tâche en présentant la structure de la tâche d'envoi d'un message électronique.

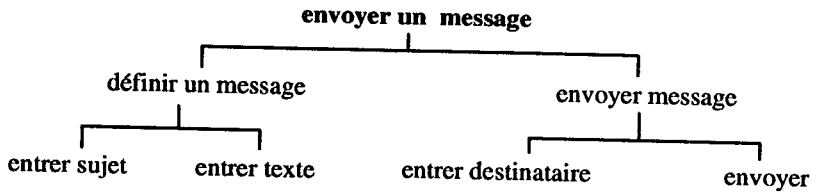


Figure I-3 *Illustration du principe de décomposition hiérarchique d'une tâche.*
(relations séquentielles entre les sous-tâches)

L'ensemble des tâches d'un système s'organise en un **arbre des tâches** dont la racine est la tâche ou le problème général auquel répond le système, et dont les feuilles sont les tâches élémentaires. Un grand degré de variabilité peut être observé entre les arbres des tâches en général. Selon le domaine de l'application considérée, l'arbre est plus ou moins fortement structuré : d'une part, la profondeur de l'arbre est plus ou moins grande ; d'autre part, la décomposition d'une tâche est plus ou moins rigide, dans le sens où l'exécution de cette tâche donne lieu à une séquence de tâches élémentaires bien déterminée ou bien au contraire sujette à alternatives. Le degré de prévisibilité de la séquence de décomposition effective d'une tâche composée influe notamment sur le type de dialogue mis en œuvre dans l'interface du système, comme nous le verrons en section 4.

Deux exemples extrêmes : une tâche de contrôle de processus et la tâche de production d'un document. Dans le premier cas, la tâche est fortement structurée, dans le sens où son exécution passe par une série d'étapes (sous-tâches) bien définies, éventuellement strictement séquentielles. Dans le second cas, la tâche est très peu structurée : on sait qu'un large éventail de sous-tâches peut être appelé lors de la réalisation de la tâche, avec des probabilités différentes, selon certaines contraintes. Il n'est pas possible de prévoir de plans très précis de réalisation de cette tâche : la tâche se décompose en un ensemble de sous-tâches dont l'exécution est alternative, a priori sans contraintes.

Nous avons jusqu'à présent parlé d'un modèle de tâche générique pour décrire la raison d'être d'un système. Notons que ce modèle de la tâche n'est pas unique : chaque intervenant, concepteur ou utilisateur du système, possède son propre modèle de la tâche.

Le concepteur définit le système en fonction d'un modèle de la tâche conçu à partir du cahier des charges qui lui est remis, et d'une analyse de l'existant ; ce modèle de la tâche répond aux besoins exprimés dans le cahier des charges en intégrant les caractéristiques de l'environnement de travail de l'utilisateur, telles que mises en évidence lors d'études ergonomiques. Cette élaboration du modèle de la tâche du concepteur détermine les caractéristiques des utilisateurs du système futur, les fonctions à réaliser dans ce système, les objets manipulés par ce système, et la façon dont ces fonctions et objets s'insèrent dans l'environnement de travail des utilisateurs.

Un utilisateur particulier quant à lui aborde un système en ayant (a priori) en tête un ensemble de buts à réaliser. Le modèle que l'utilisateur a de la tâche à réaliser est très variable selon les individus, reflétant son expérience et son niveau d'apprentissage, comme nous le verrons en section 5. La qualité de l'interaction, qualité de la communication entre utilisateur et concepteur, dépend en grande partie de la cohérence entre le modèle de la tâche détenu par l'utilisateur et celui du concepteur du système.

4 L'interaction

Nous distinguons deux aspects de l'interaction entre l'utilisateur et un système : d'une part, le style général de l'interface mise en œuvre, caractéristique d'une métaphore d'interaction ; d'autre part, indépendamment du style d'interface mis en œuvre, le mode de contrôle du dialogue.

Style d'interface

Diverses classifications ont été proposées dans la littérature afin de caractériser les interfaces utilisateur et le type d'interaction offerts. On retiendra notamment celle de B. Laurel dans [Laurel 86], qui distingue les interfaces "première personne", et les interfaces "deuxième personne", distinguées selon le degré d'engagement de l'utilisateur.

Une *interface "première personne"* apparente l'application à un monde à l'intérieur duquel évolue l'utilisateur en vue de réaliser une tâche ; l'engagement de l'utilisateur dans l'interaction est maximal, ce dernier manipulant directement les éléments de l'application. Dans les termes mieux connus de [Hutchins 86], la métaphore utilisée est celle du *monde réel* : l'interface est une représentation d'un monde proche du monde réel de l'utilisateur ; l'interaction consiste pour ce dernier à agir directement sur les représentations des éléments de ce monde, en mimant les actions au moyen notamment de la souris. Un exemple typique de ce style d'interface est celui du "desktop" du Macintosh ; pour détruire un fichier, l'utilisateur saisira (avec la souris) l'icône représentant ce fichier, et la déposera dans la corbeille à papier.

Une *interface "deuxième personne"* dénote un engagement minimal de l'utilisateur dans l'interaction : ce dernier est amené à donner des ordres à un intermédiaire informatique afin de réaliser son objectif. La métaphore utilisée est, dans les termes de [Hutchins 86], celle de la *conversation* : l'utilisateur n'agit pas directement sur les objets du domaine, mais décrit au système les actions à réaliser. Le langage de description utilisé peut être très divers : langage de commandes (tel le langage du shell sous Unix), sous-ensemble d'un langage naturel, désignation directe d'éléments de l'interface (notamment, éléments de menus), ...

Il est à remarquer qu'une interface ne comportant pas d'aspects "deuxième personne" est rare. C'est que la métaphore du monde réel a des limites : elle peut difficilement s'appliquer pour toute commande. C'est en fait bien souvent un mélange des deux styles qui est réalisé.

Contrôle du dialogue

Le dialogue entre l'utilisateur et le système se caractérise tout d'abord par le type de séquencement autorisé. On distingue les dialogues simples et les dialogues multifils.

Un *dialogue simple* limite l'interaction à un seul fil d'activité : pour lancer une commande, il est nécessaire d'attendre que la commande précédente soit achevée.

Un *dialogue multifil* autorise au contraire l'exécution de plusieurs activités en parallèle : l'utilisateur peut mener de front plusieurs dialogues avec le système, en passant de l'un à l'autre. L'intérêt de ce type de dialogue est de permettre à l'utilisateur de tirer parti du "parallélisme" offert par l'ordinateur pour mener à bien sa tâche. La création d'un nouveau fil de dialogue dénote couramment une des situations suivantes :

- cas d'une opération longue, qui ne nécessite pas un contrôle continu de la part de l'usager (par exemple, une impression) : l'utilisateur désire "reprendre la main" pour poursuivre son activité sans attendre la fin de cette opération.
- cas d'une tâche nécessitant la réalisation d'autres tâches : l'utilisateur a besoin de lancer d'autres opérations afin de mener à bien une tâche entamée. (Par exemple, opérations de recherche d'information, de modification du contexte d'exécution de la première tâche, etc.).

Le terme de "contrôle du dialogue" exprime le mode de synchronisation entre l'utilisateur et le système : qui dirige qui ? Que le dialogue soit simple ou multifil, le contrôle de ce dialogue peut prendre l'une des trois formes suivantes, identifiées dans [Tanner 83] (cité dans [Coutaz 90]) :

- contrôle interne : c'est le système qui dirige l'activité de l'utilisateur.
  L'interaction est une suite de questions du système et de réponses de l'utilisateur. Le contrôle interne limite la liberté d'action de l'utilisateur au séquencement des opérations prévu par le système. Les applications sur minitel sont typiques de ce type de contrôle.
- contrôle externe : c'est l'utilisateur qui dirige le système.
  L'interaction est une suite d'actions de l'utilisateur et de réponses du système, le séquencement du dialogue étant entièrement conditionné par les interventions de l'utilisateur ; l'application a le rôle passif d'un serveur de fonctions.
- contrôle mixte : c'est tour à tour le système et l'utilisateur qui dirigent les échanges.
  Le contrôle mixte est caractéristique de la plupart des applications actuelles : l'utilisateur possède le contrôle général du dialogue, mais le système prend de temps en temps la direction des échanges afin, par exemple, de recueillir une information nécessaire à la mise en œuvre d'un traitement, ou bien pour avertir l'utilisateur d'une condition inhabituelle, ou encore pour des raisons de sécurité (cas du contrôle de processus notamment), etc.

5 L'utilisateur

Les phénomènes mentaux chez l'utilisateur d'un système sont encore mal cernés. La notion de modèle mental reste floue ; s'il est aujourd'hui admis que les connaissances d'un utilisateur sur un système sont partiellement organisées en une structure appelée "modèle mental", le contenu et la forme de ce modèle sont mal définis. Les travaux rapportés dans [Tauber 90] par exemple sont instructifs quant aux directions empruntées par la recherche dans ce domaine, et quant à l'état de ces recherches.

La compréhension et la représentation du modèle mental de l'utilisateur d'un système est rendue particulièrement complexe par le fait que le terme générique d'utilisateur dissimule une grande variabilité. L'utilisateur d'un système interactif n'est pas un mais multiple : d'une part, un système s'adresse a priori à un ensemble d'utilisateurs plus ou moins bien défini ; d'autre part, la variabilité intervient également dans le temps pour un utilisateur donné. Cette variabilité inter ou intra utilisateurs porte sur les objectifs, l'expérience, et le degré d'apprentissage de ces derniers ; ce sont là les principales dimensions caractéristiques du modèle mental d'un utilisateur. Cette variabilité est la plupart du temps délibérément ignorée dans les travaux de recherche sur les modèles mentaux, faute de savoir la gérer ; ce sont des modèles mentaux idéaux qui sont alors envisagés.

Nous ne cherchons pas à définir plus avant le contenu d'un modèle mental. Notre objectif est ici d'indiquer quelques résultats fournissant un cadre pour la compréhension des caractéristiques de l'utilisateur : son activité lors de l'utilisation d'un système, les contraintes physiques de cette activité, la perception qu'a l'utilisateur des concepts, et les phénomènes d'apprentissage.

5.1 L'activité de l'utilisateur

Nous rapportons ici brièvement quelques résultats théoriques rendant compte de l'activité d'un utilisateur lors de l'interaction de celui-ci avec un système quelconque. Ces résultats sont issus des travaux de Norman ([Norman 86]) et Hutchins et al. ([Hutchins 86]).

L'activité d'un utilisateur consiste en l'accomplissement d'une suite de tâches. La réalisation d'une tâche se décompose de façon schématique en un cycle de six étapes ainsi définies :

1. la formulation d'une intention.
   Une intention se définit comme une sous-tâche élémentaire participant à la réalisation du but. La formulation d'une intention suppose un processus de décomposition de la tâche à réaliser jusqu'à l'obtention de sous-tâches élémentaires susceptibles d'être effectuées avec le système.
2. la spécification d'une séquence d'actions concrètes, ou procédure opérative.
   Une *procédure opérative* est une suite d'opérations du système interactif dont l'exécution correspond à l'accomplissement d'une sous-tâche élémentaire. Cette étape de spécification consiste en la construction d'une représentation mentale de la suite d'actions à effectuer.
3. l'exécution effective de cette suite d'actions.
   Les processus en jeu sont ici des processus moteurs : l'utilisateur communique au système la suite des opérations qu'il souhaite exécuter, dans le langage de ce système.

4. la perception de l'état du système.
   Il s'agit ici principalement d'une activité physique de perception des modifications éventuelles de l'image du système.
5. l'interprétation de l'état du système.
   Il s'agit pour l'utilisateur de rattacher l'état perçu du système à un état du monde de la tâche.
6. l'évaluation de l'état du système par rapport au but.
   A l'intérieur du monde de la tâche, l'utilisateur détermine la distance qu'il lui reste à parcourir pour atteindre le but fixé.

Les étapes 1, 2 et 3 constituent la partie exécution du cycle de réalisation d'une tâche, les étapes 4, 5 et 6 mettant en œuvre la partie évaluation de ce cycle. Comme le note en particulier J. Nanard dans [Nanard 90], la mise en œuvre de ces six étapes conduit l'utilisateur à considérer tour à tour le monde de la tâche et le monde réel du système, au travers des représentations mentales que l'utilisateur se fait de ces deux mondes : modèle de la tâche et modèle du système. Nous explicitons en Figure I-4 les liens de localisation entre l'activité de l'utilisateur et les deux modèles mentaux que possède ce dernier : la représentation mentale de la tâche est le monde à l'intérieur duquel se situent les étapes 1 et 6 ; les étapes 2 et 5 ont pour support le modèle du système développé par l'utilisateur ; enfin, les étapes 3 et 4 représentent les actions concrètes et visibles de l'utilisateur, par opposition aux actions mentales des autres étapes.

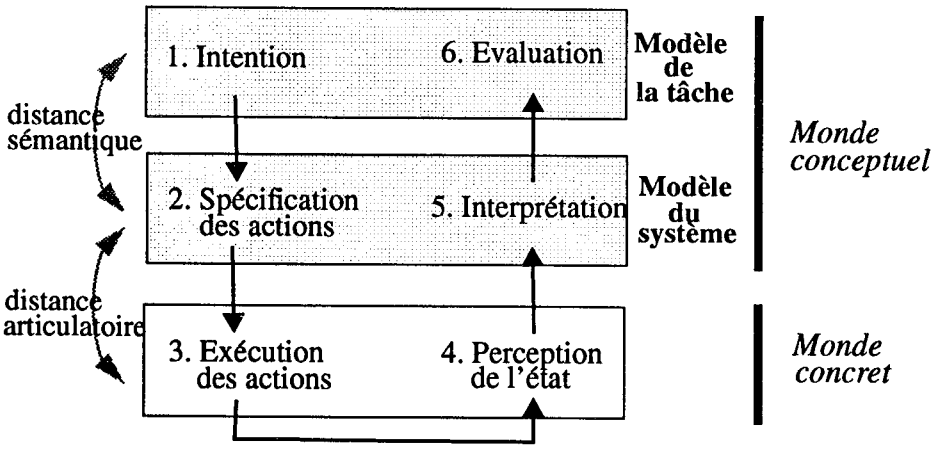


Figure I-4 *Modélisation de l'activité de l'utilisateur.*

L'activité de l'utilisateur nécessite le franchissement d'une part d'une distance sémantique, et d'autre part d'une distance articulatoire. La *distance sémantique*, terme défini par Hutchins (op. cité), dénote l'effort de transition entre modèle de la tâche et modèle du système ; il s'agit pour l'utilisateur de faire correspondre la connaissance qu'il a de la tâche avec la connaissance qu'il a du système, et vice-versa. La *distance articulatoire* dénote l'effort de transition entre le modèle conceptuel du système et le monde réel ; il s'agit pour l'utilisateur de traduire une action en une séquence d'actions physiques sur le système (enfoncement de touches du clavier ou clics souris, par exemple), et vice-versa d'élaborer une

information sémantique sur l'état du système à partir d'éléments concrets de l'image du système.

Les distances sémantique et articulatoire caractérisent l'"utilisabilité" d'un système pour un utilisateur donné : ces deux distances constituent un cadre d'analyse des difficultés ou facilités d'utilisation d'un système, relativement aux caractéristiques de l'utilisateur considéré : contraintes physiques, expérience passée, degré d'apprentissage.

5.2 Contraintes physiques

L'utilisation d'un outil quelconque, informatique ou non, est régie par un ensemble de limitations physiques du corps humain. Il ne s'agit pas ici d'énumérer la liste exhaustive de ces limitations, mais d'en énoncer les plus pertinentes pour le domaine qui nous intéresse ([Card 83]) :

- contraintes du système sensoriel : deux images affichées à moins de 10 ms d'intervalle sont perçues comme confondues.
- contraintes du système moteur : la loi de Fitts décrit l'inertie des mouvements humains, ou le temps nécessaire au placement de la main sur une cible de dimension donnée, et située à une distance donnée.
- contraintes du système cognitif : la capacité de la mémoire à court terme, lieu de stockage de l'information du travail courant, est reconnue comme étant de 7±2 éléments.

Les contraintes physiques ont une influence directe sur la distance articulatoire que doit franchir l'utilisateur. Les contraintes sensorielles délimitent les conditions à respecter pour que les modifications de l'image soient correctement perçues par l'utilisateur : deux modifications trop rapprochées ne seront pas perçues distinctement. Les contraintes motrices définissent les limites des séquences d'actions qu'un utilisateur est capable d'accomplir aisément. Les contraintes cognitives donnent lieu à la définition de la notion de charge cognitive imposée par un système : il est impératif de limiter au maximum la charge cognitive, afin de ne pas saturer la mémoire à court terme de l'utilisateur.

5.3 Phénomènes d'apprentissage

L'élaboration par l'utilisateur d'un modèle de la tâche et d'un modèle du système M_σ résulte d'une phase d'apprentissage de ce système. L'apprentissage est influencé par deux ensembles de facteurs :

- l'image I_σ que présente le système (non seulement l'image à l'écran mais également la documentation, ou la formation en relation avec le système).
- l'expérience préalable que possède l'utilisateur d'outils ou de situations analogues.

L'objectif de l'apprentissage est de constituer un modèle mental M_σ le plus proche possible du modèle conceptuel C_σ défini par le concepteur du système. De cette proximité dépend directement la distance sémantique ressentie par l'utilisateur. En effet, la distance sémantique varie selon la qualité des modèles formés par l'utilisateur pour la tâche et le système : elle sera d'autant plus grande que le modèle de la tâche est éloigné de celui défini par le concepteur, et que le modèle du système est incomplet ou erro

L'analyse des phénomènes intervenant dans l'apprentissage d'un système est hors de notre propos. Nous nous contentons ici de souligner le mode de perception par l'utilisateur des concepts d'un système à partir de leur représentation dans l'image I_σ. J. Nanard dans [Nanard 90] nous rappelle les principaux mécanismes intervenant dans la perception des concepts d'un système : d'une part le transfert métaphorique, et d'autre part l'application de l'hypothèse de cohérence[1].

Le transfert métaphorique stipule que la reconnaissance d'un nouveau concept par l'utilisateur repose sur la connaissance antérieure qu'a ce dernier de situations analogues ; l'évocation de cette connaissance se fonde sur une analogie de représentation et/ou une analogie de manipulation, qui induisent alors une analogie sémantique.

L'analogie de représentation et de manipulation entre plusieurs éléments de l'interface d'un système est implicitement perçue par l'utilisateur (hypothèse de cohérence) comme traduisant le partage d'une propriété sémantique commune par les objets et les opérations correspondant à ces représentations ou manipulations. L'application de l'hypothèse de cohérence a pour effet de structurer la connaissance de l'usager et de lui permettre d'identifier des entités et leurs relations.

1. Si les travaux de J. Nanard s'inscrivent dans le cadre des systèmes à manipulation directe, les mécanismes de perception décrits peuvent cependant être généralisés à tout système interactif.

6 Conception et développement : acteurs et processus

Avant-propos : En abordant le sujet du processus de mise en œuvre d'un système interactif, nous sommes frappés par l'imprécision et l'ambiguïté des termes employés dans ce domaine. Dans certains cas (par exemple, "conception"), un même terme pourra être appliqué dans plusieurs sens différents ; selon les communautés de chercheurs, un même terme (par exemple, "développement"), se verra attribuer des significations opposées. Nous tentons ici de nous conformer à une terminologie en cours dans la communauté des ergonomes et informaticiens. Cette terminologie est cependant étendue afin de répondre à nos intérêts propres. De façon générale, chaque terme susceptible d'ambiguïté fera l'objet d'une définition explicite.

Concepteur et développeur sont deux termes génériques qui, dans la réalité, dissimulent un ensemble de personnes de compétences variées : analystes, spécialistes du domaine, ergonomes, psychologues, graphistes, etc. pour le travail de conception ; programmeurs du domaine, spécialistes en gestion d'interface, etc. pour le travail de développement effectif du logiciel sur support informatique. Il est clair que analyste et développeur sont parfois une même et unique personne, chaque rôle étant tour à tour mené par cette même personne lors du déroulement du processus global de conception. Dans d'autres cas, c'est à une véritable équipe de conception et de développement que nous avons affaire, constituée de plusieurs personnes distinctes et coopérantes.

Le travail de conception et de développement consiste à explorer l'espace "problème" dont les sections précédentes ont ébauché les grandes dimensions, et à modeler peu à peu les formes du système en considérant chacune de ces dimensions. Plusieurs ébauches successives ou parallèles seront nécessaires avant d'aboutir au système solution. Un système peut en fait être perçu comme le *résultat de multiples transformations* ; le point de départ de cette activité est un cahier des charges décrivant les fonctions attendues du système ; les ébauches intermédiaires sont les spécifications et prototypes ; le résultat est un ensemble de modules de code exécutable P_σ réalisant le système.

L'objectif général est de produire un outil répondant aux besoins fonctionnels de l'utilisateur. Chaque rôle intervenant dans la tâche de conception et de développement apporte ses objectifs particuliers, éventuellement contradictoires, dont le logiciel produit représente un compromis :

- assurer la qualité de la communication entre l'utilisateur et le système.
- assurer la fiabilité et les performances du logiciel.
- assurer la qualité du logiciel développé : la facilité de maintenance, ses possibilités d'évolution, etc.
- etc.

La décomposition du processus global de conception permet d'apprécier la façon dont interviennent les différentes compétences du concepteur et du développeur. La progression depuis le cahier des charges jusqu'au système résultat s'effectue selon un cycle d'étapes dont la section suivante identifie les caractéristiques principales. Il s'agit également de préciser la terminologie dont nous faisons usage dans ce document. Nous détaillons ensuite les activités de conception et de développement.

6.1 Processus général de conception

La notion de *processus de conception*[1] recouvre l'ensemble des tâches nécessaires à la création d'un système logiciel. Le processus de conception est de façon classique représenté par une boucle reliant les grandes tâches que sont l'analyse des besoins et de l'existant, la spécification, le codage, et l'évaluation.

L'*analyse des besoins* a pour but de rassembler tous les éléments descriptifs de l'environnement dans lequel s'insère le système, et des tâches pour lesquelles le système est conçu. C'est à ce niveau que nous situons notamment l'analyse de la tâche, qui définit les arbres de tâches (cf. section 3). Les résultats de l'analyse des besoins constituent une plateforme à partir de laquelle est effectuée la *spécification*. La spécification donne lieu à un ensemble de descriptions généralement informelles portant sur les concepts (*spécification conceptuelle*), les aspects perceptibles de l'interface (*spécification externe*), et la représentation logicielle du système (*spécification interne*[2], globale ou détaillée). La spécification conceptuelle peut être grossièrement rapportée au modèle conceptuel C_σ ; la spécification externe décrit l'image I_σ ; la spécification interne résulte en un modèle de réalisation R_σ. Le *codage*, étape de réalisation effective du système, transforme R_σ pour obtenir les modules de code P_σ du système.

Rien n'est simple cependant ; la définition du contenu et des relations de dépendance entre ces grandes tâches nécessite de prendre en compte de nombreux paramètres, parmi lesquels les faits et recommandations suivantes :

- Un processus d'évaluation doit intervenir à chaque étape.
- Il est souhaitable de faire intervenir l'utilisateur le plus tôt possible dans l'évaluation du produit en cours de développement, notamment au moyen d'opérations de prototypage.
- Spécification et codage ne peuvent être séquentiels : un aller-retour entre ces deux tâches permet de tester la validité de la conception.
- etc.

Notre propos n'est pas de chercher à définir précisément les étapes que doit suivre le cycle de conception d'un logiciel ; la Figure I-5 donne un exemple de décomposition générale de ce cycle intégrant les remarques précédentes, variante du modèle bien connu de la cascade [Boehm 82].

Les étapes de spécification et de réalisation constituent nos principaux axes d'intérêt. Au lieu de cette dichotomie axée sur le passage d'un langage de description (formel ou non) à un langage de programmation, nous préferrons un découpage distinguant la description d'un système de celle de sa réalisation. Nous proposons les termes de conception et de développement pour refléter ce découpage - tel est le thème des deux sections suivantes.

---

1. Dans certaines méthodes du génie logiciel, le processus général est appelé "cycle de développement".

2. La spécification interne, qui a pour but la description du système interactif sous une forme directement transcriptible en langage de programmation, peut également apparaître sous le terme d' "étape de conception" dans certaines méthodes.

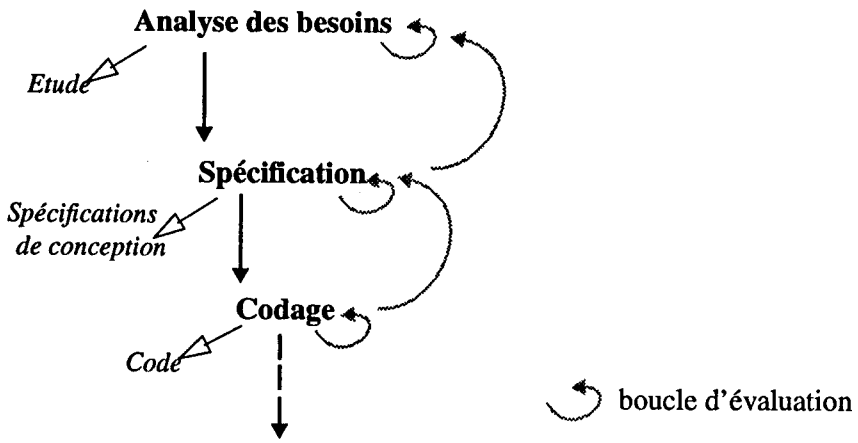


Figure I-5 *Exemple de décomposition générale du processus de conception.*

6.2 Activité de conception

Définition : Le terme de **conception** est utilisé pour représenter l'activité de spécification conceptuelle et externe d'un système interactif. Le **concepteur** est le ou les individus intervenant dans les activités de conception ainsi définies.

L'activité de conception met en jeu les deux facettes d'un système que sont le modèle conceptuel C_σ et l'image I_σ d'un système. La Figure I-6 représente les deux composantes de cette activité : la spécification conceptuelle et la spécification externe. De par la nature du processus de conception en général (cf Chapitre II, section 1), une stricte séquentialité entre spécification conceptuelle et spécification externe n'est pas toujours respectée.

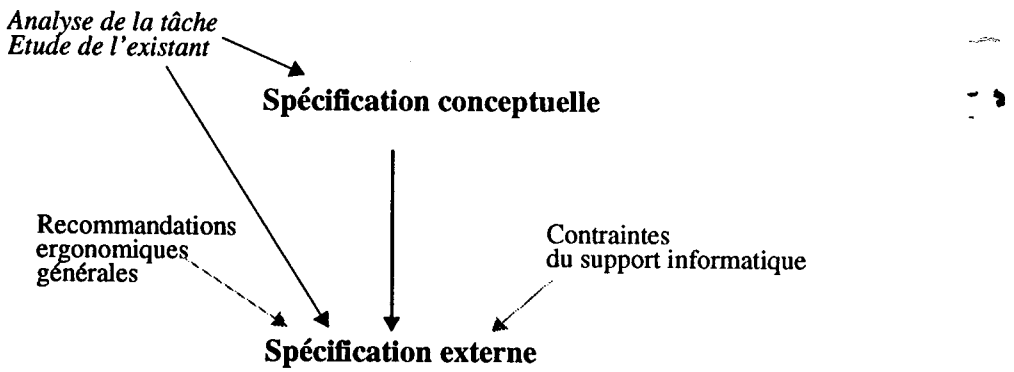


Figure I-6 *Décomposition de la tâche de conception.*

La *spécification conceptuelle* vise à définir le modèle conceptuel du système, c'est-à-dire le contenu sémantique et logique de ce système. C'est principalement à partir d'une analyse de la tâche que la spécification conceptuelle est réalisée ; l'objectif consiste à limiter la distance "sémantique" (cf 5.1) entre la représentation qu'a l'utilisateur de la tâche, et le modèle conceptuel qui lui est proposé dans le système.

La *spécification externe* s'attache à définir l'image du système. Elle est effectuée à partir des résultats de la spécification conceptuelle : il s'agit de définir la représentation externe des éléments conceptuels intervenant dans l'interface. La représentation des concepts dans l'image du système doit permettre à l'utilisateur de "reconstruire" ces concepts, grâce à une distance "articulatoire" (cf 5.1) réduite. Plusieurs aspects entrent en jeu afin de limiter cette distance "articulatoire" ; au-delà de considérations ergonomiques générales, une analyse des connaissances de l'utilisateur, et notamment de l'existant, est fondamentale. (Par exemple, l'existence de formulaires papier peut amener à adapter la présentation des données d'une application de bureautique). Enfin, le support informatique à disposition est une source de contraintes portant sur la faisabilité des choix de représentation externe.

6.3 Activité de développement

Définition : Le terme de **développement** recouvre la spécification interne et le codage du système. Le **développeur** est le ou les individus intervenant dans les activités de développement ainsi définies.

L'activité de développement recouvre l'ensemble des tâches directement liées au support informatique. Deux tâches principales peuvent être distinguées, illustrées en Figure I-7 : d'une part la tâche de conception du logiciel, que l'on nomme spécification interne, et d'autre part la tâche de programmation effective (codage).

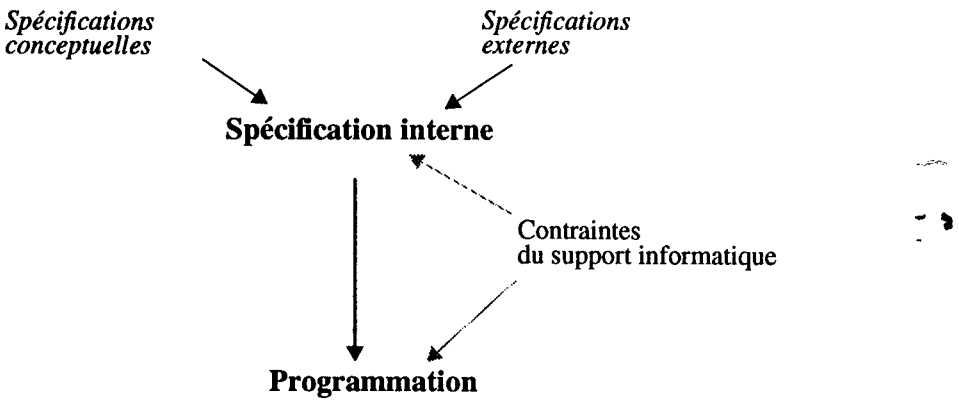


Figure I-7 *Décomposition de la tâche de développement.*

La *spécification interne* concerne le processus de conception du système physique à partir du résultat des spécifications conceptuelle et externe : il s'agit de concevoir les différents composants intervenant dans l'organisation du code réalisant les décisions conceptuelles et externes. Le résultat de la spécification interne est un modèle du code à réaliser, R_σ. Cette

tâche de spécification interne est à la limite entre conception et réalisation puisqu'elle effectue la transition entre ces deux grandes étapes. Elle est menée à partir des spécifications conceptuelles et externes, et subit l'influence des possibilités et limitations du support informatique d'accueil, et de l'environnement informatique de développement en général (les outils, langages et bibliothèques logicielles à disposition).

La tâche de *programmation* est la mise en œuvre effective du code réalisant le système, P_σ ; elle est guidée par les décisions prises lors de la spécification interne, et bien entendu directement dépendante de l'environnement informatique.

7 Conclusion

Les définitions proposées dans ce chapitre fournissent le cadre général de nos travaux.

Nous avons d'une part introduit une terminologie adaptée à nos intérêts, qui sera utilisée dans la suite de ce document. Sont à retenir les termes de description abstraite d'un système, notamment ceux de modèle conceptuel et modèle de réalisation ; la notion de tâche de l'utilisateur ; de style d'interface et de dialogue ; les caractéristiques de l'activité de l'utilisateur ; les notions de processus de conception, d'activité de conception et de développement d'un système.

Plus qu'un simple cadre terminologique, nous avons défini la perspective adoptée sur un système interactif. Trois acteurs sont mis en lumière : l'utilisateur, le concepteur et le développeur du système. Tout en gardant à l'esprit les caractéristiques complexes de cet utilisateur à qui est dédié le système, nous orientons nos travaux de recherche sur le concepteur et le développeur du système : les tâches de conception et de développement constituent l'objet d'intérêt de nos travaux. Le support de ces tâches vise un double objectif : en offrant de meilleurs outils au concepteur et au développeur, l'on cherche à améliorer la qualité du travail de ces derniers ; de façon implicite, un meilleur support de la conception et la réalisation va permettre de mieux prendre en compte les besoins et caractéristiques de l'utilisateur, et de lui offrir des interfaces plus robustes et mieux adaptées. Si les travaux décrits dans ce document s'adressent avant tout au concepteur et au développeur du système interactif, c'est l'utilisateur qui, finalement, en recueillera les fruits.

Chapitre II

Méthodes et modèles

1 Introduction

Comment définit-on le modèle conceptuel d'un système? Comment décrire le dialogue? A quel moment définir une syntaxe? Quel procédé utiliser pour spécifier l'interaction? Comment et quand tester la validité des choix de conception? Comment organiser le code? Comment passer des spécifications au développement?... Autant de questions qui traduisent les difficultés que doivent résoudre les concepteurs et les développeurs lors de la définition d'un système. Face à la complexité du processus de conception d'une application interactive, l'on ressent un besoin de modèles de représentation, et de méthodes.

Les réponses que nous apportent les travaux décrits dans la littérature sont diverses. En raison de la relative "jeunesse" du domaine des interfaces, et peut-être simplement de la complexité du problème considéré, il n'existe pas à l'heure actuelle de solution complète embrassant l'activité de conception et de développement dans son ensemble. Au contraire, c'est un éventail de solutions restreintes que nous renvoie la littérature, chaque solution concernant un aspect donné du problème. Nous tentons dans cette section de présenter une synthèse des travaux relatifs aux méthodes de conception et de développement, et de mettre en lumière les résultats actuels.

Notre propos n'est pas ici de rendre compte de façon exhaustive de l'état de l'art dans le domaine ; le lecteur trouvera dans [Nanard 90], [Coutaz 90], [Marchionini 91], etc. un panorama relativement complet. C'est une approche pratique que nous proposons : partant des besoins réels des développeurs, notre démarche est d'explorer l'ensemble des contributions de la recherche en IHM et de relever les propositions répondant directement à ces besoins, en indiquant l'apport et les insuffisances de ces propositions.

Nous distinguons les méthodes de conception et les méthodes de développement, en accordant attention au problème de la transition entre ces deux ensembles de méthodes. Les méthodes de conception sont variées, et peuvent être étiquetées selon qu'elles concernent la spécification conceptuelle ou externe ; là encore se pose le problème de la liaison entre les deux types de tâches. Les méthodes de développement sont nettement plus uniformes, dans le sens où un consensus semble atteint quant aux principales règles à suivre lors de la réalisation.

Avant de débuter notre étude des modèles et méthodes, nous rappelons quelques notions sur le processus de conception en général, et précisons la signification des termes de "méthode" et de "modèle".

Conception et méthode

La définition d'un système interactif, l'écriture d'un programme informatique, tout comme la production de texte ou la composition musicale sont des *activités de conception*. Si les processus cognitifs réalisant l'activité de conception sont encore mal connus, la recherche expérimentale (cf. [Carroll 85]) nous livre les résultats suivants :

- La conception s'effectue de façon non hiérarchique : ni strictement descendante (par raffinements successifs), ni strictement ascendante (par abstractions successives).
- La conception est un processus profondément évolutif, qui implique le développement de solutions partielles et intérimaires dont la trace peut éventuellement se révéler inexistante dans la solution finale.

- La conception est un processus qui, de façon intrinsèque, implique la découverte de nouveaux objectifs.

Les observations sur le processus de conception d'un système informatique rapportées dans [Guindon 87] corroborent ces résultats. Les activités de conception sont hautement itératives, entrelacées, et pour certaines faiblement ordonnées. La stratégie d'analyse descendante n'est pas toujours appliquée ; un concepteur peut travailler en même temps sur des niveaux d'abstraction différents, produisant des solutions partielles en fonction notamment de la reconnaissance de problèmes familiers dans l'environnement de conception. Les auteurs qualifient ce type de comportement de "serendipitous" - grossièrement assimilable au terme "opportuniste", insistant sur le fait qu'une solution peut être le fruit de la détection d'aspects familiers dans le problème étudié, sans avoir au préalable décomposé ce problème en sous-problèmes.

Dans ces conditions, que peut-être l'objectif d'une *méthode de conception*? Il s'agit tout d'abord de comprendre les points faibles du concepteur - principalement, la limitation de ses ressources cognitives, et d'aider le concepteur en conséquence. D'une façon générale, une méthode fournit d'une part des points d'ancrage de l'activité de conception, et d'autre part un langage d'expression pour cette activité.

Une méthode se définit de façon stricte comme un guide conçu pour diriger et assister une activité. Le contenu de ce guide prend la forme de conseils et de recommandations, de "recettes" décrivant pas-à-pas la démarche à suivre, et éventuellement de techniques de représentation spécifiques (par exemple, des diagrammes). On le conçoit, une méthode de conception ne pourra jamais être appliquée à la lettre, car trop statique et trop rigide. L'intérêt d'une méthode réside dans la proposition d'un cadre à l'intérieur duquel s'effectuera la conception ; ce cadre fournit des *points d'ancrage* qui seront utilisés comme points de départ ou points de synchronisation de l'activité de conception.

Une méthode a pour fondement une représentation de l'objet étudié (dans notre cas, le système interactif), représentation référence à laquelle sont rattachés les éléments du guide ; cette représentation résulte d'une modélisation de l'objet étudié. Ce modèle de l'objet fournit en quelque sorte un *langage d'expression* pour l'activité de conception : un moyen d'expression permettant l'externalisation des résultats de la conception et, pour certaines méthodes, des choix effectués lors de la conception.

Enfin, une méthode offre un outil de communication non négligeable : communication entre les concepteurs, entre les concepteurs et le client auquel est destiné le système, et éventuellement entre les concepteurs et l'utilisateur du système.

"Méthode" et "modèle"

Une méthode suppose l'existence d'un modèle de l'objet.

La réciproque n'est pas vraie : la définition d'un modèle ne s'accompagne pas forcément de celle d'une méthode. Pour un même modèle, plusieurs méthodes ou modes d'emploi peuvent être définis. On peut cependant considérer que la définition d'un modèle recèle de façon implicite celle d'une méthode.

Dans la suite de ce chapitre, l'on présentera sur un même plan méthodes et simples modèles.

2 Méthodes de conception

Question : On vous confie un cahier des charges répertoriant les besoins et souhaits d'un client ; vous êtes chargé de la conception de l'application répondant aux contraintes de ce cahier des charges. Comment procédez-vous?

En milieu industriel, la conception d'un système passe par l'application d'une méthode générale réglant les rapports entre le développeur et le client, les différentes étapes à mettre en œuvre, ainsi que le format et contenu des spécifications et études intermédiaires, jusqu'à l'obtention du système final. De nombreuses méthodes d'analyse et de conception existent sur le marché, parmi lesquelles des méthodes normalisées et des méthodes "maison", définies localement à une entreprise. Une caractéristique commune à la plupart de ces méthodes générales est leur inadaptation à prendre en compte l'interaction, et plus précisément l'utilisateur : ce sont avant tout des méthodes qui permettent l'analyse et la définition du fonctionnement d'un système informatique. Certaines de ces méthodes, telle MERISE [Tardieu 83], ne fournissent aucun support à la définition précise du dialogue homme-ordinateur, limitant l'effort d'analyse aux données et aux traitements du système. D'autres méthodes, telles AXIAL [Pellaumail 86], qui autorisent la spécification détaillée d'un dialogue, ne prennent pas en considération le point de vue de l'utilisateur, comme le montre M-F Barthet dans [Barthet 88].

Le besoin de modèles et de méthodes de conception des interfaces a suscité de nombreux travaux de recherche, dont le résultat est un ensemble de propositions diverses que nous allons chercher à démêler dans cette section. Comme l'illustre la Figure II-1, nous distinguons les travaux portant sur la spécification de l'interface, les propositions concernant l'évaluation de l'interface, et les propositions axées sur la représentation des choix de conception, raison d'être de l'interface[1].

1. Le partage entre spécification, évaluation et raison d'être de l'interface n'a d'autre intérêt que de classer les méthodes relativement aux besoins du processus de conception. Toute classification est simplificatrice ; il est clair que bon nombre de méthodes souffriront de se voir enfermées dans un des compartiments plutôt que dans un autre. Notre souci dans cette classification est de mettre l'accent sur le principal intérêt qu'offre, à notre sens, chacune des méthodes.

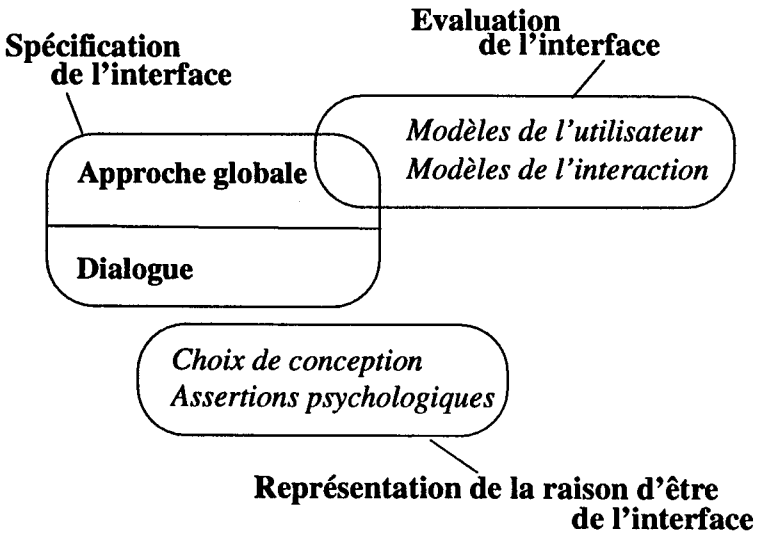


Figure II-1 *Classification des méthodes de conception.*

En ce qui concerne la spécification de l'interface, deux grands types d'approches sont envisagés : l'approche globale et l'approche "dialogue". Dans le domaine des méthodes générales d'analyse et de conception, nous relevons quelques travaux caractérisés par une *approche globale* de la conception, et aboutissant à des méthodes plus ou moins complètes. L'intérêt de ces travaux est d'offrir un cadre aux tâches d'analyse et de spécification conceptuelle d'un système, tout en offrant un support à la tâche de spécification externe. Nous verrons cependant que cette approche globale laisse de nombreuses lacunes dans le support de la conception d'un système, tout particulièrement en ce qui concerne la spécification du dialogue. Par opposition, un ensemble de travaux adoptent une *approche "dialogue"*, centrée sur l'expression de l'interaction, comblant ainsi les lacunes des méthodes globales.

2.1 Spécification de l'interface : approche globale

Le terme "approche globale" regroupe l'ensemble des travaux caractérisé par une vision globale, ou macroscopique de l'interface d'un système : c'est l'ensemble d'un système que l'on perçoit, et que l'on cherche à concevoir et à définir. Les solutions apportées par ces travaux sont des modèles et méthodes qui traitent à la fois les tâches d'analyse, de spécification conceptuelle, et de spécification externe d'un système interactif.

Une approche globale de l'activité de conception nécessite une modélisation de l'ensemble d'un système interactif. Les travaux de cette nature ont comme caractéristique commune d'adopter une *perspective "en couches"* sur un système : l'identification de plusieurs niveaux d'abstraction représentant un système favorise une méthode de conception descendante, du plus abstrait vers le plus concret, approche bien connue du génie logiciel en général.

Foley et al. dans [Foley 84] sont les premiers à plaquer une *vision linguistique* sur la modélisation d'un système interactif. L'idée est simple : l'interaction entre l'homme et l'ordinateur est un dialogue qui s'effectue dans un langage. Le principe linguistique distingue quatre niveaux de représentation d'un système interactif :

- le *niveau conceptuel*, qui s'attache à la façon dont la tâche est représentée dans le système.
- le *niveau sémantique*, qui décrit les entités et les fonctions manipulées dans le système.
- le *niveau syntaxique*, qui définit la structure des échanges entre le système et l'utilisateur : les séquences d'entrées valides, et les séquences de sortie produites par le système.
- le *niveau lexical*, qui définit l'apparence concrète des éléments des séquences d'entrée et de sortie : vocabulaire employé, images graphiques, techniques d'interaction, etc.

Les niveaux conceptuel et sémantique correspondent au résultat des étapes d'analyse et de spécification conceptuelle ; les niveaux syntaxique et lexical correspondent au contenu de la spécification externe.

La vision linguistique offre un cadre de pensée à la base de nombreux travaux dans le domaine de l'IHM - nous aurons par la suite l'occasion de décrire les implications et limitations de cette vision sur les méthodes de développement. Dans le contexte des méthodes de conception, cette vision est sous-jacente aux travaux les plus significatifs.

Nous choisissons de présenter deux grandes familles de méthodes globales : d'une part CLG, et les travaux de M. Green, issus d'une démarche que nous qualifions d'analytique, et d'autre part les travaux de M.-F. Barthet, résultant d'une démarche avant tout ergonomique.

Nous présentons les principes de CLG, qui prolonge et approfondit la vision linguistique pour aboutir à un modèle de représentation d'un système axé sur l'utilisateur, doublé d'une véritable méthode de conception. Les travaux de M. Green reprennent, en les formalisant, les principes de CLG.

Les travaux de M.-F. Barthet, dont la présentation complète cette section, se démarquent de ceux de Moran et de Green en proposant une démarche globale qui dépasse le modèle linguistique, directement inspirée des principes ergonomiques.

2.1.1 CLG

The Command Language Grammar (CLG) [Moran 81] est le résultat des travaux de Moran sur la modélisation des systèmes à langages de commande. L'objectif de Moran est de capter l'organisation conceptuelle et structurale sous-jacente à ces systèmes et à leurs interfaces, en décrivant le modèle conceptuel que forme l'utilisateur (idéal) de ce système. La modélisation que définit CLG couvre à la fois le modèle conceptuel et l'image du système, respectant la vision linguistique ci-dessus décrite.

CLG distingue trois grands composants dans un système : le composant conceptuel, le composant de communication, et le composant physique.

Composant conceptuel :

Le composant conceptuel comprend tout d'abord un *niveau "tâche"*, où sont définis les tâches et concepts du domaine considéré, en explicitant l'arbre de décomposition hiérarchique des tâches ; il s'agit là d'une classique "analyse de la tâche". Le second niveau est un *niveau sémantique* ; les entités et opérations conceptuelles du système sont identifiées ; la mise en relation de ces entités et opérations avec les éléments du niveau "tâche" est explicitée : le

concept de méthode est introduit pour décrire la procédure de réalisation d'une tâche à partir des entités et opérations sémantiques. Le niveau sémantique définit ainsi les éléments d'un modèle conceptuel du système.

Composant de communication :

Le composant de communication met en jeu les niveaux syntaxe et interaction. Au *niveau syntaxique*, la structure du langage de commandes est décrite au moyen des notions de commandes et de contextes. L'effet d'une commande est explicité dans les termes du niveau sémantique ; les procédures de réalisation des tâches sont récrites en termes syntaxiques. Le *niveau interaction* traduit les éléments syntaxiques en une séquence d'actions physiques effectuées par le système et par l'utilisateur. On trouve à ce niveau les notions de messages d'invite, de spécification d'une commande, d'actions de terminaison d'une spécification, de retour ("feedback"), etc. Le niveau interaction de CLG recouvre ainsi ce que plus classiquement l'on nommerait règles syntaxiques élémentaires et lexique.

Composant physique :

Le composant physique comprend le niveau de la *disposition spatiale* et le niveau des *périphériques*. Il s'agit dans ce composant de décrire l'organisation et les caractéristiques des périphériques d'entrée/sortie, et de l'affichage sur l'écran. On se situe là à un niveau d'abstraction au-delà du niveau lexical classique.

La puissance de CLG est de constituer à la fois un modèle de description de la vision qu'a l'utilisateur du système, et un support pour l'activité de conception.

CLG en tant que représentation du modèle de l'utilisateur ouvre d'intéressantes perspectives sur le plan de l'évaluation d'un système. Dans un premier temps, l'on peut considérer la vérification de la cohérence syntaxique et conceptuelle des procédures de réalisation d'une tâche. Dans un second temps, c'est une évaluation de la complexité du modèle décrit qui peut être envisagée. On peut déjà entrevoir ici les prémices des travaux de Polson & Kieras [Polson 85].

Comme support de l'activité de conception, CLG constitue un cadre de référence pour les décisions de conception plutôt qu'une véritable méthode. CLG favorise une démarche descendante, par affinages successifs, mais ne détermine pas l'ordre dans lequel doivent être définis chacun des éléments du système. Une des principales limites de CLG est le manque de formalisation.

Les travaux de Moran que nous venons d'évoquer ont largement influencé la communauté IHM, de façon directe ou indirecte. Nous rapportons dans la section suivante une approche de l'activité de conception qui reprend et prolonge les enseignements tirés de CLG.

2.1.2 Dans la lignée de CLG : travaux de M. Green

Les travaux présentés par M. Green dans [Green 85] ont l'originalité d'aborder de front le problème de la conception des interfaces utilisateur graphiques. Avec la contribution de M.-F. Barthet, que nous évoquerons dans la suite de ce chapitre, c'est à notre connaissance la seule proposition d'une réelle *méthode* de conception en IHM.

La méthode proposée par M. Green comporte sept étapes : l'analyse des tâches, la conception sémantique, la conception détaillée, l'évaluation, l'intégration, l'intervention de l'utilisateur, et la documentation.

Etapes de conception :

Seules les trois premières étapes relèvent de la conception pure : l'analyse des tâches correspond au niveau "tâche" de CLG ; la conception sémantique au niveau sémantique ; la conception détaillée recouvre les aspects syntaxiques et lexicaux. Il est à noter que tandis que CLG ne s'intéresse qu'aux interfaces à langage de commande, les travaux de M. Green ont pour objectif les interfaces graphiques.

Etape d'évaluation de la conception :

L'étape d'évaluation est double. D'une part, il s'agit de vérifier la compatibilité entre les spécifications issues des trois premières étapes. D'autre part, il s'agit d'effectuer une évaluation des performances.

La vérification de la *compatibilité* consiste à expliciter les procédures de mise en correspondance entre :

1. les tâches et les opérateurs du niveau sémantique.
2. les opérateurs sémantiques et les opérateurs issus de la conception détaillée.

La vérification de la compatibilité revient ainsi à définir ce que Moran appelle les "méthodes" dans CLG.

L'évaluation des *performances* s'effectue à partir des procédures de mise en correspondance évoquées précédemment. A l'aide de formules caractérisant les performances élémentaires des périphériques utilisés (par exemple, le temps nécessaire au déplacement moyen de la souris lors de la sélection d'un élément dans un menu), une estimation du temps d'exécution et du taux d'erreur de manipulation peut être calculée pour chacun des opérateurs sémantiques, puis des opérateurs du niveau tâche. Les résultats obtenus sont comparés aux critères de performance établis lors de l'étape de définition des tâches. En cas de désaccord, un processus de calcul inverse permet de détecter le ou les opérateurs syntaxiques responsables des mauvaises performances, et de remédier à la situation en modifiant les aspects de la conception en cause. Dans le cadre d'une application d'édition interactive d'un graphe par exemple, M. Green décrit un processus d'évaluation des taux d'erreur de manipulation ; il est détecté que l'opérateur de sélection d'un nœud du graphe est à la source d'un fort taux d'erreur ; la solution adoptée consiste à augmenter la taille de l'affichage.

Etapes terminales :

L'étape d'intégration est une étape de maintenance des descriptions de la conception. Notamment, c'est à cette étape que la modification de la conception en fonction des résultats de l'évaluation est effectuée.

L'étape d'intervention de l'utilisateur repose sur la construction d'un prototype du système décrit lors des étapes précédentes. Ce prototype est confronté aux utilisateurs potentiels du système, de façon à recueillir les réactions de ces derniers.

L'étape de documentation produit le document de conception final à partir des spécifications issues des étapes précédentes.

Langages de spécification :

Contrairement à CLG, qui ne formalise pas complètement les résultats de chaque niveau de la conception, M. Green propose une famille de trois langages de description formelle : UML, GUSL, et ML.

Le User Modelling Language (UML) est utilisé lors des deux étapes d'analyse des tâches et de conception sémantique. Ce langage permet la déclaration d'objets et d'opérateurs, d'invariants, de pré- et de post-conditions pour les opérateurs.

Le Graphical User Interface Specification Language (GUSL) est associé à l'étape de conception détaillée ; le Mapping Language (ML) est utilisé lors de l'étape d'évaluation de la conception. Ces deux langages sont très proches de langages de programmation.

2.1.3 La méthode DIANE

DIANE est une méthode de conception des applications interactives proposée par M.-F. Barthet dans [Barthet 88]. DIANE s'inscrit dans le domaine de l'informatique des organisations ; cette méthode est issue de travaux spécifiquement dédiés à la bureautique, dans le cadre de l'automatisation du poste de travail. Cette remarque est importante : l'étude de l'existant, et les relations entre l'existant et le système à construire entrent pour une large part dans les propositions de cette méthode.

La méthode DIANE a pour caractéristique première de promouvoir une approche ergonomique de la conception des interfaces : les notions de tâche et de procédure de réalisation d'une tâche sont au cœur de cette méthode. L'approche de DIANE est symptomatique des travaux de recherche effectués par les ergonomes, tels qu'ils apparaissent dans [Scapin 88] par exemple. L'intérêt de DIANE est d'aller au-delà des classiques listes de recommandations et autres règles ergonomiques : DIANE propose une véritable méthode de conception, cristallisant un ensemble de recommandations ergonomiques autour d'un modèle de représentation d'un système interactif.

DIANE aborde séparément la spécification conceptuelle et la spécification externe. Dans les deux cas, un modèle de représentation est proposé, muni d'un ensemble de recommandations.

Conception de la Représentation Conceptuelle

DIANE exprime la représentation conceptuelle d'un système à travers la notion de tâche de l'utilisateur, et de procédure d'utilisation de ce système : description de l'enchaînement des opérations du système permettant la réalisation d'une tâche. Une même tâche donne lieu à la définition de trois ensembles de procédures différentes : la procédure minimale, une ou plusieurs procédures prévues, une ou plusieurs procédures effectives.

La *procédure minimale* est l'ensemble des opérations et enchaînements minimaux pour que le but de la tâche puisse être considéré comme atteint. Elle est définie en précisant la latitude décisionnelle associée au poste de travail : opérations obligatoires ou facultatives, relations de précédence permanente entre les opérations, marge de manœuvre de l'utilisateur, etc.

La *procédure prévue* est la description de l'exécution d'une tâche telle qu'elle est prévue dans son déroulement normal ou usuel. On peut définir une procédure prévue à partir de la procédure minimale, en ajoutant à cette dernière des précédences indicatives et des

déclenchements automatiques qui rendront l'exécution de la procédure prévue plus commode et plus rapide.

Une *procédure effective* est un modèle observé de l'activité : une procédure de réalisation d'une tâche adoptée par les utilisateurs suivant leurs réponses aux aléas, leur propre organisation de leur travail, ou bien les caractéristiques générales de chaque utilisateur (novices, expérimentés, occasionnels, ...). La notion de procédure effective n'intervient au cours de la spécification conceptuelle que dans le contexte d'une automatisation de procédures effectives observées dans l'existant.

Les procédures définissent les aspects variables d'un système. DIANE isole de plus un ensemble de *paramètres constants*, qui désignent les fonctions générales offertes à l'utilisateur indépendamment de toute application :

- aide au travail (interrompre, quitter, annuler, reprendre, etc.)
- aide à l'apprentissage (guidage)
- possibilités d'évolution (ajout de procédures effectives).

C'est un formalisme graphique à base de réseaux de Petri qui est utilisé pour décrire les événements, règles d'émission, et synchronisations lors de la spécification d'une procédure.

Conception de la Représentation Externe

La méthode DIANE analyse les différents éléments intervenant dans la définition de la représentation externe d'un système, et propose une démarche intégrant simultanément ces éléments. Cette démarche repose de façon classique sur la traduction des paramètres conceptuels, l'application de recommandations ergonomiques, et la prise en compte des contraintes matérielles.

Dans un premier temps, il s'agit de traduire les aspects conceptuels en leur donnant une représentation externe : commandes, noms, formes de présentation, mode de désignation, messages d'erreur, hiérarchies de commandes, menus, etc. sont à définir.

La traduction des éléments conceptuels doit s'effectuer en tenant compte d'une part des critères ergonomiques, et d'autre part des possibilités techniques.

Parmi les critères ergonomiques, citons le critère d'homogénéité, qui influe à la fois sur la syntaxe des entrées et sur la présentation des sorties du système. Il est d'autre part nécessaire de prendre en compte certains aspects de l'existant, tels le vocabulaire des spécialistes du domaine de l'application, la présentation des formulaires existants, etc.

Sur le plan matériel comme sur le plan logiciel, de fortes contraintes influent directement sur les choix de la représentation externe.

2.1.4 Discussion

Les travaux de Moran, Green et Barthet sont représentatifs des principaux courants de la recherche concernant les méthodes de conception globales. La confrontation de ces différents travaux est intéressante car trois grands types d'approches sont en jeu, résumant les différents domaines d'inspiration de l'IHM en général : une approche "psychologique" avec CLG, une approche ancrée dans le génie logiciel avec les travaux de Green, et une approche ergonomique avec DIANE.

Les mêmes éléments apparaissent dans chacune des méthodes, sous une forme différente, munis d'une importance différente.

L'importance de l'étape d'analyse de la tâche est unanimement reconnue, bien que diversement exprimée. L'utilisateur, la tâche, la procédure opérative, etc. sont des notions qui constituent le fondement de DIANE, qui dans CLG sont utilisées comme référence tout au long de la spécification pour vérifier la validité des composants spécifiés, et qui pour M. Green n'apparaissent que lors de l'étape d'évaluation de la conception.

Les notions de procédure minimale, prévue, et effective n'apparaissent que dans les travaux de M-F Barthet, et semblent directement issues du contexte de l'informatique des organisations qui caractérise DIANE. La notion d'alternative dans la réalisation d'une tâche apparaît dans CLG. Il est intéressant de noter que cette notion intervient dans d'autres travaux du domaine de l'ergonomie. Notamment, MAD [Pierret-Golbreich 89] est un outil d'acquisition et de représentation analytique des tâches qui admet plusieurs procédures de réalisation d'une même tâche.

> *Question : Vous avez analysé les tâches que met en jeu votre système ; vous avez défini les concepts et fonctions offerts dans votre système pour réaliser ces tâches. Il vous faut maintenant préciser les éléments du dialogue et de la présentation de votre système. Comment procédez-vous ?*

Si les différentes méthodes présentées offrent un support relativement satisfaisant aux étapes d'analyse et de spécification conceptuelle d'un système, il en est autrement de la spécification externe, et notamment de la spécification du dialogue. Les propositions de CLG se cantonnent aux langages de commande, et sont difficilement applicables aux interfaces utilisateur actuelles. DIANE n'offre que des recommandations pour guider l'étape de spécification externe, mais offre une extension intéressante concernant la modélisation de la structure de contrôle général de l'application, au travers des réseaux de Petri. Green propose un formalisme de spécification complet, tenant de sa méthode de conception. La critique de ce formalisme est relativement aisée cependant : d'une part les descriptions obtenues sont incomplètes en ce qui concerne notamment la dynamique du graphisme, et d'autre part le volume de ces descriptions successives peut atteindre, voire dépasser celui du code de réalisation des interfaces spécifiées.

En conclusion, les méthodes générales que nous fournit la recherche offrent un cadre aux étapes d'analyse des tâches, et de spécification conceptuelle d'un système. En ce qui concerne la spécification du dialogue que va mettre en œuvre le système, ces méthodes sont inégales ; il est nécessaire de disposer de méthodes spécialisées issues d'une approche non plus globale mais centrée sur le dialogue.

2.2 Spécification de l'interface : approche "dialogue"

Le terme "approche dialogue" désigne l'ensemble des travaux dont la portée se limite à la spécification externe d'une interface. Ces travaux sont symptomatiques d'une vision microscopique sur le système, en opposition à la vision globale exposée dans la section précédente.

L'approche "dialogue" est celle de la plupart des travaux de recherche effectués dans le domaine de l'aide à la spécification des interfaces. Cette approche a pour avantage de considérer un sujet relativement bien cerné, moins flou et moins complexe qu'une approche

globale. Une autre raison du nombre des contributions sur le dialogue est que la modélisation du dialogue, et la proposition de formalismes de spécification sont le passage obligé vers la construction d'une classe d'outils d'aide à la réalisation d'un système - nous aurons par la suite l'occasion de revenir sur les travaux de modélisation du dialogue, lorsque nous aborderons les outils de développement.

Les travaux portant sur la spécification du dialogue ont donné lieu à la définition de modèles de représentation du dialogue plutôt qu'à des méthodes de description du dialogue. On peut distinguer deux grands types de modèles : le modèle conversationnel et le modèle à événements. Nous relevons quelques travaux significatifs de ces deux classes de modèles.

2.2.1 Modèle conversationnel

L'interaction entre l'utilisateur et le système peut être modélisée comme une conversation s'effectuant dans un langage particulier. Le dialogue est alors perçu comme une séquence d'interventions mettant en jeu tour à tour l'homme et le système ; chaque intervention produit un ensemble d'expressions dont la structure, l'ordre et l'occurrence sont prédéterminés, régis par des règles qui constituent un véritable langage d'interaction.

La définition du langage d'interaction s'effectue bien entendu dans le cadre général de la perspective linguistique décrite par Foley, et présentée en 2.1 : l'on distinguera les niveaux sémantique, syntaxique et lexical du dialogue. Le modèle conversationnel donne lieu à deux grands types de formalisations, qui diffèrent selon que l'on cherche à exprimer la forme du langage ou bien l'état du dialogue.

Modélisation de la forme du langage

La forme d'un langage définit les séquences d'expression autorisées dans ce langage. Les *grammaires* hors-contexte, de par leur nature linguistique, constituent l'outil de base pour la description d'un langage d'interaction. Elles donnent lieu à la définition de règles BNF exprimant la syntaxe et le lexique du langage d'interaction. Les grammaires permettent difficilement l'expression des aspects sémantiques du langage : elles décrivent la forme des expressions du langage, et non pas l'état du dialogue qui en découle.

Nombreux sont les travaux utilisant une grammaire pour la spécification du dialogue. A titre d'exemple, CLG, que l'on a présenté précédemment comme une approche globale de la spécification d'une interface, utilise une grammaire pour la description du dialogue.

Les principales limites des grammaires concernent l'expression des termes lexicaux, et le déséquilibre en faveur du langage d'entrée de l'utilisateur, au détriment de l'expression des sorties du système.

Si les grammaires sont parfaitement adaptées à la spécification d'un langage de commandes, la modélisation d'un dialogue à base graphique est plus difficile : le lexique ne contient alors pas simplement des chaînes de caractères, mais aussi par exemple des actions élémentaires utilisant la souris. Les travaux de Olsen relatifs à SYNGRAPH [Olsen 83] abordent ce problème en définissant une grammaire dont le lexique est enrichi pour prendre en compte les facilités des environnements graphiques.

Si, à l'aide d'un lexique enrichi, une grammaire permet de façon naturelle l'expression du langage d'entrée de l'utilisateur, le second interlocuteur (c'est-à-dire le système) est la plupart du temps mal représenté. Ici encore, le langage de commandes apparaît comme le mieux

adapté aux possibilités de la spécification grammaticale. Les grammaires multiparties [Shneiderman 82] dénotent un effort pour mettre sur le même plan les interventions de l'utilisateur et du système lors du déroulement du dialogue.

Modélisation de l'état du dialogue

La modélisation de l'état du dialogue met en jeu des formalismes de type réseau de transitions ([Green 86]) : le dialogue est décrit sous la forme d'un graphe dont les nœuds dénotent les états du dialogue, et les arcs les transitions possibles entre les états. Le réseau de transitions représente la structure du dialogue ; la sémantique ainsi que la forme du dialogue vont être spécifiées en "étiquetant" les éléments de cette structure. Les aspects sémantiques s'expriment au travers d'étiquettes associées aux états, décrivant les traitements sémantiques effectués lorsque ces états sont atteints (sous la forme d'appels de procédures). Le langage d'entrée est spécifié en attachant des termes du lexique aux arcs de transition ; la syntaxe est alors déterminée par les séquences possibles de transitions. Le langage de sortie est spécifié en associant des séquences de sortie aux nœuds du graphe.

Divers types de réseaux de transitions ont été développés, dans le but de pallier les inconvénients des diagrammes de transitions élémentaires, et d'augmenter le pouvoir d'expression de ce type de formalisme. Notons les réseaux de transitions récursifs, qui permettent de limiter le volume des descriptions en autorisant la définition et l'utilisation de réseaux de sous-dialogue, et les réseaux de transitions augmentés, qui comportent registres et fonctions booléennes, et permettent la modélisation de contextes.

Conclusion sur les modèles conversationnels

L'intérêt des modèles conversationnels est d'offrir une passerelle vers le domaine de la linguistique ; cette passerelle nous apporte les formalismes et outils développés dans ce domaine, constituant ainsi une base d'outils expérimentés pour la spécification et la mise en œuvre des dialogues. Pour cette raison, les modèles conversationnels sont le résultat des premières tentatives de représentation des interfaces. L'expérience montre que si ces modèles sont bien adaptés à la description des interfaces tels les langages de commandes, ils présentent d'importantes limitations qui restreignent leur utilisation dans un contexte plus général.

Que l'on choisisse de représenter la forme ou bien les états d'un dialogue, les modèles conversationnels imposent de décrire explicitement toutes les séquences autorisées par le langage. Cette exhaustivité a pour conséquence l'importance du volume des spécifications du langage. Dans le cas des interfaces peu dirigistes, cette *contrainte à l'exhaustivité* peut conduire au rejet du modèle. La spécification d'une interface à manipulation directe par exemple nécessite de décrire toutes les séquences d'actions possibles de l'utilisateur, ce qui de façon pratique résulte en une explosion combinatoire. Un second exemple est celui des interfaces autorisant la tenue de plusieurs dialogues en parallèle (dialogues "multifils") ; la spécification de tels dialogues nécessiterait de décrire toutes les séquences d'interventions de l'utilisateur d'un sous-dialogue à un autre ; ici encore, l'échec des modèles conversationnels est patent.

2.2.2 Modèle à événements

Les modèles à événements se démarquent entièrement des modèles linguistiques ; leur objectif est de modéliser des comportements non pas séquentiels mais asynchrones, palliant ainsi la contrainte à l'exhaustivité imposée par les modèles conversationnels.

Un *événement* se définit comme l'occurrence d'un fait d'un type donné ; un événement possède une identification, ainsi qu'une liste d'attributs le caractérisant. La notion d'événement apparaît très tôt dans la réalisation des serveurs graphiques, à l'origine du développement des modèles à événements pour représenter les interfaces. Les événements de base sont ceux générés par les périphériques d'entrée : frappe de touches, pression de boutons de la souris, déplacements élémentaires de la souris, etc. Au-delà des événements décrivant les actions physiques de l'utilisateur, des événements peuvent être produits par l'interface elle-même, ou bien par l'application. Divers niveaux d'événements peuvent être envisagés : événements de sortie, tels l'affichage d'une fenêtre, d'un texte, la mise en surbrillance d'une zone, etc., ou bien événements de l'application, tels la mise à jour d'une donnée, la détection d'une situation anormale, etc.

Lorsqu'il est généré, un événement est transmis à un ou plusieurs *gestionnaires d'événements*, qui ont pour charge de l'examiner et, le cas échéant, de réagir en provoquant l'exécution d'actions en conséquence ; ces actions peuvent exécuter des fonctions de l'application, modifier le contexte de l'interaction, modifier l'image de l'interface, mais également générer de nouveaux événements, et créer ou détruire des gestionnaires d'événements.

La spécification d'un dialogue selon le modèle à événements suppose la définition des différents types d'événements pouvant être produits, ainsi que la définition d'un ensemble de gestionnaires de ces événements. Le comportement d'un gestionnaire d'événements se décrit en spécifiant les types d'événements traités par le gestionnaire, ainsi que les procédures de traitement associées.

On trouvera dans [Green 86] une définition formelle du modèle à événements.

Différents formalismes ont été proposés pour décrire un dialogue selon le modèle à événements ; ils reposent sur les résultats issus de la recherche en programmation parallèle. On peut relever ERL, Event Response Language [Hill 87], fondé sur des règles de production, Squeak [Cardelli 85], ou ALGAE [Flecchia 87], une extension du langage Pascal.

L'intérêt principal du modèle à événement est la souplesse d'expression des dialogues multifils (cf. section 4, Chapitre I) : chaque fil de dialogue est spécifié isolément au travers d'un ou de plusieurs gestionnaires d'événements, le dialogue multifil résultant de la conjonction de ces gestionnaires d'événements. Les séquences d'actions autorisées de l'utilisateur sont exprimées implicitement par la combinaison des événements attendus par les gestionnaires à un moment donné.

Notons que la division lexique/syntaxe/sémantique reste opportune dans le contexte d'un modèle à événements, même si elle n'est pas toujours explicite. Les termes du lexique apparaissent au niveau des événements d'entrée ; la syntaxe transparaît dans les séquences d'événements que peuvent traiter les gestionnaires ; la sémantique s'exprime au niveau des procédures de traitement mises en œuvre par les gestionnaires.

2.2.3 Conclusion sur les modèles de représentation du dialogue

Nous avons présenté deux grandes familles de modèles de représentation du dialogue : les modèles conversationnels, fondés sur une perspective linguistique, et les modèles à événements, dont les concepts sont issus de la programmation parallèle.

Comme le rappelle J. Nanard dans [Nanard 90], les modèles de dialogue sont en général difficilement dissociables des formalismes de description proposés. C'est le cas tout particulièrement pour les grammaires et réseaux de transitions, formalismes mais aussi supports intrinsèques du modèle conversationnel.

L'intérêt des grammaires et automates est que ces modèles de représentation possèdent des fondements théoriques permettant notamment le calcul de preuves. Il est clair cependant que le modèle à événements offre une plus grande *puissance d'expression* que le modèle conversationnel. M. Green en fait la démonstration dans [Green 86], en comparant les trois formalismes de description du dialogue que sont les grammaires hors contexte, les réseaux de transitions et les modèles à événements. Il remarque que certaines fonctions d'une interface sont difficiles, voire impossibles à représenter avec les deux premiers formalismes - notamment, toute forme d'interruption ; il exprime la puissance d'expression du modèle à événements en décrivant des algorithmes de conversion de spécifications sous forme de grammaire ou de réseau de transitions vers ce modèle à événements.

Les interfaces actuelles, bien souvent multifils, ne peuvent être modélisées comme une conversation séquentielle. C'est un modèle à événements qui est le plus souvent retenu. Nous verrons au chapitre III comment les outils prennent en compte cette réalité.

Au-delà de l'expression des séquences du dialogue, les modèles de dialogue présentent des limitations quant à la *représentation de l'image* de l'interface, et de l'évolution de cette dernière. Il est clair que la spécification de l'image passe par des schémas et croquis informels, voire par la mise au point de prototypes d'interface, comme nous le verrons au Chapitre III. La dynamique de l'image, intimement liée à celle du dialogue, est difficilement formalisable. Notamment, la description de l'information de retour ("feedback") lors d'une action de l'utilisateur est malaisée, dépendante du domaine pour une large part.

Par ailleurs, les modèles de représentation du dialogue ont généralement le défaut de ne pas expliciter la relation entre les séquences de dialogue et *les tâches de l'utilisateur* : c'est une représentation de la structure générale du fonctionnement du dialogue qui est donnée. Ceci a pour conséquence de mettre en péril l'adéquation du système à la réalisation des tâches de l'utilisateur. Des travaux tels ceux rapportés dans [Siochi 89] visent à remédier à ce péril, en proposant des notations explicitement axées sur les tâches.

2.3 Evaluation de l'interface

Nous avons, lors de la présentation des méthodes de représentation globales, donné quelques aperçus des méthodes d'évaluation en cours dans la communauté IHM. Le lecteur interessé trouvera dans [Senach 90] une présentation des différentes approches de l'évaluation ergonomique des interfaces. Deux grands types d'évaluation coexistent actuellement : l'expérimentation et l'analyse des interfaces, la mise en œuvre de l'un n'excluant pas celle de l'autre.

En ce qui concerne l'évaluation par expérimentation, la littérature regorge de comptes rendus d'expériences confirmant telle ou telle hypothèse. Le lecteur intéressé trouvera par exemple dans [Roberts 83] une méthodologie d'évaluation expérimentale pour les éditeurs de texte, revue et corrigée dans [Borenstein 85].

Nous nous intéressons ici à l'évaluation des interfaces par analyse (évaluation prédictive). La littérature nous renvoie un ensemble de méthodes ; les différences portent sur le contenu des modèles de représentation utilisés, et de façon générale sur les objectifs et possibilités de

ces méthodes. Nous percevons deux grandes approches, selon que l'analyse est axée sur un modèle de l'utilisateur ou bien sur un modèle de l'interaction.

La première approche met en jeu un modèle de représentation de l'interface, mais aussi de la tâche, et de l'utilisateur. L'évaluation s'effectue par l'analyse des structures de tâches, et, selon les méthodes, par la simulation de l'utilisation de l'interface, et l'analyse des comportements engendrés.

La seconde approche repose sur un langage de spécification formelle d'une interface susceptible de permettre la mise en œuvre d'analyses et de calculs en vue de l'évaluation de propriétés de l'interaction.

2.3.1 Modèles de l'utilisateur

Les méthodes d'évaluation les plus significatives sont issues (de façon directe ou indirecte) du *modèle GOMS* [Card 83] conçu pour modéliser le comportement humain en situation de réalisation d'une tâche à l'aide d'un système. GOMS est un modèle générique qui permet l'évaluation prédictive des performances d'utilisation d'un système. GOMS est à la source de toute une famille de modèles issus d'une application particulière de GOMS, tel Keystroke, ou bien résultant d'une extension de GOMS en vue de dépasser les simples évaluations quantitatives : la "Théorie de la Complexité Cognitive" de Kieras & Polson.

En dehors de la famille des modèles GOMS, quelques contributions commencent à apparaître, encore peu nombreuses. Nous relevons le "Modèle Utilisateur Programmable" de R. Young et al.

Le modèle GOMS et ses applications

GOMS (Goal, Operator, Method, Selection) représente l'activité d'un individu en termes de Buts, d'Opérateurs, de Méthodes et de règles de Sélection.

Un But décrit un état recherché ; les Buts sont organisés selon une structure hiérarchisée ; la notion de But peut être assimilée à la notion de tâche présentée en 3, Chapitre I.

Un Opérateur est une action élémentaire dont l'exécution provoque un changement d'état (état mental de l'utilisateur et/ou état de l'environnement) ; selon le niveau d'abstraction considéré, un Opérateur est une commande du système, un déplacement de la main de l'utilisateur, une action de perception sensorielle, etc.

Une Méthode décrit le procédé qui permet d'atteindre un but, sous la forme d'une suite conditionnelle de sous-Buts et d'Opérateurs ; les Méthodes figent un savoir-faire sur l'utilisation d'un système pour réaliser une tâche : il ne s'agit pas de plans d'actions construits dynamiquement.

Une règle de Sélection exprime le choix d'une Méthode en cas de conflit : selon les caractéristiques de la situation courante, telle Méthode sera choisie pour réaliser tel But.

GOMS est un modèle de représentation qui fournit une description mesurable du comportement de l'utilisateur, et permet ainsi une évaluation quantitative des performances d'utilisation d'un système. Connaissant le temps d'exécution d'un Opérateur, il est possible de déterminer le temps nécessaire à la réalisation d'une tâche. Il ne s'agit cependant que d'évaluations quantitatives de procédures de routine, sans erreurs : GOMS ne permet de représenter que le comportement d'un utilisateur expert, qui ne laisse aucune place à l'erreur.

Le modèle GOMS est un modèle générique dans le sens où il est appliquable à divers niveaux d'abstraction. Il est cependant clair que l'application de GOMS au niveau des mécanismes cognitifs par exemple serait trop réductrice : GOMS ne décrit qu'un aspect limité des mécanismes en jeu lors de la réalisation d'une tâche. GOMS est aujourd'hui encore largement utilisé pour évaluer des comportements d'experts, comme en témoignent notamment les travaux de B. John, rapportés par exemple dans [John 90].

La principale application réaliste de GOMS est le modèle Keystroke, qui concerne les niveaux syntaxiques et lexicaux de l'interaction, selon la perspective linguistique évoquée en section 2.1.

Keystroke [Card 83] applique le modèle GOMS à la représentation des actions physiques que doit effectuer l'utilisateur pour spécifier une commande. Les Opérateurs en jeu sont la frappe de touches du clavier ou la pression de boutons de la souris, la désignation, le rapatriement de la main, l'action de dessiner, l'activité mentale, et le temps de réponse du système.

Keystroke permet de calculer et de comparer les performances de l'utilisateur en fonction d'une syntaxe donnée. Les évaluations Keystroke sont à considérer avec précaution cependant : imprécises, elles ne tiennent par exemple absolument pas compte de la logique sémantique globale qui dirige l'activité de l'utilisateur.

Extensions du modèle GOMS

Nous relevons ici des travaux dont l'objectif est, en partant d'une modélisation GOMS, de dépasser la simple évaluation quantitative des comportements experts. Les travaux de Kieras & Polson [Kieras 85] [Polson 85] visent à évaluer la complexité cognitive d'une interface.

L'approche est de construire un système de règles de production à partir des concepts d'une modélisation GOMS (notamment, les Buts et les règles de Sélection). Ce système de règles est ensuite utilisé pour simuler la réalisation d'une tâche. A partir de l'observation et de l'analyse du comportement de ce système, il est possible de formuler des prédictions quant au temps d'exécution, mais aussi le temps d'apprentissage (fonction du nombre de nouvelles productions nécessaires à la réalisation de la tâche), et les possibilités de transfert de connaissances entre deux systèmes (fonction des productions communes aux deux systèmes).

Le "Modèle Utilisateur Programmable"

Le "Modèle Utilisateur Programmable" (PUM) [Young 90] résulte de travaux concernent la prédiction des erreurs. L'approche adoptée se démarque entièrement de celle de la famille GOMS, puisque les procédures de réalisation d'une tâche ne sont pas modélisées. Au contraire, PUM repose sur une architecture cognitive programmable. Cette architecture recouvre un ensemble de principes et de contraintes psychologiques. L'utilisation de cette architecture nécessite de la "programmer", en lui apprenant à réaliser un ensemble de tâches avec une interface donnée. Cette "programmation" s'effectue en spécifiant les opérateurs à disposition, ainsi que leurs préconditions et leur effet sur les objets du domaine. La transcription de l'architecture programmée sur un système de résolution de problèmes (les expériences décrites utilisent SOAR [Newell 89]) fournit un modèle exécutable. L'utilisation de ce modèle s'effectue en lui donnant à résoudre une tâche décrite par l'état final des objets du domaine ; l'observation du comportement de ce modèle permet de détecter des problèmes

potentiels d'utilisation de l'interface : à partir de l'objectif final, quels sont les sous-buts choisis par le modèle, quels sont les opérateurs appliqués? Est-ce le comportement souhaité? L'objectif final est-il atteint?

L'ambition de PUM est de permettre la détection d'erreurs de conception dans une interface. Cette ambition, tout à fait louable, se heurte cependant aux fortes limitations des résultats actuels. La programmation du modèle est une tâche complexe, qui requiert un effort démesuré par rapport au gain retiré : les expériences rapportées, bien qu'encourageantes, ne traitent que de problèmes triviaux. Il est encore trop tôt pour se prononcer sur l'avenir de ces travaux.

2.3.2 Modèles de l'interaction

Nous relevons ici un courant de recherche axé sur la spécification formelle des systèmes interactifs - l'aperçu que nous proposons se fonde sur les travaux effectués à l'Université de York, significatifs de ce domaine, tels qu'ils sont évoqués dans [Dix 87] et [Harrison 91].

Les techniques formelles ont pour motivation première la *clarté d'expression*. Elles fournissent en effet des descriptions précises et non ambigües d'entités et de comportements, sous la forme de constructions mathématiques. Le formalisme développé à York, par exemple, définit un système interactif comme une collection d'agents communicants ; la notation proposée est une combinaison de deux techniques formelles :

- les types de données abstraits (ADT), qui permettent la description des fonctions et des états d'un système.
- un formalisme à base d'événements, CSP, qui permet l'expression d'une dynamique de l'interaction.

Au-delà de la précision de l'expression, les descriptions formelles ont l'intérêt d'offrir prise aux opérations de calcul. L'utilisation de ces techniques dans le but de *prouver les propriétés d'un système* est une des perspectives les plus intéressantes des travaux sur la formalisation - c'est pourquoi nous évoquons ces travaux dans le contexte de l'évaluation des interfaces.

Formalisation des propriétés de l'interaction

A. Dix et al. proposent dans [Dix 87] la formalisation mathématique d'un ensemble de principes et de propriétés d'un système interactif qui affectent la compréhension par l'utilisateur des commandes de ce système :

- la prévisibilité.
  Il s'agit d'énoncer les règles vérifiées par le système, et permettant à l'utilisateur de prévoir le comportement futur de ce système ; ces règles se fondent sur l'image du système, mais aussi la notion de contexte et de mémoire des actions passées (cf. [Harrison 91]).
- la passivité d'une commande.
  Une telle commande n'a pas d'effet sur les données du système - il s'agit de commandes que l'on qualifie parfois de "syntaxiques", telles les commandes de déplacement, d'ouverture et de fermeture des fenêtres dans un environnement multi-fenêtré.
- l'observabilité.

Il s'agit d'exprimer qu'à tout moment l'ensemble de l'état d'un système peut être perçu par l'utilisateur, au moyen de l'application de commandes passives.

- le caractère global ou local d'une commande.
- etc.

La formalisation de ces propriétés d'une part, et l'utilisation de techniques de spécification formelle d'un système interactif d'autre part, offrent les moyens nécessaires à une analyse d'un système, et à l'évaluation des propriétés vérifiées par ce système. Cette approche nous semble prometteuse : elle conduit à expliciter certaines hypothèses sur l'utilisation d'un système, qui la plupart du temps restent implicites ; elle permet ainsi de relever des incohérences dans une interface, ou plus simplement des aspects sujets à de potentiels problèmes, nécessitant une étude approfondie.

2.3.3 Discussion sur les méthodes d'évaluation

Les travaux sur l'évaluation prédictive des interfaces offrent une grande diversité de solutions dont la plupart sont encore à l'état de recherche. Un regard critique sur les méthodes d'évaluation nous conduit à considérer deux questions : quelle est la portée de ces méthodes d'évaluation, et quel est leur champ d'application dans le monde réel?

La portée des méthodes d'évaluation : que cherche-t-on à évaluer ?

Qu'est-ce qui fait la qualité d'une interface? Ce problème complexe, non complétement rationnel, ne peut être traité que de façon simplificatrice et partielle par des travaux d'évaluation prédictive automatisée. De tels travaux reposent sur l'adoption d'un point de vue restreint sur une interface, et sur le choix et l'évaluation de critères de qualité précis.

Une première constatation s'impose : *l'évaluation prédictive ne concerne que le point de vue du concepteur du système*. Si l'on référence la représentation adoptée en Figure I-2, Chapitre I, n'interviennent dans l'évaluation que le modèle du système C_σ (modèle du concepteur), une partie de l'image du système I_σ (principalement la syntaxe d'entrée des commandes), et, selon les méthodes, les modèles de la tâche et de l'utilisateur dont dispose le concepteur.

Les travaux que nous avons présentés sous l'appellation "modèle de l'interaction" axent l'évaluation sur une mesure des propriétés intrinsèques à un système représenté par C_σ et I_σ.

Les méthodes que nous avons isolées sous l'appellation "modèle de l'utilisateur" permettent une évaluation d'un système au travers de l'utilisation qui en est prévue par le concepteur : C_σ et, de façon partielle, I_σ sont évalués relativement au modèle de la tâche connu du concepteur. Les mesures effectuées visent à répondre aux questions suivantes :

- Quel est le *temps moyen de réalisation d'une tâche*, étant donnés le modèle conceptuel C_σ et l'image I_σ du système interactif?
  (cf les méthodes GOMS et Keystroke). La ou les procédures de réalisation de la tâche sont celles d'un utilisateur idéal qui ne commet pas d'erreur.
- Quel est le *degré de complexité* d'une procédure de réalisation d'une tâche, étant donnés le modèle conceptuel C_σ et l'image I_σ? Dans quelle mesure cette procédure pourra-t-elle être déduite de connaissances sur un autre système? Quel pourra être le temps d'apprentissage de cette procédure?

(cf les travaux de Kieras & Polson). Il s'agit de mesurer les connaissances en jeu lors de l'exécution d'une procédure de réalisation d'une tâche (d'un utilisateur idéal).

- Etant donnés C_σ et I_σ, quelles *erreurs potentielles* peut-on prédire dans la réalisation par l'utilisateur d'une tâche donnée?
  (cf le "modèle utilisateur programmable"). Il s'agit d'évaluer les procédures de réalisation elles-mêmes, et de détecter de potentiels problèmes d'utilisation de l'interface.

Ces méthodes permettent la détection d'anomalies : la réalisation d'une tâche prend trop de temps, est trop complexe, ou bien comporte un risque d'erreur ; ces anomalies amènent à reconsidérer la conception du système C_σ, ou bien certains choix de I_σ, de manière à réparer les insuffisances.

Le point de vue de l'utilisateur n'est certes pas totalement absent de ces méthodes : la mesure de la complexité d'une procédure ou bien la détection d'erreurs potentielles relèvent de ce point de vue. Cependant, ces méthodes s'exercent dans les limites des modèles du concepteur ; le modèle mental M_σ de l'utilisateur n'intervient pas, ou plus précisément c'est un utilisateur idéal qui est considéré, dont le modèle mental est assimilé au modèle C_σ du concepteur. L'adéquation du modèle conceptuel C_σ aux connaissances et au modèle de la tâche d'un utilisateur réel, l'adéquation de l'image I_σ pour représenter le modèle conceptuel C_σ et entraîner une perception correcte de ce modèle par l'utilisateur, etc., sont hors du champ de portée des méthodes d'évaluation prédictives.

Le champ d'application des méthodes d'évaluation prédictive : passer du cas de laboratoire au monde réel.

Quels que soient les mérites respectifs des travaux rapportés, l'emploi des méthodes d'évaluation dans un contexte réel n'est pas évident. L'application d'une méthode d'évaluation nécessite de disposer d'une représentation fidèle du système à évaluer ; ce qui est possible pour un cas de laboratoire s'avère beaucoup plus délicat dans le contexte d'un système réel : c'est le problème classique du passage du laboratoire au monde réel qui se pose, caractérisé par un facteur de complexité potentiellement dissuasif.

Le passage du laboratoire au monde réel est un problème particulièrement sensible pour l'application des méthodes d'évaluation prédictive en raison de la relative complexité des modélisations nécessaires à l'étude des cas les plus "simples". Jusqu'à présent, quelle que soit la méthode choisie, l'évaluation d'un système réel passe par un volume et une complexité de descriptions qui, à notre sens, met en péril l'application de la méthode à l'ensemble de ce système.

Malgré leurs limitations actuelles, les méthodes d'évaluation prédictive demeurent l'objet de recherches ambitieuses. Les méthodes décrites à la section suivante, si elles ne sont pas exclusivement et directement dédiées à l'évaluation des interfaces, offrent également des facilités dans ce sens ; l'on notera des approches très différentes de celles des méthodes d'évaluation prédictive, fondées sur une représentation non pas du système interactif lui-même mais des choix effectués lors de la conception de ce système.

2.4 Représentation de la raison d'être d'une interface

Il nous semble intéressant de relever ici un courant de recherche récent s'adonnant à la représentation non pas des interfaces mais plutôt des choix de conception qui ont conduit à ces interfaces.

Si l'on reprend la métaphore de l'espace "problème", la conception d'une interface donne lieu à un ensemble de décisions de parcours de cet espace, dont le résultat est le produit que constitue la spécification de cette interface. Les sections 2.1 et 2.2 traitent de la représentation de ce produit. Or ce produit final ne représente que la "surface" du résultat de la conception, la partie visible d'un iceberg construit peu à peu, accumulant les choix et compromis dirigeant le processus de conception.

La connaissance d'une interface nécessite parfois de comprendre les raisons pour lesquelles cette interface est ce qu'elle est : comment elle aurait pu être différente, et pourquoi tels choix ont été faits. La représentation de la partie cachée de l'iceberg, c'est-à-dire des choix de conception, offre plusieurs intérêts, principalement une aide à la conception, et un support de communication entre concepteurs.

Nous relevons deux approches très différentes de ce problème de la représentation de l'argumentation ou raison d'être[1] d'une interface : la décomposition analytique des choix de conception, principalement abordée dans les travaux de A. MacLean et al., et l'analyse des assertions psychologiques contenues dans une interface, telle que décrite dans les travaux de J. Carroll et al.

Décomposition analytique des choix de conception

Les travaux de A. MacLean et al., rapportés dans [MacLean 89], constituent la principale contribution au problème de la représentation analytique de l'argumentation d'une conception. Ces travaux décomposent l'espace de la conception en un espace de décision et un espace d'évaluation, et proposent une notation semi-formelle pour représenter les choix effectués dans chacun de ces deux espaces. Les choix considérés concernent le dialogue et les détails de présentation et de comportement.

L'*espace de décision* décrit ce que les composants du produit fini peuvent être ; les éléments de base de cet espace sont les *options*, organisées autour de *questions* de conception. La notation proposée est très simple : à partir d'une question de conception, l'ensemble des options, ou réponses potentielles est représenté, pouvant éventuellement conduire à poser des questions supplémentaires. Par exemple, la question : "Où et quand afficher la barre de défilement?" provoque les options : "De façon permanente, attachée à un bord de la fenêtre", ou bien "De façon transitoire, la barre n'apparaissant que lorsque nécessaire" ; cette dernière option soulève alors la question : "Comment faire apparaître la barre?", etc.

L'*espace d'évaluation* décrit les raisons pour lesquelles une option est choisie ; les éléments de base de cet espace sont les *liens de cohérence*, qui mettent en valeur les relations de dépendance internes à l'espace de conception, ou bien entre cet espace et le monde extérieur, et les *critères*, qui sont les principes régissant les choix. Le choix d'une option doit

1. En anglais, le terme employé est celui de "rationale" d'une interface. Un tel mot n'existe pas en français. "Argumentation" et "raison d'être" sont deux termes approchants que nous utiliserons indifféremment..

en effet d'une part préserver la cohérence de l'interface considérée comme un tout, et d'autre part s'effectuer de façon raisonnée, à partir de critères de jugement explicites.

Les liens de cohérence interne s'expriment en reconnaissant des classes de questions de conception, et en figeant l'option correspondant à une classe de questions donnée. Les critères de choix doivent être définis par le concepteur de façon à qualifier les caractéristiques recherchées : faible encombrement de l'écran, retour ("feedback") en continu, déplacements de la souris minimaux, effort minimal de la part de l'utilisateur, etc. La confrontation des options aux critères d'évaluation prend naturellement la forme d'une matrice de comparaison.

La représentation des espaces de décision et d'évaluation offre un support au processus de conception même. D'une part la représentation explicite des questions, options et critères offre, à défaut d'une méthode de conception, une structure organisatrice autour de laquelle va s'effectuer la conception - le problème est maintenant de poser les bonnes questions de conception. D'autre part, cette représentation offre un support au raisonnement, limitant la charge cognitive du concepteur en lui adjoignant une formulation externe. Une expérimentation décrite dans [MacLean 90] vient renforcer ces idées, tout en montrant la nécessité d'un outil mettant en œuvre le schéma d'analyse rationnelle.

La représentation des espaces de décision et d'évaluation permet au concepteur de mieux communiquer avec lui-même, ainsi qu'avec les autres intervenants dans la conception d'un système : c'est un support de communication entre les concepteurs, et entre les concepteurs et le client auquel le système est destiné. A. MacLean et al. présentent également cette représentation comme une aide aux utilisateurs : elle peut les aider à mieux comprendre un système, notamment dans le contexte d'une particularisation de l'interface. Nous sommes à ce sujet plus réservés : une telle représentation nous semble avant tout un outil de concepteur, éloigné des préoccupations et schémas de langage de l'utilisateur en général.

Analyse des assertions psychologiques

Plutôt que de chercher à représenter les choix élémentaires de la conception d'une interface, J. Carroll et al. adoptent une approche axée sur l'analyse des assertions psychologiques sous-jacentes aux interfaces.

Cette démarche, présentée dans [Carroll 89], a pour ambition de favoriser l'utilisation de la psychologie dans la conception des interfaces, au-delà des théories quantitatives en cours dans le domaine de l'évaluation de la conception (cf. 2.3). Le moyen proposé est une interprétation systématique de l'utilisabilité d'une interface, au travers de l'analyse des assertions psychologiques contenues dans cette interface.

Cette démarche repose sur l'idée qu'une interface concrétise un ensemble de postulats psychologiques : les éléments d'une interface engendrent des conséquences psychologiques lors de leur utilisation ; la conception de ces éléments dénote ainsi l'existence de postulats sur le comportement et l'expérience de l'utilisateur de l'interface. Considérons l'exemple des "training wheels", un mécanisme générique que l'on trouve dans certaines applications dans un contexte d'apprentissage : ce mécanisme inhibe certaines fonctions complexes ou dangereuses de l'application, protégeant l'utilisateur contre ses propres erreurs, et le contraignant à un apprentissage progressif de l'ensemble des fonctions de l'application. Parmi les assertions psychologiques contenues dans ce concept des "training wheels", J. Carroll et al. relèvent les suivantes :

- Inhiber les conséquences tout en autorisant le déclenchement des erreurs rend plus perceptible la mise en correspondance des objectifs et des plans.
- Limiter la distraction qu'entraîne le recouvrement des erreurs concentre l'attention de l'utilisateur sur les actions correctes.
- La pratique de scénarios élémentaires permet d'établir une maîtrise intégrée des opérations de base.
- etc.

L'analyse des assertions psychologiques s'organise selon les différentes étapes de l'activité de l'utilisateur, présentées en au chapitre I. Ce sont bien entendu les assertions principales qui seront extraites, la liste ne pouvant être exhaustive. Si l'on considère les tâches réalisées dans un système, les assertions s'organisent autour de deux dimensions : d'une part, la couverture des tâches (quelles sont les tâches concernées par les assertions?), et d'autre part la profondeur (combien d'assertions traitent de la même tâche?).

Cette perception d'une conception en termes psychologiques peut être construite et utilisée au cours de l'activité de conception. Une expérience rapportée dans [Bellamy 90] montre l'intérêt de ce schéma d'analyse psychologique pour pallier certaines difficultés de conception : notamment, en explicitant les assertions, et en introduisant un nombre limité de dimensions d'analyse, la complexité et la charge cognitive peuvent être réduites.

2.5 Conclusion sur les méthodes de conception

Nous avons rapidement décrit les principaux modèles et méthodes actuellement proposés pour fournir une aide à la conception ; la spécification de l'interface, son évaluation, et l'expression des choix de conception ont été successivement abordés.

Au terme de ce panorama, il apparaît que le concepteur d'une interface dispose d'un large éventail de modèles et méthodes répondant de façon locale et restreinte aux besoins de ce dernier, mais dont la variété est à la hauteur de la diversité de ces besoins.

Un point essentiel, bien qu'évident, est de remarquer que toutes ces solutions sont liées : les contenus des modèles de représentation mis en jeu se recoupent, mais également se complètent. La spécification du dialogue définit les opérateurs d'une modélisation Keystroke ; une analyse des choix de conception permet de justifier l'existence et les caractéristiques de ces opérateurs ; l'analyse des tâches d'un système fournit les buts d'une modélisation GOMS, structure l'analyse des choix de conception ou des assertions psychologiques, etc.

La discussion sur ces solutions multiples s'effectue sur deux plans distincts. Tout d'abord, nous souhaitons relever le besoin de concepts génériques ; ensuite, l'absence de cohésion.

Qu'il s'agisse de définir les concepts d'un système, ou bien de spécifier le dialogue, la littérature fait abstraction de toute **dimension générique** pour un système interactif. Il semblerait que la conception d'un système doive à chaque fois être effectuée à partir de zéro. En fait, il nous semble plus juste de penser que cette généricité - car elle existe! - s'exprime mal au niveau des modèles de représentation de l'interface, faute d'une théorie dans le domaine de l'interaction. Nous verrons au Chapitre III comment cette dimension générique est exprimée au niveau des outils de développement pratique des systèmes interactifs.

Manque une **méthode globale** intégrant un ensemble choisi de solutions partielles, effectuant le lien entre ces solutions, et offrant une méthode de conception véritable et

complète. Une telle méthode globale nécessiterait la mise en œuvre de transitions entre formalismes (par exemple, le passage d'un formalisme de représentation du dialogue vers un modèle permettant l'évaluation de ce dialogue), et de façon générale l'adaptation des différentes notations afin de mettre en lumière les points communs à ces différents langages, de gérer les dépendances entre les modèles de représentation, et obtenir un ensemble cohérent.

Si cette méthode globale est encore du domaine de l'imagination, le point de départ d'une telle méthode est, nous semble-t-il, l'*analyse des tâches* d'un système, et la représentation de ces tâches. Nous souhaitons d'autre part mettre l'accent sur la représentation des *informations de niveau méta* sur une interface (ce que nous avons appelé l'argumentation ou la raison d'être d'une interface). Ces informations sont à l'heure actuelle mal formalisées, et très faiblement exploitées - elles demeurent la plupart du temps implicites.

3 Méthodes de développement

Question : On vous confie le développement d'un système dont les spécifications conceptuelle et externe sont achevées. Vous disposez de la description des tâches réalisées par le système, des niveaux conceptuels, sémantiques, syntaxiques et lexicaux de l'interaction.

Comment procédez-vous?

Les méthodes de développement du monde industriel définissent un ensemble d'étapes pour la réalisation du système physique : spécifications internes, globales puis détaillées, prototypes, évaluation, codage du produit final, tests unitaires, d'intégration, etc. Cette décomposition de l'activité de développement peut tout à fait être appliquée à la réalisation d'un système interactif.

Question : Comment organisez-vous le code?

Le code d'un système interactif présente un ensemble de caractéristiques qui nécessitent l'adoption de stratégies de réalisation particulières. Ces caractéristiques sont relatives au cycle de développement, au problème de la portabilité, et à la diversité des compétences en jeu.

Le développement d'un système interactif est un cycle (cf. 6.3, Chapire I) ; de nombreuses itérations sont nécessaires pour parvenir au produit final, sans tenir compte des itérations de maintenance de ce produit. Une large part de ces itérations est relative à l'interface : la faible portée des méthodes d'évaluation de la conception (cf. 2.3) conduit à préférer les méthodes d'évaluation par l'expérimentation. Une partie de ces itérations cependant concerne également les fonctions de l'application. Le code d'un système est donc soumis à une succession de modifications variées bien que localisées ; il doit être structuré en conséquence.

L'environnement de développement d'un système se caractérise par la diversité des matériels, des logiciels graphiques, et des utilisateurs. Cette diversité nécessite de prendre en compte les problèmes de portabilité : le code d'un système doit être structuré de façon à identifier et isoler les dépendances envers l'environnement.

Enfin, la diversité porte également sur les compétences nécessaires à la réalisation du code du système. Une étude présentée dans [Olsen 84] ne décrit pas moins de neuf rôles intervenant dans le développement d'un système interactif! Pour notre part, nous identifions deux grands ensembles de compétences : le programmeur d'application, responsable de la réalisation des fonctions du système, et le programmeur d'interface, qui réalise la partie du système dédiée à l'interface.

Le génie logiciel apporte à la complexité la solution de la modularité. Si le principe de la modularité est connu, et apprécié, son application est malaisée. Ici encore, l'on ressent un besoin de méthodes. Ces méthodes vont s'exprimer sous la forme de *modèles d'architecture des applications*.

Avant de présenter les principaux modèles d'architecture, une constatation s'impose : le domaine des méthodes de développement, moins complexe que celui des méthodes de conception, présente une maturité plus grande, et une diversité nettement moindre. Si l'on peut identifier un ensemble de modèles distincts, il semble cependant qu'un consensus soit atteint dans la communauté IHM quant aux caractéristiques générales de ces modèles. Notre présentation est axée sur ces caractéristiques consensuelles.

Après la présentation des principes consensuels, nous décrivons quelques modèles proposés dans la littérature, puis nous précisons les grandes lignes de notre proposition dans ce domaine. Notre conclusion porte sur les difficultés d'application de ces modèles, et le besoin de disposer d'un plus grand degré de précision et de détail dans la formulation des composants des modèles d'architecture.

3.1 Modèles d'architecture : principes consensuels

Les travaux menés dans le domaine des modèles d'architecture des systèmes interactifs, bien que relativement récents puisque les premières recherches n'apparaissent qu'au début des années 80, parviennent aujourd'hui à un consensus sur un ensemble de lignes directrices[1] - tendance confirmée par les résultats récents de deux groupes de travail, rapportés respectivement dans [Lisbon 90] et dans [Arch 91]. Parmi ces dernières, nous relevons trois principes généraux, que nous présentons en suivant l'ordre chronologique de leur apparition : la séparation composant fonctionnel/interface, l'organisation de l'interface en trois composants, et le modèle multiagent.

3.1.1 Principe numéro 1 : la séparation composant fonctionnel / interface

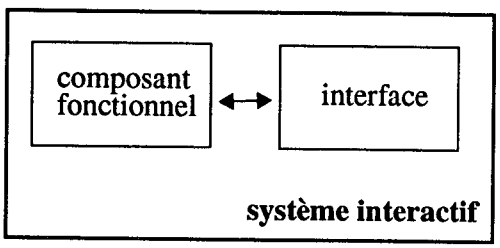


Figure II-2 *Le composant fonctionnel et l'interface.*

Un principe de base, admis dès l'origine, est celui qui décompose un système interactif en deux grands morceaux :

- le *composant fonctionnel*, ou noyau fonctionnel, qui contient les fonctions du système. Le composant fonctionnel contient la partie du code qui est indépendante du médium d'interaction.

1. C'est un sentiment personnel que nous exprimons ici quant à l'existence de ce consensus.

Notons que cette stabilité s'inscrit dans le cadre des interfaces "traditionnelles", alliant écran, clavier et souris, et développées à partir de bibliothèques graphiques telle X Window - la grande majorité des interfaces actuelles..

L'apparition de nouvelles modalités d'entrée (parole, geste), l'extension des médias de restitution (son, video) sont des phénomènes fortement déstabilisants ; non seulement ils nécessitent la définition d'architectures adaptées, mais peuvent également remettre en cause certains des principes des architectures traditionnelles, notamment celui des trois composants de l'interface (principe numéro 2). S'il est encore trop tôt pour parler de résultats sur les architectures multimodales, l'on suivra avec intérêt les travaux en cours, notamment au sein de l'équipe IHM du Laboratoire de Génie Informatique de Grenoble.

Ces fonctions relèvent principalement du domaine de l'application considérée. Par exemple, dans le domaine des bases de données, le composant fonctionnel d'une application contient des fonctions d'interrogation et de modification de la base. Le composant fonctionnel peut également proposer des traitements ou services généraux indépendant du domaine de l'application (par exemple, un moteur de système expert).

- l'*interface*, qui contient le code dédié à la mise en œuvre de l'interaction entre l'utilisateur et le système interactif.

La Figure II-2 représente ce découpage. Initialement logique, ce découpage se veut également physique : l'organisation du code d'un système reflétera cette décomposition, assurant ainsi un degré d'indépendance entre fonctions et interfaces, permettant l'évolution séparée de ces deux composants.

Remarquons que l'indépendance des deux composants n'est que relative. Sur le plan physique, la suite de cette section le démontrera. Sur le plan logique, le contenu du composant fonctionnel détermine certains aspects de l'interface. Un exemple classique est celui de l'indication de l'état d'avancement d'une activité longue lancée par l'utilisateur (sous la forme d'un pourcentage des données traitées par exemple) ; la présentation de ce type d'indications dans une interface nécessite que le composant fonctionnel puisse fournir l'information, et soit donc construit en conséquence.

3.1.2 Principe numéro 2 : les trois composants de l'interface

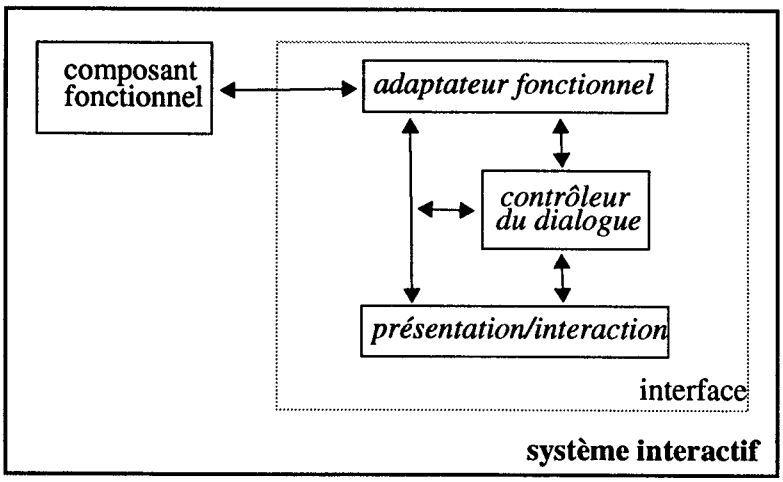


Figure II-3 *Les trois composants de l'interface.*

Le second principe aujourd'hui admis concerne la structuration du composant d'interface. Trois sous-parties sont identifiées : l'adaptateur du composant fonctionnel, le contrôle du dialogue, et le composant de gestion de la présentation et de l'interaction, représentées en Figure II-3.

L'*adaptateur du composant fonctionnel* dénote la vue qu'a l'interface des fonctions et données du composant fonctionnel. Ce composant repose sur l'interface de programmation du composant fonctionnel ; son rôle est de fournir aux autres composants de l'interface les fonctions et données adaptées aux commandes et informations présentées à l'utilisateur - éléments qui ne sont pas forcément directement disponibles dans le composant fonctionnel[1].

Le *composant de présentation et d'interaction* contient le code de mise en œuvre de la présentation physique de l'interface, ainsi que de la réception des interactions de l'utilisateur sur les périphériques d'entrée. La réalisation de ce composant repose sur l'utilisation des fonctions d'un serveur graphique (par exemple, X Window, NeWS, NeXTStep, etc., sur lesquels nous reviendrons au Chapitre III).

Le *contrôleur du dialogue* a un rôle central dans l'interface : celui de la mise en œuvre des commandes à partir de l'interprétation des interactions de l'utilisateur, du maintien de la cohérence entre données affichées et données du composant fonctionnel (par l'intermédiaire de l'adaptateur) ; le contrôleur assure de plus l'éventuel séquencement des opérations du système.

La décomposition de l'interface présentée ci-dessus est celle qui prévaut dans le modèle Seeheim [Pfaff 85], cité dans [Coutaz 90]. Ce modèle, élaboré en 1984, peut être considéré comme la référence de base des travaux sur les architectures.

3.1.3 Principe numéro 3 : le modèle multiagent

Le modèle multiagent structure un système en un certain nombre d'agents spécialisés qui réagissent à des événements en effectuant des traitements, et qui à leur tour peuvent générer des événements. Le modèle multiagent constitue en quelque sorte un modèle de réalisation du modèle à événements présenté en 2.2.2.

Le modèle multiagent se caractérise par une organisation fortement modulaire, des traitements exécutés (théoriquement) en parallèle, et une communication entre modules par événements. L'application du modèle multiagent lors de la structuration de l'interface permet de briser les trois blocs monolithiques que sont l'adaptateur, le contrôleur, et le gestionnaire de présentation/interaction, tels qu'ils apparaissent dans la section précédente. Le bénéfice retiré est double : d'une part la modularité, et d'autre part des facilités de gestion des dialogues multifils.

Le développement des modèles multiagent est à rapprocher de celui des langages de programmation à objets : les concepts véhiculés par les langages à objets fournissent une base de réalisation tout à fait adaptée aux concepts des modèles multiagent. Un agent peut être mis en œuvre sous la forme de un ou plusieurs objets ; les événements de communication correspondent alors au message d'invocation d'une opération sur un objet.

L'application du modèle multiagent n'est pas contradictoire avec la décomposition en trois blocs définie comme principe numéro 2. Si elle conduit à l'éclatement de cette décomposition, elle s'effectue à l'intérieur des frontières logiques (et dans de nombreux cas physiques) définies pour ces trois blocs.

---

1. A mi-chemin entre l'interface et le composant fonctionnel, l'adaptateur peut être considéré comme partie de chacun des deux composants.

Reste à préciser comment s'effectue la structuration en agents des trois composants de l'interface. Les solutions varient selon les modèles d'architecture. Quelques unes de ces solutions sont présentées dans la section suivante.

3.2 Modèles d'architecture : quelques exemples

Nous donnons ici quelques exemples de modèles d'architecture des systèmes interactifs : les modèles MVC (antérieur au modèle Seeheim), GWUIMS et PAC (postérieurs au modèle Seeheim) sont rapidement décrits. L'originalité de chaque modèle est mise en lumière, ainsi que le mode d'application (ou de non application) des trois principes ci-dessus présentés.

3.2.1 MVC

MVC (Model View Controller) est un modèle utilisé dans l'environnement Smalltalk [Goldberg 84]. Les éléments représentés par les lettres M, V et C sont le Modèle, la Vue, et le Contrôleur. Ces termes, qui rappellent ceux du modèle Seeheim, présentent cependant de notables différences.

Le Modèle M est celui des fonctions du domaine du système ; il s'apparente à l'adaptateur du composant fonctionnel.

La Vue V réalise la présentation des données contenues dans le Modèle. Cet élément regroupe une partie des fonctions du gestionnaire de présentation et d'interaction (seulement la partie présentation), mais également les fonctions du contrôleur du dialogue, relatives à l'affichage des données.

Le Contrôleur C gère les interactions de l'utilisateur sur la présentation, et initialise les traitements correspondants. Son rôle intègre une partie des fonctions du gestionnaire de présentation et d'interaction (seulement l'interaction), mais également certaines fonctions du contrôleur de dialogue, relatives à la mise en œuvre des commandes du système.

Ainsi, la notion de contrôleur du dialogue du modèle de Seeheim n'est pas représentée explicitement : les fonctions correspondantes sont disséminées entre les composants V et C de MVC.

Un système interactif est structuré en un ensemble d'agents comportant chacun les trois facettes M, V et C. Chaque facette est réalisée comme un objet Smalltalk. Un agent MVC est donc gestionnaire de tous les aspects (interaction, présentation, contrôle, adaptateur) d'une fraction de l'interface, résultant d'un découpage vertical[1] du système représenté en Figure II-3.

1. Les notions de découpage vertical ou horizontal d'un système sont largement utilisées dans la suite de ce chapitre. C'est de la décomposition de la structure illustrée en Figure II-3 dont il s'agit ; la verticalité et l'horizontalité se rapportent à la disposition adoptée dans cette figure - elles peuvent également être perçues par analogie avec, dans le domaine de l'économie, les termes de "marché vertical ou horizontal".

Plus précisément, un découpage horizontal fait intervenir des agents spécialisés dans la gestion d'un des trois aspects d'un système que sont l'adaptateur, le contrôle et la présentation.

Un découpage vertical propose des agents de compétence plus étendue, dont le rôle participe de plusieurs des trois aspects d'un système.

3.2.2 GWUIMS

GWUIMS (George Washigton User Interface Management System) [Sibert 86] propose un modèle d'architecture à objets qui structure un système interactif en cinq classes d'objets : les Objets-A, Objets-I, Objets-R, Objets-T et Objets-G. Les concepts utilisés sont ceux du modèle linguistique du dialogue : l'on retrouve la distinction entre sémantique, syntaxe, et lexique.

Les Objets-A (A pour Abstraction) contiennent la sémantique de l'application. Les Objets-A sont décrits comme les objets du composant fonctionnel. Si l'adaptateur du composant fonctionnel n'est pas mentionné, l'on peut cependant considérer que sa réalisation repose sur un ensemble d'Objets-A.

Les Objets-R (R pour Représentation) contrôlent la présentation. A la frontière entre syntaxe et lexique, ces objets reçoivent les aspects lexicaux d'entrée au travers des Objets-T (T pour "Typing"), et contrôlent les aspects lexicaux de sortie par l'intermédiaire des Objets-G (G pour Graphique). Les fonctions mises en œuvre par l'ensemble des Objets-R, Objets-T et Objets-G sont celles du composant de présentation et d'interaction.

Les Objets-I (I pour Interaction) sont intermédiaires entre la syntaxe et la sémantique du système. Ils font le pont entre les Objets-A et les Objets-R. Les Objets-I ont les fonctions du contrôleur du dialogue.

Un système apparaît comme un ensemble d'agents spécialisés dans un traitement particulier : Objet-A, Objet-I, Objet-R, Objet-T ou Objet-G. Le découpage est cette fois horizontal, si l'on considère la Figure II-3.

3.2.3 PAC

PAC (Présentation, Abstraction, Contrôle) [Coutaz 87] est un modèle d'architecture théorique, non lié à un outil ou langage de réalisation.

PAC structure un système comme un ensemble d'objets ("objet" pris au sens large) interactifs composés de trois facettes : la Présentation (P), qui gère le comportement en entrée et en sortie de l'objet ; le Contrôle (C), qui maintient la cohérence entre les deux autres facettes ; l'Abstraction (A), qui contient les concepts et fonctions du système. Les facettes de Présentation et de Contrôle mettent en œuvre des fonctions analogues à celles des composants du modèle Seeheim. L'Abstraction a une signification variable : elle représente l'adaptateur du composant fonctionnel, mais aussi, plus simplement, les éléments abstraits sous-jacents à une présentation (par exemple, la valeur d'une chaîne affichée, les bornes d'un curseur, etc.).

Le modèle organise l'ensemble d'agents selon une structure de contrôle hiérarchisée, au moyen de la notion d'objet interactif composé. Un objet composé définit un agent structuré dont le comportement dépend de lui-même mais aussi des objets qui le constituent : si son Abstraction lui est propre, sa Présentation repose sur une composition de celle des objets constituants, et son Contrôle a la charge de gérer la collaboration des objets constituants. Un système interactif peut être perçu comme un objet unique dont l'Abstraction est le composant fonctionnel (ou son adaptateur) en entier, et le Contrôle supervise l'ensemble de l'application ; cet objet est récursivement décomposable en objets composés, jusqu'à l'obtention d'agents PAC simples. Ainsi, les agents PAC mettent en œuvre un découpage vertical de l'organisation présentée en Figure II-3, mais ce découpage ne concerne pas l'adaptateur du composant fonctionnel, qui reste monolithique.

3.3 Discussion, et proposition

Les trois grands principes énoncés en 3.1 trouvent une application variable selon les modèles d'architecture ; notamment, les concepts du modèle de Seeheim sont soumis à des variations plus ou moins prononcées. Dans l'ensemble cependant, les modèles d'architecture - dont certains ont été développés indépendamment de Seeheim (notamment MVC), sont cohérents avec les principes énoncés.

A partir d'une observation critique des propriétés des modèles d'architecture présentés à la section précédente, nous élaborons les grandes lignes d'un modèle d'architecture qui constitue le point de départ de nos travaux. Notre proposition se veut "concrète" : elle prend en compte les particularités de l'environnement de développement.

3.3.1 Propriétés générales retenues

Les propriétés à partir desquelles nous élaborons notre proposition sont au nombre de trois : le regroupement de la gestion des entrées et des sorties, le découpage vertical complet, et les relations hiérarchiques entre agents.

Entrées et sorties regroupées

La gestion séparée des entrées de l'utilisateur d'un côté, et de la présentation de l'autre côté est un principe retenu dans certains modèles (notamment MVC et GWUIMS). Ce principe s'avère mal adapté à la réalisation des interfaces graphiques vérifiant les propriétés suivantes :

- La plupart des entrées de l'utilisateur nécessitent de façon immédiate un retour ("feedback") au niveau de l'affichage.
- Une entrée de l'utilisateur fait souvent référence à un élément produit en sortie. Ce phénomène, identifié dans la littérature sous le terme d'inter-références ([Draper 86], cité dans [Nanard 90]), est caractéristique des interfaces à désignation directe.

L'exemple le plus frappant concerne le déplacement d'une icône : à chaque déplacement de la souris, la présentation doit être modifiée afin d'assurer le retour visuel montrant le déplacement réalisé ; un autre exemple est celui d'un menu déroulant : l'affichage du menu, et la sélection par l'utilisateur d'un élément de ce menu requiert un enchevêtrement d'interactions d'entrée (pression d'un bouton de la souris, déplacement, relâchement de la souris) et de modifications de la présentation (affichage du menu, mise en évidence de l'élément du menu sur lequel se trouve le curseur, affichage en vidéo inverse de l'élément effectivement désigné).

Il est nécessaire de disposer au niveau de l'architecture des systèmes de moyens adéquats pour traiter efficacement ces phénomènes. La réalisation des opérations d'entrée et de sortie enchevêtrées sera plus aisée au sein d'un même composant, et plus propice aux temps de réponse courts, cruciaux dans le cas précis du retour.

Il est à noter que, en pratique, les Vues et Contrôleurs de MVC sont souvent définis de façon combinée, ce qui corrobore nos remarques précédentes ; il apparaît dans [Krasner 88] que la définition d'éléments "pluggable", objets "prêts à l'emploi" utilisables par instanciation, tend à présenter les paires Vue-Contrôleur comme une entité unique.

Découpage vertical complet

La validité d'un découpage vertical nous semble incontournable. Ce découpage considère un agent comme une machine abstraite entière, éventuellement dépendante d'autres agents

pour son fonctionnement, mais capable de mettre en œuvre une fraction entière de l'interface, depuis la présentation jusqu'aux relations avec le composant fonctionnel.

Contrairement à PAC, nous considérons un découpage vertical complet, comprenant les fonctions de l'adaptateur du composant fonctionnel. D'une part, la répartition des concepts et fonctions de l'adaptateur relativement aux agents de l'interface nous semble assez naturelle : à une fraction de l'interface correspond un ensemble d'éléments fonctionnels clairement défini. D'autre part, ce choix est dicté par des considérations pratiques : la concentration des accès au composant fonctionnel au sein d'un unique objet, tel que PAC le préconise, implique une circulation de messages entre les agents, au détriment des temps de réponse du système. Notons que nos observations et choix se trouvent confirmés dans les récents travaux menés autour de PAC. [Coutaz 91b] présente une version nouvelle de ce modèle, appelée PAC-Seeheim, qui rejoint notre notion de découpage vertical complet.

Relations hiérarchiques entre agents

La gestion de la répartition des compétences entre agents passe par la mise en œuvre de niveaux d'abstraction que l'on exprimera au travers de relations hiérarchisées entre les agents d'un système. Nous retenons ainsi la propriété générale du modèle PAC concernant les relations entre agents : un système est organisé comme une hiérarchie d'agents coopérant. Certains agents supervisent d'autres agents ; ils sont responsables du bon fonctionnement de ces agents : ils leur transmettent les paramètres nécessaires, mettent en œuvre leur instanciation, leur initialisation, leur activation, et leur destruction.

3.3.2 Proposition

Notre proposition [Normand 90a] s'insère dans le contexte d'un environnement de développement à objets, muni des logiciels graphiques X Window et Motif. Nous aurons par la suite l'occasion de revenir sur les propriétés de cet environnement, notamment sur le langage de programmation à disposition, et sur le contenu des logiciels graphiques. A ce stade de notre exposé, les éléments à relever sont les suivants :

- L'entité de base de la structuration d'une application est l'objet. Un objet contient un ensemble de données, et offre un ensemble d'opérations.
- La construction d'une application s'effectue en définissant de nouveaux objets, ou bien en réutilisant des objets existants.
- Les logiciels graphiques fournissent une bibliothèque d'objets élémentaires graphiques, couramment appelée "boîte-à-outils". Ces objets élémentaires constituent la plate-forme de réalisation de l'interface d'une application.

Notre modèle décompose un système interactif en cinq grandes classes d'objets : les objets internes, les objets montrables, les objets de contrôle, les objets vue, et les objets interactifs. Outre cette décomposition horizontale d'un système, des regroupements verticaux sont mis en œuvre, dénotant les agents du système.

Décomposition horizontale

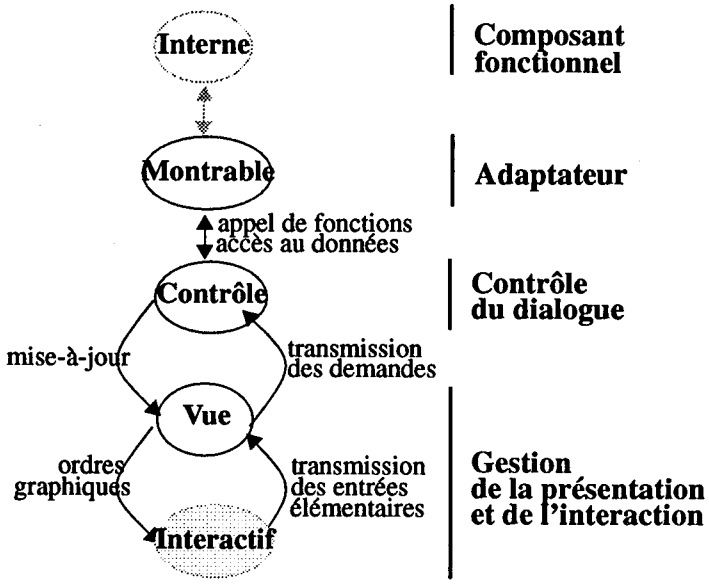


Figure II-4 *Les cinq classes d'objets.*

La Figure II-4 illustre les cinq classes d'objets intervenant dans la décomposition d'un système.

Les *objets internes* constituent le composant fonctionnel du système. Les *objets montrables* représentent l'adaptateur du composant fonctionnel ; ce sont les données contenues dans ces objets qui sont affichées dans la présentation ; les opérations définies sur ces objets sont les fonctions à partir desquelles sont mises en œuvre les commandes du système.

Les *objets vue* gèrent l'interaction et la présentation du système interactif. Leur rôle est de recevoir les actions de l'utilisateur sur les périphériques d'entrée, d'analyser ces actions, et de les traduire dans le formalisme des objets de contrôle, et de les communiquer à ces derniers. En retour, un objet vue est capable de présenter à l'image des données qui lui sont communiquées par un objet de contrôle. Les fonctions d'un objet vue sont réalisées au travers d'un ensemble d'*objets interactifs*. Ce niveau d'objets est introduit afin de prendre en compte les logiciels graphiques au-dessus desquels est réalisée l'interface d'une application. Les objets interactifs sont les composants élémentaires ("widgets") des boîtes-à-outils fournies par ces logiciels graphiques.

Les *objets de contrôle* ont pour rôle de mettre en œuvre le dialogue : exécution des commandes en fonction des demandes de l'utilisateur transmises par les objets Vue ; maintient de la cohérence entre les objets Montrables et les objets Vue, de façon à assurer la validité des affichages ; gestion de l'état du dialogue, et des éventuels séquencements à l'intérieur d'une session.

Regroupements verticaux : les agents

L'ensemble des objets d'un système est partitionné en unités logiques représentant les *agents du système*. Un agent regroupe un objet de contrôle, un ou plusieurs objets vue,

l'ensemble des objets interactifs associés aux objets vue, et utilise un ou plusieurs objets montrables.

L'objet de contrôle identifie l'agent et son rôle. Les objets montrables contiennent les concepts et fonctions utilisés par l'agent. Les objets vue mettent en œuvre la présentation de l'agent, ainsi que l'interaction associée. Un objet vue, de même qu'un objet de contrôle, est propre à un agent ; a fortiori, il en va de même pour un objet interactif. Un objet montrable peut être partagé entre plusieurs agents, dans le cas de vues multiples sur cet objet.

La Figure II-5 présente un exemple d'application de notre modèle. Le cas étudié est celui d'un système (ou partie d'un système) permettant la visualisation d'une liste d'éléments - par exemple, une liste d'enregistrements, de fichiers, etc. Ce système fournit la liste des éléments représentés par leur nom ; l'utilisateur a la possibilité de parcourir cette liste, et d'afficher le détail d'un élément choisi ; plusieurs éléments peuvent être détaillés à la fois. La présentation externe de ce système est suggérée en partie basse de la figure. La partie supérieure représente une structuration possible pour le code réalisant ce système (pour plus de clarté, ni les objets internes ni les objets interactifs ne sont représentés).

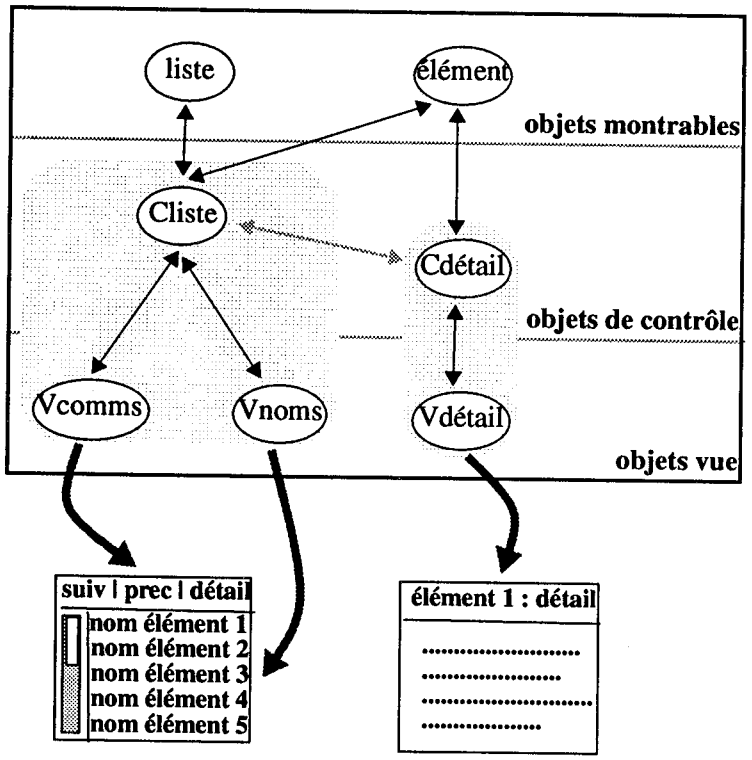


Figure II-5 *Les agents (exemple).*
Les doubles flèches indiquent les flux de contrôle entre les objets.

Deux agents sont définis :

- L'agent principal, gestionnaire de la liste d'éléments.
  Cet agent est composé d'un objet de contrôle (Cliste), et de deux objets vue (Vcomms et Vnoms) qui se partagent la charge de la présentation des commandes de navigation et de la liste des noms d'éléments. Cet

agent est en relation avec l'objet montrable de la liste, mais aussi avec chacun des objets montrables des éléments de la liste.

- L'agent gestionnaire de la vue détaillée sur un élément.
  Cet agent est composé d'un objet de contrôle (Cdétail), un objet vue (Vdétail), et n'accède qu'à un seul objet montrable, celui de l'élément présenté.

Les relations entre agents sont doubles. D'une part, des relations hiérarchiques peuvent exister entre les objets de contrôle de deux agents, et d'autre part des relations d'inclusion peuvent survenir entre les objets vue de ces deux agents. Dans le cadre de notre exemple, l'agent gestionnaire de la liste supervise l'agent de détail de la façon suivante : à la demande de l'utilisateur (transmise par la vue Vcomms), Cliste instancie un agent détail, et l'initialise en lui transmettant l'identification de l'élément à visualiser (information rapportée par la vue Vnoms).

Il est clair que la définition de ces deux agents n'est qu'une des structurations possibles de cet exemple. Par rapport à une solution qui n'emploierait qu'un seul agent cumulant les deux rôles ci-dessus identifiés, l'agent de détail a pour intérêt d'isoler un comportement générique réutilisable : la présentation détaillée et simultanée de plusieurs éléments de la liste est réalisée par l'instanciation de plusieurs agents de détail.

Une remarque est également nécessaire quant au mode de gestion de la cohérence des vues multiples sous-jacent à la solution proposée. Comment assurer que la modification d'un nom d'élément (par l'utilisateur, de façon directe, ou bien au cours d'un traitement fonctionnel, éventuellement issu d'un autre système s'exécutant en parallèle) sera correctement reportée à l'écran, sur les présentations générale et (le cas échéant) détaillées? La solution que nous proposons est celle d'un objet montrable "actif", capable d'avertir lui-même les objets de contrôle intéressés par une éventuelle modification de ses données internes. Nous reviendrons au Chapitre III sur le mode de réalisation de cet objet montrable "actif", lors de l'introduction de nos travaux sur les squelettes d'application.

Evaluation du modèle

Le modèle proposé se situe clairement dans la lignée des modèles présentés dans cette section. Outre les choix de propriétés présentés en section 3.3.1, ce modèle se caractérise par la définition d'une couche architecturale (les objets vue) isolant l'utilisation des bibliothèques graphiques en vue de la réalisation de la présentation de l'interface, et de la gestion locale de l'interaction. En ce sens, notre proposition dénote une approche "pratique" du problème, puisqu'elle tient compte des spécificités de l'environnement de réalisation des systèmes interactifs.

La couche des objets vue a deux intérêts :

- Elle permet d'isoler l'interface des bibliothèques graphiques, assurant un degré d'indépendance de l'interface vis-à-vis de la plate-forme de réalisation.
- Elle vise à favoriser la définition d'objets de présentation réutilisables.
  Un objet vue procède de l'abstraction de fonctions de présentation, et de la réalisation de ces fonctions par la combinaison d'objets interactifs. Ce procédé d'abstraction est la condition nécessaire (et cependant non suffisante) à l'obtention d'objets de présentation réutilisables - soit à

l'intérieur de la même interface, soit lors de la réalisation d'un autre système interactif.

3.4 Discussion sur les méthodes de développement : comment aller plus loin?

Les méthodes de développement des systèmes interactifs reposent sur l'utilisation de modèles d'architecture guidant la décomposition modulaire du code, et déterminant le type des communications entre les différents modules. La structuration obtenue garantit la facilité de maintenance du code et son indépendance à l'égard de l'environnement extérieur ; elle peut de plus favoriser la production de code réutilisable, élément particulièrement intéressant au regard du volume du code nécessaire à la mise en œuvre des interfaces les plus simples.

Nous avons dans cette section insisté sur un ensemble de grands principes dont l'adoption est semble-t-il dorénavant admise dans la communauté IHM. En précisant le mode d'application de ces principes, les modèles d'architecture offrent un indéniable support à la réalisation. Nous verrons par la suite comment les outils de développement utilisent ces modèles.

La force d'un modèle d'architecture est sa généricité. Sa faiblesse est son manque de précision. Le caractère général et théorique des éléments de structuration proposés rendent ces derniers difficiles à utiliser. Il suffit pour s'en persuader d'observer les difficultés rencontrées par les développeurs lors de l'application du modèle. Quel que soit le modèle (par exemple PAC, ou bien le nôtre), le problème de l'identification des agents est patent. Quel peut être le rôle d'un agent? Comment définit-on un agent? Comment déterminer si un seul agent remplit la fonction, ou bien s'il est préférable d'en définir plusieurs? Ce problème de l'identification des agents est un problème de fond, de l'ordre, par exemple, de celui de l'identification des objets dans une modélisation à objets : l'ajustement d'une structure abstraite pour représenter une situation concrète.

Pour dissiper l'imprécision d'un modèle d'architecture, et simplifier la tâche d'identification des agents d'un système, deux approches complémentaires peuvent être envisagées : développer une méthode d'application du modèle, ou bien pousser plus avant l'analyse des composants du système.

J. Coutaz décrit dans [Coutaz 91a], au travers d'un exemple, quelques règles heuristiques guidant la décomposition d'un système selon le modèle d'architecture PAC-Seeheim précédemment évoqué. Les règles énoncées sont trop spécifiques à l'exemple étudié pour que l'on puisse les généraliser, et véritablement parler de méthode de développement - de fait, ce terme n'est pas employé par J. Coutaz. On peut cependant considérer la formulation d'heuristiques comme un pas intéressant vers l'obtention de cette méthode ; reste à savoir jusqu'où l'on peut aller sur cette voie.

[Lisbon 90] rapporte une démarche de recherche allant dans le sens de l'analyse des composants du modèle d'architecture. Parmi les objets de contrôle de l'interface, sont identifiés deux types d'objets particuliers : les Transformateurs, gérant les relations d'intégrité entre les objets de l'adaptateur et les objets de présentation/interaction, ainsi que les relations de cohérence entre les objets de présentation/interaction ; les Moniteurs, superviseurs des différentes transactions survenant à l'intérieur du système, et dédiés à la mise en œuvre de fonctions "meta" telle l'aide, le défaire-refaire, etc. Cette tentative de classification des composants de contrôle reste timide. Il faut aller plus loin. Sans mettre en péril la généricité du

modèle, des rôles d'agents doivent être identifiés, porteurs d'une sémantique plus proche des besoins pratiques des développeurs.

Insuffler des notions pratiques dans un modèle d'architecture nécessite un langage d'expression de cette sémantique. Plus précisément, il est nécessaire de disposer d'une description des besoins du développeur pouvant servir de référence lors de l'expression de ces notions pratiques. La suite de ce document apportera notre réponse à ce problème.

4 Conclusion

Nous l'indiquions en introduction de ce chapitre, il n'existe pas de méthode couvrant l'ensemble des tâches du processus global de conception d'un système interactif. En ce qui concerne la stricte conception du système, nous disposons d'un ensemble varié de méthodes et modèles traitant de façon locale les différents aspects des activités de spécification ; c'est un foisonnement de propositions qui apparaît sans harmonie, faute de disposer d'une méthode unificatrice. Du côté de la réalisation, la recherche nous offre plusieurs modèles d'architecture guidant la structuration du code du système - parmi lesquels notre propre proposition.

Entre les méthodes de conception et les méthodes de développement, aucune passerelle. Deux mondes se côtoient en s'ignorant.

Les travaux rapportés dans ce document ont à cœur d'affronter ce fossé. Notre sentiment est que la modélisation conceptuelle d'un système peut apporter aux modèles d'architecture le cadre de référence dont nous soulignions le besoin en 3.4. La suite de ce mémoire décrit notre démarche en ce sens : la construction d'un outil faisant le pont entre modélisation conceptuelle et réalisation.

Chapitre III

Outils

1 Introduction

La section précédente explorait les méthodes et modèles offerts comme support théorique du processus de conception d'un système interactif. Nous abordons maintenant le support pratique de cette activité : les outils informatiques d'aide à la conception et au développement.

Une remarque préliminaire s'impose : les outils existants ont pour finalité la réalisation effective de l'interface du système interactif. Certains outils permettant la collecte de connaissances - tel MAD, précédemment cité, qui vise à la saisie structurée des connaissances sur les tâches d'un système, offrent un support informatique partiel à l'activité de conception d'une interface. A notre connaissance cependant, il n'existe pas d'outils logiciels véritablement dédiés à l'aide à la conception d'une interface.

Cette remarque étant faite, les moyens qu'offrent les outils pour parvenir à cette finalité de réalisation sont très diversifiés ; selon les cas, ces moyens peuvent ou non influencer l'activité de conception. L'objectif de cette section est de préciser la place des outils dans le processus général de conception d'un système interactif.

Il est courant de distinguer deux grandes catégories d'outils en IHM : les *outils de programmation* et les *outils de spécification*. Les premiers peuvent être ramenés à des bibliothèques logicielles utilisables lors de la programmation de l'interface; les seconds reposent sur une méthode de spécification (langage textuel ou méthode interactive), et comportent un générateur capable de produire le code nécessaire à la réalisation de l'interface spécifiée. Bibliothèques graphiques, "boîtes-à-outils" et squelettes d'application constituent la première catégorie d'outils ; langages spécialisés, grammaires, éditeurs interactifs font partie de la seconde catégorie d'outils.

Cette dichotomie classique dissimule cependant une grande disparité entre les outils, concernant notamment :

- le niveau d'abstraction des éléments manipulés par ces outils,
- le type et le nombre des services réalisés par ces outils,
- la façon dont ces outils s'insèrent dans le cycle de développement de l'application,
- l'influence éventuelle de ces outils sur l'architecture de l'application,
- etc.

Considérant que le niveau d'abstraction des concepts d'un outil détermine la plupart des caractéristiques de ce dernier, nous privilégions cette dimension dans notre analyse de l'espace des outils en IHM. L'intérêt de cette dimension, outre l'axe d'analyse qu'elle représente, est d'offrir des éléments de liaison avec les approches théoriques présentées au Chapitre II.

Cette section s'organise ainsi : après une définition des différents niveaux de concepts qu'un outil peut comporter, nous analysons quelques outils représentatifs de façon à déterminer les concepts dont ils sont porteurs. La confrontation des résultats de cette analyse nous amène à une évaluation des mérites respectifs de ces différents outils, relativement au cycle de développement. De cette évaluation se dégagent les caractéristiques d'un outil qui constitue l'objectif de nos travaux.

2 Quels concepts ?

Un outil offre un ensemble de fonctions (au sens large) accessibles au travers d'un langage d'expression : l'exploitation par le développeur d'une fonction d'un outil nécessite de pouvoir exprimer la fonction désirée, ainsi que d'éventuels paramètres pour cette fonction. Le langage d'expression peut prendre des formes très diverses : langage de programmation, langage de description déclarative, langage gestuel (spécification interactive), etc. Sans pour l'instant nous attacher à la forme de ce langage d'expression, ce sont les concepts manipulés dans ces langages que nous considérons ici.

Plutôt que de partir des outils, nous procédons à une analyse fine de la forme et du contenu d'une interface et des séquences de dialogue en général - c'est naturellement une analyse orientée vers la réalisation de l'interface qui est effectuée. L'objectif est d'isoler les concepts génériques existant à tous les niveaux d'une interface. Le résultat de cette analyse fournit un étalon que nous utiliserons pour mesurer la couverture des outils d'IHM.

Notre analyse nous conduit à distinguer sept grands types de concepts, dans un ordre d'abstraction croissant :

- concepts de gestion des ressources de l'interaction
- techniques d'interaction
- concepts de dialogue
- services d'interface
- concepts du domaine de l'application.
- concepts de gestion du dialogue
- concepts architecturaux

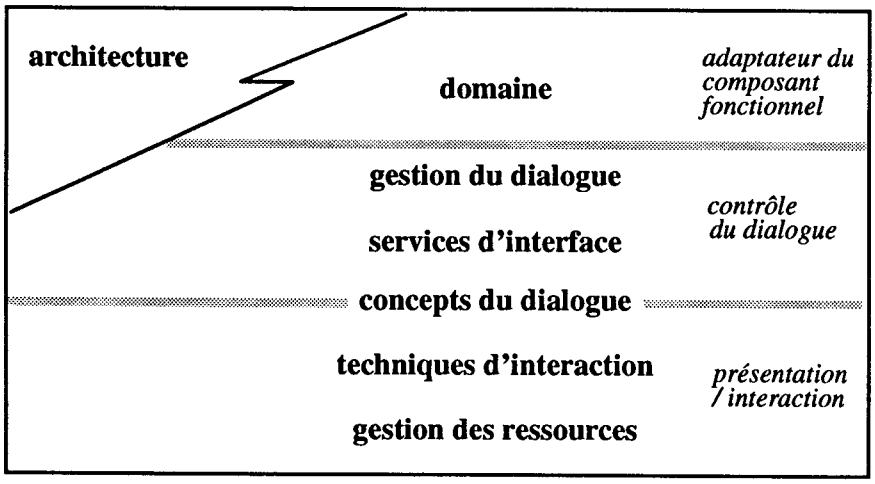


Figure III-1 *Les catégories de concepts des outils.*

La Figure III-1 représente ces sept catégories de concepts, en indiquant le composant de réalisation du système (cf. la section 3, Chapitre II) auquel elles peuvent grossièrement être rattachées. La suite de cette section reprend en l'explicitant chaque élément de cette figure.

2.1 Gestion des ressources de l'interaction

Nous désignons sous le terme de ressources de l'interaction l'ensemble des mécanismes élémentaires de gestion des entrées/sorties d'une interface. Dans un environnement classique, ces mécanismes concernent l'affichage textuel ou graphique à l'écran, ainsi que la détection des actions de l'utilisateur sur le clavier et la souris. L'avènement de techniques nouvelles étend notamment la gestion des ressources à l'acquisition de la parole et du geste, et à l'affichage de séquences vidéo.

La gestion des ressources de l'interaction repose sur de multiples couches logicielles soigneusement cloisonnées. Au niveau du système d'exploitation, les *pilotes* sont chargés de gérer chacun une classe de périphériques. Ces pilotes sont masqués par une couche logicielle, garante de l'indépendance des couches supérieures vis-à-vis du matériel ; elle définit un "terminal abstrait", base de réalisation de la gestion des ressources.

Nous décrivons les principaux concepts intervenant dans la gestion des ressources d'un environnement "classique", puis nous donnons quelques précisions relatives à l'utilisation des techniques nouvelles.

2.1.1 Ressources "classiques"

La gestion des ressources de l'interaction classiques est le fait des systèmes de fenêtrage (NeWS [SUN 87], X Window [Scheifler 86], etc.) et des machines graphiques (PostScript [Adobe 85], GKS [ISO 85], boîtes abstraites [Knuth 79, Quint 85], etc.). Sans entrer dans le détail de ces couches supérieures, nous listons les principaux concepts inhérents aux ressources de l'interaction : la surface d'affichage, la fenêtre, les événements, les requêtes graphiques.

La *surface d'affichage* est un concept permettant de s'abstraire des limites physiques de l'écran : un dispositif abstrait défini comme une surface dotée d'un système de coordonnées, et d'une forme précise (X Window [Young 89] propose la notion de "drawable", de forme strictement rectangulaire, tandis que NeWS offre des "canvas" de forme quelconque). Tout affichage a lieu sur une surface d'affichage, et non pas directement sur l'écran. Plusieurs surfaces d'affichage peuvent apparaître sur l'écran à la fois. Une surface peut être contenue dans une autre, dénotant des relations de hiérarchie entre surfaces, ou bien en recouvrir partiellement une autre. Pour ces raisons, la surface d'affichage est l'élément de structuration physique de l'image-écran d'une interface.

La *fenêtre* est l'élément de décomposition logique d'une interface. Selon les systèmes, une fenêtre est constituée d'une simple surface d'affichage (notamment dans X Window), ou bien d'un assemblage de surfaces d'affichage au sommet de la hiérarchie (sur Macintosh). Dans tous les cas, les fenêtres situées au sommet de la hiérarchie des surfaces d'affichage nécessitent une attention particulière. Réceptacles de l'image-écran d'une interface, elles constituent "l'enveloppe" d'un système interactif dans l'environnement du poste de travail. Leur manipulation par l'utilisateur (ouverture, fermeture, déplacement, etc.) est, dans la plupart des systèmes, prise en charge par un serveur commun à toutes les applications s'exécutant sur un même poste de travail : le *gestionnaire de fenêtres*.

La gestion des ressources de l'interaction s'effectue selon un modèle à *événements* (cf. Chapitre II, 2.2.2). Divers types d'événements peuvent intervenir, associés à une fenêtre de l'interface :

- Les événements d'entrée : frappe de touches, pression de boutons de la souris, déplacements élémentaires de la souris, etc.
- Les événements dénotant la modification d'un attribut d'une fenêtre : exposition d'une zone précédemment cachée (ce qui nécessite le réaffichage du contenu de cette zone), réception du "focus" du clavier, modification des caractéristiques d'une fenêtre (par exemple, changement de sa taille), etc.
- Les événements liés à une application. Par exemple, X Window et le serveur du Macintosh permettent la génération et réception d'événements propres à une application.

L'affichage de texte ou de graphiques sur une surface s'effectue au travers de concepts variés, que nous ne détaillerons pas ici - le lecteur trouvera une présentation de ces concepts dans [Coutaz 90].

2.1.2 Ressources "nouvelles"

Dans le domaine des techniques nouvelles, la gestion des ressources de l'interaction est encore "tâtonnante". Des solutions sont développées localement aux laboratoires de recherche sur ces techniques, sans réelle unité, et de façon "ad hoc" : pour répondre à des besoins précis et immédiats. Grâce à de récentes avancées, ces techniques commencent à sortir des laboratoires où elles ont été jusqu'à présent confinées - c'est tout particulièrement le cas pour les techniques de reconnaissance de la parole. Face à un besoin croissant de systèmes de gestion adaptés, la recherche commence à s'intéresser aux problèmes sous-jacents à l'utilisation de ces techniques : la définition de concepts prenant en compte toutes les particularités des nouvelles modalités, et l'intégration de ces concepts dans les environnements classiques existants, en vue de la gestion de l'interaction multimodale.

En ce qui concerne les techniques de reconnaissance de la parole, nous relevons un ensemble de travaux effectués dans le projet Sphinx, et décrits dans [Lunati 91], et [Normand 91b]. CM-SLS (The Carnegie Mellon Spoken Language Shell) est une plate-forme logicielle relativement complète qui met en œuvre le support de la parole dans un environnement graphique multifenêtré, assurant à la fois l'acquisition de la parole et la transmission à l'application du résultat de la reconnaissance.

Le concept de base est celui de *phrase reconnue*. Ce sont des phrases entières qui sont traitées, et non des mots. La transmission à l'application d'une phrase de l'utilisateur prend la forme d'un "*événement parole*" : la reconnaissance d'une phrase prononcée donne lieu à la génération d'un événement de type parole, porteur de la chaîne de caractères décrivant la phrase reconnue ; cet événement est inséré dans une file d'événements parallèle à celle du serveur graphique (en l'occurrence, NeXTStep), dont le traitement est effectué de façon analogue aux événements d'entrée classiques.

La génération des événements parole nécessite un ensemble de services afin d'assurer la *synchronisation* entre l'utilisateur et le système, la définition de l'application *focus* de la parole, et d'éventuelles procédures de *confirmation/correction* des résultats de la reconnaissance. (Bien que certains systèmes de reconnaissance, tel Sphinx, soient considérés comme robustes, la marge d'erreur dans la reconnaissance des phrases reste sensible.) CM-SLS met en œuvre ces différents services sous la forme de gestionnaires spécialisés,

s'exécutant sur le poste de travail de façon analogue au gestionnaire de fenêtres du serveur graphique.

Il est clair que l'approche adoptée dans CM-SLS impose des contraintes sur le style d'interaction et plus généralement sur le type des applications développées dans cet environnement. Le niveau de granularité des événements parole (la phrase reconnue) est adapté à l'expression de commandes ne nécessitant pas d'évaluation et de retour d'information ("feedback") au cours de leur formulation - typiquement, des requêtes sur des bases de données. Notre propre expérience d'utilisation de CM-SLS, décrite dans [Normand 91a], concerne la réalisation d'une application de traduction automatique ; un événement parole correspond à une phrase à traduire.

La construction d'interfaces réactives nécessite des événements parole de granularité fine. Il en va de même pour les interfaces multimodales combinant, par exemple, la souris et la parole pour la formulation des commandes. Dans ce dernier cas, interviennent également des problèmes de synchronisation entre événements de modalités différentes. Ces questions alimentent un courant de recherche actuel sur les événements multimodaux [Caelen 91].

2.2 Techniques d'interaction

Les techniques d'interaction utilisent les services de gestion des ressources. Les fonctions offertes par ces techniques d'interaction résultent de la définition d'éléments d'interface "standard" répondant aux besoins élémentaires du dialogue homme-ordinateur : boutons à pousser, menus, champs de saisie, etc. L'apparition des techniques d'interaction est indissociable de l'avènement des systèmes de gestion des ressources graphiques, évoqués dans la section précédente ; elle découle naturellement d'un souci d'abstraction visant à la réutilisation de composants logiciels, mais également à l'obtention d'interfaces homogènes.

La conception de ces composantes d'interface s'inspire à la fois des formulaires papier (titres, champs d'édition, fiches, etc.) et des outils mécaniques (boutons à presser, interrupteurs, témoins, jauges, etc.). En dépit d'une relative diversité des styles des interfaces actuellement produites, l'on peut noter un consensus général quant aux composants d'interaction utilisés, leurs fonctions dans l'interface, et leurs grandes caractéristiques. Nous relevons trois grandes catégories de techniques d'interaction :

- éléments affichés, non désignables.
  Il s'agit d'afficher à l'écran une information de nature textuelle, sous la forme d'une étiquette, ou bien de nature iconique, sous la forme d'une image.
- éléments de choix.
  Il s'agit de permettre la désignation par l'utilisateur d'un ou plusieurs éléments parmi une liste d'options. Les choix simples (un choix parmi deux options) s'expriment sous la forme de boutons à presser, ou de bascules à positionner. Les choix complexes 1-n (un choix parmi n options) s'effectuent à travers des menus ou des listes ; les choix complexes m-n (m choix parmi n) font intervenir des listes à choix multiples.
  L'utilisateur exprime son choix au moyen de la souris et/ou du clavier.
- éléments de saisie.

Il s'agit de permettre l'entrée par l'utilisateur de données textuelles ou numériques. Les éléments en jeu sont des champs de texte, des curseurs numériques, etc.

Les outils fondés sur les techniques d'interaction sont la grande majorité des outils d'IHM actuels. La raison en est simple : les techniques d'interaction constituent un ensemble clairement défini de "briques" de construction d'une interface utilisateur ; le caractère élémentaire de ces briques permet une grande souplesse dans la construction des interfaces ; si ces "briques" sont définies dans le contexte d'un "style" d'interface bien précis, elles sont néanmoins susceptibles d'être adaptées aux besoins propres d'une application grâce à un système de paramètrage bien développé (mais aussi difficile à maîtriser).

Dans la catégorie des outils de programmation, les "boîtes-à-outils" sont des bibliothèques de modules logiciels réalisant des techniques d'interaction (par exemple Motif [OSF 89a], AppKit de NeXTStep [Webster 89], la Toolbox du Macintosh, etc.).

En ce qui concerne les outils de spécification, nombreux sont les éditeurs interactifs d'interfaces directement construits au-dessus d'une boîte-à-outils, et permettant un assemblage interactif des briques de construction d'une interface (par exemple, UIMX de Hewlett Packard, Egérie [Egeria 90] de Bull, etc.).

Lapidary [Vander Zanden 91] représente une catégorie différente d'outils de spécification fondés sur les techniques d'interaction. Développé à l'intérieur du projet Garnet [Myers 89], cet éditeur graphique permet non seulement l'assemblage de techniques d'interaction mais également la spécification des techniques elles-même (présentation et comportement), au moyen d'interacteurs de niveau élémentaire.

2.3 Concepts du dialogue

Les concepts du dialogue relèvent d'un souci d'analyse et de décomposition du contenu d'une interface visant à identifier des constantes dans l'organisation de ces interfaces utilisateur, ainsi que des fonctions ou services communs à ces interfaces. Les concepts du dialogue définissent des règles de présentation et d'interaction, mais comportent également des aspects relatifs à la gestion du dialogue. Pour cette raison, la Figure III-1 représente ces concepts à la frontière des composants de contrôle du dialogue et de gestion de la présentation et de l'interaction. Nous relevons deux grands ensembles de concepts du dialogue portant respectivement sur l'organisation du dialogue et sur la mise en œuvre de sous-dialogues spécialisés.

2.3.1 Concepts organisateurs

Les concepts organisateurs sont généralement définis dans le contexte d'un *guide de style*.

Un guide de style est une liste de recommandations plus ou moins précises, relatives notamment au choix des composants d'une interface, et dont le but est de rendre homogènes les interfaces des applications. Divers guides de style ont été définis, pour la plupart dans le contexte de boîtes-à-outils. Notons le guide de style Open Look du groupement SUN Microsystems et AT&T, initialement proposé indépendamment d'un outil logiciel.

Les concepts organisateurs apparaissent dans certains guides de style (notamment celui de Motif). Ils proposent une organisation "standard" de certains aspects de l'interface en définissant par exemple les notions de :

- fenêtre principale et fenêtres secondaires,
- zone de présentation des principales commandes de l'application, (barre de menus au sommet d'une fenêtre pour Motif, menu vertical pour NeXTStep, etc.)
- zone de message.

Si ces concepts ne s'expriment bien souvent que sous la forme de règles de style plus ou moins précises et informelles destinées à guider les choix d'assemblage de techniques d'interaction, ils peuvent donner lieu à des modules logiciels prêts à l'emploi, super-éléments d'une boîte-à-outils – c'est notamment le cas pour Motif (widget "fenêtre principale") et NeXTStep.

2.3.2 Concepts de sous-dialogues

Le dialogue homme-ordinateur se caractérise par une séquence d'interactions initialisées par l'un ou l'autre des deux partenaires - dans le cadre d'un dialogue multifil, plusieurs séquences peuvent être menées en parallèle. L'observation d'une séquence de dialogue révèle des phénomènes de nature générique : les sous-dialogues. Un *sous-dialogue* dénote l'occurence d'interactions en dehors de la séquence principale du dialogue, en vue de traiter une situation particulière, avant de reprendre et poursuivre la séquence principale.

Selon le domaine de l'application, ou le type d'interface proposé, différents sous-dialogues pourront survenir. Nous nous intéressons ici aux sous-dialogues génériques que constituent les demandes de confirmation ou d'annulation : de par la séquence de dialogue précédente, ou bien en raison d'un événement de l'environnement extérieur, le système se retrouve dans une situation requérant une confirmation ou une annulation de l'utilisateur. Parmi ces situations, citons :

- les situations d'erreur.
  Les entrées précédentes de l'utilisateur sont incorrectes, ou bien un événement extérieur s'est produit (par exemple, le système de fichiers est saturé, et le système a besoin de créer un nouveau fichier).
- les situations à risque.
  Un risque de modification irréversible des données est détecté par le système.
- l'affichage d'informations annexes.
  Le système présente à l'utilisateur des panneaux d'information, et attend une confirmation pour supprimer l'affichage.

Selon la situation en cours, la demande de confirmation/annulation sera plus ou moins pressante, et l'utilisateur sera (demande modale) ou non contraint de donner une réponse avant de pouvoir poursuivre le dialogue.

Ces éléments génériques du dialogue sont pris en compte dans certains outils. La boîte-à-outils Motif propose par exemple la notion de "fenêtre de dialogue", fenêtre arborant un message textuel, une icône, et un, deux ou trois boutons (par exemple : "OK", "Cancel", "Help"), et qui répond aux besoins précis de communication directe, éventuellement modale, entre le programme et l'utilisateur.

2.4 Services d'interface

La section précédente traitait des aspects génériques d'une séquence de dialogue ; c'est le contenu fonctionnel d'une interface qui est examiné ici. L'observation d'une interface conduit à décomposer ses commandes en deux groupes : les *commandes de l'application*, et les *commandes de l'interface*[1], dont la fonction est de permettre la manipulation par l'utilisateur de cette interface : création de vues sur des données, "zoom" sur des données, changement de paramètres d'affichage, etc. (Rappelons que les commandes de manipulation des fenêtres englobantes de l'interface d'un système : iconification, ouverture, déplacement, retaillage, sont prises en charge par le gestionnaire de fenêtres évoqué en 2.1.) Parmi ces commandes de l'interface, il est un ensemble de fonctions indépendantes du domaine ou du type de l'interface, que l'on retrouve (ou souhaite retrouver) de façon générique d'interface en interface ; nous appelons ces fonctions *services d'interface*.

Parmi les services d'interface :

- la sortie de l'application (!).
- le copier-couper-coller, intra et inter-applications.
  Ces trois classiques fonctions d'édition ont un double intérêt : elles facilitent l'édition, mais également permettent la transmission de données entre applications.
- le défaire-refaire.
- le service d'aide statique.
  L'aide statique consiste à associer des données d'information immuables à chaque élément d'une interface, et à afficher ces données sur la demande de l'utilisateur. Nous n'indiquons pas l'aide contextuelle comme un service d'interface, car non indépendante de l'état de l'interaction.

Nous verrons par la suite que les services d'interface sont encore mal réalisés dans la plupart des outils. Nous pouvons dès à présent relever le presse-papier Macintosh ou NeXTStep, base des fonctions copier-couper-coller, le service d'information sur l'application existant dans NeXTStep, etc.

2.5 Concepts du domaine de l'application

Les concepts du domaine de l'application font intervenir les notions de données et d'opérations du système ; ce sont ces opérations qui servent de base aux commandes d'application évoquées dans la section précédente.

Nous verrons que très peu d'outils permettent la manipulation de tels concepts ; certains outils permettent l'expression des structures de données de l'application, notamment sous la forme de bases de données actives (par exemple, Serpent [Bass 88]) ; seuls les outils de spécification dite fonctionnelle permettent véritablement la manipulation de concepts de données et de commandes de l'application : UIDE [Foley 88], MacIDA [Petoud 90].

---

1. Ces commandes de l'interface sont parfois appelées commandes "syntaxiques" dans la littératures ; l'on trouve également le terme de commande "passive" (cf. section 2.3.2, Chapitre II).

2.6 Concepts de gestion du dialogue

La gestion du dialogue nécessite la mise en œuvre de mécanismes évoqués en section 3 du Chapitre II. Parmi ces mécanismes, certains comportent une large part de généricité : ce sont nos concepts de gestion du dialogue. Citons notamment :

- le maintien de la cohérence entre données et image de l'application.
- la mise en œuvre des traitements de l'application.

Les outils comportant des concepts de gestion du dialogue sont peu nombreux. Certains squelettes d'application offrent ces concepts. Ce type de concept est également présent dans la plupart des outils de spécification textuelle. Ainsi Serpent et ses données actives, UIDE, ITS [Boies 88], etc.

2.7 Concepts architecturaux

Le terme de concept d'architecture désigne un concept visant à guider la tâche de programmation de l'interface d'une application. Ce type de concept est lié à la notion de modèle d'architecture présentée au Chapitre II.

Les squelettes d'application sont des outils privilégiant ce type de concepts. Citons MacApp [Schmucker 86] et notre propre contribution [Normand 90b].

La plupart des outils de spécification n'intègrent pas ce type de concepts puisque par définition ces outils ont pour charge de produire le code de support de l'interface. Suivant le niveau d'abstraction des concepts maniés par le langage de spécification, des concepts d'architecture peuvent néanmoins transparaître. Ainsi Serpent, dont le langage de spécification Slang a la puissance d'expression d'un langage de programmation traditionnel, offre le concept de "View Controller", agent de contrôle, concept de structuration d'une spécification Slang.

2.8 Récapitulatif

Concepts	*Rôle*	*Exemples*
gestion des ressources	gestion des entrées/sorties	*surface d'affichage, fenêtre, événement, requête graphique*
techniques d'interaction	composants standard de l'interface	*bouton, menu, champ d'édition, icône, étiquette*
concepts du dialogue	organisation de l'interface sous-dialogues génériques	*fenêtre principale/secondaire barre de commandes demande de confirmation*
services d'interface	fonctions génériques de l'interface	*copier-couper-coller aide défaire-refaire*
gestion du dialogue	mécanismes génériques de gestion de la dynamique du dialogue	*maintien cohérence données/image enchaînements*
domaine de l'application	concepts du domaine	*données et opérations du domaine*
architecture	structuration du code de réalisation du système interactif	*composants d'un modèle d'architecture*

Figure III-2 *Les catégories de concepts des outils : récapitulatif.*

Au terme de cette longue analyse, la Figure III-2 récapitule les sept catégories de concepts mises en lumière. Sont succintement présentés le rôle de chaque catégorie de concepts, ainsi que quelques exemples représentatifs.

La section suivante utilise les résultats de cette analyse pour confronter les outils d'IHM.

3 Quels concepts pour quels outils ?

Cette section examine un sous-ensemble représentatif des outils d'IHM, de façon à déterminer la couverture des concepts réalisée par chacun de ces outils. L'étude aborde dans un premier temps les outils de programmation, et dans un second temps les outils de spécification.

3.1 Outils de programmation

Les outils de programmation offrent des services qui ne sont utilisables par le développeur que lors de la programmation du système. Le langage d'expression associé à ces outils admet deux formes :

- l'appel de fonctions logicielles.
  Ces outils sont alors des bibliothèques logicielles. On distingue les *bibliothèques* de base, et les bibliothèques de composants d'interface, dénommées *boîtes-à-outils.*
- le complément de structures de code, ou bien le sous-classage, dans le cadre d'un environnement de programmation à objets.
  Ces outils sont alors des *squelettes d'application.*

Notre étude porte sur un sous-ensemble de quatre outils de programmation, cités au cours de la section précédente : la bibliothèque graphique Xlib, la boîte-à-outils Motif, et deux squelettes d'application, MacApp, et le squelette Guide, résultat de nos propres travaux.

Les outils de programmation étudiés

Xlib [MIT 87] est une bibliothèque graphique de base dans l'environnement X Window.

Motif [OSF 89a] est une boîte-à-outils construite dans l'environnement X Window (en utilisant notamment Xlib).

MacApp [Schmucker 86] est un squelette d'application de l'environnement Macintosh. Conçu pour simplifier l'utilisation de la Toolbox (boîte-à-outils) Macintosh, MacApp factorise un ensemble de services à l'intérieur d'objets surchargeables par le programmeur. L'utilisation de MacApp par le programmeur s'effectue conjointement avec celle de la Toolbox.

Le **squelette Guide** [Normand 90b, Normand 90c] est le résultat de nos travaux sur les modèles d'architecture des systèmes interactifs. Ce squelette s'insère dans l'environnement de développement Guide [Krakowiak 90], et véhicule le modèle d'architecture décrit en 3.3.2, au Chapitre II. Il utilise (de façon directe) Xlib et Motif. Son utilisation par le programmeur s'effectue conjointement avec celle des bibliothèques X Window.

L'étude

La table présentée en Figure III-3 détaille les concepts véhiculés par chacun de ces outils de programmation.

	bibliothèques	*boîtes-à-outils*	*squelettes d'application*	
Concepts	*Xlib*	*Motif*	*MacApp*	*squel. Guide*
gestion des ressources	(par définition)	—	—	—
techniques d'interaction	—	(par définition)	—	—
concepts du dialogue	—	style "Motif" barre de commandes fenêtres de dialogue	style "desktop" du Macintosh boîtesde dialogue de la Toolbox	—
services d'interface	—	—	presse-papier défaire-refaire	presse-papier
gestion du dialogue	—	—	—	cohérence données / image
domaine de l'application	—	—	objetsTDocument	objets montrables
architecture	—	—	objets TApplication, TDocument, TView	objets de contrôle, objets montrables, objets vue

Figure III-3 *Les concepts des outils de programmation.*
(Un "-" dans une case signifie que l'outil ne véhicule pas le concept correspondant.)

L'on constate une grande diversité dans la couverture des concepts : à côté des outils très ciblés que sont les bibliothèques graphiques et les boîtes-à-outils, les squelettes d'application sont porteurs d'une gamme (restreinte) de concepts de niveau d'abstraction élevé (partie inférieure du tableau) ; cette différence est caractéristique de la complémentarité entre ces deux ensembles d'outils.

Xlib, par définition, ne véhicule que des concepts de gestion des ressources.

Motif, par définition, véhicule des techniques d'interaction. Motif est également porteur de concepts de dialogue : Motif est muni de composants d'interface complexes réglant l'organisation générale de l'interface (fenêtre principale, barre de menus, zones de travail, etc.), ainsi que de composants offrant un support aux demandes de confirmation/annulation (fenêtres de dialogue).

Les **squelettes d'application** ont la caractéristique de véhiculer des concepts architecturaux. MacApp est porteur d'un modèle d'architecture assez simple, et sujet à controverse (cf. [Coutaz 90]). Le squelette Guide offre des classes d'objets correspondant aux différents éléments du modèle d'architecture exposé au Chapitre II. Les concepts du domaine de l'application apparaissent dans les squelettes d'application au travers des éléments correspondants dans les modèles d'architecture sous-jacents.

La seconde caractéristique des squelettes d'application est de factoriser la réalisation de services d'interface. MacApp offre un support au presse-papier, et au défaire-refaire ; le squelette Guide offre la notion de presse-papier.

En ce qui concerne les concepts de dialogue, le squelette Guide laisse entière liberté au développeur. MacApp, au contraire, véhicule l'organisation "desktop" du Macintosh.

Le squelette Guide est porteur d'un concept de gestion du dialogue relatif au maintien de la cohérence entre les données et l'image. Ce concept prend la forme d'un mécanisme de propagation des modifications survenant dans les objets montrables du système, de façon à ce que les objets de contrôle gérant ces objets soient avertis de ces modifications. Mis en œuvre au niveau des super-classes des objets de l'interface, ce mécanisme est transparent au programmeur de l'application ; incombe à ce dernier le respect d'un protocole d'enregistrement des objets de contrôle, et d'expression d'une modification.

3.2 Outils de spécification

Les outils de spécification offrent des services dont l'utilisation s'effectue en dehors de la tâche de programmation du système. Le langage d'expression associé à ces outils peut être de nature textuelle, ou bien gestuelle.

Les langages d'expression textuels peuvent être très divers : diagrammes de transition, grammaires, langages à événements, langages déclaratifs, etc. Le lecteur trouvera dans [Nanard 90] une classification des principaux outils de spécification selon la forme de leur langage d'expression.

Les langages d'expression gestuelle permettent une spécification interactive, par manipulation graphique.

Notre étude porte sur un sous-ensemble de cinq outils de programmation, dont certains ont été précédemment cités : les éditeurs interactifs Egérie et Interface Builder, et les langages de spécification Serpent, ITS, et UIDE.

Les outils de spécification étudiés

Egérie [Egeria 90] est un éditeur représentatif d'une large classe d'outils de spécification interactive d'interface. Reposant sur Motif, Egérie permet la construction interactive de l'image d'une interface par la composition de techniques d'interaction issues de Motif.

Interface Builder[Webster 89] est un éditeur interactif de l'environnement de développement NeXT. Cet éditeur assure des fonctions analogues à celles d'éditeurs classiques tel Egérie, mais offre également un support à l'expression d'une partie de la dynamique de l'interface.

Serpent [Bass 88] est un système de développement comportant un langage spécialisé dont la puissance est celle d'un langage de programmation classique. Serpent assure la spécification de l'interface en entier : présentation mais aussi contrôle du dialogue, ainsi que les données de l'adaptateur du composant fonctionnel.

ITS [Boies 88] est un système complexe dont nous ne retenons ici que la partie relative à la description du contenu de l'interface utilisateur d'une application ; cette description s'effectue dans un langage déclaratif.

UIDE [Foley 88] est un système fondé sur un langage déclaratif de spécification fonctionnelle de l'interface.

L'étude

La table présentée en Figure III-4 détaille les concepts véhiculés par chacun de ces outils de spécification.

	spécification interactive		*spécification textuelle*		
Concepts	*Egérie*	*Int. Builder*	*Serpent*	*ITS*	*UIDE*
gestion des ressources	—	—	—	—	—
techniques d'interaction	concepts de Motif	concepts de NeXTStep	concepts d'une boîte-à-outils paramétrable	formulaires, listes, choix, blocs d'information, etc.	—
concepts du dialogue	concepts de Motif (style non intégré)	concepts de NeXTStep style intégré	concepts de la boîte-à-outils paramétrable	session, cadres	—
services d'interface	—	presse-papier, information statique	—	historique	—
gestion du dialogue	—	une part de dynamique	cohérence données/image création/destruct	séquencement des cadres	—
domaine de l'application	—	—	données actives	tables de données	objets, opérations et paramètres
architecture	—	—	contrôleurs de vue	—	—

Figure III-4 *Les concepts des outils de spécification.*
(Un "-" dans une case signifie que l'outil ne véhicule pas le concept correspondant.)

L'on constate ici encore une grande diversité, sans cependant que la dichotomie selon le type du langage d'expression ne transparaisse dans la table : que la spécification soit interactive ou non, les concepts manipulés sont bien souvent de même nature. Seul UIDE apparaît isolé dans le tableau - nous le commenterons séparément.

Aucun outil de spécification ne présente de concepts de *gestion des ressources*. Tous les outils (sauf UIDE) sont par contre porteurs de concepts du niveau des *techniques d'interaction*. Aucun outil ne permet la création de techniques d'interaction ; tous reposent sur la composition de techniques d'interaction pré-définies. Egérie et Interface Builder permettent la composition interactive des techniques d'interaction intervenant dans la présentation d'une interface. Tous deux, et de façon générale tous les éditeurs interactifs sont fondés sur une boîte-à-outils (en l'occurence, Motif et NeXTStep). Serpent et ITS offrent également des techniques d'interaction, en assurant cependant un degré d'indépendance vis-à-vis des logiciels graphiques de réalisation de ces techniques. Serpent est paramétrable par une

boîte-à-outils ; les techniques d'interaction manipulées sont celles de la boîte-à-outils utilisée. ITS va nettement plus loin dans l'indépendance : ITS permet la manipulation de techniques d'interaction génériques. Ces techniques (formulaires, listes, choix, et blocs d'information) dénotent les composants génériques d'une interface ; la présentation de ces techniques génériques est déterminée séparément ; leur réalisation au travers de techniques d'interaction effectives fait l'objet d'une génération automatique.

En ce qui concerne les *concepts de dialogue*, les outils héritent de ceux des boîtes-à-outils sous-jacentes. Seul Interface Builder intègre des règles d'organisation de l'interface (guide de style) favorisant l'obtention d'interfaces homogènes.

Les *services d'interface* sont mal assurés. Interface Builder hérite des services de l'environnement NeXTStep, notamment, le presse-papier, et le dispositif d'information générale sur une application, mais aussi le mécanisme d'exportation de services d'une application vers une autre application[1]. ITS propose des services d'historique d'une session - le contenu de cet historique n'est pas décrit dans la littérature dont nous disposons.

En dehors des éditeurs simples tel Egérie, les outils apportent quelques *concepts de gestion du dialogue*. Interface Builder permet l'expression d'une partie restreinte de la dynamique d'une interface : l'image affichée au lancement de la session, et, au travers de la spécification interactive des procédures appelées lors des actions de l'utilisateur sur les techniques d'interaction[2], une partie de la dynamique de séquencement des fenêtres. ITS permet de même l'expression du séquencement des fenêtres (au travers de la notion de "frame"). Serpent permet l'expression d'une part de dynamique sous la forme de conditions déclenchant l'instanciation[3] ou la destruction d'entités de l'interface (les contrôleurs de vue). Ce mécanisme d'instanciation d'entités décrites de façon générique est particulièrement intéressant puisqu'il permet notamment la spécification non exhaustive des configurations possibles pour une interface, à l'exécution : par exemple, Serpent permet de spécifier qu'une même fenêtre pourra être instanciée autant de fois que le demande l'utilisateur. Serpent offre de plus un mécanisme facilitant la gestion de la cohérence entre données et image, par la notion de donnée active, dont les modifications sont automatiquement communiquées à l'interface.

Les *concepts du domaine de l'application* apparaissent dans Serpent et ITS, sous la forme, respectivement, de données actives et de tables de données. En ce qui concerne les

1. Ce dernier concept est particulièrement intéressant car il permet l'obtention d'applications intégrées. Prenons pour exemple l'application Webster disponible sur la machine NeXT. Cette application permet l'accès "en ligne" au dictionnaire de même nom, et la recherche de la définition de mots anglais. L'accès au dictionnaire Webster peut être réalisé depuis le shell, mais également à l'intérieur de toute autre application, pour rechercher la signification d'un mot apparaissant dans le contexte de cette application ; l'appel de ce service est alors effectué par l'utilisateur depuis l'interface de l'application, le mot recherché étant le mot courant sélectionné.

2. Par exemple, on indiquera de façon graphique que l'enfoncement de tel bouton de l'interface donnera lieu à l'appel de telle procédure.

3. Rares sont les outils qui offrent ces facilités d'instanciation. La plupart (notamment Egérie) requièrent une spécification exhaustive de toutes les instances d'éléments pouvant intervenir dans l'interface à un moment donné. Interface Builder permet la définition d'entités génériques (notion de "module"), mais l'instanciation et la destruction de ces entités est à la charge du programmeur. En ce qui concerne ITS, la littérature n'est pas claire.

opérations du domaine, la spécification est moins nette. Par exemple, la notion d'action apparaissant dans ITS peut se rapporter aussi bien à une procédure du composant fonctionnel ou bien à une procédure de contrôle du dialogue.

UIDE est un outil de spécification dont la particularité est de ne comporter que des concepts du domaine de l'application. UIDE propose un langage de spécification des informations relatives à ces concepts, et prend en charge la génération automatique de l'interface à partir de ces informations.

Les *concepts architecturaux* sont absents de la plupart des outils. C'est naturel en ce qui concerne les outils tels ITS et UIDE, qui visent à la génération de l'ensemble de l'interface. Serpent, bien que faisant également partie de cette catégorie d'outils, offre un langage d'expression proche d'un langage de programmation ; Serpent propose le concept de contrôleur de vue, agent de structuration d'une spécification, analogue aux agents des modèles d'architecture tel PAC.

4 Discussion

Après avoir défini les différents niveaux de concepts intervenant dans le support pratique du développement d'un système interactif, nous avons montré comment les outils actuels véhiculent ou non ces concepts. A partir de ces résultats, nous cherchons à préciser la place que peut occuper un outil dans le cycle de conception ainsi que, de façon générale, l'influence d'un outil sur le cycle de développement.

4.1 Outils et conception

En dépit de sa finalité de réalisation, un outil peut-il intervenir dans l'activité de conception d'un système interactif ? Les concepts véhiculés par un outil fournissent des arguments de réponse à cette question.

4.1.1 Les concepts des outils et l'activité de conception du système

Les concepts définis en section 2 résultent d'une analyse pratique du contenu et de la forme d'une interface. Comment cette analyse pratique se situe-t-elle par rapport aux approches théoriques présentées au Chapitre II ? Si les concepts isolés sont clairement orientés vers la réalisation du système interactif, certains d'entre eux n'en présentent pas moins un intérêt pour l'activité de conception. L'apport de ces concepts s'inscrit notamment dans la dimension générique dont nous déplorions l'absence en section 2.5 du Chapitre II.

Les concepts de gestion des ressources sont trop proches du fonctionnement du système pour intervenir au niveau de la conception. De même, les concepts architecturaux sont par nature dédiés à la réalisation du système.

Les techniques d'interaction offrent des éléments descriptifs de l'image d'une interface utilisables lors de la *spécification externe*. De même pour les concepts du dialogue - à un niveau de généricité encore supérieur.

Les concepts de gestion du dialogue dénotent une dynamique exprimée lors de la *spécification du dialogue*. Notons cependant que les concepts de gestion du dialogue expriment cette dynamique dans les termes du fonctionnement du système, et non pas selon le point de vue de l'utilisateur.

Les services d'interface dénotent des fonctions génériques qui n'apparaissent généralement pas explicitement dans les modèles de représentation d'un système. Ces services représentent des tâches du niveau de l'interface et non pas du domaine de l'application. Leur définition intervient a priori lors de la *spécification conceptuelle*. D'autre part, les concepts du domaine peuvent être perçus comme les données et opérations définies lors de la spécification conceptuelle du système.

4.1.2 La place d'un outil dans le processus de conception

C'est ici que la nature du langage d'expression de l'outil prend une importance déterminante : un langage de programmation est trop complexe pour être utilisable lors de la conception. Seuls les langages de spécification sont propices à l'utilisation des outils lors de l'activité de conception.

Un outil peut donner lieu à deux types d'utilisation : comme formalisme de représentation d'une interface, et comme outil de prototypage des choix de conception. Dans

les deux cas, c'est un support partiel de l'activité de conception qui est réalisé. Deux familles d'outils peuvent être distinguées en fonction du support offert aux activités de conception :

- les outils axés sur les concepts du domaine,
- les outils axés sur les techniques d'interaction.

Les outils axés sur les concepts du domaine interviennent au niveau de la spécification conceptuelle. UIDE est un outil représentatif de cette famille. Axé sur les concepts du domaine, UIDE offre une notation utilisable par le concepteur pour spécifier (partiellement) les entités fonctionnelles de son modèle conceptuel. La génération automatique d'une présentation à partir de ces spécifications permet un prototypage très rapide de l'interface. Le contrôle des choix de présentation - apparentés à la spécification externe - est souvent fourni au travers d'un éditeur permettant la modification du résultat de la génération.

Les outils de spécification fondés sur les techniques d'interaction (notamment les éditeurs interactifs) sont particulièrement intéressants dans le contexte de la spécification externe : ils remplacent avantageusement les dessins et croquis de représentation de l'image de l'interface, mentionnés au Chapitre II. Dotés de concepts de gestion du dialogue (c'est par exemple le cas pour Interface Builder), ils peuvent être utilisés pour construire des prototypes reflétant une partie (élémentaire) de la dynamique de l'interface ; dans la pratique, les limites des outils sont très rapidement atteintes dès lors que l'on s'intéresse à la dynamique de l'interface, et il faut alors recourir à la programmation.

Certains outils de cette seconde famille véhiculent des concepts du domaine dont la spécification permet de capter les aspects fonctionnels ; l'intérêt est alors de pouvoir définir le lien entre un concept du domaine et les techniques d'interaction le représentant, explicitant les racines des choix externes dans les choix fonctionnels.

4.2 Influence sur le cycle de développement

Il est clair que le choix du niveau d'abstraction des concepts d'un outil implique un compromis entre souplesse d'utilisation d'une part, facilité d'emploi et temps de développement, puissance des fonctions offertes d'autre part. Le développement d'une interface à partir de concepts de gestion des ressources laisse au développeur une marge de manœuvre maximale, tandis que l'utilisation de techniques d'interaction ou de concepts de dialogue pose des contraintes sur le travail de ce dernier : le développeur mène sa tâche dans les termes que lui impose l'outil. D'un autre côté, le développement d'une interface à partir de concepts de gestion des ressources nécessite une maîtrise parfaite de quelques centaines de fonctions logicielles, et implique un temps de développement considérable - ce que les outils de plus haut niveau d'abstraction s'efforcent de combattre.

La Figure III-5 donne un résumé des avantages et inconvénients apportés par chacun des différents types de concepts. Nous listons quatre propriétés recherchées lors du développement d'une interface :

- La souplesse permise dans la définition de l'interface, c'est-à-dire la marge de manœuvre accordée au développeur dans l'expression de ses souhaits - le degré de finesse du contrôle laissé au développeur sur le résultat.
- L'homogénéité des interfaces construites.
- La limitation du temps de développement.

- Les facilités de maintenance du logiciel résultant.

N.B : cette évaluation des différents types de concepts se veut indépendante des outils mettant en œuvre ces concepts - d'où le caractère indicatif des résultats de cette évaluation. C'est une tendance qui est évaluée. Il est clair que les caractéristiques propres des outils ont une influence importante dont il est fait ici abstraction ; notamment, le temps de développement d'une interface sera grandement modifié selon que l'on considère un langage de programmation ou un éditeur interactif.

	souplesse des résultats	*homogénéité*	*temps de dév. court*	*maintenance facilitée*
gestion des ressources	++++	-	-	-
techniques d'interaction	++	++	++	
dialogue	+	++++	+++	
services d'interface	+	++++	+++	
gestion du dialogue	+	+	++	++
domaine de l'application	-*	++++*	++++*	+
architecture			+	++++

Figure III-5 *avantages et inconvénients inhérents aux types de concepts.*

* : lorsque les concepts du domaine sont utilisés pour générer l'interface.
(Un "+" signifie que le type de concept considéré tend à favoriser l'obtention de la propriété courante ; un "-" indique le contraire ; une case vide signifie que le concept considéré n'a pas d'influence immédiate sur la propriété.)

Au regard du volume de code nécessaire au développement d'une interface utilisateur, la puissance des fonctions offertes par un outil est une caractéristique fondamentale. Il n'est aujourd'hui plus envisageable de développer une interface à partir de seules fonctions d'entrée-sortie. Les outils actuellement les plus nombreux, et les plus couramment utilisés sont des outils de techniques d'interaction, pour les raisons citées en 2.2.

Les concepts d'interface, et tout particulièrement les services fonctionnels d'interface sont encore peu répandus. Une des raisons de cet état des choses provient du fait que la mise en œuvre de ce niveau de concepts implique la normalisation de certains aspects des interfaces. D'autre part, la réalisation de services d'interface nécessite une véritable plate-forme d'exécution des applications ; la réalisation d'un "presse-papier" par exemple suppose la définition et la mise en œuvre de conventions d'écriture et de protocoles d'exécution des applications s'exécutant sur une même station de travail (notamment, conventions quant au type des données contenues dans le presse-papier, et quant aux transformations de types possibles).

4.3 Conclusion

En reprenant les différents aspects du processus de conception d'un système, nous effectuons la synthèse des contributions et limitations qu'offrent les outils actuels.

La *spécification conceptuelle* reçoit un support très limité. D'une part, les outils axés sur les concepts du domaine sont à l'heure actuelle peu nombreux, et constituent un domaine de recherche encore mal exploré ; les outils les plus répandus sont ceux qui offrent la notion de concept du domaine comme point d'ancrage des techniques d'interaction. D'autre part, les concepts du domaine véhiculés par les outils actuels ne permettent de capter que certains aspects fonctionnels d'un modèle conceptuel : la structure des données et des opérations offertes dans un système, laissant dans l'ombre une large partie des informations de conception.

La *spécification externe* reçoit un très large support, puisqu'elle constitue l'objectif de la majorité des outils d'IHM. Au-delà de la spécification de l'image statique d'une interface, il doit être possible d'indiquer une part de la dynamique du dialogue. L'homogénéité des interfaces doit être préservée, tout en réalisant un compromis sur la finesse du contrôle de la spécification.

Que la spécification soit conceptuelle ou externe, un travail d'extraction des *aspects génériques* a été accompli, qui a conduit à la définition des techniques d'interaction, concepts de dialogue ou services d'interface. L'intégration de ces concepts génériques (notamment les concepts de dialogue et les services d'interface) au sein des outils est loin d'être achevée. D'autres développements sont nécessaires dans ce domaine.

Le faible support de la spécification conceptuelle, allié aux facilités de spécification externe offertes dans un grand nombre d'outils, peut à nos yeux présenter un danger quant à la qualité des interfaces et systèmes produits. Le développeur peut en effet être entraîné à négliger les aspects conceptuels d'un système au profit de ses aspects externes.[1]

Le *prototypage rapide* est un objectif primordial. Au-delà de la phase de prototypage, la réalisation du système nécessite des facilités de maintenance du code que seuls des *principes architecturaux* peuvent assurer. En effet, même dans le cas d'un outil de génération automatique d'interface, reste à la charge du programmeur le soin de gérer une part de dynamique de l'interface, ou bien simplement le rattachement du composant fonctionnel au code de l'interface. L'absence de principes architecturaux dans une grande partie des outils (notamment, les éditeurs interactifs) peut conduire à des situations dangereuses.[2]

1. Cette réflexion est issue de notre expérience d'utilisation et d'observation de l'utilisation de Interface Builder, sur la machine NeXT.

2. Ici encore, c'est l'observation de l'utilisation de Interface Builder qui nous conduit à cette remarque. Les développeurs observés, peu expérimentés et non munis d'une méthode de développement, ont tendance à réserver leur attention aux détails de présentation plutôt qu'à se soucier de l'organisation du code. Cette attitude peut conduire à des résultats aberrants, telle la programmation du code de gestion de l'interface en un seul gros objet, etc.

Paradoxalement, la grande souplesse et simplicité d'utilisation d'un outil tel Interface Builder, peut conduire à des résultats désastreux sur les plans conceptuel et architectural. Par contre, allié à des principes architecturaux, Interface Builder constitue un outil de production inégalé.

5 Notre démarche

Au terme de notre analyse de l'existant, c'est vers un outil de développement axé sur une représentation conceptuelle de l'interface que nous orientons nos recherches.

Nos objectifs trouvent racine dans nos observations sur les méthodes de conception et les modèles d'architecture, rapportées au Chapitre II, et dans notre étude des outils d'IHM et de leurs concepts, développée précédemment dans ce chapitre. Ces objectifs sont triples : le support à la conception, le pont entre modèle de conception et modèle de réalisation, et la génération automatique d'interface.

Nos travaux s'inscrivent dans le cadre de l'environnement de développement Guide, dont nous décrivons plus loin les principales caractéristiques. Les systèmes interactifs visés sont des applications axées sur les données plutôt que sur les traitements. Typiquement, ce sont des applications de bureautique (par exemple, agenda électronique, messagerie, etc.). Les interfaces de ces applications sont relativement simples, l'interaction reposant sur les entrées clavier et la désignation directe d'éléments par la souris.

La démarche que nous adoptons repose sur une décomposition des choix de conception d'un système interactif en trois ensembles : les choix conceptuels, les choix relatifs au style d'interaction mis en œuvre, et les choix concernant les détails de présentation de l'interface. Ce partage nous permet d'aborder séparément les concepts, l'interaction et la présentation d'une interface ; nous en choisissons une application privilégiant les choix conceptuels.

Nous proposons de définir un modèle de représentation conceptuelle d'un système interactif, accompagné d'un formalisme de description. Ce *langage de spécification conceptuelle* doit permettre la formalisation du contenu d'un système interactif, relativement aux tâches de l'utilisateur de ce système, offrant ainsi un outil d'expression des résultats de l'activité de conception.

Le modèle de représentation conceptuelle défini va nous permettre d'explorer les possibilité de *passerelle entre les modèles de conception et les modèles de réalisation*. Il s'agit principalement d'exploiter les données modélisées lors de la conception afin d'en retirer des enseignements quant aux agents de réalisation du système (cf. section 4, Chapitre II).

Enfin, le langage de spécification conceptuelle sera complété par des facilités de description du style d'interaction et de présentation de l'interface du système - les deux ensembles restant de notre décomposition des choix de conception. La totalité de ces données de description sera utilisée comme entrée d'un *générateur automatique* d'interface.

La suite de ce document décrit les travaux effectués comme application de la démarche ci-dessus évoquée. Le résultat de ces travaux est un outil que nous avons conçu et développé sous le nom de SIROCO [Normand 91c, Normand 91e, Normand 92].

Le Chapitre IV présente le langage de spécification SIROCO, en mettant en lumière le modèle de représentation conceptuelle sous-jacent à ce langage. Le Chapitre V décrit notre approche de la réalisation d'un système interactif à partir d'une spécification en langage SIROCO : la passerelle jetée en direction des modèles de réalisation, et les principes du générateur de code réalisé. Le Chapitre VI donne une évaluation des travaux réalisés autour de SIROCO, relativement à l'existant.

Chapitre IV

Le langage de spécification SIROCO

1 Introduction

Le langage SIROCO est un formalisme de spécification conceptuelle d'une interface. Son objectif est de permettre la formalisation des données et opérations qui constituent les concepts du domaine de l'application, relativement aux tâches de l'utilisateur.

Ce chapitre décrit les principaux éléments du langage SIROCO et de son modèle de représentation d'un système interactif.

Les applications visées :

Les applications visées sont celles de l'environnement Guide : des applications centrées sur les données plutôt que sur les traitements. Ce sont des applications qui permettent la visualisation et la gestion de données de nature principalement textuelle, et la plupart du temps permanentes : "browser" de répertoires, ou de types et de classes d'objets, agenda électronique, messagerie, système de circulation de documents, etc.

Organisation de la présentation :

La première section présente les principes dont est issu SIROCO - plus précisément, les principes régissant le contenu et la forme du langage : la perspective de la logique d'utilisation d'un système, et le modèle de structuration à objets.

La deuxième section décrit le modèle de représentation sur lequel repose le langage SIROCO.

La troisième section rapporte les principaux aspects du formalisme même - sans entrer pour autant dans les détails de la syntaxe du langage SIROCO.

Enfin, nous proposons d'éclairer notre présentation du langage SIROCO par une section offrant un exemple de spécification d'un système interactif selon ce langage.

2 Principes

Parmi l'ensemble des choix de conception, que cherche-t-on à modéliser ? Comment va-t-on structurer la formalisation ? Telles sont les deux questions de base qui se posent lors de la conception du langage SIROCO. Nous indiquons ici les principes des réponses apportées à ces deux questions.

2.1 Ce que l'on cherche à modéliser

C'est sur la notion de *modèle conceptuel* d'un système interactif que nous axons nos travaux. Cette notion, introduite au Chapitre I, section 2, sous l'appellation C_σ, décrit les éléments du système définis par le concepteur pour répondre aux besoins de l'utilisateur. Le modèle conceptuel décrit le contenu sémantique du système (en opposition à sa présentation, ou aux séquences du dialogue, contenues dans l'image I_σ) : les concepts en jeu, données et opérations du système. Le modèle conceptuel constitue la représentation abstraite du système que l'interface doit transmettre à l'utilisateur pour en assurer une utilisation correcte. En quelque sorte, le modèle conceptuel détient les informations sémantiques du modèle mental M_σ de l'utilisateur expert idéal.

Notre modélisation du contenu conceptuel d'un système se veut orientée vers l'utilisateur, tout en explicitant les règles de fonctionnement du système, reflet de la dualité utilisation/ fonctionnement. Notre réflexion porte également sur la notion de donnée de conception ; l'on retrouve ici les informations de niveau "méta" évoquées au Chapitre II, section 2.5.

2.1.1 Dimension d'utilisation et dimension de fonctionnement

La distinction entre logique de fonctionnement et logique d'utilisation apparaît dans les travaux de Richard, cités dans [Barthet 88]. Dans la *logique de fonctionnement*, on apprend à l'utilisateur le fonctionnement du système et les effets de chaque commande : "si l'on effectue l'opération O alors on obtient le résultat R". Dans la *logique d'utilisation*, c'est le mode d'utilisation des commandes relativement aux tâches de l'utilisateur qui est en jeu ; on explique à l'utilisateur comment faire pour arriver à un résultat donné : "si l'objectif est le résultat R, alors on peut effectuer l'opération O".

Nous proposons de conserver la dualité utilisation/fonctionnement lors de notre modélisation du contenu conceptuel d'un système. Nous distinguons deux dimensions de modélisation : la dimension de fonctionnement et la dimension d'utilisation.

La *dimension de fonctionnement* décrit les données et opérations contenues dans le système en explicitant le statut des données, les préconditions et l'effet des opérations, etc. - les règles de fonctionnement du système.

La *dimension d'utilisation* décrit la façon dont les données et opérations du système sont en relation avec les tâches de l'utilisateur. Les informations d'utilisation reposent sur les notions de tâche de l'utilisateur, procédure opérative, et image opérative.

Les notions de tâche et de procédure opérative ont été introduites au Chapitre I. La *tâche* représente un objectif que l'utilisateur cherche à atteindre avec le système ; la *procédure opérative* décrit la séquence d'opérations du système permettant la réalisation d'une tâche. La notion d'*image opérative* sur une entité caractérise la perception qu'a l'utilisateur de cette entité lorsque cette dernière est impliquée dans la réalisation d'une tâche. Cette perception

subit une déformation fonctionnelle qui accentue les aspects de l'entité importants pour la tâche, et élimine les aspects de l'entité non pertinents dans le contexte de cette tâche.

L'identification des tâches, leur éventuelle structuration en sous-tâches élémentaires, l'identification des entités et de leurs images opératives pertinentes dans le contexte des tâches, et la mise en correspondance des tâches élémentaires et des procédures (pas forcément uniques) les réalisant constituent les données d'utilisation du système. Ces données définissent une dimension organisatrice de l'espace des données et opérations offertes dans un système interactif ; cette dimension doit être reflétée au niveau de l'interface afin de faciliter l'acquisition et l'exécution des procédures par l'utilisateur.

Notons que les données de logique d'utilisation ne sont et ne peuvent être figées, tout particulièrement dans le cas des tâches faiblement structurées (cf. section 3, Chapitre I). D'une part, plusieurs procédures peuvent coexister pour la réalisation d'une même tâche ; le choix d'une procédure, qu'elle soit optimale ou non, dépendra du contexte et de l'utilisateur. D'autre part, dans le cas des tâches faiblement structurées, il n'est pas possible de produire des procédures de réalisation des tâches complexes (la traduction d'une tâche complexe en une suite de tâches élémentaires ne peut être figée).

2.1.2 Données de conception

L'activité de conception produit un ensemble de données de tendance volatile, pour certaines implicites - que ces données s'inscrivent dans la dimension d'utilisation du système ou bien dans la dimension de fonctionnement. Trois grands types de données de conception peuvent être distingués : les informations structurelles, sémantiques et logiques.

Les *informations structurelles* décrivent les éléments intervenant dans la structuration de cette interface - ces données constituent le noyau autour duquel s'organise la conception. Les *informations sémantiques et logiques* apportent des données complémentaires attachées aux informations de structure. Il s'agit de capter d'une part la sémantique d'un élément de structure, et d'autre part les liens logiques existant entre les éléments de structure.

Les informations sémantiques sont bien entendu les plus subtiles à capter, les plus difficiles à représenter. Les relations sémantiques se placent à un niveau méta de description de l'interface : elles participent à l'argument des choix de conception (cf. Chapitre II) plus qu'à la description des caractéristiques de l'interface.

Quel que soit le type de donnée, la part doit être faite entre les éléments *génériques* aux applications interactives et les éléments *spécifiques* à une application particulière - nous nous attacherons à la recherche de la généricité lors de la définition de SIROCO.

2.2 La forme de la spécification

C'est un langage de nature déclarative que nous envisageons, puisque les informations à représenter ne sont pas de type procédural. Quelle forme de structuration adopter ? Nos travaux s'inscrivant dans un environnement de programmation à objets, c'est tout naturellement que les notions d'objet, de classe, de type et d'héritage interviennent comme base de structuration d'une modélisation SIROCO.

Nous passons en revue les différentes notions issues des langages de programmation à objets, en précisant leur application et leur signification dans le contexte de nos travaux de modélisation. Nous montrons également les limites du modèle à objets, et proposons une

extension permettant l'expression de l'ensemble des données de conception précédemment évoquées.

2.2.1 Un modèle de structuration à objets

Nos travaux ont pour cadre un environnement de développement à objets : l'environnement Guide [Krakowiak 90]. Guide est un langage de programmation doublé d'un système d'exploitation réparti permettant l'exécution des applications Guide. Parmi les caractéristiques de Guide, l'on retiendra ici l'adoption "en profondeur" d'un modèle à objets : non seulement pour la programmation des applications mais également pour le stockage des données permanentes. Dans Guide, l'objet est l'unité de décomposition du code d'une application ; c'est également l'unité de stockage des données, tout objet étant potentiellement permanent ; enfin, l'objet est l'unité de partage de données entre applications.

Le modèle à objet dont nous nous inspirons est celui du langage de programmation Guide. La suite de cette section présente les éléments que nous retenons de ce modèle, en explicitant l'application de ces derniers dans le contexte de nos travaux[1].

Objets et classes

Un *objet* modélise une entité perçue du monde réel ou abstrait. Nous considérons un objet comme un module renfermant un état interne, un comportement, et offrant une interface d'accès composée d'une collection de données et d'une collection d'opérations (d'un point de vue théorique, la présence d'une donnée dans l'interface d'un objet dénote la déclaration d'une opération de lecture sur cette donnée, et éventuellement d'une opération d'écriture - on rejoint ainsi la définition théorique qui présente un objet comme une collection d'opérations).

Une *classe* décrit un modèle générique d'objets ayant une structuration, une interface et un comportement similaires. Tout objet est produit à partir d'une classe, par un procédé d'*instanciation* de cette classe ; un objet est en fait une *instance*[2] (un exemplaire) de classe.

Types et restrictions de types

Un *type* (ou type abstrait) se définit théoriquement comme une interface d'objet : la liste des données et opérations exportées par l'objet. La notion de type apparaît dans le contexte de l'utilisation d'un objet : il s'agit de préciser le protocole d'accès à cet objet depuis l'environnement extérieur.

Le type décrit le comportement d'un objet, tandis que la classe définit une réalisation spécifique d'un type, en vue de la génération d'instances de ce type - un même type peut être réalisé par plusieurs classes. Dans le contexte de nos travaux, ce sont des types que nous allons manipuler, et non des classes : notre approche de la modélisation d'un système interactif passe

1. Notons que les notions développées dans cette section s'insèrent dans le contexte général et bien connu de la programmation à objets, telle que décrite par exemple dans [Cox 86] ou [Meyer 88]. Certaines de ces notions sont cependant marginales, dans le sens où elles ne sont pas présentes dans tout modèle à objet ; c'est le cas notamment des notions de type abstrait et d'héritage conforme présentes dans Guide.

2. Les termes "instance" et "instanciation" sont des anglicismes. Nous les adoptons cependant dans la suite de ce document, car ce sont les termes consacrés pour exprimer les notions d'exemplaire de classe, et de création d'un exemplaire de classe.

par la déclaration de structures d'entités, de rôles et de comportements, plutôt que par la définition de procédures de réalisation.

Pour un objet donné, l'on distingue le type complet et les types restreints. Le *type complet* est l'interface originale de l'objet. Un *type restreint* présente un sous-ensemble des définitions du type complet.

La notion de type restreint est à la base de nos travaux. Restreindre un type, cela signifie écarter un certain nombre de définitions de ce type pour ne retenir qu'un sous-ensemble des données et opérations de l'interface originale. Les motivations d'une restriction, lors d'une tâche de programmation, sont la mise en œuvre de filtres de protection, ou bien des manipulations de programmation (cf. [Normand 90d]). Nos travaux feront apparaître de nouvelles motivations liées aux besoins de la modélisation des interfaces.

La définition du protocole d'accès à un objet que constitue un type offre deux implications : d'une part elle peut effectuer un filtrage sur l'interface originale de l'objet effectivement manipulé, et d'autre part elle définit l'interface minimale que doit satisfaire un objet pour pouvoir subir les manipulations en question. Dans tous les cas, elle est indissociable de la notion de polymorphisme des objets. La Figure IV-1 illustre les notions de type complet ou restreint, et de protocole de communication entre objets.

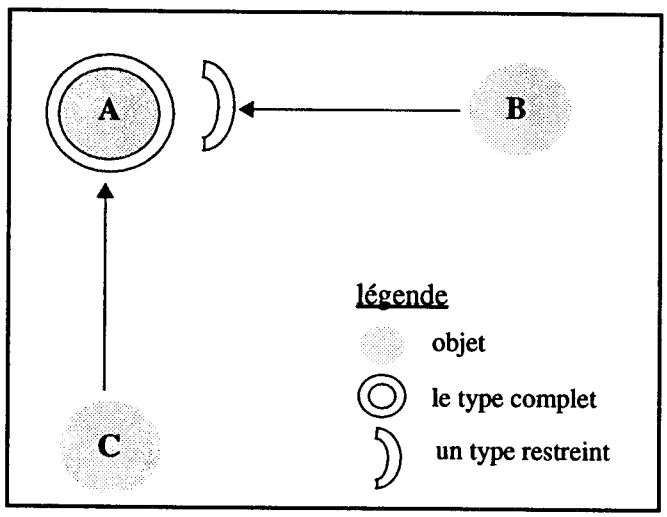


Figure IV-1 *Objets et types d'objets.*

La figure représente un objet A dont les données et opérations sont accédées par les objets B et C.
L'objet C accède à l'objet A au travers du type complet de A ; l'objet B n'a accès qu'à une partie restreinte des opérations et données de A.

Héritage

C'est l'*héritage* simple entre types que nous considérons ici. L'héritage est un procédé qui permet la définition d'un type d'objet à partir d'un autre type, en ajoutant de nouvelles définitions à ce nouveau type.

Dans le contexte de la programmation, l'héritage est souvent utilisé comme un procédé de factorisation/réutilisation de code. Ici, c'est une perspective sémantique qui est adoptée : l'héritage dénote une relation EST-UN entre deux types.

2.2.2 Extension du modèle à objets

Les concepts du modèle à objets de Guide ci-dessus présentés fournissent un cadre général pour l'expression des aspects structurels de la modélisation d'un système interactif. Les objets, les types complets et restreints, les données et les opérations d'un objet constituent les éléments d'un modèle de structuration permettant la collecte des informations structurelles évoquées en 2.1.2. En ce qui concerne les informations sémantiques et logiques, le modèle à objets atteint ses limites. En dehors de certaines relations sémantiques, dont l'héritage entre types d'objets permet l'expression, la spécification des données de conception nécessite un dispositif autorisant l'association de déclarations à chaque élément structurel. Nous introduisons la notion de propriété comme une extension au modèle à objets de Guide mettant en œuvre ce dispositif.

Une *propriété* est un couple étiquette/valeur. L'étiquette identifie la propriété ; la valeur représente le contenu de la propriété. Une propriété permet le recueil d'une information de type donné, propre à la propriété considérée : textuel, numérique, booléen, ou procédural.

Une propriété se définit relativement à un élément structurel décrit par application du modèle à objets. Tout élément structurel (type d'objet, donnée, opération) est doté d'un ensemble de propriétés dont les valeurs contiennent les informations sémantiques et logiques qui complètent l'information structurelle inhérente à cet élément.

Un exemple simple va nous permettre d'expliciter la notion de "propriété", extension du modèle à objets.

Un compte bancaire peut être perçu - de façon volontairement simpliste - comme une "bourse" virtuelle repérée par un numéro, contenant un montant d'argent, et à laquelle l'on peut ajouter ou retirer des sommes d'argent. La notion de compte bancaire peut ainsi être représentée comme un objet dont le type comporte deux attributs (le numéro du compte et son solde), et admet deux opérations (l'ajout et le retrait d'argent) :

Type compte-bancaire :
attribut Numéro : entier
attribut Solde : nombre décimal
opération Ajout (somme : nombre décimal)
opération Retrait (somme : nombre décimal)

Le type ainsi défini n'exprime que les aspects structurels de la notion de compte bancaire : les attributs de données qui lui sont rattachés, et les opérations auxquelles le compte peut donner lieu. Cette description structurelle laisse non exprimées un ensemble d'informations sémantiques ou logiques caractérisant le contenu tout comme le mode de fonctionnement du compte bancaire. Par exemple :

- le fait que l'attribut Numéro est la clé d'identification du compte dans l'environnement externe de la banque.
- le fait que l'opération de Retrait ne peut être effectuée qu'à condition que la somme retirée soit inférieure au montant de l'attribut Solde.

L'expression de ces deux informations prend la forme de deux propriétés ajoutées aux informations structurelles du type ci-dessus défini :

Type compte-bancaire :
attribut Numéro : entier
attribut Solde : nombre décimal
propriété de nature : clé d'identification
opération Ajout (somme : nombre décimal)
opération Retrait (somme : nombre décimal)
propriété de pré-condition : somme < Solde

La section suivante reprend les principes que nous venons d'énoncer, et montre comment nous construisons notre formalisme de représentation conceptuelle à partir de ces principes.

3 Modèle de représentation conceptuelle d'un système interactif

Notre proposition est une concrétisation des principes exposés dans la section précédente : un modèle orienté objet qui articule la représentation d'un système interactif selon la dualité fonctionnement/utilisation.

Nous traçons dans un premier temps les grandes lignes de notre modèle de représentation conceptuelle d'un système interactif. Nous examinons ensuite plus précisément les composants des dimensions de fonctionnement et d'utilisation du modèle.

3.1 Présentation générale

Le modèle SIROCO décompose la représentation d'un système interactif en deux ensembles de données correspondant aux deux points de vue que sont la dimension du fonctionnement et la dimension de l'utilisation de ce système.

La *dimension du fonctionnement* donne lieu à la description des données conceptuelles du système, et des opérations pouvant être effectuées sur ces données. La *dimension d'utilisation* s'exprime dans l'organisation de l'espace des opérations et des données de fonctionnement en modules utilitaires, cette organisation reflètant notamment l'arbre des tâches de l'utilisateur du système.

Les éléments de base du modèle sont au nombre de trois : les Concepts de l'Application, qui expriment la dimension de fonctionnement de l'application, et les Perspectives et Espaces de Travail, qui expriment la dimension d'utilisation de l'application. La suite de cette section procède à la définition de ces trois composants de base de notre modèle. Nous proposons tout d'abord une définition statique de ces trois notions, puis donnons un aperçu du comportement dynamique du système ainsi modélisé. Enfin, nous complétons ces descriptions informelles en donnant une définition formelle au travers d'une notation ensembliste..

3.1.1 Définitions

Concept de l'Application

Définition : *Un Concept de l'Application est un objet sémantique du système interactif.*

Par objet sémantique, il faut comprendre toute entité fonctionnelle du système susceptible d'intervenir directement dans la réalisation des tâches de l'utilisateur. La définition d'un Concept de l'Application repose sur les entités du domaine de la tâche. Elle reflète le point de vue adopté par le système sur ces entités, caractéristiques du modèle conceptuel C_σ défini par le concepteur (cf. Chapitre I). Dans ce sens, la définition d'un Concept de l'Application est dépendante des éventuelles métaphores véhiculées par l'interface du système.

Un Concept de l'Application se définit comme un objet comportant un ensemble d'attributs de données, et sur lequel un ensemble d'opérations peuvent être appliquées.

Prenons l'exemple d'un système de messagerie électronique. Les Concepts de l'Application élémentaires sont l'utilisateur (abonné à la messagerie), la boîte-à-lettres, et le message. Selon les fonctions réalisées dans le système considéré, l'on peut envisager des Concepts d'Application tels la

liste d'utilisateurs (pour envois groupés de messages), le dossier de rangement (pour gérer les messages reçus par un utilisateur), etc. Selon la métaphore choisie par le concepteur du système, l'on peut également songer au Concept de "poubelle" (à la Macintosh) qui permet de jeter un message en retardant sa destruction physique.

Espaces de Travail et Perspectives

Soit un système interactif conçu pour permettre la réalisation d'un ensemble de tâches de l'utilisateur. Les Concepts du système constituent une base d'objets offerte à l'utilisateur pour la réalisation des tâches de ce dernier. L'utilisation du système donne lieu à un ensemble d'activités de réalisation des tâches ; chaque activité peut être perçue de façon générale comme une suite de manipulations d'objets Concepts - une manipulation pouvant conduire à la création de nouveaux objets Concepts, la modification des attributs d'un objet, ou à l'appel d'une opération sur un objet.

Espaces de travail et perspectives sont les éléments du modèle dédiés à l'organisation de l'ensemble des Concepts de l'Application dans le contexte de l'utilisation du système. C'est une organisation spatiale que nous proposons, selon une métaphore que nous décrivons ci-après.

On peut considérer que les objets Concepts sont placés dans un espace auquel le système est la porte d'accès. La réalisation de ses tâches conduit l'utilisateur à des activités de parcours de l'espace des Concepts, et de manipulations sur ces objets. Deux remarques s'imposent :

1. Le parcours de l'espace des Concepts est régi par les objectifs de l'utilisateur mais aussi par les contraintes de fonctionnement du système.
2. L'activité de réalisation d'une tâche donnée nécessite la manipulation d'un ensemble déterminé de Concepts ; de plus, cette manipulation met en jeu un ensemble précis d'éléments de ces Concepts.

Afin de faciliter le travail de l'utilisateur, il est nécessaire de guider le parcours de ce dernier à l'intérieur de l'espace global des Concepts. Ce guidage est le rôle de l'interface. Il passe par l'organisation de l'espace global en un ensemble de sous-espaces que nous appelons *Espaces de Travail.* La notion d'Espace de Travail est l'unité de structuration de l'espace des Concepts.

Définition : *Un Espace de Travail est un lieu d'activité virtuel offrant les éléments nécessaires à la réalisation d'une ou plusieurs tâches de l'utilisateur.*

Un Espace de Travail se définit comme un lieu d'accès virtuel à un ensemble déterminé de Concepts de l'Application - les Concepts intervenant dans les tâches de l'Espace de Travail ; ce concept de lieu d'activité virtuel rejoint la notion de "room" développée dans [Henderson 86]. L'accès à un Concept se caractérise de plus par une Perspective, filtre qui définit une image opérative sur ce Concept et ne retient que les éléments du Concept pertinents pour les tâches de l'Espace de Travail (cf. la notion d'image opérative définie en 2.1.1).

Corollaire : *Un Espace de Travail se définit comme un ensemble de Perspectives sur des Concepts de l'Application.*

Définition : *Une Perspective décrit une vue restrictive sur les données et les opérations d'un Concept de l'Application. Cette vue est caractéristique d'une image opérative du Concept concerné.*

Dans l'exemple du système de messagerie, l'on peut définir un Espace de Travail correspondant à la tâche d'envoi d'un message. Cet Espace de Travail donne accès à un Concept de message, avec une Perspective autorisant la modification du sujet et du texte du message, ainsi que l'activation de l'opération d'envoi sur ce message.

La Figure IV-2 illustre les définitions ci-dessus énoncées dans le contexte de la métaphore spatiale.

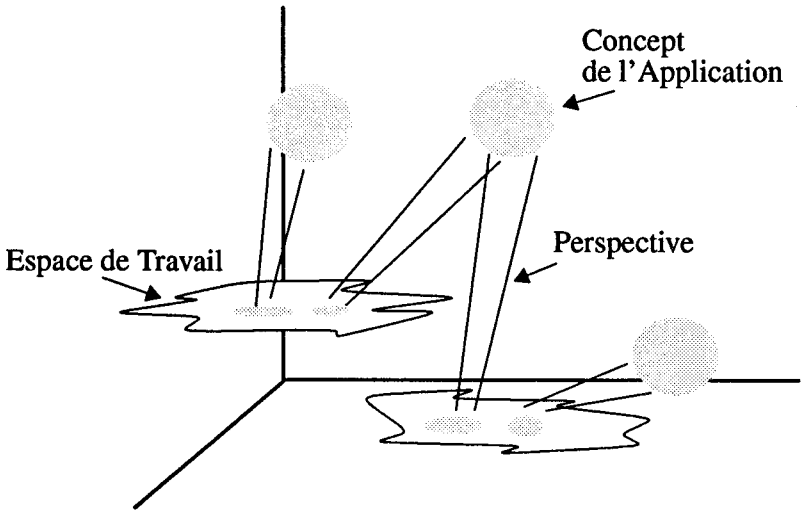


Figure IV-2 *Organisation de l'espace des Concepts de l'Application.*

La figure représente un espace comportant trois Concepts de l'Application. Deux Espaces de Travail sont définis ; les Perspectives les caractérisant sont représentées comme des projections des Concepts sur les Espaces de Travail.

Notation : Concepts de l'Application, Perspectives et Espaces de Travail pourront dans la suite de ce document être désignés par les sigles anglais[1] (respectivement) AC (Application Concept), P (Perspective), et WS (WorkSpace).

3.1.2 Comportement dynamique

Au sens de la modélisation à objets, Concepts de l'Application et Espaces de Travail sont des objets, et plus précisément sont décrits comme les types de ces objets. Les Perspectives ne sont pas des objets, mais des filtres sur des objets ; ce sont des restrictions sur les types des Concepts de l'Application.

1. L'usage de l'anglais pour ces acronymes vise à assurer une unité de ton entre nos descriptions et les termes du langage SIROCO lui-même.

De façon générale, notre utilisation de l'anglais dénote un souci de cohésion avec le langage Guide, dont les termes sont en anglais.

L'utilisation d'un système donne lieu à une *session* d'exécution de ce système. La modélisation d'une session met en jeu des instances de Concepts de l'Application et d'Espaces de Travail. Quelles sont les règles de création et de destruction de ces instances ? Comment désigne-t-on une instance ? Nous verrons dans la suite de ce document le détail du mode de spécification de ces règles de comportement dynamique dans le modèle SIROCO. Pour l'heure, nous précisons les principes adoptés par SIROCO pour modéliser la dynamique d'un système.

L'objet Session

Notre modèle est pourvu d'un élément de structuration supplémentaire, la Session. L'objet **Session**, qui apparaît en un unique exemplaire dans la représentation d'un système, a pour rôle d'une part de définir le contexte global de l'interface du système, et d'autre part de décrire l'état initial de cette interface.

Le *contexte global de l'interface du système* est une collection de données définies de façon globale au système, désignables à l'intérieur de tout Concept de l'Application, de toute Perspective, et de tout Espace de Travail.

L'*état initial de l'interface du système* décrit les instances d'Espaces de Travail présentes au lancement du système. Ces instances représentent l'initialisation de la session ; toute la dynamique du système est mise en œuvre à partir de ces Espaces de Travail initiaux.

Dynamique d'instanciation des objets du système

Le mode d'instanciation d'un objet diffère selon que l'objet est un Concept de l'Application ou un Espace de Travail.

Concepts de l'Application :

En ce qui concerne les objets Concepts de l'Application, nous distinguons deux cas, selon que l'instanciation se situe en aval ou en amont des traitements fonctionnels.

Pour une part des objets Concepts de l'Application, l'instanciation résulte de traitements fonctionnels. Il s'agit là d'aspects relatifs à la dynamique interne du composant fonctionnel du système, dont la modélisation est hors de notre propos. La représentation de l'instanciation de ces Concepts de l'Application est par conséquent hors du champ de notre modèle.

Pour l'autre part des objets Concepts de l'Application, l'instanciation se situe non pas en aval mais en amont des traitements fonctionnels : l'instanciation de ces objets a un rôle d'initialisation des traitements fonctionnels. Typiquement, il s'agit d'instancier un objet, de l'initialiser, puis de lui appliquer une opération exécutant un traitement fonctionnel. Les instanciations de cette nature sont du domaine de l'interface du système ; SIROCO permet de modéliser la génération d'objets au travers de la notion de prototype de Concept.

La notion de *prototype* représente un mécanisme permettant l'instanciation et l'initialisation d'un Concept à partir de la saisie de l'utilisateur. Un prototype est en quelque sorte un "moule" destiné à produire des instances de Concepts à partir des données saisies. La présence d'un Concept prototype à l'intérieur d'un Espace de Travail dénote un objet "fantôme" (ou objet potentiel), sans existence réelle, destiné au support de la saisie de données d'initialisation qui seront utilisées lors de l'instanciation effective de l'objet. Cette instanciation peut être déclenchée de multiples façons, notamment par effet de bord, lors de l'appel d'opérations du système.

Dans l'exemple du système de messagerie, la tâche d'envoi d'un message peut être modélisée en utilisant la notion de prototype.

De façon fonctionnelle, envoyer un message, cela signifie instancier un Concept de message, initialiser les champs de données de cet objet, puis appeler l'opération d'envoi sur cet objet. Du point de vue de l'utilisation du système, l'on choisit de décomposer la tâche en la saisie du texte d'un message, suivie de l'envoi effectif du message ; cette tâche est de plus interruptible à tout moment.

C'est l'opération d'envoi qui valide le message ; tant qu'il n'est pas envoyé, le message ne possède pas d'existence réelle en dehors des données saisies par l'utilisateur, temporairement mémorisées au niveau de l'interface du système. Cette situation trouve son expression dans l'utilisation d'un prototype de message qui permet de retarder l'instanciation de l'objet message jusqu'à l'envoi effectif du message.

Espaces de Travail :

L'instanciation des objets Espaces de Travail représente une large part de la dynamique d'utilisation du système. L'instanciation d'un Espace de Travail a pour rôle de mettre de façon effective à la disposition de l'utilisateur le champ d'activité correspondant à cet Espace. Cette extension de l'activité de l'utilisateur s'inscrit dans la continuité de l'utilisation du système : l'instanciation s'effectue depuis un Espace de Travail existant, sur demande explicite de l'utilisateur, ou bien comme effet de bord d'une opération. Nous introduisons la notion d'opération d'accès à un Espace de Travail depuis un autre Espace afin de représenter les liens de transition entre Espaces de Travail.

Une *opération d'accès à un Espace de Travail* (Espace destination) se définit localement à un Espace donné (Espace source). L'accès à l'Espace destination depuis l'Espace source passe par l'instanciation et l'initialisation d'un objet Espace destination (ou bien le rappel d'un objet Espace destination existant) ; nous verrons par la suite que cette opération comporte plusieurs paramètres permettant notamment de décrire des relations logiques entre les deux Espaces.

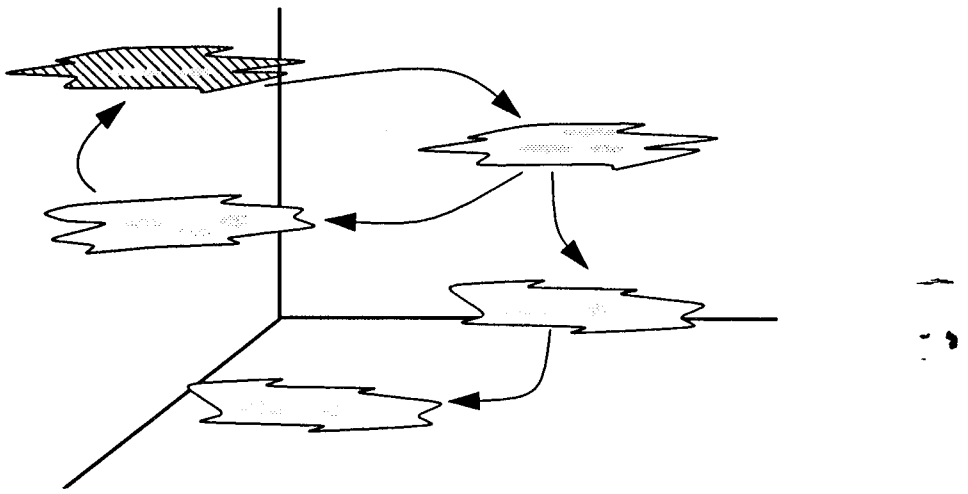


Figure IV-3 *Navigation entre les Espaces de Travail d'un système.*

Les flèches représentent les opérations d'accès à un Espace offertes par chaque Espace. Ce sont des types d'Espaces qui sont représentés, et non des instances. L'Espace hachuré est un des Espaces initiaux instanciés à l'initialisation du système.

Une opération d'accès représente une passerelle entre deux Espaces ; comme l'illustre la Figure IV-3, l'ensemble des opérations d'accès tracent les voies de navigation autorisées à l'intérieur du système. La réalisation de ses tâches nécessite pour l'utilisateur de déplacer son activité d'un Espace de Travail à un autre ; le parcours de l'utilisateur a pour point de départ les Espaces de Travail décrits dans l'état initial du système (cf. l'objet Session), et se poursuit en empruntant les voies de navigation offertes par les opérations d'accès définies dans ces Espaces de Travail initiaux.

3.1.3 Formalisation

Dans un souci de précision, nous reprenons les notions de système interactif, Concept de l'Application, Espace de Travail, et Perspective, et proposons une définition formelle de ces différentes notions[1].

Notations :

Les notations utilisées sont des notations ensemblistes élémentaires. Seul notre usage des indices mérite quelques remarques : afin de ne pas surcharger nos formules, les domaines de valeur de ces indices sont la plupart du temps omis. Chaque indice apparaissant dans une formule, représenté par un des signes i, j, ou k, représente un entier naturel dont l'intervalle des valeurs est fini, et débute à la valeur 0.

Définitions générales :

La notion de *type* est à la base de nos définitions. Qu'est-ce qu'un type? Notre objectif n'est pas de proposer une formalisation de cette notion générale, mais d'en exprimer les aspects pertinents dans le cadre de cette section.

De façon générale, on distinguera les types simples et les types construits.

Les *types simples* sont d'une part les types élémentaires (par exemple, type entier, type caractère, type booléen), et d'autre part les types fonctions.

Un *type construit* dénote une structure mettant en jeu un ou plusieurs types quelconques. La définition de cette structure repose sur l'utilisation d'un constructeur de type (par exemple, tableau, enregistrement).

Un objet peut être perçu comme un enregistrement dont les champs représentent les variables et méthodes (fonctions) de cet objet. Un *type d'objet* sera défini comme un type contruit, de constructeur "enregistrement".

On note :

Types : l'ensemble des types.

TypesObjets : l'ensemble des types d'objets.

1. Seuls les aspects structurels du modèle sont ici formalisés ; les aspects logiques, qui à ce stade de notre présentation du modèle sont à peine esquissés, ne sont pas représentés

On considère un *ensemble de tâches* noté T, et un *système interactif* noté S, défini comme support de T.

Définition théorique du système S :

Nous définissons S comme le quadruplet d'ensembles suivant :

$S : <C, W, g, i>$

avec

- C : l'ensemble des Concepts d'Application de S,
- W : l'ensemble des Espaces de Travail de S,
- g : le contexte global de S.
- i : le contexte initial de S.

– Un Concept de l'Application est un objet représentant une entité sémantique du système interactif. Un Concept de l'Application peut se définir comme un type d'objet.

$$C \subset \textbf{TypesObjets}$$

– On définit l'opérateur *perspectives* qui, à un Concept donné, associe l'ensemble des Perspectives pouvant lui être appliquées. *perspectives* est un opérateur défini sur l'ensemble des Concepts C, avec pour domaine général de valeurs l'ensemble des parties de l'ensemble des types d'objets :

$$perspectives : C \rightarrow P(\textbf{TypesObjets})$$

$$perspectives(c) = \{ p_i \}_{i \in [0,n]}$$

où c est un Concept : $c \in C$
p_i est un type d'objet : $p_i \in$ **TypesObjets**
$\forall j \in [0,n]$, p_i est différent de p_j
et chaque champ de p_i apparaît de façon unique dans c

– W est l'ensemble des Espaces de Travail de S : $W = \{W_i\}_{1<i<n}$

Un Espace de Travail W_i peut ainsi se définir :

$W_i : <\wp^i, V^i>$

avec

$\wp^i$ est l'ensemble des Concepts accessibles dans l'Espace de Travail W_i. $\wp^i$ peut être représenté comme un ensemble de couples associant un Concept et la ou les Perspectives d'utilisation de ce Concept à l'intérieur de l'Espace de Travail:

$\wp^i = \{ (c^i_j, \{p^{i,j}_k\}) \}$ où c^i_j est un Concept du système : $c^i_j \in C$
$p^{i,j}_k$ est une Perspective sur c^i_j : $p^{i,j}_k \in perspectives(c^i_j)$.

ν^i est l'ensemble des opérations de navigation définies sur l'Espace de Travail W_i. ν^i peut être représenté comme l'ensemble des Espaces de Travail destinations de ces opérations :

$\nu^i = \{ W^i_j \}$ où W^i_j est un Espace de Travail de S : $W^i_j \in \mathbf{W}$.

– $\mathbf{g}$ est le contexte global de S. $\mathbf{g}$ peut être défini comme un ensemble de Concepts :

$\mathbf{g} = \{ c_i \}$ où c_i est un Concept du système : $c_i \in \mathbf{C}$

– $\dot{\mathbf{i}}$ est le contexte initial de S. $\dot{\mathbf{i}}$ peut être représenté comme un ensemble d'Espaces de Travail :

$\dot{\mathbf{i}} = \{ W_i \}$ où W_i est un Espace de Travail de S : $W_i \in \mathbf{W}$.

A l'exécution :

A un instant donné d'une exécution de S, l'état du système se compose d'un ensemble d'instances de Concepts et d'Espaces de Travail.

On note $S(\mu(t))$ l'état de S à l'instant t d'une session donnée.

μ représente la fonction de mémorisation des interactions qui caractérisent la session considérée : μ dénote d'une part les conditions initiales de lancement de la session, et d'autre part les actions de l'utilisateur sur le système, et les interactions de l'environnement extérieur à S.

$\mu(0)$ représente les conditions d'initialisation de la session ;

$\mu(t)$ représente les conditions d'initialisation de la session, ainsi que l'ensemble des actions d'interaction de l'utilisateur et de l'environnement extérieur depuis l'instant 0, date de lancement de la session, jusqu'à l'instant t.

En reprenant les formules de définition statique de S, $S(\mu(t))$ peut être représenté par le quadruplet suivant :

$S(\mu(t)) : \langle \mathbf{C}(\mu(t)), \mathbf{W}(\mu(t)), \mathbf{g}(\mu(t)), \dot{\mathbf{i}}(\mu(0)) \rangle$

avec, à l'instant t de la session considérée,

$\mathbf{C}(\mu(t))$: l'ensemble des instances de Concepts d'Application de S,

$\mathbf{W}(\mu(t))$: l'ensemble des instances d'Espaces de Travail de S,

$\mathbf{g}(\mu(t))$: l'ensemble des instances de Concepts du contexte global de S,

$\dot{\mathbf{i}}(\mu(0))$: l'ensemble des instances d'Espaces de Travail initiales de S.

La réalisation d'une tâche à l'aide du système S, à l'instant t de la session considérée, peut se décrire de la façon suivante :

Soit τ une tâche, $\tau \in T$.

τ est réalisable dans $S(\mu(t))$ si et seulement si il existe une suite d'actions de l'utilisateur $(a_i)_{0<i<n}$ constituant une procédure de réalisation de τ à l'intérieur de la session considérée. Une action a_i répond à l'une ou l'autre des deux définitions suivantes, correspondant à l'instant t+i - soit après l'application des i-1 premières actions de la suite :

- Soit a_i est une opération sur une instance de Concept c accessible dans une instance d'Espace de Travail W, à l'instant t+i :
  $\exists \quad W \in \mathbf{W}(\mu(t+i)), W = <\wp(\mu(t+i)), \nu(\mu(t+i))>$
  $\exists \quad c \in \mathbf{C}(\mu(t+i)),$
  $\exists \quad p \in \textit{perspectives}(c)$
  tels que
  $(c, \{p_k\}) \in \wp(\mu(t+i))$ et $\exists$ k tel que $p_k = p$,
  a_i est un élément de type fonction, champ de p,
  et a_i est activable[1] à l'instant t+i.
- Soit a_i est une opération de navigation d'une instance d'Espace W, à l'instant t+i :
  $\exists \quad W \in \mathbf{W}(\mu(t+i)), W = <\wp(\mu(t+i)), \nu(\mu(t+i))>$
  tel que
  $a_i \in \nu(\mu(t+i))$ et a_i est activable[1].

Au terme de cette présentation générale, les notions de Concept de l'Application, Espace de Travail et Perspective ont été définies. Ce sont cependant des "coques" vides qui ont été présentées : bien des aspects de notre modèle restent dans l'ombre, qu'il s'agisse du contenu de ces "coques", ou bien de leur mode d'emploi par le développeur.

La suite de cette section s'attache à décrire le contenu des composants du modèle ; elle reprend tour à tour chaque composant, et précise sa structure interne ainsi que les éléments d'information qui lui sont associés.

Le mode d'emploi du modèle sera décrit à la section suivante, à l'aide d'un exemple de spécification d'un système interactif.

3.2 Dimension de fonctionnement

Le fonctionnement d'un système interactif se définit par les caractéristiques fonctionnelles de ce dernier : il s'agit de décrire les éléments de données intervenant dans le

1. Interviennent à ce niveau des aspects logiques non formalisés : une opération sur un Concept, tout comme une opération de navigation, peut être à un instant donné rendue inactivable par le jeu de contraintes logiques, ainsi que nous le verrons ultérieurement.

système et les opérations offertes par ce système, mais également les contraintes régissant la manipulation des données et l'activation des opérations du système.

L'unité de structuration de la dimension de fonctionnement est le Concept de l'Application. Un Concept de l'Application se décrit comme un objet ; plus précisément, comme le type complet d'un objet. Cette section présente les principales caractéristiques d'un Concept de l'Application : ses composants structuraux, mais également les informations sémantiques et logiques qui lui sont attachées.

Notre présentation s'articule de façon "classique", en décrivant tout d'abord le modèle des données, puis le modèle des traitements. Notons que cette décomposition est un artifice de présentation puisque c'est un modèle à objets liant données et opérations qui est utilisé.

3.2.1 Modèle des données

Les données du système sont regroupées dans les attributs des Concepts de l'Application. Un ***attribut*** dénote une donnée contenue dans le Concept. Un attribut est caractérisé par un identificateur, et par un type ; ce type peut être d'une part simple ou complexe, et d'autre part élémentaire ou construit.

Un *type simple* est un des types élémentaires suivants : entier, chaîne de caractères, caractère, ou booléen. Un *type complexe* dénote une donnée qui est elle-même un Concept de l'Application ; il s'agit d'une référence sur un Concept de l'Application, et non d'une inclusion. Un *type élémentaire* caractérise une donnée unique, non décomposable, qu'elle soit de type simple ou de type complexe. Un *type construit* représente une donnée agrégat de données de type quelconque ; la définition d'un type construit fait mention d'un constructeur caractérisant l'agrégat : liste ordonnée, ensemble non ordonné, graphe d'éléments, etc.

Dans l'exemple du système de messagerie, le Concept de Message a pour structure un ensemble d'attributs parmi lesquels :

- le sujet du message, attribut de type simple et élémentaire : chaîne de caractères,
- le texte du message, également de type chaîne de caractères,
- la date de réception de ce message, de type simple et élémentaire : date,
- l'auteur du message, de type complexe et élémentaire : une référence sur un Concept d'Application décrivant un utilisateur du système,
- etc.

Ce modèle des données, relativement simple, est enrichi d'un ensemble de propriétés (notion introduite en 2.2.2) permettant l'expression d'informations sémantiques, logiques, et structurelles sur les Concepts de l'Application et leurs attributs. Nous distinguons les propriétés intrinsèques et les propriétés contextuelles.

Une *propriété intrinsèque* est une propriété dont la valeur est liée au Concept de l'Application auquel elle est attachée. Une *propriété contextuelle* est une propriété dont la valeur est susceptible de varier en fonction du contexte d'utilisation du Concept. La modification des propriétés contextuelles sera abordée lors de la description des éléments de la dimension d'utilisation du système.

La Figure IV-4 énumère les principales propriétés attachées au modèle des données.

		informations sémantiques	informations logiques et structurelles
sur un Concept	**propriétés intrinsèques**	traits sémantiques	cohérence entre attributs
	propriétés contextuelles	identification	
sur un attribut	**propriétés intrinsèques**	*nature* traits sémantiques	*composition* taille maximale domaine de valeurs
	propriétés contextuelles	identification	valeur initiale

Figure IV-4 *Propriétés du modèle des données (Concepts et attributs).*

Les propriétés sont classées selon qu'elles ont pour but le recueil d'informations sémantiques ou bien logiques/structurelles, et qu'elles sont intrinsèques ou contextuelles. Les propriétés apparaissant en italiques admettent des valeurs génériques, tandis que les autres sont spécifiques à l'application.

Recueil d'informations sémantiques

Les informations sémantiques s'expriment à travers une liste de propriétés que l'on définit de façon générale pour tout élément structurel d'un Concept d'Application - notamment, pour le Concept lui-même, et pour chacun de ses attributs. On distingue les propriétés destinées au recueil d'informations génériques, et les propriétés dont les valeurs ont une signification spécifique à l'application.

Les *informations sémantiques génériques* sont recueillies comme valeurs de la **propriété de nature**. Cette propriété a pour contenu une liste de valeurs pré-définies ; c'est une propriété intrinsèque à l'élément structurel concerné. Les valeurs déclarées pour la propriété de nature d'un élément ont pour fonction de préciser le rôle de cet élément à l'intérieur du contexte de sa définition.

Nous disposons pour les éléments du modèle de données d'une liste réduite de valeurs génériques. La propriété de nature ne s'applique pas au Concept de l'Application même. En ce qui concerne les attributs d'un Concept, la seule valeur possible est une valeur indiquant que l'attribut participe à la clé d'identification du Concept ("clé" au sens habituel dans les bases de données) : l'attribut joue un rôle d'identificateur du Concept (éventuellement en combinaison avec d'autres attributs, en cas de clé composite).

Les *informations sémantiques spécifiques* sont recueillies au travers du double mécanisme des propriétés contextuelles et intrinsèques. D'un côté, un ensemble de propriétés contextuelles permet le recueil d'informations d'identification en langage naturel ; de l'autre côté, une propriété admet comme valeurs intrinsèques une liste de traits sémantiques.

Les propriétés d'identification ont pour objectif le recueil des informations d'identification textuelle d'un élément, en langage naturel ; nous distinguons trois niveaux de description textuelle, sous la forme de trois propriétés distinctes :

- **propriété de nom** : identification de l'élément en langue naturelle, par opposition à l'identificateur formel de l'élément.
- **propriété de descriptif court** : texte de description de l'élément.
- **propriété de descriptif long** : texte de description détaillée de l'élément.

Les descriptifs court et long permettent l'expression de deux niveaux d'information sur l'élément.

En dehors de la propriété de nature, qui permet le recueil d'informations sémantiques prédéfinies sur un élément, nous définissons la **propriété de traits sémantiques** afin de recueillir des informations sémantiques non génériques, issues du domaine particulier de l'application considérée. Un trait sémantique se présente comme un identificateur défini par le concepteur pour représenter une valeur sémantique spécifique. La propriété de traits sémantiques admet une liste de valeurs, chaque valeur dénotant un trait sémantique défini sur l'élément.

Dans l'exemple du Concept de Message, chaque attribut pourra être doté d'un ensemble de propriétés sémantiques. Pour l'*attribut sujet*, on pourra par exemple définir les propriétés suivantes :
- nature : clé (le sujet fait partie de la clé d'identification d'un message),
- nom : "sujet", descriptif court : "intitulé du message" (information en langage naturel),
- traits sémantiques : "contenu".

Pour l'*attribut texte* :
- nom : "texte", descriptif court : "corps du message" (information en langage naturel),
- traits sémantiques : "contenu".

Pour l'*attribut date de réception* :
- nature : clé (la date fait partie de la clé d'identification d'un message),
- nom : "date", descriptif court : "date de réception du message" (information en langage naturel),
- traits sémantiques : "information".

Pour l'*attribut envoyeur* :
- nature : clé (l'identité de l'envoyeur fait partie de la clé d'identification d'un message),
- nom : "envoyeur", descriptif court : "envoyeur du message" (information en langage naturel),
- traits sémantiques : "information".

Dans cet exemple, deux traits sémantiques ont été définis : "contenu" et "information", qui ont pour but de distinguer deux ensembles parmi les attributs du Concept Message : texte et sujet d'une part, qui représentent le contenu du message, et date et envoyeur d'autre part, qui représentent des informationssur le message.

Recueil d'informations logiques et structurelles

Les informations logiques et structurelles sont l'objet de propriétés différentes selon la nature de l'élément considéré.

Un Concept admet une **propriété de cohérence des attributs**. Cette propriété intrinsèque prend pour valeur une expression booléenne définissant d'éventuelles contraintes de validité sur les valeurs des attributs du Concept.

Un attribut admet trois propriétés portant sur sa valeur. Une **propriété de taille** indique la taille de l'attribut (de type chaîne de caracères ou de type construit). On définit également une **propriété de valeur initiale** et une **propriété de domaine de valeurs**.

Un attribut comporte de plus une **propriété de composition**. Cette propriété n'a de sens que pour un attribut de type complexe ; elle précise la relation existant entre le Concept référencé par la valeur de l'attribut et le Concept à l'intérieur duquel est défini l'attribut. Deux valeurs possibles pour cette propriété : il s'agit ou bien d'une relation de référence, ou bien d'une relation de composition. Dans le premier cas, les deux Concepts sont indépendants ; dans le second cas, le Concept valeur de l'attribut n'a pas d'existence indépendante de celle du Concept sur lequel est défini l'attribut ; notamment, la création du second entraîne celle du premier.

Relations entre Concepts de l'Application :

Le modèle de données dont nous venons de tracer les grandes lignes permet la définition de relations structurelles, logiques ou sémantiques entre deux Concepts de l'Application.

Soient A et B deux Concepts de l'Application. La Figure IV-5 illustre les différentes relations pouvant être définies entre A et B.

Les *relations structurelles* pouvant exister entre A et B s'expriment au niveau des attributs de l'un ou de l'autre. Par exemple, A peut comporter un attribut complexe défini comme référençant un Concept B. Au-delà de la relation de simple référence entre Concepts, A peut comporter un attribut dénotant un Concept B composant de A (propriété de composition définie sur l'attribut). La relation de composition admet les caractéristiques suivantes : l'instanciation de A entraîne l'instanciation de B ; la destruction de A entraîne celle de B ; B ne peut apparaître comme composant ou référence d'un autre Concept que A.

Dans l'exemple du Concept message, l'attribut "auteur du message" dénote une relation de référence entre le Concept message et le Concept utilisateur.
La relation de composition peut par exemple être illustrée dans une représentation plus structurée du Concept de message. Un message peut être défini comme comportant deux attributs :

- le texte du message, de type simple.
- un attribut de type complexe portant sur le Concept d'entête.
  Le Concept en-tête comporte les attributs auteur, sujet et date précédemment définis.
  La relation entre les Concepts message et en-tête est une relation de composition : un message a forcément un en-tête, un en-tête est propre à un message, et ne peut exister en dehors d'un message.

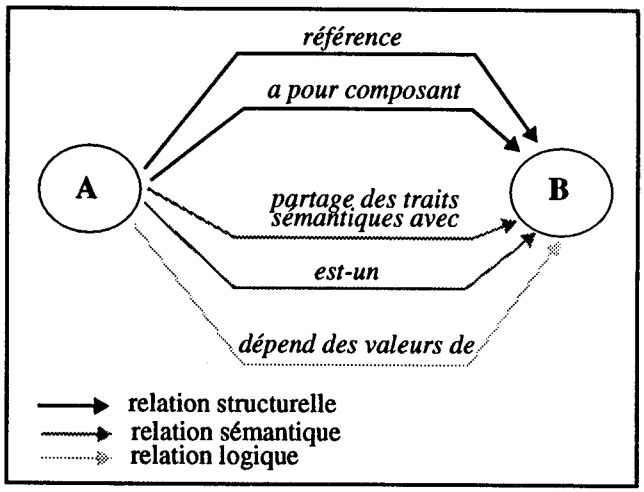


Figure IV-5 *Relations entre deux Concepts d'Application.*

Les *relations sémantiques* entre A et B peuvent s'exprimer de deux façons. D'une part, A et B peuvent comporter des traits sémantiques communs. D'autre part, B peut être défini comme une spécialisation du Concept A (cf. 2.2.1), explicitant une relation EST-UN.

Des *relations logiques* peuvent être exprimées entre A et B au travers des expressions définies comme valeur des propriétés de cohérence des attributs : une expression définie pour A peut référencer des valeurs d'attributs d'un Concept B connu de A (relation de référence ou de composition).

3.2.2 Modèle des traitements

Les traitements pouvant être effectués à l'intérieur du système interactif sont exprimés sous la forme d'opérations définies sur les Concepts d'Application.

En plus d'une liste d'attributs, la structure d'un Concept de l'Application comporte une liste d'opérations appelables sur ce Concept. Une **opération** est définie par un nom et d'éventuels paramètres d'entrée ou de sortie. Nous prenons comme convention que toute opération retourne un code exprimant le résultat de l'opération ; ce code peut prendre une valeur conventionnelle : succès ou échec, mais peut également prendre des valeurs spécifiques à l'application, définies par le concepteur du système.

Un **paramètre** décrit une donnée d'entrée ou de sortie de l'opération. Un paramètre est décrit par un identificateur, un mode de passage (entrée, sortie, ou les deux), et par un type : type simple, ou complexe, construit ou non, analogue au type d'un attribut.

Dans l'exemple du système de messagerie, la structure du Concept de Message comporte un ensemble d'opérations, parmi lesquelles :

- l'opération d'envoi du message dans la boîte-à-lettres d'un utilisateur.
  Cette opération admet pour paramètre d'entrée un paramètre de type complexe portant sur l'utilisateur destinataire de l'envoi.
- l'opération de destruction de ce message, qui ne comporte pas de paramètres.
- etc.

De même que pour le modèle des données, les éléments du modèle des traitements est enrichi d'un ensemble de propriétés intrinsèques ou contextuelles permettant l'association d'informations additionnelles tant au niveau sémantique que logique.

La Figure IV-6 énumère les principales propriétés attachées au modèle des traitement.

		informations sémantiques	informations logiques et structurelles
sur une Opération	**propriétés intrinsèques**	*nature* traits sémantiques	cohérence entre paranètres pré-condition post-action
	propriétés contextuelles	identification	
sur un paramètre	**propriétés intrinsèques**	*nature* traits sémantiques	taille maximale domaine de valeurs
	propriétés contextuelles	identification	valeur initiale d'un param. d'entrée liaison d'un param. de sortie

Figure IV-6 *Propriétés du modèle des traitements (opérations d'un Concept).*

Les propriétés sont classées selon qu'elles ont pour but le recueil d'informations sémantiques ou bien logiques, et qu'elles sont intrinsèques ou contextuelles. Les propriétés apparaissant en italiques admettent des valeurs génériques, tandis que les autres sont spécifiques à l'application.

Recueil d'informations sémantiques

Les propriétés de nature, d'identification, et de traits sémantiques définies pour le modèle des données s'appliquent également pour le modèle des traitements : toute opération, tout paramètre est doté de ce jeu de propriétés.

En ce qui concerne la propriété de nature, notons la valeur générique de "danger" qui, définie pour une opération, signifie que cette opération est susceptible de modifier de façon définitive l'état du système. D'autres valeurs sont définies, indiquant grossièrement l'effet de l'opération, en ce qui concerne les données du système : création, destruction ou modification.

Recueil d'informations logiques

Les informations logiques sont l'objet de propriétés différentes selon la nature de l'élément considéré.

Une opération admet une **propriété de pré-condition** dont la valeur est une expression booléenne indiquant une contrainte sur l'activation de l'opération. Une **propriété de cohérence des paramètres** permet la définition d'une expression booléenne indiquant d'éventuelles contraintes de validation des paramètres lors de l'appel de l'opération. Ces deux propriétés sont définies de façon intrinsèques à l'opération. Une **propriété de post-action** permet de spécifier des actions à exécuter après l'activation de l'opération. Ces actions sont notamment l'appel à des opérations sur des Concepts ; nous verrons en section 5 que des actions plus complexes peuvent être définies.

Un paramètre admet un ensemble de propriétés analogues à celles des attributs : propriétés de taille, et, pour un paramètre d'entrée, de valeur initiale, et de domaine de valeurs. Une **propriété de liaison** est définie pour les paramètres de sortie; cette propriété contextuelle

permet la spécification de l'élément (par exemple, un attribut d'un Concept connu) qui recevra la valeur retournée par ce paramètre lors de l'appel de l'opération.

Logique d'enchaînement des traitements

La logique d'enchaînement des traitements s'exprime à ce niveau sous la forme d'expressions de contraintes sur la validité de l'appel aux opérations (propriétés de pré-condition et de validation des paramètres définies sur une opération).

3.3 Dimension d'utilisation

La dimension d'utilisation d'un système interactif décrit une organisation de l'ensemble des données et opérations des Concepts du système proposant un mode d'utilisation du système. Pour un système donné, plusieurs utilisations peuvent être définies. A l'aide d'un exemple de spécification d'un système, nous préciserons en section 5 la façon dont se définit une dimension d'utilisation.

L'unité de structuration de la dimension d'utilisation est l'Espace de Travail. Un Espace de Travail est un lieu abstrait d'activité permettant la mise en œuvre d'une ou plusieurs grandes tâches de l'application conceptuellement liées. Ces tâches font intervenir un ensemble de Concepts de l'Application ; la visualisation et la manipulation d'un Concept à l'intérieur d'un Espace s'effectue au travers d'une Perspective traduisant une image opérative sur ce Concept.

Cette section présente les principales caractéristiques des Espaces de Travail et des Perspectives : leurs composants structuraux, mais également les informations sémantiques et logiques qui leur sont attachées.

3.3.1 Espaces de Travail

Un Espace de Travail se décrit comme un objet - plus précisément, comme le type complet d'un objet. La structure de cet objet fait apparaître deux types d'éléments : d'une part des attributs de données référençant des Concepts de l'Application, et d'autre part des opérations propres aux Espaces.

Les **attributs** d'un Espace de Travail définissent les Concepts d'Application auquel cet Espace donne accès. Un attribut se définit par un identificateur, un type complexe référençant un Concept, une nature, et une liste de Perspectives.

La *nature d'un attribut* porte sur le mode d'initialisation de la valeur de l'attribut : le Concept référencé peut être connu globalement au système, ou bien peut être transmis lors de la création de l'Espace, ou bien encore peut être un Concept local à cet Espace.

A chaque attribut est associée une *liste de Perspectives* décrivant l'accès au Concept valeur de l'attribut.

Les propriétés d'un attribut ont une composition analogue à celle des propriétés contextuelles d'un Concept de l'Application.

Les **opérations** d'un Espace de Travail ne représentent pas des traitements du système, comme les opérations des Concepts de l'Application : ce sont au contraire des opérations du niveau interface dont la fonction est de permettre la navigation entre les Espaces de Travail du

système, assurant ainsi l'enchaînement des tâches effectuées par l'utilisateur. Une opération d'un Espace de Travail est une opération d'accès à un autre Espace de Travail. Elle se définit par un identificateur, une liste de paramètres, et l'identificateur de l'Espace de Travail destination.

Dans l'exemple du système de messagerie, l'Espace de Travail dédié à la tâche d'envoi d'un message peut avoir la structure suivante :

- un attribut de nature locale, Concept de message.
  Cet attribut contient le message à envoyer. On lui associe une Perspective adéquate, dont nous précisons le contenu un peu plus loin.
- une opération d'accès à un Espace de Travail permettant la visualisation des utilisateurs du système.
  Cette opération n'a d'utilité que pour faciliter la réalisation de la tâche d'envoi. L'accès à l'Espace de visualisation des utilisateurs permet à l'utilisateur de prendre connaissance des destinataires possibles.
- etc.

De même que pour la dimension de fonctionnement, le modèle de la dimension d'utilisation est enrichi d'un ensemble de propriétés permettant le recueil d'informations additionnelles tant au niveau sémantique que logique et structurel.

La Figure IV-7 énumère les principales propriétés attachées aux Espaces de Travail.

		informations sémantiques	**informations logiques et structurelles**
sur un Espace	**propriétés intrinsèques**	*nature* traits sémantiques identification	*instanciation* *unicité*
sur un attribut	**propriétés intrinsèques**	*nature* traits sémantiques identification	valeur initiale
sur une Opération	**propriétés intrinsèques**	*nature* traits sémantiques identification	cohérence entre paranètres pré-condition post-action transmission de données *relation*
sur un paramètre d'Opération	**propriétés intrinsèques**	*nature* traits sémantiques identification	taille maximale domaine de valeurs valeur initiale d'un param. d'entrée liaison d'un param. de sortie

Figure IV-7 *Propriétés d'un Espace de Travail.*

Les propriétés sont classées selon qu'elles ont pour but le recueil d'informations sémantiques ou bien logiques, et qu'elles sont intrinsèques ou contextuelles. Les propriétés apparaissant en italiques admettent des valeurs génériques, tandis que les autres sont spécifiques à l'application.

Recueil d'informations sémantiques

Les propriétés de nature, d'identification, et de traits sémantiques définies pour la dimension de fonctionnement s'appliquent également ici : tout Espace de Travail, tout attribut et toute opération est dotée de ces propriétés.

En ce qui concerne la propriété de *nature d'un Espace de Travail*, deux valeurs génériques sont définies pour préciser le rôle potentiel d'un Espace lors d'une session d'utilisation du système : la valeur d'Espace principal, qui accorde à l'Espace considéré le statut de représentant de la session, et la valeur de "porte de sortie", qui dénote la possibilité de mettre fin à la session depuis l'Espace considéré.

Recueil d'informations logiques

Les informations logiques sont l'objet de propriétés différentes selon l'élément considéré.

Propriétés d'un Espace de Travail :

Un Espace de Travail admet deux propriétés portant sur la logique d'instanciation de cet Espace : une **propriété d'instanciation** permet d'exprimer si l'Espace peut être instancié plus d'une fois ou non ; une **propriété d'unicité** permet d'indiquer si l'on souhaite ou non autoriser deux instances identiques (c'est-à-dire présentant les mêmes valeurs de Concepts).

La restriction ou non du nombre d'instanciations d'un Espace de Travail résulte à la fois d'une politique de gestion de l'espace écran et de choix d'interaction (veut-on favoriser l'activité multitâche, ou bien veut-on limiter l'encombrement de l'écran?), et de contraintes internes (exclusion mutuelle sur l'accès à un concept, par exemple).

Propriétés d'une opération :

De façon analogue à une opération d'un Concept de l'Application, une opération de navigation comporte les propriétés de pré-condition, de post-action, et de validation des paramètres définies précédemment. Une opération d'un Espace de Travail admet de plus un ensemble de propriétés décrivant la façon dont s'effectue l'accès à l'Espace de Travail destination.

Une **propriété de transmission de données** permet de décrire, le cas échéant, l'initialisation de valeurs d'attributs de l'Espace destination à partir des données connues de l'Espace courant.

Une **propriété de relation** permet de caractériser la relation entre l'Espace destination et l'Espace courant. Les relations possibles sont les suivantes :

- suspension de l'Espace courant jusqu'à la terminaison de l'Espace destination.
  Toute activité à l'intérieur de l'Espace courant est interdite tant que l'Espace destination est en cours ; à la fin de l'Espace destination, l'activité peut reprendre dans l'Espace courant.
- séquentialité entre les deux Espaces : l'accès à l'Espace destination s'effectue en mettant fin à l'Espace courant.
- relation d'appartenance : l'Espace destination prend la place de l'Espace courant ; à la fin de l'Espace destination, l'on retrouve l'Espace courant.

- parallélisme : les deux Espaces sont indépendants, l'activité de l'utilisateur pouvant entrelacer les traitements de l'un et de l'autre.

Logique d'enchaînement des tâches

Le modèle des Espaces de travail ci-dessus décrit permet au travers des opérations de navigation l'expression d'une logique d'enchaînement des traitements que nous souhaitons préciser ici. Cette logique d'enchaînement reflète les contraintes de fonctionnement du système, mais également une logique d'utilisation du système dont la définition est propre au concepteur.

On propose de décrire les transitions entre Espaces de Travail selon deux critères : le critère des données et le critère des contraintes d'exécution.

Soient WS1 et WS2 deux Espaces de Travail reliés par une relation de transition de WS1 vers WS2 (c'est-à-dire que WS1 définit une opération d'accès à WS2).

Critère des données :

Si l'on choisit comme axe d'observation les valeurs de Concepts présentées dans chacun des deux Espaces, trois cas peuvent être distingués :

- WS2 effectue un sous-traitement sur un ou plusieurs Concepts gérés dans WS1.
  Il y a transmission de valeurs d'attributs de WS1 vers WS2. Le sous-traitement effectué dans WS2 peut éventuellement modifier les données présentées dans WS1. Les données en question constituent les données principales de WS2. Les Perspectives adoptées par WS1 et WS2 sur un même concept donné sont a priori différentes.

Un exemple typique : la double Perspective sur un concept. WS1 offre une liste de Concepts dont seules les données-clef sont présentées ; WS2 présente le détail du contenu d'un Concept de cette liste.

- WS2 utilise un ou plusieurs Concepts gérés par WS1, en entrée seulement. Il y a transmission de valeurs de concepts de WS1 vers WS2. Les données ne sont pas modifiées ; elles participent simplement en entrée aux traitements effectués dans WS2. (Elles ne constituent pas les données principales de WS2).

Un exemple : WS1 met en œuvre la saisie d'un formulaire de requêtes "bases de données", et WS2 met en œuvre l'extraction des données selon les critères spécifiés dans WS1.

- Les données de WS1 et de WS2 sont disjointes.

Critère des contraintes d'exécution :

Si l'on choisit comme axe d'observation les relations définies entre les deux Espaces, trois cas peuvent être distingués :

- WS1 est désactivé, et ne peut être à nouveau actif que lorsque l'existence de WS2 prend fin.

WS2 est alors le lieu de traitements invalidant WS1.
Ce type de contrainte apparaît dans les relations dites de suspension et d'appartenance.

- Le lancement de WS2 signifie la terminaison de WS1.
(Notion de séquence entre WS1 et WS2).
- WS1 et WS2 peuvent être poursuivis en parallèle.

3.3.2 Perspectives

Une Perspective définit le mode d'accès à un Concept de l'Application depuis un Espace de Travail. Ce mode d'accès comporte deux grands aspects : d'une part, un filtre sur les éléments du Concept, et d'autre part, la description du mode d'utilisation des éléments retenus.

Une Perspective se décrit comme une restriction sur le type du Concept de l'Application : la structure d'une Perspective est une liste d'attributs et d'opérations définie relativement aux attributs et opérations d'un Concept de l'Application. La définition d'une Perspective met en jeu un identificateur, l'identificateur du Concept sur lequel porte la Perspective, et la spécification du sous-ensemble d'attributs et d'opérations constituant la Perspective.

L'apparition d'un attribut dans une Perspective signifie que l'accès à cet attribut est autorisé en lecture et/ou en écriture, selon la façon dont l'attribut est décrit dans la Perspective. En ce qui concerne les attributs de type complexe (dont la valeur est elle-même un Concept), il est nécessaire de préciser la forme de cet accès, c'est-à-dire la ou les Perspectives portées sur cet attribut ; la définition d'une Perspective associe récursivement une liste de Perspectives à chaque attribut de type complexe apparaissant dans cette Perspective. En ce qui concerne les attributs de type construit, il est également nécessaire de préciser le mode d'accès à l'agrégat de données, par l'intermédiaire de Perspectives prédéfinies.

Comme tout élément du modèle, une Perspective ainsi que ses éléments structurels (attributs et opérations) sont dotés d'un ensemble de propriétés permettant le recueil d'informations sémantiques, logiques et structurelles. Le jeu de propriétés associé à une Perspective et à ses éléments a la particularité de permettre la surcharge des propriétés contextuelles attachées aux éléments correspondants du Concept sous-jacent. Toute propriété contextuelle d'un élément du Concept (par exemple, la propriété de nom d'un attribut du Concept) est susceptible d'être surchargée au niveau de la Perspective, en redéfinissant cette propriété sur l'attribut référencé dans la Perspective.

La Figure IV-8 illustre le contenu d'une Perspective par rapport à celui du Concept sur lequel cette Perspective est définie.

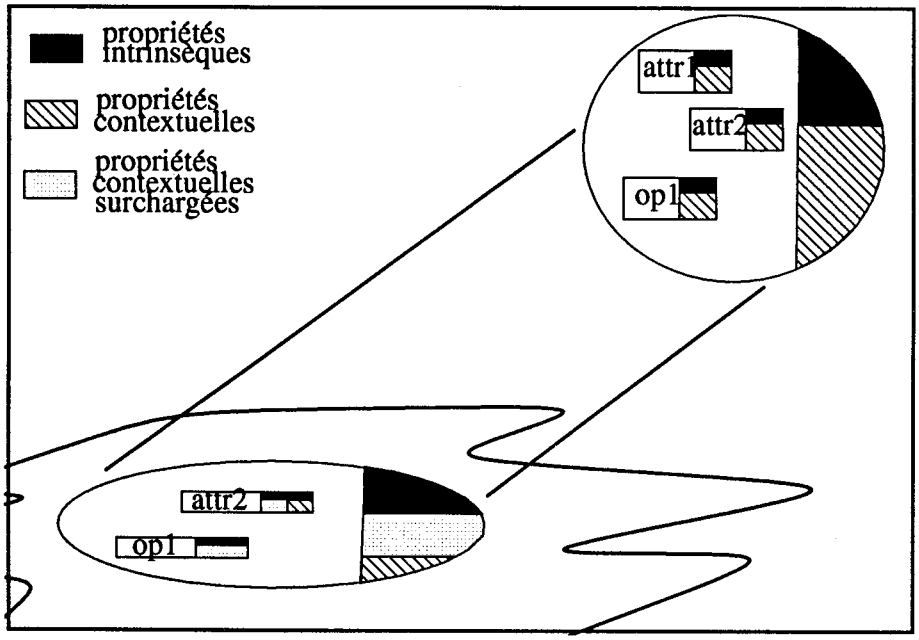


Figure IV-8 *La Perspective par rapport au Concept de l'Application.*

Une Perspective opère une sélection sur les attributs et les opérations d'un Concept, ainsi qu'une éventuelle surcharge des propriétés contextuelles définies sur les éléments du Concept.

En dehors de la possibilité de redéfinir les propriétés contextuelles d'un élément du Concept de l'Application, une Perspective admet un ensemble de propriétés intrinsèques à la dimension d'utilisation (ces propriétés ne sont pas représentées dans la Figure IV-8).

La Figure IV-9 énumère les principales propriétés attachées aux Perspectives.

		informations sémantiques	informations logiques et structurelles
sur une Perspective	**propriétés intrinsèques**	*nature* traits sémantiques	*mode de validation*
	propriétés en surcharge	identification	
sur un attribut	**propriétés intrinsèques**	*nature* traits sémantiques	
	propriétés en surcharge	identification	valeur initiale
sur une Opération	**propriétés intrinsèques**	*nature* traits sémantiques	cohérence entre paranètres pré-conditionpost-action
	propriétés en surcharge	identification	
sur un paramètre d'Opération	**propriétés intrinsèques**	*nature* traits sémantiques	
	propriétés en surcharge	identification	valeur initiale d'un param. d'entrée liaison d'un param. de sortie

Figure IV-9 *Propriétés d'une Perspective.*

Les propriétés sont classées selon qu'elles ont pour but le recueil d'informations sémantiques ou bien logiques. Nous distinguons de plus les propriétés intrinsèques à la Perspective et les propriétés définies en surcharge des propriétés contextuelles du Concept sous-jacent. Les propriétés apparaissant en italiques admettent des valeurs génériques, tandis que les autres sont spécifiques à l'application.

Recueil d'informations sémantiques

Les propriétés de nature, d'identification, et de traits sémantiques définies pour la dimension de fonctionnement s'appliquent également ici : toute Perspective, tout attribut et toute opération d'une Perspective est doté de ces propriétés. Si les propriétés d'identification s'utilisent en surcharge des propriétés définies sur les éléments du Concept sous-jacent, les propriétés de nature et de traits sémantiques sont propres à la Perspective et à ses éléments.

En ce qui concerne la **propriété de nature** d'une Perspective, trois valeurs génériques permettent de préciser d'une part la nature de la Perspective portée sur le Concept de l'Application, et d'autre part le mode d'emploi du Concept sous-jacent.

La valeur de "prototype" permet de préciser si le Concept considéré est un objet existant ou bien si c'est un prototype d'objet, la Perspective étant destinée à la collecte de données d'initialisation en vue de l'instanciation de l'objet Concept (notions introduites en 3.1.2).

La valeur de "conteneur" exprime le fait que la valeur du Concept sous-jacent est potentiellement modifiable au travers de la Perspective.

La valeur de "sélectionnable" indique une Perspective représentative du Concept sous-jacent, manipulable comme une entité.

Recueil d'informations logiques

Une **propriété de mode de validation** permet de préciser la façon dont opèrent les accès en écriture sur les attributs. Il s'agit de définir si les modifications des attributs sont sujet à une validation explicite ou bien si cette validation est implicitement mise en œuvre.

Quelques propriétés supplémentaires sont définies, notamment pour les Perspectives sur des attributs de type construit. Nous ne les détaillerons pas ici. Le lecteur intéressé se reportera au manuel du langage SIROCO [Normand 91d].

4 Le langage SIROCO

Le modèle SIROCO, décrit dans la section précédente, offre un support pour la représentation d'un système interactif. Ce modèle est muni d'un langage permettant de formaliser la spécification d'un système : le langage SIROCO.

Nous présentons ici les principaux aspects du langage SIROCO : la forme générale de la syntaxe définie, et surtout les contrôles mis en œuvre sur une spécification.

4.1 Forme d'une spécification SIROCO

Dans sa forme actuelle, le langage SIROCO est fortement ancré dans le langage Guide [Nguyen Van 90]. Si la forme générale des déclarations SIROCO est propre à notre langage, les éléments procéduraux : conditions booléennes et listes d'instructions apparaissant dans le langage SIROCO (notamment, les préconditions et post-actions définies sur les opérations) sont exprimées selon la syntaxe Guide.

Il s'agit là d'un choix de conception très net : notre objectif n'est pas de définir un nouveau langage d'expression procédurale, mais d'intégrer un langage existant comme support de SIROCO. Le choix de Guide est dicté par notre environnement de travail. Dans l'optique de la génération de code qui sera présentée au Chapitre V, ce choix est de plus simplificateur puisque Guide est également le langage cible de la génération. Reste à déterminer si le langage Guide est ou non adapté à l'emploi qui en est fait ; les perspectives d'évolution de SIROCO seront évoquées lors de la Conclusion Générale de ce document.

Pour l'heure, la syntaxe du langage est très grossièrement ébauchée ; quelques précisions sont de plus apportées relativement à la mise en œuvre de la désignation.

4.1.1 Syntaxe

Une spécification SIROCO est une suite de définitions de types abstraits étiquetés. Une spécification SIROCO comporte quatre grandes parties : définition des types des Concepts de l'application, des Perspectives (éventuellement vide), et des Espaces de Travail, puis définition de l'objet Session.

<spécification> = *<Concept>**
*<Perspective>**
<Espace>$^+$
< Session >

La définition d'un élément du modèle SIROCO (Concept, Perspective, attribut, opération, paramètre, etc.) s'effectue sous la forme d'un bloc déclaratif indépendant. Seule la définition d'une Perspective peut intervenir à l'intérieur d'un bloc Espace, ou bien à l'intérieur d'une autre Perspective. Cette définition suit un schéma régulier que l'on peut définir ainsi (seul le format de la Session s'écarte de ce shéma) :

<élément> = *<en-tête_élément>*
*<propriété_élément>**
*[<sous-élément>**
<fin_élément>]

<en-tête_élément> définit le type de l'élément, son identificateur, ainsi que ses caractéristiques principales dans le contexte courant de la définition.

*<propriété_élément>** définit les propriétés logiques et sémantiques de l'élément sous la forme d'une suite de couples étiquette/valeur.

Dans le cas d'un élément structuré (par exemple, de type Concept), *<sous-élément>** définit les sous-éléments qui décomposent l'élément. La définition de ces sous-éléments respecte le schéma ci-dessus.

<fin_élément> met fin à la définition de l'élément. Cette clause n'est requise que dans le cas d'un élément structuré.

Nous ne donnerons pas plus de détails quant à la syntaxe du langage SIROCO dans cette section. Nous fournissons en Annexe A le descriptif de la grammaire du langage. De plus, la section 5 donne lieu à la présentation de larges extraits d'une spécification en langage SIROCO. Le lecteur intéressé se reportera au manuel du langage [Normand 91d].

4.1.2 Désignation

Deux niveaux de désignation peuvent être distingués : d'une part un niveau global, et d'autre part un niveau local aux blocs de définition des éléments de base de la spécification.

Tout identificateur de Concept, d'Espace de Travail et de Session a une portée globale au fichier de spécification. Il en va de même pour les identificateurs des Perspectives définies au niveau global.

Tout identificateur d'attribut ou d'opération d'un Concept a une portée locale au bloc de définition correspondant ; cependant un tel identificateur est "accessible" globalement, en le faisant précéder, séparé par un point, du nom de la variable contenant l'objet sur lequel cet identificateur est défini, à la manière des variables visibles et des méthodes d'un objet Guide.

Toute variable définie dans la Session est connue globalement, et est accédée en faisant précéder son identificateur de l'identificateur donné à l'objet Session, séparé par un point.

Tout identificateur d'attribut ou d'opération défini à l'intérieur d'un Espace de Travail a une portée locale à ce bloc.

4.2 Contrôles sur une spécification

L'utilisation du langage SIROCO offre un support à la mise en œuvre de divers types de vérifications sur une spécification. Au-delà des simples contrôles de validité syntaxique et lexicale, il s'agit de vérifier l'état d'achèvement et la cohérence de la spécification.

L'*état d'achèvement* d'une spécification est principalement évalué en testant la connexité des éléments décrits dans cette spécification. On vérifie que l'ensemble des éléments spécifiés est "connexe", c'est-à-dire que chaque élément est "accessible" lors de l'exécution de l'application. Cette vérification s'effectue en construisant le graphe complet des Espaces de Travail pouvant être instanciés lors d'une exécution, à partir des Espaces apparaissant dans la description de l'état initial de l'objet Session. Ce graphe permet de déterminer l'ensemble des Espaces qui peuvent intervenir lors de l'exécution, mais aussi, par ricochet, l'ensemble des Concepts "accessibles" lors de l'exécution, ainsi que l'ensemble des Perspectives mises en jeu. L'analyse de ce graphe permet de déterminer :

- les Espaces, Perspectives et Concepts spécifiés mais n'intervenant pas dans l'exécution.
- pour chaque Concept, les attributs et opérations spécifiés mais n'intervenant pas dans l'exécution.

Ce type de vérification engendre d'éventuels avertissements ; la notion d'erreur n'existe pas, un manquement à la règle de connexité pouvant résulter d'une intention du concepteur (notamment en phase de test d'une spécification en cours d'élaboration).

Plusieurs types de *vérifications de cohérence* peuvent être envisagés. Nous nous contentons pour l'instant de vérifier que l'interface spécifiée permet à l'utilisateur de quitter l'application (!), propriété élémentaire de toute interface.

Le principe des contrôles sur une spécification sera repris et étendu lors de la présentation des perspectives de nos travaux, en Conclusion Générale.

5 Un mode d'emploi, par l'exemple

A ce stade de notre présentation, les différents composants du modèle SIROCO ont été définis de façon théorique. Cette section aborde les aspects pratiques du mode d'emploi de ce modèle.

Avant toute chose, une mise au point est nécessaire quant à la notion de mode d'emploi. Nous définissons dans cette section *un* mode d'emploi, et non *le* mode d'emploi du modèle SIROCO. L'approche que nous décrivons, et notamment le séquencement des étapes de conception et de spécification doivent être considérés comme indicatifs plutôt que définitifs.

Nous proposons de construire notre présentation sur l'étude de la spécification d'un système interactif particulier. Cet exercice de spécification offre deux intérêts : d'une part il permet de concrétiser les définitions théoriques données précédemment, et d'autre part il permet de préciser la façon dont la spécification SIROCO s'insère dans le travail de conception d'un système interactif.

L'exemple choisi est celui d'un système offrant les fonctions bien connues d'une *messagerie électronique* - système d'où est issue la plupart des exemples émaillant nos précédentes définitions. Ce système, simple mais non trivial, constitue un excellent terrain d'expérimentation du modèle SIROCO.

L'organisation de cette section suit la démarche de conception et de spécification du système de messagerie. Elle a pour point de départ naturel l'analyse des tâches, puis aborde successivement la définition des Concepts de l'Application, des Espaces de Travail et des Perspectives du système.

Avertissement :

La spécification SIROCO présentée dans cette section est "minimale" dans le sens où, dans un souci de simplification, seules les propriétés fondamentales des éléments du modèle sont définies. Notamment, aucun descriptif (long ou court), aucun trait sémantique, etc. ne sont définis.

5.1 Analyse des tâches

La conception d'un système nécessite tout d'abord de définir l'objectif de ce système : ce que l'utilisateur va pouvoir effectuer avec ce système. En accord avec les principes évoqués aux Chapitres I et II, c'est une analyse des tâches du domaine qui précède tout travail de conception.

L'objectif du système considéré est de permettre la communication par messages entre les utilisateurs. Sont à préciser les entités du domaine ainsi que l'arbre des tâches que le système doit permettre de réaliser.

5.1.1 Les entités du domaine

Le système de messagerie, ou application Messagerie, repose sur les notions de message, d'utilisateur, et de file de messages, ou boîte-à-lettres. Telles sont les *entités du domaine*. Un message comporte un sujet et un contenu ; un utilisateur s'identifie par un nom et un mot-de-passe, et possède une boîte-à-lettres.

C'est principalement à partir des entités du domaine que seront définis les Concepts de l'Application.

5.1.2 L'arbre des tâches du domaine

L'application Messagerie gère un ensemble d'utilisateurs munis de boîtes-à-lettres, et offre les services connus d'envoi de messages à un utilisateur et de lecture d'un message reçu en provenance d'un utilisateur. Une analyse détaillée du système, et des grandes tâches d'envoi et de lecture d'un message permet de déterminer l'ensemble des tâches élémentaires auxquelles le système doit apporter un support. Ces *tâches du domaine*, liées par une structure hiérarchisée, constituent un arbre des tâches présenté en Figure IV-10.

L'arbre représenté est un arbre de décomposition générale des tâches intervenant dans l'application Messagerie : aucune hypothèse n'est faite sur le logiciel lui-même, sur le type d'interface de ce logiciel, sur les modalités d'entrée et de sorties mises en œuvre, etc. Sans lister en détail chacune des tâches apparaissant dans l'arbre, remarquons les *tâches principales*, qui figurent en caractères gras dans la figure : se connecter, lire un message, répondre à un message, et envoyer un message. Notons également les différents types de décomposition des tâches reflétés dans la figure : une tâche peut se décomposer en une séquence de sous-tâches, une suite non ordonnée de sous-tâches, la répétition en boucle d'une sous-tâche, une alternative entre plusieurs sous-tâches, ou bien une alternative non exclusive entre plusieurs sous-tâches (c'est-à-dire que zéro, une ou plusieurs des sous-tâches pourront être réalisées par l'utilisateur).

C'est à partir des différentes caractéristiques de l'arbre des tâches que s'effectuera la conception des Espaces de Travail et Perspectives du système.

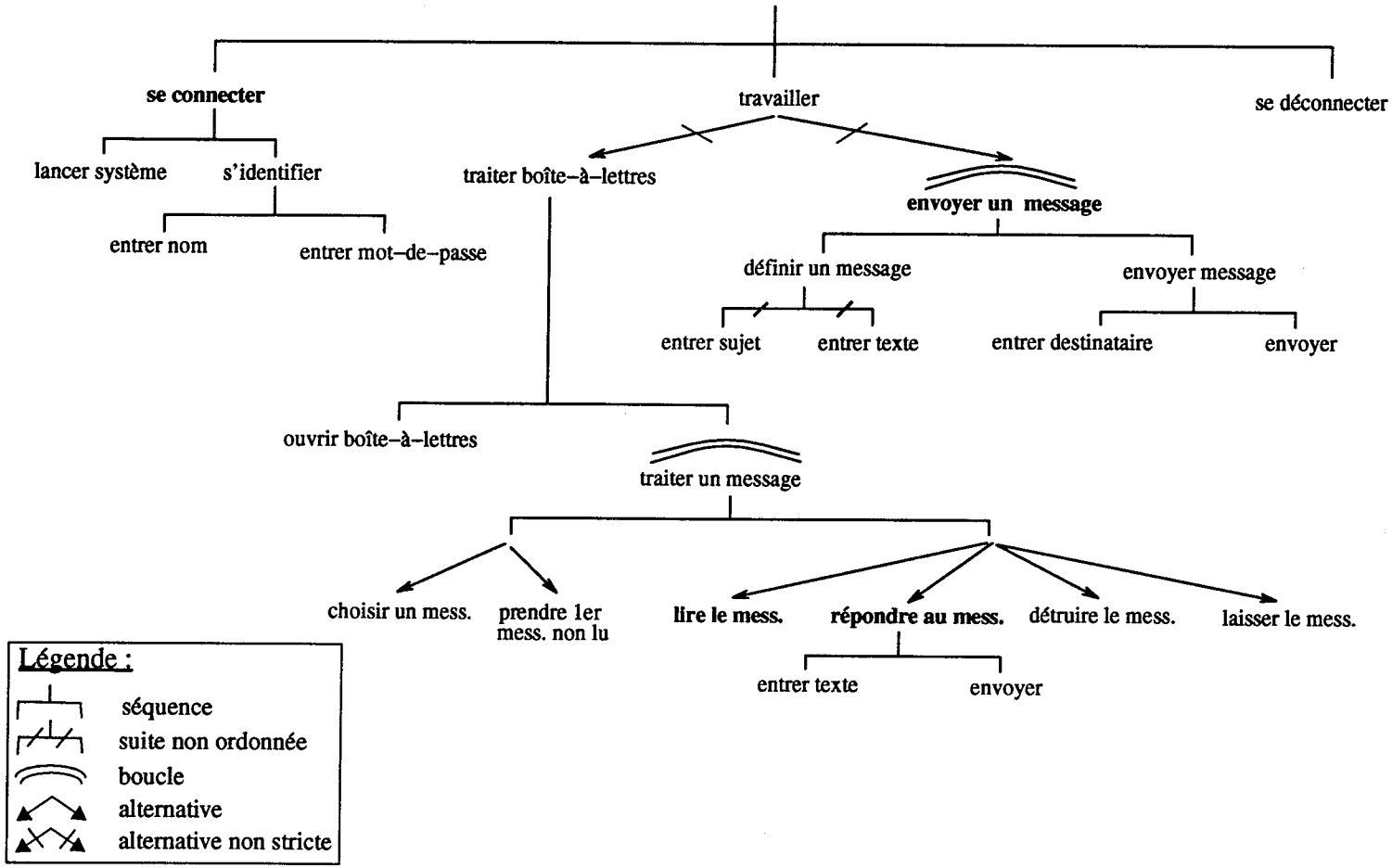


Figure IV-10 *L'arbre des tâches.*
(Les tâches principales sont soulignées par des caractères gras.)

5.2 Spécification des Concepts de l'Application

A partir des entités du domaine de la tâche, l'on va chercher à définir les objets conceptuels du système. La définition d'un Concept de l'Application repose sur un ou plusieurs des éléments suivants :

- une entité du domaine de la tâche,
- une métaphore d'interface,
- des précisions et contraintes fonctionnelles.

Les Concepts que nous proposons pour l'application Messagerie sont au nombre de six : utilisateur, base d'utilisateurs, boîte-à-lettres, message, message reçu, et message à envoyer. La suite de cette section passe chaque Concept en revue, explicitant son origine, et détaillant son contenu en donnant sa spécification en langage SIROCO.

5.2.1 Utilisateur

Le Concept d'utilisateur est appelé User_AC[1] ; sa définition est directement issue de l'entité du domaine correspondante. Une fois le Concept identifié, il s'agit de définir avec précision sa structure et ses propriétés.

La structure du Concept : Le Concept User_AC dénote un pur objet de données, sans traitement associé. Les attributs du Concept User_AC sont définis à partir des éléments caractéristiques de l'entité du domaine : nom, mot de passe, et boîte-à-lettres. La définition précise des attributs fait intervenir celle de leur type, mais aussi celle de l'accès autorisé sur ces attributs : lecture ou écriture.

Les propriétés du Concept : Les propriétés principales du Concept et de ses éléments donnent leur identification textuelle, leur taille, ainsi que leur nature.

On définit pour User_AC trois attributs : *name* et *password*, de type chaîne de caractères, et *mbox*, qui référence un Concept boîte-à-lettres, composant de User_AC. La spécification de User_AC dans le langage SIROCO est la suivante[2] :

```
CONCEPT User_AC IS
    _name : "utilisateur"                  // propriété d'identification
    R_ATTRIBUTE name : string              // attribut de type simple, non modifiable
        _name : "nom"                      // prop. d'identification de l'attribut
        _kind : KEY;                       // prop. de nature "clé" de l'attribut
        _size : 30                         // prop. de taille de l'attribut
    R_ATTRIBUTE password : string          // attribut non modifiable
        _name : "mot de passe"
        _size : 8
    R_ATTRIBUTE mbox : MailBox_AC          // attribut de type complexe, non modifiable
        _name : "boîte à lettres"
        _composition : COMPONENT           // prop. de composition de l'attribut
END User_AC.
```

1. Rappel : AC signifie "Application Concept".

2. Pour plus de clarté, les mots clefs du langage ainsi que les valeurs prédéfinies des propriétés apparaissent en majuscules. Les étiquettes des propriétés ont pour point commun de commencer par le caractère '_'. Les commentaires sont introduits par la séquence "//".

5.2.2 Base des utilisateurs

Le Concept de base des utilisateurs est appelé UserList_AC. Ce Concept représente une notion restée implicite lors de l'analyse de la tâche, celle de la liste des utilisateurs connus du système Messagerie.

La base des utilisateurs est une simple liste d'utilisateurs, sur laquelle peut être effectuée une opération de recherche et d'extraction d'un utilisateur.

On définit pour UserList_AC un attribut de type construit, *users*, et une opération, *GetUser*, selon la spécification suivante :

```
CONCEPT UserList_AC IS
        _name : "base des utilisateurs"
        R_ATTRIBUTE users : list of User_AC          // attribut construit, non modifiable
                _name : "utilisateurs"
        OPERATION GetUser                            // opération du Concept
                _name : "rechercher un utilisateur"
                IN name : string                     // paramètre d'entrée de l'op.
                        _name : "nom"
                        _size : 30
                IN password : string                 // paramètre d'entrée de l'op.
                        _name : "mot de passe"
                        _size : 8
                OUT user_retrieved : User_AC         // paramètre de sortie de l'op.
                        _name : "utilisateur"
END UserList_AC.
```

5.2.3 Boîte-à-lettres

Le Concept de boîte-à-lettres est appelé MailBox_AC. Comme l'entité du domaine identifiée lors de l'analyse des tâches, le Concept de boîte-à-lettres est une simple liste de messages reçus.

La spécification SIROCO est la suivante :

```
CONCEPT MailBox_AC IS
        _name : "boîte à lettres"
        R_ATTRIBUTE messages : list of ReceivedMess_AC
END MailBox_AC.
```

5.2.4 Message, message reçu et message à envoyer

Définis à partir de l'entité de message, les trois Concepts de message, message reçu et message à envoyer permettent l'expression de contraintes fonctionnelles différentes selon que le message apparaît dans une situation de réception ou d'émission.

Le Concept de message est appelé Mess_AC. Il comporte deux attributs permettant d'identifier le message : l'envoyeur (l'utilisateur qui l'a envoyé), et la date de réception, et deux attributs décrivant le contenu du message : le sujet, et le texte. Un message peut subir deux opérations : d'une part il peut être envoyé à un utilisateur, et d'autre part il peut être détruit.

La spécification SIROCO est la suivante :

```
CONCEPT Mess_AC IS
        _name : "message"
        ATTRIBUTE sender : User_AC          // attribut modifiable
                _name : "envoyeur"
                _kind : KEY;                // attribut participant à la "clé" du AC
        ATTRIBUTE time : string             // attribut modifiable
                _name : "date de reception"
                _kind : KEY;
                _size : 30
        ATTRIBUTE subject : string          // attribut modifiable
                _name : "sujet"
                _kind : KEY;
                _size : 40
        ATTRIBUTE text : string             // attribut modifiable
                _name : "texte"
                _size : 400
        OPERATION Delete                    // opération sans paramètres
                _name : "destruction"
                _kind : destroy;            // opération de nature "dangereuse"
        OPERATION Post                      // opération ayant un paramètre
                _name : "envoi du message"
                IN whom : string
                        _name : "nom du destinataire"
                        _size : 30
END Mess_AC.
```

Le concept de message reçu est appelé RecMess_AC. Un message reçu est un message dont le sujet et le texte ne sont pas modifiables, et sur lequel l'opération d'envoi n'est pas appliquable. (N.B. : Notre système exemple, dans un souci de simplification, n'autorise pas la retransmission ("forwarding") d'un message reçu.)

La spécification SIROCO utilise l'opérateur de restriction de la façon suivante :

```
CONCEPT RecMess_AC RESTRICTS Mess_AC IS   // restriction de type
        _name : "message reçu"
                        // attributs retenus
        R_ATTRIBUTE sender                // accès restreint à la lecture seule
        R_ATTRIBUTE time                  // idem
        R_ATTRIBUTE subject               // idem
        R_ATTRIBUTE text                  // idem
                        // opération retenue
        OPERATION Delete
END RecMess_AC.
```

Le concept de message à envoyer est appelé SendMess_AC. Un message à envoyer est un message sur lequel on ne peut appliquer que l'opération d'envoi.

La spécification SIROCO est la suivante :

```
CONCEPT SendMess_AC RESTRICTS Mess_AC IS  // restriction de type
        _name : "message à envoyer"
                        // attributs retenus
        R_ATTRIBUTE sender                // accès restreint à la lecture seule
        R_ATTRIBUTE time                  // idem
        ATTRIBUTE subject                 // accès non restreint
        ATTRIBUTE text                    // idem
                        // opération retenue
        OPERATION Post
END SendMess_AC.
```

5.2.5 Récapitulatif

La Figure IV-11 représente les différents Concepts de l'Application que nous venons de définir en mettant l'accent sur les relations existant entre ces Concepts.

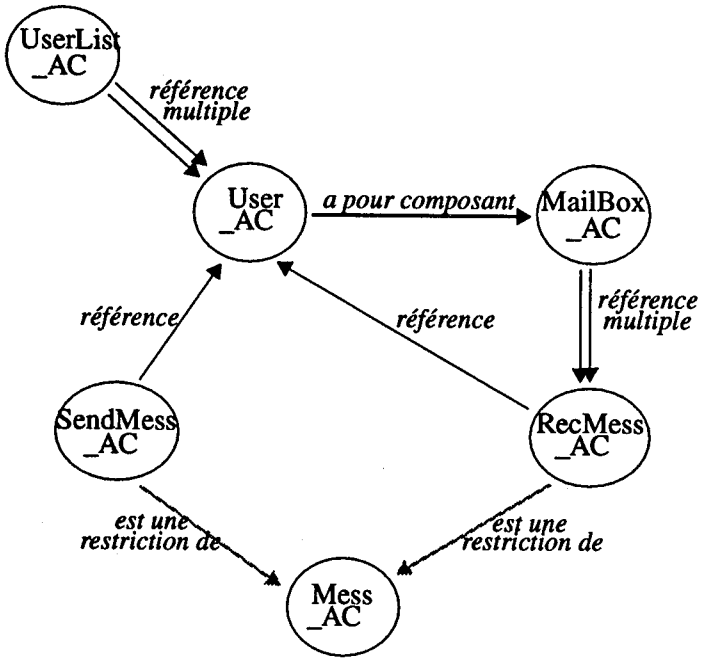


Figure IV-11 *Les Concepts du système Messagerie.*

5.3 Spécification des Espaces de Travail et Perspectives

Il s'agit maintenant de concevoir le mode d'utilisation des Concepts de l'Application : définir les Espaces de Travail.

La conception d'un Espace de Travail peut se décomposer ainsi :

1. Choix des tâches que l'Espace permettra de réaliser.
2. Identification des Concepts de l'Application nécessaires à la réalisation des tâches choisies.
   Il s'agit d'identifier non seulement les types de Concepts en jeu mais également les instances de Concepts.
3. Définition des Perspectives sur les Concepts de l'Application identifiés.
4. Insertion de l'Espace dans la logique de la session.
   Il s'agit de définir comment l'Espace sera mis à la disposition de l'Utilisateur.

Notons que si les trois premières étapes se suivent naturellement, la quatrième peut tout à fait survenir à un moment quelconque de la conception de l'Espace.

Nous proposons pour le système Messagerie un découpage de l'arbre des tâches en cinq groupes de tâches, à partir desquels nous définissons cinq Espaces de Travail. Le découpage de

l'arbre est représenté en Figure IV-12. On remarque qu'une tâche n'est pas couverte (l'ouverture de la boîte-à-lettres) ; elle est mise en œuvre automatiquement par le système.

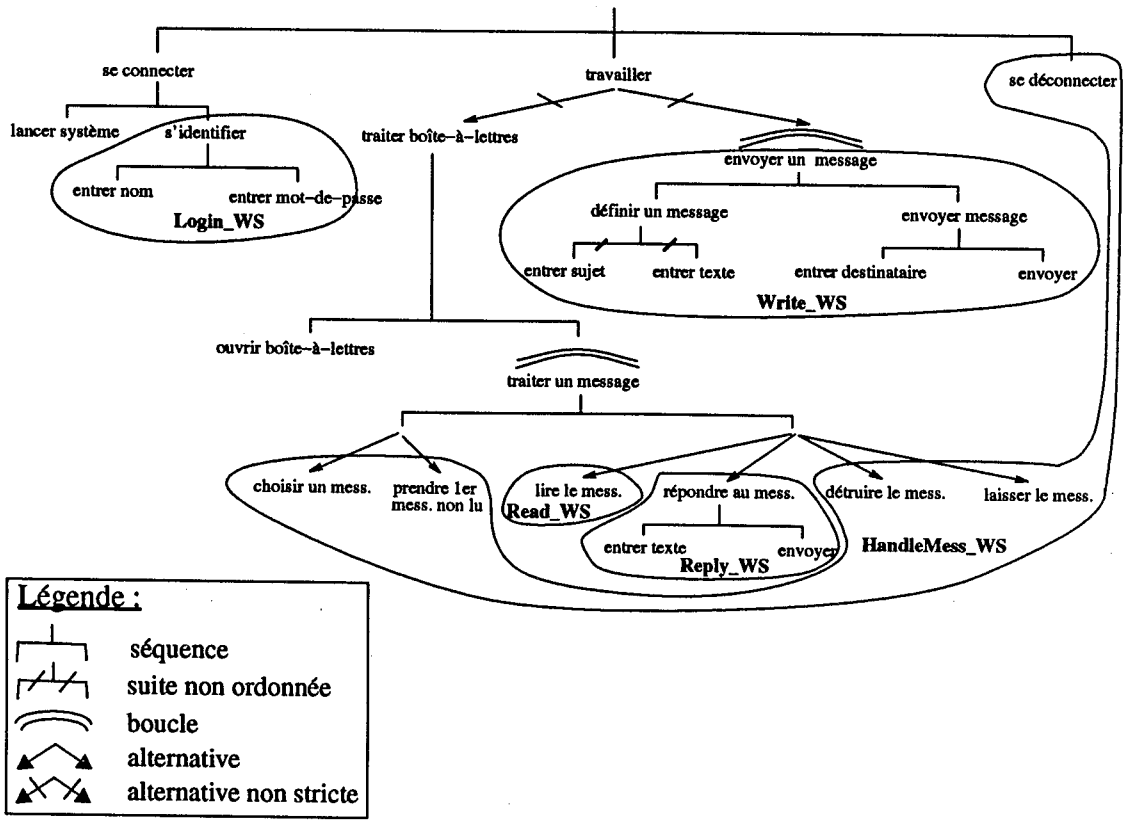


Figure IV-12 *Tâches et Espaces de Travail.*

Parmi les tâches de l'arbre, nous privilégions les cinq tâches suivantes : "se connecter", tâche préalable à toute autre, "traiter un message" et "envoyer un message", qui sont les tâches importantes du système, "lire le message", qui est la principale sous-tâche de "traiter un message", et "répondre au message", qui peut être considérée comme une variante de la tâche "envoyer un message". Parmi ces cinq tâches, nous considérons que la tâche "traiter un message" est la tâche principale de l'utilisateur.

Notre analyse nous amène à définir les Espaces de Travail suivants : connexion (Login_WS[1]), traitement (HandleMess_WS), lecture (Read_WS), réponse (Reply_WS), et écriture (Write_WS). La suite de cette section décrit chacun de ces Espaces de Travail en mettant en valeur les quatre facettes résultant des quatre étapes précédemment définies : les tâches, les Concepts, les Perspectives, et l'insertion dans la dynamique d'utilisation du système.

1. Rappel : WS signifie "Workspace", pour Espace de Travail.

Avant d'aborder le détail de la spécification des Espaces de Travail de notre système exemple, nous rapportons quelques éléments de réflexion concernant le choix des tâches figurant dans un Espace de Travail.

5.3.1 Le choix des tâches d'un Espace de Travail

L'étape du choix des tâches figurant dans un Espace de Travail est bien entendu déterminante pour la spécification de cet Espace. Quels sont les critères, quelles sont les règles applicables lors de cette étape? En dehors des recommandations générales que peuvent fournir les ergonomes, l'on ne dispose pas à l'heure actuelle de règles méthodiques précises. Notre propre analyse met en lumière quelques axes de réflexion, que nous rapportons ici sous la forme d'une suite d'indications.

Indication 1 : *Si nécessaire, compléter l'arbre des tâches du domaine par des tâches "syntaxiques".*

C'est de façon privilégiée à l'intérieur de l'arbre des tâches que le choix des tâches d'un Espace de Travail sera effectué. La définition d'un Espace de Travail peut cependant également résulter de l'identification de besoins n'apparaissant pas dans l'arbre des tâches. De tels besoins mettent en jeu des tâches qui ne sont pas liées au domaine de l'application mais à l'interface : tâches de l'interface, au sens des commandes du niveau interface evoquées au Chapitre III, section 2.4. Par analogie avec ces commandes, l'on parlera de tâches "syntaxiques" ou "passives". Ces tâches syntaxiques sont principalement des tâches de visualisation et de manipulation de données.

Par exemple, la visualisation d'une structure de données complexe pourra nécessiter la définition de plusieurs points de vue sur cette structure ; le passage d'un point de vue à un autre est une tâche syntaxique.

Indication 2 : *Distinguer les tâches automatisées et les tâches dont la réalisation incombe à l'utilisateur.*

L'ensemble des Espaces de Travail définis pour un système doit couvrir l'arbre des tâches dans sa totalité ; il est cependant possible que certaines tâches ne soient pas explicitement affectées à un Espace, mais soient automatiquement réalisées par le système. L'automatisation d'une tâche revient à raccourcir ou supprimer une branche de l'arbre. Les options offertes à l'utilisateur sont réduites en conséquence, ce qui a pour effet de limiter les parcours de ce dernier à l'intérieur de l'arbre des tâches.

Dans l'exemple de la Messagerie, l'on choisit d'automatiser la tâche d'ouverture de la boîte-à-lettres. Ce choix repose sur l'hypothèse que l'utilisateur de la Messagerie est interessé par le contenu de sa boîte-à-lettres la plupart du temps, lorsqu'il se connecte au système.

Indication 3 : *Décider si l'interface doit ou non favoriser le multitâche.*

En fonction des contraintes de l'environnement (notamment, la disponibilité ou non d'un système de multifenêtrage), l'on définit l'orientation générale de l'interface relativement au multitâche : souhaite-t-on permettre à l'utilisateur de conduire plusieurs tâches en parallèle, ou bien lui impose-t-on un déroulement strictement séquentiel?

Indication 4 : *Dans l'optique du multitâche, identifier les tâches ou ensembles de tâches parallélisables.*

Les informations contenues dans l'arbre des tâches nous renseignent sur les parallélismes possibles entre les procédures de réalisation des tâches. Dans l'exemple de la Messagerie, l'arbre défini en 5.1 explicite un possible parallélisme entre :

- le traitement de la boîte-à-lettres et l'envoi d'un message,
- l'envoi de deux messages,
- le traitement de deux messages.

Indication 5 : *Dans l'optique du multitâche, déterminer les parallélismes utiles.*

Ce n'est pas parce que deux tâches sont parallélisables qu'elles doivent donner lieu à deux Espaces de Travail distincts. Intervient notamment la notion d'*utilité*. Les informations contenues dans un arbre des tâches tel celui défini précédemment décrivent une procédure *standard* de réalisation des tâches. En référence aux travaux de M.F. Barthet, c'est aux procédures *effectives* de réalisation des tâches que nous nous intéressons ici. La tâche effective correspond à ce que l'utilisateur fait effectivement. L'analyse des tâches effectives permet d'identifier les parallélismes utiles.

Dans l'exemple de la Messagerie, une analyse rapide des activités de lecture et d'envoi de messages nous conduit aux réflexions suivantes :

- la composition d'un message (lors des tâches d'envoi et de réponse) peut nécessiter d'avoir sous les yeux un ou plusieurs messages précédemment reçus.
- il peut être utile de composer deux messages en même temps.
- la lecture d'un message peut nécessiter un retour sur un message précédemment reçu.

En conséquence, l'on choisit de définir trois Espaces de Travail distincts pour les tâches d'envoi, de lecture et de réponse.

Indication 6 : *Parmi les tâches strictement séquentielles, isoler les tâches effectivement distinctes.*

L'arbre des tâches nous renseigne quant aux séquences de tâches (non terminales). Dans certains cas, une séquence donnera lieu à des Espaces de Travail distincts ; dans d'autres cas, la séquence de tâches sera réalisée à l'intérieur du même Espace de Travail. Quels sont les critères conduisant à préférer une solution à l'autre? Une heuristique simple met en jeu les objets manipulés dans la séquence de tâches : si le recouvrement entre tâches est nul, c'est la solution des Espaces de Travail distincts qui tend à être préférable ; à l'inverse, si la séquence porte sur les mêmes objets, un seul Espace de Travail sera défini. Il est clair que de nombreux autres critères entrent en jeu, parmi lesquels :

- des critères de "volume" : il peut être nécessaire de partitionner une longue séquence de tâches en Espaces distincts afin de limiter la taille et la complexité des Espaces.

- des critères de cohérence : dans le cas où une sous-séquence apparaît dans plusieurs séquences, il peut être utile de l'isoler dans un Espace de Travail distinct.
- ...

Dans le système Messagerie, la tâche de connexion est clairement distincte des autres tâches, puisqu'elle ne porte ni sur la boîte-à-lettres, ni sur un message ; d'où la définition d'un Espace de Travail particulier. La séquence de réalisation de la tâche d'envoi d'un message n'est par contre pas décomposée, car elle porte sur le même objet message ; elle sera réalisée dans le même Espace de Travail.

Indication 7 : *Eventuellement, déterminer si une tâche doit être représentée dans deux Espaces différents.*

Le découpage de l'arbre des tâches mis en œuvre lors de la définition de ces Espaces n'est pas forcément une partition : dans certains cas, il pourra être intéressant qu'une même tâche figure dans deux Espaces différents. Cette possibilité est à manier avec prudence : en cherchant à simplifier la réalisation des tâches de l'utilisateur, l'on risque de compliquer le modèle mental de ce dernier (la même tâche pourra être réalisée dans deux contextes de travail différents). Ici encore, il est nécessaire d'observer des procédures effectives de réalisation des tâches.

La question se pose par exemple pour la tâche de destruction de message dans le système Messagerie. Cette tâche, que nous choisissons d'associer à l'Espace général de traitement d'un message, pourrait également figurer dans les Espaces de lecture et de réponse.

5.3.2 L'Espace de connexion

L'Espace de Travail connexion est appelé Login_WS. Il permet la mise en œuvre de la tâche composée "s'identifier".

Concepts et Perspectives

Les Concepts de l'Application mis en jeu sont la base des utilisateurs du système, et l'utilisateur courant de la Messagerie. Plus précisément, l'Espace Login_WS est le lieu de réalisation d'un unique traitement : la récupération de l'objet User_AC représentant l'utilisateur dans la base des utilisateurs (opération GetUser de UserList_AC).

Spécification

L'objet UserList_AC, base des utilisateurs du système, est défini comme étant global au système ; de même, l'objet User_AC représentant l'utilisateur du système sera stocké globalement au système. Ainsi, la spécification de Login_WS fait référence aux données globales définies dans la session du système Messagerie (cf. la définition de l'objet Session, en fin de section) : *bd* (base des utilisateurs) et *user* (utilisateur courant). Elle est la suivante :

```
WORKSPACE Login_WS IS
    _name : "Connexion à la messagerie"     // prop. de nom
    _instances : UNIQUE;                    // prop. d'instanciation unique
    _kind : EXIT_GATE;                      // prop. de nature " porte de sortie"
```

```
                    // objet Concept accessible dans l'Espace :
                    // la base des utilisateurs du système
GLOBAL session.bd : UserList_AC          // donnée globale à la session
                    // Perspective décrivant le mode d'accès au Concept
    WITH PERSPECTIVE Login_P IS      // définition locale de la Perspective
        OPERATION GetUser                    // opération retenue
            _name : "Identifiez-vous"
            OUT user_retrieved
                _link : session.user         // liaison du paramètre de
                                             // sortie à une variable de la
                                             // session
            ON SUCCESS DO                    // post actions
                WS_ACCESS TO HandleMess_WS // accès à un WS
                    _relation : SEQUENCE;  // en séquence
            END
    END Login_P      // fin de la définition locale de la Perspective

END Login_WS.
```

Dynamique d'utilisation

L'Espace de Travail Login_WS est destiné à une instanciation unique (propriété *_instances*), à l'initialisation de l'application (cf. la définition de l'objet Session). L'utilisation de cet Espace met en jeu la dynamique suivante :

- soit l'utilisateur met fin à l'Espace, ce qui entraîne la terminaison de la session (propriété *_kind*, qui indique une nature de "porte de sortie").
- soit l'utilisateur lance avec succès l'opération GetUser, en lui fournissant des valeurs de paramètres correctes. L'exécution de GetUser donne alors lieu à une opération de transition vers un Espace de type HandleMess_WS (post-action définie sur GetUser : *ON SUCCESS DO...*). Un Espace HandleMess_WS est instancié, qui entraîne la terminaison de l'Espace courant (propriété *_relation*, qui indique une relation de séquencement entre les deux Espaces).

5.3.3 L'Espace de traitement

L'Espace de Travail dédié au traitement général des messages est appelé HandleMess_WS. C'est l'Espace principal de l'application, point de lancement de toutes les activités de l'utilisateur. Les tâches effectivement mises en œuvre à l'intérieur de l'Espace sont celles du choix d'un message, et de la destruction d'un message. HandleMess_WS offre de plus l'accès aux Espaces permettant la réalisation des tâches de lecture d'un message, de réponse à un message, et d'envoi d'un message.

Concepts

Les Concepts de l'Application mis en jeu se réduisent à l'utilisateur courant de la Messagerie, Concept User_AC défini globalement au système.

Spécification

La spécification de HandleMess_WS est la suivante :

```
WORKSPACE HandleMess_WS IS
    _name : "boîte-à-lettres"
```

```
    _instances : UNIQUE;                    // prop. d'instanciation unique
    _kind : MAIN;                           // prop. de nature "WS principal"

                        // objet Concept accessible dans l'Espace :
                        // l'utilisateur du système
GLOBAL session.user : User_AC               // donnée globale à la session
        WITH PERSPECTIVE User_P             // Perspective utilisée

                        // opération d'accès à un Espace de Travail :
                        // accès à l'Espace de lecture d'un message
WS_ACCESS Read TO Read_WS
    _name : "Lire"
    _relation : PARALLEL;                   // parallélisme entre les deux WSs
    _data_transmission : init_mess := mess;     // transmission de données
                                                // au WS destination
    IN mess : RecMess_AC                    // paramètre de l'opération d'accès

                        // opération d'accès à un Espace de Travail :
                        // accès à l'Espace de réponse à un message
WS_ACCESS Reply TO Reply_WS
    _name : "Répondre"
    _relation : PARALLEL;                   // parallélisme entre les deux WSs
    _data_transmission : init_mess := mess;     // transmission de données
                                                // au WS destination
    IN mess : RecMess_AC                    // paramètre de l'opération d'accès

                        // opération d'accès à un Espace de Travail :
                        // accès à l'Espace d'écriture d'un message
WS_ACCESS Write TO Write_WS
    _name : "Ecrire"
    _relation : PARALLEL;                   // parallélisme entre les deux WSs

END HandleMess_WS.
```

Perspectives

L'accès au Concept User_AC mis en œuvre dans l'Espace de traitement est régi par une Perspective nommée User_P, qui présente le contenu de la boîte-à-lettres de l'utilisateur sous la forme d'une liste d'entêtes de messages autorisant l'opération de destruction d'un message. D'où la définition des perspectives suivantes :

Perspective User_P, sur l'utilisateur courant :

```
PERSPECTIVE User_P ON User_AC IS
    _name : "boîte-à-lettres"           // surcharge d'une propriété contextuelle

            // attributs retenus :
    ATTRIBUTE name
    ATTRIBUTE mbox WITH PERSPECTIVE MailBox_P

END User_P.
```

Perspective MailBox_P, sur la boîte-à-lettres de l'utilisateur courant :

```
PERSPECTIVE MailBox_P ON MailBox_AC IS
            // attributs retenus :
    ATTRIBUTE messages WITH PERSPECTIVE HeaderList_P
END MailBox_P.
```

Perspective HeaderList_P, sur la liste de messages contenus dans la boîte-à-lettres :

```
PERSPECTIVE HeaderList_P ON list of RecMess_AC IS
    _elem_persp : Header_P              // Perspective adoptée sur un élément
                                        // de la liste
    _nb_of_visible_elems : 10           // taille de la zone de visualisation
END HeaderList_P.
```

Perspective Header_P, sur un message de la liste :

```
PERSPECTIVE Header_P ON RecMess_AC IS
                // attributs retenus :
    R_ATTRIBUTE sender WITH PERSPECTIVE Sender_P
    R_ATTRIBUTE subject
    R_ATTRIBUTE time
                // opération retenue :
    OPERATION Delete
END Header_P.
```

Perspective Sender_P, sur l'envoyeur du message :

```
PERSPECTIVE Sender_P ON User_AC IS
    _name : "envoyeur"

                // attributs retenus :
    R_ATTRIBUTE name

END Sender_P.
```

Dynamique d'utilisation

L'Espace HandleMess_WS est destiné à une instanciation unique (propriété *_instances*), lancée à partir de l'unique instance de Login_WS (cf. 5.3.2). L'utilisation de cet Espace met en jeu la dynamique suivante :

- l'utilisateur peut mettre fin à l'exécution de l'Espace, ce qui entraîne la terminaison de l'application (propriété *_kind* indiquant une nature d'Espace principal de l'application, qui implique la notion de "porte de sortie" de l'application).
- l'utilisateur peut désigner un message comme objet d'une opération de destruction (opération Delete de Header_P).
- l'utilisateur peut désigner un message comme paramètre d'une activité de lecture (*WS_ACCESS Read*) et/ou de réponse (*WS_ACCESS Reply*), et ainsi activer l'Espace correspondant.
- l'utilisateur peut lancer une activité d'écriture et d'envoi d'un message (*WS_ACCESS Write*), et ainsi activer l'Espace correspondant.

Quel que soit l'Espace activé, la nouvelle activité est menée en parallèle avec celle de l'Espace courant (propriété *_relation*), en "multitâche".

5.3.4 L'Espace de lecture d'un message

L'Espace de lecture d'un message est appelé Read_WS. Il permet la lecture d'un message reçu, et offre l'accès à la tâche de réponse au message lu.

Concepts

Les Concepts mis en jeu se réduisent à l'objet RecMess_AC représentant le message à lire. Ce Concept est conservé localement à l'Espace.

Spécification

La spécification de Read_WS est la suivante :

```
WORKSPACE Read_WS IS
    _name : "lecture d'un message"
    _instances : MULTIPLE;                  // prop. d'instanciation multiple
    _unique_idem? : TRUE                    // prop. d'unicité

                        // objet Concept accessible dans l'Espace :
                        // le message reçu
    IN init_mess : RecMess_AC WITH PERSPECTIVE ReadMess_P

                        // opération d'accès à un Espace de Travail :
                        // accès à l'Espace de réponse à un message
    WS_ACCESS Reply TO Reply_WS
        _name : "Répondre"
        _relation : PARALLEL;               // parallélisme entre les deux WSs
        _data_transmission : init_mess := init_mess;  // transmission de données
                                                      // à l'Espace destination

END Read_WS.
```

Perspectives

La Perspective définissant l'accès au message lu offre une visualisation détaillée du contenu du message :

```
PERSPECTIVE ReadMess_P ON RecMess_AC IS
            // attributs retenus :
    R_ATTRIBUTE sender WITH PERSPECTIVE Sender_P
    R_ATTRIBUTE subject
    R_ATTRIBUTE time
    R_ATTRIBUTE text
END ReadMess_P.
```

Dynamique d'utilisation

L'Espace Read_WS est destiné à d'éventuelles instanciations multiples (propriétés *_instances*), à travers l'opération d'accès définie sur l'Espace de traitement des messages. Une instanciation ne sera cependant réellement effectuée que s'il n'existe pas déjà une instance de Read_WS opérant sur le même Concept de message (propriété *_unique_idem?*).

5.3.5 L'Espace de réponse à un message

L'Espace de réponse à un message est appelé Reply_WS. Il permet la définition et l'envoi d'un message de réponse à un message reçu.

Concepts

Les Concepts mis en jeu sont deux : l'objet RecMess_AC représentant le message reçu, et l'objet SendMess_AC représentant le message à envoyer en guise de réponse. Les deux objets Concept sont conservés localement à l'Espace.

Spécification

La spécification de Reply_WS est la suivante :

```
WORKSPACE Reply_WS IS
    _name : "réponse à un message"
    _instances : MULTIPLE;                    // prop. d'instanciation multiple
    _unique_idem? : TRUE                      // prop. d'unicité

                        // objet Concept accessible dans l'Espace :
                        // le message reçu, donnée d'initialisation
IN init_mess : RecMess_AC WITH PERSPECTIVE Header_P

                        // objet Concept accessible dans l'Espace :
                        // le message à envoyer en réponse, donnée locale
LOCAL mess : SendMess_AC
                        // Perspective utilisée sur le Concept
    WITH PERSPECTIVE SendMess1_P IS       // définition locale
        _kind : PROTOTYPE;                // prop. de nature "prototype"
        _input_validation_mode : IMPLICIT // prop. de mode de validation

                        // attributs retenus
        ATTRIBUTE subject
            _init_value : init_mess.subject   // valeur initiale
        ATTRIBUTE text

                        // opérations retenues
        OPERATION Post
            IN whom
                                          // valeur initiale
                _init_value : init_mess.sender.name
                                          // pré-actions de l'opération
            PRE_ACTION NEW_INSTANCE SCAN_INPUT END
                                          // post-actions
            ON SUCCESS DO RESET_PROTO END

    END SendMess1_P             // fin de la définition de la Perspective

END Reply_WS.
```

Dynamique d'utilisation

L'Espace Reply_WS est destiné à d'éventuelles instanciations multiples (propriétés *_instances*), à partir des opérations d'accès définies sur l'Espace de traitement des messages et sur l'Espace de lecture d'un message. Une instanciation ne sera cependant effectuée que s'il

n'existe pas déjà une instance de Reply_WS opérant sur le même message reçu (propriété *_unique_idem?*).

Reply_WS repose sur une donnée d'entrée, le message reçu, et une donnée locale décrite comme un prototype de message à envoyer (propriété *_kind* de la Perspective), sur laquelle est portée une Perspective permettant la modification des champs du message. Ce prototype permet la récupération de données qui seront utilisées lors de l'instanciation véritable de l'objet, en vue de l'initialisation de ce dernier. La validation des données entrées est implicite (propriété *_input_validation_mode*) ; l'instanciation et l'initialisation du message s'effectuent lors de l'activation de l'opération Post (valeur de *PRE_ACTION* : *NEW_INSTANCE* provoque l'instanciation, et *SCAN_INPUT* l'initialisation à partir des données entrées par l'utilisateur). En cas de succès de l'opération, la donnée prototype est réinitialisée (*ON SUCCESS DO...*).

5.3.6 L'Espace d'envoi d'un message

L'Espace d'envoi d'un message est appelé Write_WS. Il permet la définition et l'envoi d'un message quelconque.

Concepts

Les Concepts mis en jeu se réduisent à l'objet SendMess_AC représentant le message à envoyer. Ce Concept est conservé localement à l'Espace.

Spécification

La spécification de Write_WS est la suivante :

```
WORKSPACE Write_WS IS
    _name : "écriture d'un message"
    _instances : MULTIPLE;                    // prop. d'instanciation multiple

                        // objet Concept accessible dans l'Espace :
                        // le message à envoyer, donnée locale
    LOCAL mess : SendMess_AC
                    // Perspective utilisée, définie localement
        WITH PERSPECTIVE SendMess2_P IS
            _kind : PROTOTYPE;          // prop. de nature "prototype"
            _input_validation_mode : IMPLICIT   // prop. de mode de validation

                        // attributs retenus :
            ATTRIBUTE subject
            ATTRIBUTE text

                        // opérations retenues :
            OPERATION Post
                                        // pré-actions de l'opération
                PRE_ACTION NEW_INSTANCE SCAN_INPUT END
                                        // post-actions de l'opération
                ON SUCCESS DO RESET_PROTO END

        END SendMess2_P         // fin de la définition de la Perspective

END Write_WS.
```

Dynamique d'utilisation

L'Espace Write_WS est destiné à d'éventuelles instanciations multiples (propriétés *_instances*), à partir de l'opération d'accès définie sur l'Espace de traitement des messages.

Write_WS met en jeu une donnée locale décrite comme un prototype de message à envoyer (propriété *_kind* de la Perspective utilisée), sur lequel est portée une Perspective permettant la modification des champs du message. Ce prototype permet la récupération de données qui seront utilisées lors de l'instanciation véritable de l'objet, en vue de l'initialisation de ce dernier. La validation des données entrées est implicite (propriété *_input_validation_mode*) ; l'instanciation du message s'effectue lors de l'activation de l'opération Post (valeur de *PRE_ACTION*). En cas de succès de l'opération, la donnée prototype est réinitialisée (*ON SUCCESS DO...*) à une valeur vide de façon à permettre la composition et l'envoi d'un nouveau message.

5.3.7 Récapitulatif

La Figure IV-13 représente les différents Espaces de Travail définis pour le système Messagerie, en explicitant la dynamique d'utilisation mise en jeu.

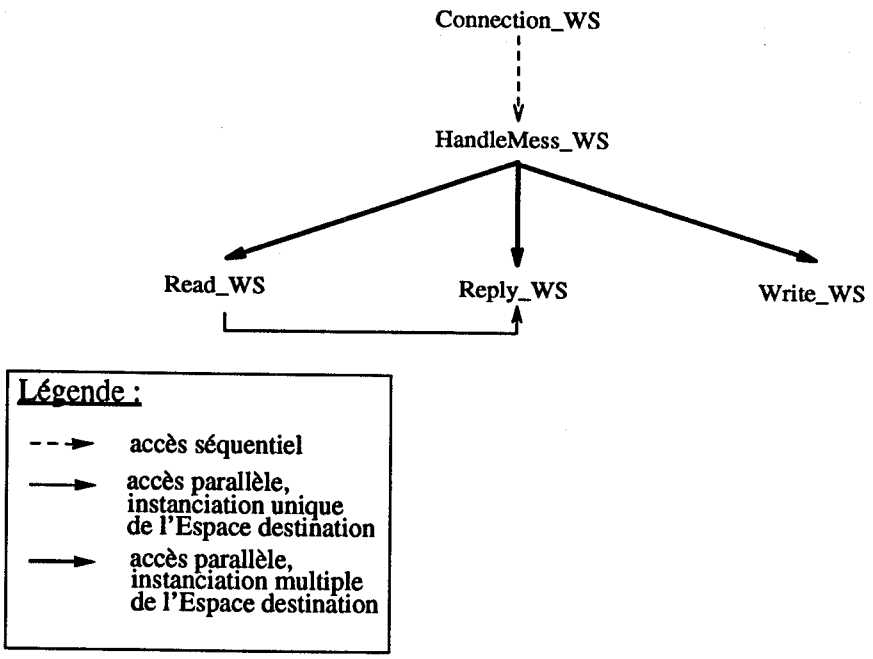


Figure IV-13 *Espaces de Travail et dynamique d'utilisation.*

5.4 Spécification de la Session du système Messagerie

Le contenu de l'objet Session du système a été élaboré au fur et à mesure de la définition des Espaces de Travail.

L'objet Session du système Messagerie offre un contexte global de données constitué d'une variable stockant l'objet UserList_AC représentant la base des utilisateurs, et d'une variable stockant l'objet User_AC représentant l'utilisateur du système.

L'état initial d'une session de l'application Messagerie offre un unique Espace de Travail, instance de Connection_WS.

La spécification de la session est la suivante :

```
SESSION session IS
    _name : "messagerie electronique"

                    // variables globales de la session
    GLOBALS
        bd : UserList_AC
        user : User_AC

                    // Espaces initiaux
    WORKSPACES
        ws1 : Login_WS

END Messagerie.
```

5.5 Conclusion

Au terme de ce travail de spécification d'un système interactif, l'on dispose d'une représentation des concepts du système, et de la structure organisant l'utilisation de ces concepts relativement aux tâches de l'utilisateur. L'activité de conception doit alors se poursuivre et aborder les aspects externes de l'interface.

Dans un premier temps, un style d'interaction devra être choisi. C'est ensuite à partir des composants d'utilisation que s'effectue le travail de conception menant à la spécification externe : il importe de définir le mode de présentation de chaque Espace de Travail, et de chaque Perspective sur un Concept, ainsi que le mode d'activation de chaque opération du système.

Chapitre V

Passerelle vers la réalisation

1 Introduction

A ce stade de nos travaux, nous avons défini un modèle de représentation conceptuelle d'un système interactif : SIROCO permet la modélisation des entités sémantiques du système, et de la structure organisant l'utilisation de ces entités relativement aux tâches de l'utilisateur. SIROCO réalise la première partie de nos objectifs, un outil de support à la conception.

Ce chapitre rapporte les résultats du deuxième volet de nos recherches, tournées cette fois vers la réalisation des systèmes interactifs. SIROCO constitue le point de départ de ces recherches. Nous montrons comment, dans le prolongement de SIROCO, un outil d'aide à la réalisation a pu être élaboré.

Plus qu'un outil, c'est une passerelle entre le monde de la conception et le monde de la réalisation que nous avons cherché à construire. Cette passerelle comporte deux parties. A la base de cette passerelle, l'utilisation de SIROCO comme argument d'un modèle de réalisation d'un système interactif. En consolidation de cette passerelle, le développement d'un outil de génération automatique d' interface.

L'organisation du chapitre reflète le caractère double de nos travaux : dans un premier temps, nous posons les fondations de la passerelle en décrivant un modèle de réalisation reposant sur SIROCO ; nous définissons ensuite les principales caractéristiques de notre outil de génération automatique d'interface.

2 Un modèle d'architecture "argumenté"

Lors de notre présentation des méthodes de réalisation des systèmes interactifs, au Chapitre II, nous déplorions l'imprécision des modèles d'architecture proposés, et nous explicitions la nécessité "d'aller plus loin". L'application d'un modèle d'architecture est bien souvent malaisée faute d'indications pratiques quant à la façon d'utiliser les principes théoriques de ce modèle. Les limitations des modèles existants proviennent, à notre sens, de l'absence de cadre de référence pour l'expression des besoins du développeur : sans un moyen de référencer précisément les caractéristiques du système à réaliser, seul un modèle d'architecture général et théorique peut être défini.

Nous proposons d'utiliser SIROCO comme ce cadre de référence qui fait défaut aux modèles d'architecture en général. A partir des éléments de représentation offerts dans le modèle SIROCO, nous définissons un modèle d'architecture qui, tout en étant générique, se veut ancré dans l'expression des besoins d'un système.

Nous décrivons dans un premier temps l'approche générale adoptée, puis nous passons en revue chacun des composants de notre modèle d'architecture.

2.1 Approche générale

Notre démarche se situe dans le prolongement de nos travaux sur les modèles d'architecture et les squelettes d'application évoqués respectivement aux chapitres II et III. Le modèle d'architecture décrit en section 3.3 du Chapitre II est repris et étendu en fonction des données apportées par SIROCO.

Le modèle de départ

Rappelons les principes élémentaires du modèle conçu lors de la première étape de nos travaux. Ce modèle repose sur le partitionnement de l'interface en trois grands composants : l'adaptateur du composant fonctionnel, le contrôleur du dialogue, et le gestionnaire de la présentation et de l'interaction. Ce modèle est un modèle multiagent doté des propriétés suivantes :

1. Chaque grand composant de l'interface est découpé en un ensemble d'objets : objets montrables pour l'adaptateur fonctionnel, objets de contrôle pour le contrôleur, objets vue et objets d'interaction pour le composant de présentation/ interaction.
2. L'ensemble des objets d'un système est partitionné en unités logiques représentant les agents du système. Un agent regroupe un objet de contrôle, un ou plusieurs objets vue, l'ensemble des objets interactifs associés aux objets vue, et utilise un ou plusieurs objets montrables.
3. Les agents sont liés par des relations hiérarchiques.

L'extension de ce modèle général passe par un travail d'analyse dont le but est l'identification de rôles d'objets et de rôles d'agents participant à la réalisation d'un système. Ces rôles sont définis relativement au contenu du système à réaliser, tel que le modèle SIROCO permet de l'exprimer.

Identification de rôles

C'est une analyse conjointe des éléments du modèle SIROCO et des composants de notre modèle d'architecture initial que nous sommes amenés à conduire. Nous cherchons d'une part à déterminer l'implication des Concepts, Espaces de Travail et Perspectives, en termes d'objets de gestion de l'interface ; d'autre part, il nous faut évaluer la part des éléments d'un système laissés implicites ou non représentés dans SIROCO - principalement les aspects externes, et déterminer l'influence de ces éléments sur le modèle d'architecture.

Notre analyse nous conduit à identifier dix types d'objets, organisés en cinq grands rôles d'agents. Les sections suivantes explicitent de façon informelle chacun des éléments de notre modèle d'architecture étendu : objets de l'adaptateur fonctionnel, objets de contrôle, et objets de présentation et d'interaction.

2.2 Objets de l'adaptateur du composant fonctionnel

Les Concepts de l'Application du modèle SIROCO constituent naturellement le contenu de l'adaptateur du composant fonctionnel. En effet, Concepts et objets de l'adaptateur dénotent une signification analogue exprimée pour les premiers en termes conceptuels et pour les seconds en termes de réalisation.

Les Concepts de l'Application représentent le contenu sémantique du système : les entités issues du domaine de l'application ou bien de la métaphore d'interaction choisie, que le concepteur définit comme éléments du modèle conceptuel du système. Ce modèle conceptuel définit les termes dans lesquels la réalisation des tâches de l'utilisateur va s'effectuer, au travers de l'interface du système dont le rôle est notamment d'assurer la communication de ce modèle à l'utilisateur. L'on rejoint ici la fonction des objets de l'adaptateur : définir les entités présentées dans l'interface, dénotant le point de vue de l'interface sur le composant fonctionnel.

D'où la règle de réalisation suivante :

Règle 1 : *Les objets de l'adaptateur du composant fonctionnel sont la réalisation des Concepts de l'Application définis pour le système.*

2.3 Objets de contrôle

Il s'agit de définir les différents types d'objets de contrôle nécessaires à la réalisation d'un système, ainsi que les relations de hiérarchie existant entre ces objets.

SIROCO nous invite à considérer dans un premier temps deux niveaux de contrôle général d'une application interactive, reflétant la structuration de la dimension d'utilisation du système : le niveau global à la session, et le niveau inférieur des Espaces de Travail. Afin d'affiner le contrôle d'un Espace de Travail, nous proposons trois niveaux de contrôle supplémentaires assurant la gestion des données et des opérations offertes dans un Espace.

La Figure V-1 résume les différents rôles identifiés en représentant les objets de contrôle ainsi que les relations hiérarchiques existant entre ces objets.

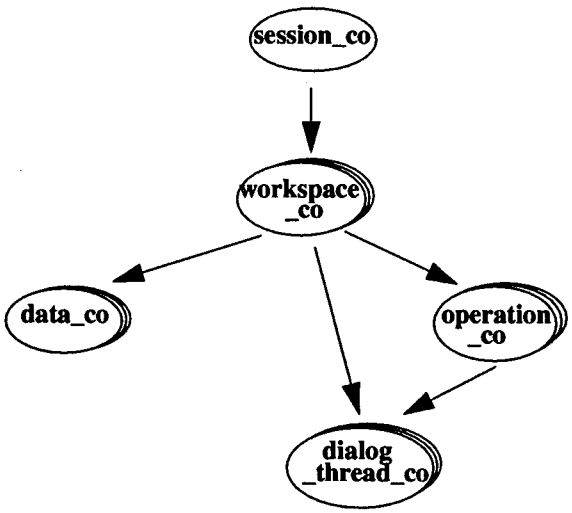


Figure V-1 *Les objets de contrôle et leurs relations de hiérarchie.*

2.3.1 Contrôle des Espaces de Travail

La notion d'Espace de Travail recèle celle de *contexte*. Contexte d'activité pour l'utilisateur, mais aussi contexte d'interaction et de dialogue pour le système. En effet, ce qui pour l'utilisateur apparaît comme une unité contextuelle d'interaction avec le système doit également être gérée comme telle au niveau de l'interface. La notion d'Espace de Travail se prolonge en une unité d'organisation de la gestion du dialogue, que l'on représente dans notre modèle d'architecture par un objet de contrôle spécialisé nommé *workspace_co*[1].

D'où la règle :

Règle 2 : *La gestion d'un Espace de Travail donne lieu à la définition d'un objet de contrôle spécialisé, appelé "workspace_co".*

L'objet workspace_co a pour rôle de gérer le contexte sous-jacent à un Espace de Travail : stockage des données de l'Espace, maintien de la cohérence entre ces données et leur présentation, gestion de l'état de l'interaction, de la présentation des opérations, mise en œuvre des traitements associés aux opérations, etc.

La diversité des fonctions intervenant dans la gestion d'un Espace de Travail conduit à une décomposition modulaire de ces fonctions. On définit des objets de contrôle supplémentaires chargés de mettre en œuvre des modules de fonctions, délégués par l'objet workspace_co, qui prend alors un rôle de superviseur de ces nouveaux objets. La décomposition que nous proposons isole le contrôle des données (objets data_co) de celui des opérations (objets operation_co) ; elle est explicitée dans les sections 2.3.3 et 2.3.4 à venir.

1. Le suffixe "_co" signifie "Control Object".

2.3.2 Contrôle de la session

Si les objets workspace_co assurent le contrôle des Espaces de Travail du système, il est nécessaire de disposer d'un mécanisme de contrôle global de la session, superviseur des objets workspace_co. Nous définissons un objet de contrôle nommé *session_co*, dont le rôle est d'assurer ce contrôle global.

Il s'agit principalement de gérer la dynamique de l'instanciation, de l'initialisation et de la terminaison des Espaces de Travail représentés par les objets workspace_co, mais également de mettre en œuvre le contexte global du système ; ce contexte global contient les données définies globalement au système.

Règle 3 : *L'initialisation et le contrôle général d'une session est assuré par un objet de contrôle appelé session_co, situé au sommet de la hiérarchie des objets de contrôle.*

2.3.3 Contrôle des données d'un Espace de Travail

On propose de conserver la structure des attributs d'un Espace au niveau de la gestion de ces attributs : l'on définit un objet de contrôle par attribut de donnée. Cet objet spécialisé, de type *data_co*, est supervisé par l'objet de contrôle de l'Espace workspace_co. Son rôle est de mettre en œuvre la gestion des données d'un attribut de l'Espace ; cette gestion comporte deux grands aspects :

- la mise en œuvre de la cohérence entre les données en mémoire et l'image de présentation de ces données,
- le recueil des modifications effectuées par l'utilisateur sur la présentation, leur interprétation et leur propagation au niveau fonctionnel.

Règle 4 : *La gestion d'un Concept présent dans un Espace de Travail est assurée par un objet data_co.*

En dehors de la phase d'initialisation, il n'y a a priori pas d'échanges entre objet workspace_co et objet data_co, l'objet data_co étant autonome. L'objet data_co est susceptible d'entrer en contact avec l'objet session_co, et avec un objet de gestion des opérations ; dans le premier cas, il s'agit d'intervenir sur le contexte global du système ; dans le second cas, il s'agit de lancer un traitement fonctionnel en réponse à une action de l'utilisateur.

2.3.4 Contrôle des opérations d'un Espace de Travail

La gestion des opérations d'un Espace de Travail implique la gestion de leur état (activable ou non), de leur présentation éventuelle, et la mise en œuvre de leur activation. On peut distinguer les opérations dotées d'une présentation (typiquement, un élément de menu), et celles qui ne sont pas représentées à l'image (cas des opérations activables uniquement à partir du clavier, ou bien des opérations de manipulation directe). Les opérations peuvent être réparties dans deux groupes selon le niveau de complexité mis en jeu par leur activation : les opérations d'activation simple, et les opérations d'activation complexe, dont le lancement nécessite un dialogue entre l'utilisateur et le système, qu'il s'agisse de choisir des valeurs de paramètres ou bien de gérer des situations d'erreur, ou de risque.

Dans le cas des opérations complexes, c'est bien souvent un véritable sous-dialogue qui doit être mis en œuvre, éventuellement asynchrone, c'est-à-dire parallèle au dialogue principal

localisé dans l'Espace de Travail. Nous proposons de déléguer la gestion de ce sous-dialogue à un objet de contrôle particulier, nommé *dialog-thread_co*.

L'objet dialog-thread_co a pour fonction, comme son nom l'indique, de gérer un fil du dialogue entre l'utilisateur et le système. Ce fil de dialogue correspond au dialogue nécessaire à la mise en œuvre d'un appel à une opération : désignation et/ou saisie des paramètres de l'opération, prise en compte des cas d'erreur, et demande de confirmation dans le cas d'une opération "sensible", etc., et enfin lancement effectif des traitements de l'opération.

Règle 5 : *La gestion de chaque opération complexe est effectuée par un objet de contrôle spécialisé indépendant appelé dialog-thread_co.*

La gestion des opérations simples peut être conduite de façon globale par un unique objet de contrôle, que l'on appelle *operation_co*. Cet objet a pour rôle de gérer d'une part la présentation des opérations de l'Espace, au travers d'un ensemble d'objets de gestion de l'image, et d'autre part de mettre en œuvre l'activation d'une opération par l'utilisateur (saisie des paramètres et transmission au composant fonctionnel).

Règle 6 : *La gestion de l'ensemble des* opérations *simples d'un Espace de Travail est mise en œuvre par l'objet de contrôle operation_co.*

2.4 Objets de présentation/interaction

Rappelons qu'un objet vue, défini dans notre modèle d'architecture initial, a pour rôle de mettre en œuvre les aspects graphiques et interactifs de parties précises de l'interface. Un objet vue est réalisé par la combinaison d'un ensemble d'objets interactifs issus la plupart du temps d'une boîte-à-outils. Nous cherchons ici à définir un mode de décomposition de l'interface en parcelles dont la gestion est confiée à un objet vue. Les notions d'Espace et de Perspectives offrent de premiers éléments de structuration, sur lesquels viennent se greffer un ensemble d'objets vue spécialisés.

2.4.1 Présentation d'un Espace de Travail

La notion d'Espace de Travail dénote une propriété de localité de présentation. Ce lieu virtuel d'activité doit apparaître comme tel aux yeux de l'utilisateur ; il se traduit concrètement à l'écran par des éléments d'interface que caractérise une unité de présentation. La notion d'Espace de Travail se prolonge en une unité d'organisation de la présentation externe de l'interface du système.

La réalisation de cette unité de présentation est dépendante des choix de conception ; elle pourra mettre en jeu une sous-fenêtre, une fenêtre, un groupe de fenêtres, un écran, etc. Dans tous les cas, l'on propose un objet vue dédié à la mise en œuvre de cette unité, que l'on appelle *workspace_view.*

Règle 7 : *La mise en œuvre de l'unité de présentation d'un Espace de Travail est à la charge d'un objet workspace_view.*

L'objet workspace_view a pour rôle de gérer les éléments de présentation assurant l'unité visuelle et interactive de l'Espace de Travail. La mise en œuvre du contenu de l'Espace de Travail passe par la définition d'objets vue supplémentaires, dont la notion de Perspective constitue une dimension de structuration privilégiée. L'organisation spatiale de ces vues passe par l'objet workspace_view.

2.4.2 Présentation d'une Perspective

En ce qui concerne la présentation d'une Perspective, il est moins aisé de définir des propriétés caractéristiques car l'on touche à ce qui fait la spécificité d'un système.

Tout d'abord, la présentation des données sera bien souvent séparée de celle des opérations. En effet, les guides de style ont tendance à regrouper les opérations dans des menus ou barres de menus indépendants du reste de l'interface. On aura ainsi des objets vue dédiés à la mise en œuvre de ces opérations : objet de présentation d'une barre de menus (noté *menubar_view*), objets de présentation des formulaires de paramètres (notés *param_view*), etc.

La présentation des données d'une Perspective est bien entendu très dépendante du système. On peut considérer que dans la plupart des cas la propriété de localité de la présentation sera respectée : une Perspective dénote une vue fonctionnelle sur un Concept dont l'individualité est clairement définie. Les données d'une Perspective seront présentées dans une parcelle bien cernée de l'interface, sans influence des autres Perspectives de l'Espace de Travail[1]. Cette situation peut alors être gérée par un objet vue spécifique à la Perspective, que l'on appellera *perspective_view.*

Règle 8 : *La gestion de la présentation/interaction des données d'une Perspective est assurée par un objet perspective_view.*

Dans le cas d'une Perspective comportant une donnée de type complexe, les Perspectives imbriquées seront gérées par des objets perspective_view distincts, l'un supervisant l'autre.

Il est possible que, malgré le caractère unitaire de la Perspective, l'on souhaite mettre en œuvre la présentation imbriquée de deux Perspectives. Dans ce cas, l'on définira un objet vue particulier, de type *combined_persp_view*, pour mettre en œuvre cette combinaison.

Règle 9 : *La gestion de la présentation et de l'interaction combinées des données de plusieurs Perspectives est assurée par un objet combined_persp_view.*

2.5 Récapitulatif : les agents

L'ensemble des objets ci-dessus identifiés constituent la base à partir de laquelle l'on peut définir les agents du système. Conformément au modèle, ce sont les objets de contrôle qui déterminent les agents du système. On définit ainsi cinq grands types d'agents, représentés dans la Figure V-2 : agent de gestion de la session, agent de gestion d'un Espace de travail, agent de gestion d'un attribut d'un Espace, agent de gestion des opérations simples, agent de gestion d'une opération complexe d'un Espace.

1. Nous faisons ici abstraction des dépendances géométriques pouvant exister entre la présentation de deux Perspectives, ou bien entre la présentation d'une Perspective et celle de l'Espace de Travail. Ce type de dépendances est dans tous les cas géré au niveau des objets Vue, et plus précisément au travers des services des boîtes-à-outils sous-jacentes.

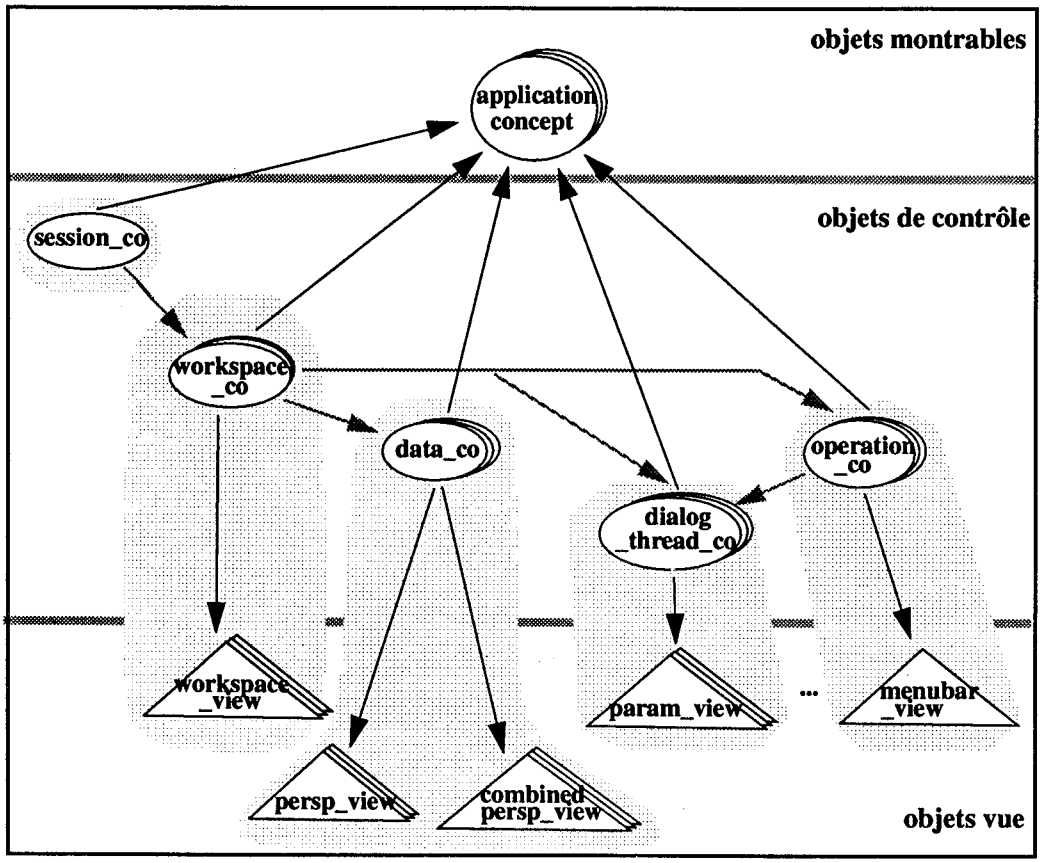


Figure V-2 *Les agents du modèle d'architecture étendu.*
(Les zones grisées délimitent les agents du modèle ; les flèches grises indiquent les relations hiérarchiques entre les agents).

L'agent de gestion de la session ne possède pas de présentation. Son rôle est de maintenir le contexte global du système (référence aux objets montrables représentant les Concepts globaux), ainsi que d'assurer le contrôle global des agents de gestion des Espaces.

L'agent de gestion d'un Espace est doté d'une présentation qui lui est propre, délimitant l'Espace à l'intérieur de l'ensemble de l'interface du système. L'agent utilise les objets montrables des Concepts attributs de l'Espace de Travail. La mise en œuvre de la présentation et de l'interaction à l'intérieur de l'Espace est déléguée aux agents de gestion des données et des opérations, que l'agent de l'Espace supervise. De façon générale, on aura un agent de gestion par attribut de données ; en ce qui concerne les opérations, un ou plusieurs agents de gestion des opérations simples pourront coexister ; on aura de plus un agent de gestion par opération complexe de l'Espace.

L'agent de gestion d'un attribut de donnée de l'Espace a pour rôle de mettre en œuvre la présentation de cet attribut, ainsi que de gérer les interactions survenant sur cette présentation. Ces fonctions sont réalisées par l'intermédiaire d'un ou plusieurs objets perspective_view ou combined_persp_view.

L'agent de gestion des opérations simples a pour rôle de gérer la présentation et l'activation d'un groupe d'opérations simples de l'Espace (par opposition aux opérations

"complexes", dont l'activation nécessite un sous-dialogue). Le regroupement des opérations simples est dépendant du système, fonction notamment du mode de présentation de ces opérations. L'agent est doté d'objets Vue permettant la présentation des opérations ; par exemple, un objet de présentation de barre de menu, etc.

L'agent de gestion d'une opération complexe a pour rôle de mettre en œuvre le sous-dialogue nécessaire à l'activation de l'opération. L'agent est doté d'objets Vue support de ce dialogue, par exemple une Vue permettant la saisie des paramètres de l'opération. La présentation de l'opération même peut être prise en charge par l'agent, ou bien peut être indissociée de celle des opérations simples et confiée alors à l'agent de gestion des opérations simples correspondant.

3 Génération automatique de code

Au-delà de l'argument d'un modèle d'architecture étendu, SIROCO fournit la matière d'un outil de génération automatique du code de réalisation d'un système. L'objectif est double : d'une part un outil de prototypage rapide, et d'autre part une fabrique d'objets réutilisables.

On souhaite disposer d'un outil de prototypage rapide fournissant un prototype d'interface permettant de tester l'ensemble des fonctions et propriétés spécifiées pour un système interactif. Ce prototype suppose la génération d'une image de présentation de l'interface, et la mise en œuvre d'un noyau de gestion de la dynamique de cette interface (cohérence de l'image, mise en œuvre de l'interaction, etc.). Les données de spécification exploitées par cet outil de prototypage sont en premier lieu la description du système interactif en langage SIROCO ; intervient également la spécification des choix du concepteur quant au style d'interaction et aux détails de présentation de l'interface, complément des choix conceptuels exprimés en langage SIROCO.

Autant que possible, le générateur ne doit pas simplement donner naissance à des prototypes d'interface, mais doit produire un noyau de composants logiciels réutilisable en dehors de la phase de prototypage, lors de la mise en œuvre effective de l'application. En ce sens, c'est une fabrique d'objets réutilisables que nous souhaitons réaliser.

Cette section décrit la démarche adoptée dans la conception de cet outil de génération de code, et présente les principales caractéristiques de la maquette de générateur réalisée pour le langage Guide[1], sous le nom de SIROCO-Guide.

Cette section traite dans un premier temps des aspects externes de l'interface, puis rend compte de la génération des modules de code en décrivant les principes d'une fabrique d'objets. La section s'achève sur une brève description des services réalisés par le code généré.

3.1 Aspects externes de l'interface

Après quelques mots sur l'approche générale adoptée, cette section détaille les traitements relatifs à la présentation et au style d'interaction.

3.1.1 Approche générale

Rappelons les principes exprimés en section 5 du Chapitre III, en prélude à la présentation de SIROCO. Notre démarche repose sur une décomposition des choix de conception d'un système interactif en trois ensembles : les choix conceptuels, les choix relatifs au style d'interaction mis en œuvre, et les choix concernant les détails de présentation de l'interface. Le langage SIROCO dénote notre intérêt privilégié pour l'expression des choix conceptuels. Nous décrivons ici le mode de spécification des choix de style d'interaction et de présentation, non couverts par SIROCO.

1. Notons que les principes de cette maquette sont indépendants du langage utilisé lors de la génération de code. L'utilisation de Guide comme langage d'application est dictée par notre environnement de travail ; tout langage de programmation à objets, est un langage d'application potentiel des outils SIROCO ; un langage de programmation traditionnel est moins indiqué puisque le modèle sous-jacent à SIROCO est un modèle à objets.

C'est une approche simplificatrice que nous proposons, fondée sur la déduction automatique de ces aspects externes à partir de la spécification d'un système interactif en langage SIROCO. L'opération de déduction est menée en appliquant un ensemble de règles fixes tempérées par une liste de paramètres génériques dont le choix constitue la marge de décision du concepteur.

La suite de cette section précise les éléments de cette approche en introduisant la notion de guide de style, et en définissant la marge de variabilité associée à ce guide de style.

Un guide de style pour les interfaces SIROCO

La déduction des aspects externes d'une interface à partir d'une spécification SIROCO suppose l'application de règles permettant de franchir la distance entre concepts abstraits et image concrète.

Le travail d'analyse et de définition du modèle d'architecture étendu, rapporté dans la section précédente, fournit de premiers résultats quant à cette distance de l'abstrait au concret. La compréhension des besoins sur le plan de la gestion de l'interface nous a en effet conduits à analyser les implications concrètes des composants de SIROCO sur le plan de la présentation. Les résultats de cette analyse nous montrent que, au-delà de certaines propriétés telle par exemple la localité de présentation pour les Espaces de Travail, la marge de variabilité dans les choix de concrétisation est appréciable.

Définir des règles de déduction de l'image revient à figer cette variabilité dans les limites d'un *guide de style*[1] régissant précisément les choix du style d'interaction mis en œuvre ainsi que les choix de présentation de l'image offerte par l'interface. Ainsi, c'est un guide de style que nous sommes amenés à définir pour les interfaces SIROCO.

De façon générale, les interfaces visées par les outils SIROCO sont des interfaces mêlant menus de commandes, boutons, formulaires, icônes, etc., et adeptes de la désignation directe. Le style général d'une interface SIROCO met en œuvre les recommandations du "style guide" Motif [OSF 89b]. Ces directions générales constituent le point de départ de notre étude. Notre guide de style intègre ces éléments, et les complète par un ensemble de choix de conception explicités sous la forme de règles.

La marge décisionnelle du concepteur

Notre objectif n'est pas de définir un unique style d'interface SIROCO, mais de définir plusieurs dimensions à l'intérieur desquelles se situe le style d'une interface SIROCO : nous souhaitons en effet mettre en œuvre un degré de souplesse dans la génération de l'interface

1. Le guide de style que nous définissons ici se place à un niveau d'abstraction et de détail très différent de celui des guides de style fournis avec certaines bibliothèques graphiques, tel le "style guide" de Motif, par exemple (cf. Chapitre III).

Ces guides définissent en général deux niveaux de règles : d'une part des règles élémentaires de navigation, de "feed-back", etc. intégrées aux composants des bibliothèques graphiques, et qui ne sont utiles que lors de la définition de composants étendant les bibliothèques ; d'autre part, des règles visant à guider le difficile choix des composants d'une interface parmi les éléments offerts dans les bibliothèques ; ces règles sont cependant très grossières, car elles se veulent très générales, et s'appuient sur des notions informelles et mal définies telles les "primary application actions", "ancillary application actions", etc. qui figurent dans le guide Motif.

d'une application, et accorder une marge décisionnelle au concepteur de l'interface, dans les limites des contraintes du guide de style.

Quelle portée attribuons-nous à cette marge décisionnelle ? On considère trois niveaux d'abstraction dans la partie externe de l'interface d'un système : le niveau des détails, le niveau des entités graphiques, et le niveau du style. Pour chacun de ces niveaux la marge décisionnelle prend une forme très différente.

Niveau des détails :

Les *détails de l'image ou de l'interaction* (par exemple, la taille ou la police de telle chaîne de caractères, la couleur ou l'emplacement de telle zone, etc.) sont des éléments que la plupart des environnements de réalisation - notamment les bibliothèques X et Macintosh - gèrent de façon indépendante du code de l'interface. Le mécanisme utilisé est celui des fichiers de "ressources" à l'intérieur desquels le développeur spécifie les valeurs à affecter aux paramètres régissant ces éléments de détail ; ces paramètres sont dynamiquement pris en compte lors de l'exécution du système, sans nécessiter de recompilation. Ceci nous permet de ramener la marge décisionnelle sur ces détails à la manipulation de fichiers de "ressources" portant sur le code généré. C'est après la génération et non avant que les choix relatifs aux détails de l'interface seront exprimés, lorsque les choix automatiquement effectués apparaîtront comme non satisfaisants.

Niveau des entités graphiques :

Le *niveau des entités graphiques* concerne le choix des techniques d'interaction et afficheurs graphiques dont le combinaison constitue l'interface externe d'un système. Dans un souci de simplification du processus de prototypage, nous ne souhaitons pas mettre en œuvre une marge décisionnelle explicite à ce niveau. Le choix des entités graphiques sera effectué de façon automatique, par l'application des règles de déduction définies dans le guide de style.

Si l'on ne fournit pas d'outil de spécification des choix des entités graphiques, le développeur conserve cependant un moyen de modifier les choix automatiques effectués en adaptant le code généré. Cette adaptation est grandement facilitée par l'adoption d'un modèle de structuration à objets, comme nous le montrerons par la suite.

Niveau du style :

Le *niveau du style* porte sur les choix généraux qui sous-tendent la construction d'une interface. Ces choix affectent de façon générique à la fois la présentation et l'interaction proposées dans l'interface : le mode d'organisation de la présentation des commandes, le mode d'activation d'une commande, etc. Ces choix ont une portée qui dépasse les couches de présentation/interaction, atteignant l'organisation de la dynamique du dialogue. En ce sens, ils sont indissociables de l'interface d'un système, au cœur de sa réalisation.

La marge décisionnelle s'exprime à ce niveau par le choix d'options déterminant le style de l'interface. Ces options sont définies dans le cadre d'un système de paramétrage des règles de déduction des aspects externes, et des règles de génération du code. Elles sont fournies au générateur parallèlement à la spécification SIROCO, et représentent la principale marge de décision du concepteur quant au dialogue et à l'image de l'interface.

L'utilisateur et les objets

Avant de poursuivre notre présentation du traitement des aspects externes de l'interface, nous souhaitons nous arrêter quelques instants sur la question des relations entre le modèle à objets et l'interface. *Dans quelle mesure le modèle à objets sous-jacent à notre modèle de spécification est-il "naturel", et doit-il être sensible au niveau de l'interface utilisateur ?* Il s'agit là d'une question de fond qui apparaît très tôt dans nos travaux de conception du générateur, et se trouve à la source d'une partie des paramètres de style ci-dessus évoqués.

La notion d'objet même n'est pas en cause : l'environnement naturel de l'utilisateur est constitué d'objets, entités dont l'identité, les propriétés et les frontières sont plus ou moins clairement définies, et sur lesquelles l'utilisateur possède une liberté d'action variable ; une table, une pomme sont perçues comme des objets, de même qu'une adresse, l'hiver, ou un voyage... Les premiers sont des objets concrets, les seconds des objets abstraits.

La question ne concerne a priori pas la modélisation des données. En effet, le module des données contenues dans un objet est la plupart du temps indissociable de l'identité de cet objet. Les relations sémantiques et structurelles qui régissent le regroupement de ces données correspondent à une réalité abstraite ou concrète du domaine de l'application. Elles doivent être perceptibles dans l'interface de l'application.

La question se pose en ce qui concerne le lien entre une opération et un objet. Le rattachement d'une opération à un objet soulève une double interrogation : la relativité du choix de ce rattachement d'une part, et d'autre part le manque de naturel de la pensée objet-action.

Si le *rattachement d'une opération à un objet* est bien souvent non ambiguë, il est des cas où le choix de l'objet de rattachement est subjectif. De façon générale, le lien entre une opération et son objet est un lien d'association et non pas d'appartenance.

> Un exemple simple, dans le contexte du système de messagerie, est celui de l'opération de suppression d'un message d'une boîte-à-lettre : si l'on considère que le message est ôté de la boîte à lettres, l'opération de suppression est à définir sur l'objet boîte-à-lettres, avec un paramètre de type message ; si au contraire l'on privilégie le message, considérant qu'il s'agit de détruire un message, c'est sur l'objet message que l'opération de suppression sera définie.

La modélisation à objets implique un *mode d'expression objet-action* plutôt que action-objet : l'utilisateur est conduit, au cours de la réalisation de ses tâches, à choisir d'abord un objet puis une action. Ce mode d'expression s'oppose au langage naturel, la tendance étant plutôt de choisir d'abord une action puis l'objet sur lequel on applique l'action.

> Par exemple, comparer "je veux détruire ce message", et "message.détruire".

En dépit de ce manque apparent de naturel[1], force est de constater que le mode de pensée objet-action est celui qui prévaut dans des interfaces dont la convivialité n'est pas en défaut, interfaces fondées sur la désignation directe : l'utilisateur est amené à désigner dans un premier temps l'objet, puis le traitement qu'il souhaite appliquer à l'objet.

---

1. Il est difficile de juger de ce qui est "naturel" ou non : c'est le mode de pensée des individus qui est en jeu ici, et il n'est pas certain que ce mode de pensée soit forcément celui que reflète le langage naturel...

   Nos propos sont issus de notre expérience d'enseignement pratique de logiciels de bureautique à un public non informaticien. Le mode d'expression objet-action s'avère pour certains individus bien difficile à pratiquer...

L'on retiendra de ces interrogations que la perception du modèle à objets - plus précisément, le modèle des opérations - dans l'interface est sujette à caution, et sollicite un choix explicite du concepteur.

3.1.2 Construction de la présentation de l'interface

Nous nous intéressons ici à l'image de l'interface : le choix d'éléments graphiques, et leur assemblage en vue de représenter l'interface utilisateur du système.

Nous décrivons dans un premier temps les principales règles fixes de notre guide de style, en justifiant les choix effectués. Afin de ne pas alourdir notre présentation, seuls les principes de ces règles sont présentés. Dans un second temps, nous passons en revue les paramètres et options représentant le degré de variabilité de ce guide de style.

En prélude aux développements annoncés, il est nécessaire de préciser les termes qui seront utilisés dans la description de l'image de l'interface. La Figure V-3 a pour objectif d'introduire ou de rappeler ces termes, pour lesquels nous nous contentons d'une définition "visuelle" : les termes de fenêtre, sous-fenêtre, formulaire, champ, barre de menus, zone de travail et zone de messages.

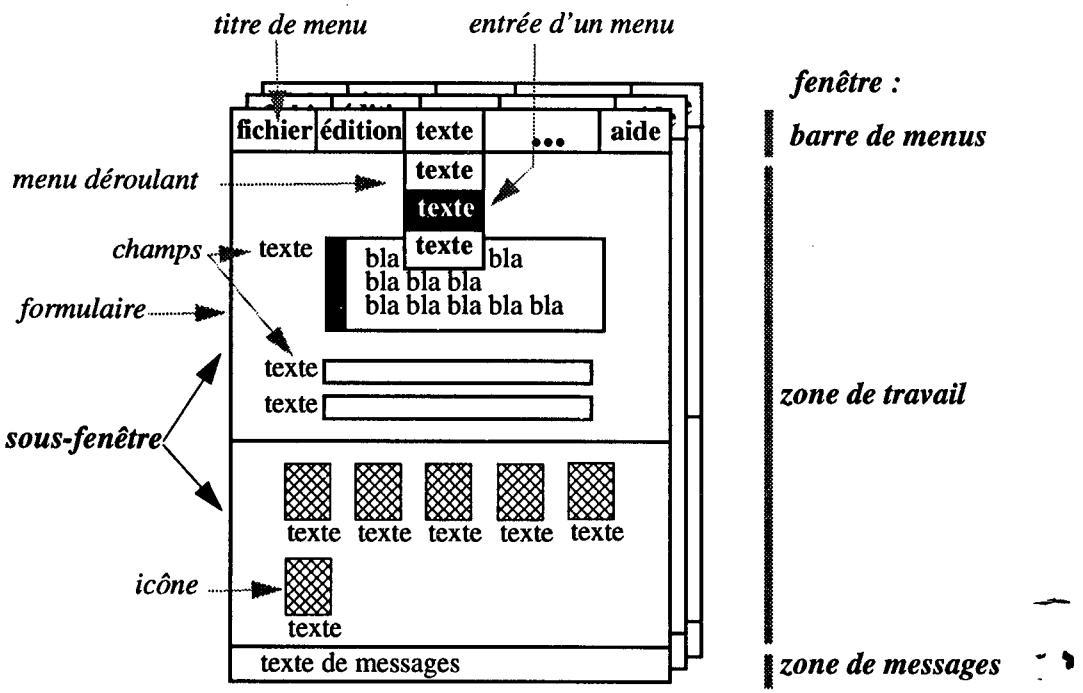


Figure V-3 *Les termes de description de la présentation.*

Règles générales

La Figure V-4 illustre les principes d'organisation définis par les règles générales de présentation. Elle reprend la fenêtre de l'interface proposée en Figure V-3, unité de structuration de la présentation, en indiquant grossièrement la correspondance entre les éléments de présentation et les composants du modèle SIROCO.

Présentation d'un Espace de Travail :

La propriété de localité de la présentation d'un Espace de Travail trouve son expression dans le choix de représentation d'un Espace par une *fenêtre* indépendante. La mise à la disposition de l'utilisateur d'un Espace de Travail passe par l'instanciation et l'affichage de la fenêtre de présentation correspondante ; la suspension de l'activité menée à l'intérieur de l'Espace se concrétise par l'inhibition de tout élément activable de la fenêtre, qui reste cependant affichée ; la terminaison de l'activité conduit à la suppression de la fenêtre.

L'image présentée à l'intérieur de la fenêtre représente le contenu de l'Espace de Travail. On distingue la présentation des commandes de celle des données de l'Espace.

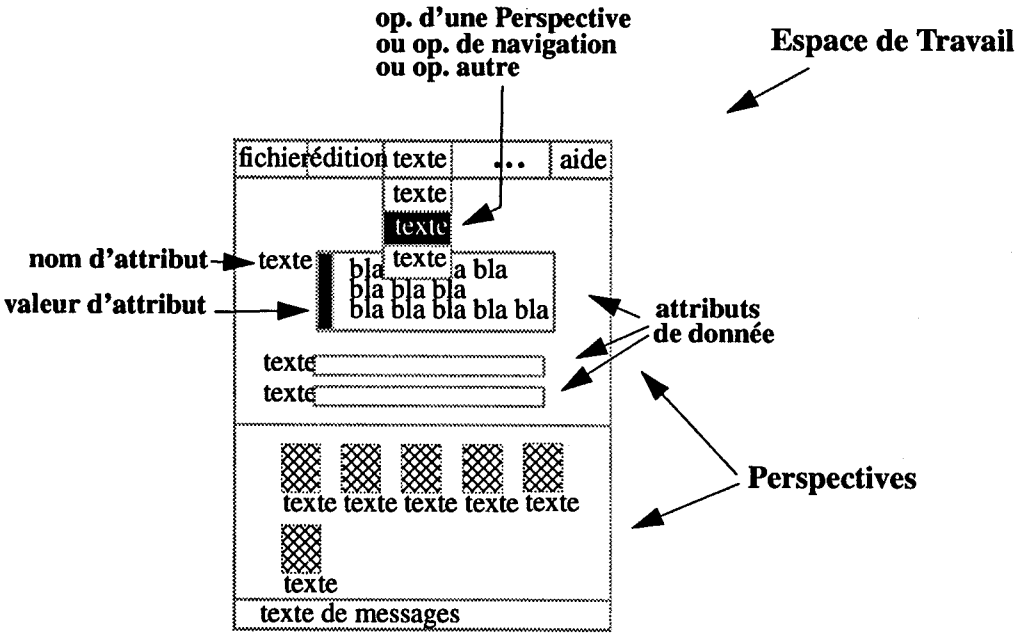


Figure V-4 *Principes de la mise en correspondance des composants SIROCO et des éléments de présentation.*

Présentation des commandes de l'Espace:

Les commandes de l'Espace comprennent quatre ensembles d'opérations :

1. les opérations qui apparaissent dans les Perspectives définies sur les Concepts de l'Espace,
2. les opérations de navigation propres à l'Espace même,
3. les opérations de validation explicite des modifications de données,
4. les opérations implicites à tout Espace, représentant des services d'interface (au sens du 2.4, Chapitre III), dont le générateur réalise de façon automatique la mise en œuvre : opération de terminaison, opérations d'édition (copier, couper, coller), et opérations de demande d'aide.
   (Nous reviendrons dans la suite de ce document sur le contenu de ces services et leur mode de réalisation).

Toute commande fait l'objet d'une présentation désignable par la souris. La présentation des ensembles 1, 2 et 4 de commandes de l'Espace s'effectue dans la barre de menus présentée au sommet de la fenêtre de l'Espace, conformément aux recommandations Motif. Les opérations de validation des modifications échappent à cette règle car fortement liées à la présentation des données sur lesquelles s'effectuent ces modifications ; elles sont déportées dans la zone de présentation de ces données.

L'organisation de la barre de menus, c'est-à-dire le choix des menus principaux et des sous-menus, fait l'objet de règles paramétriques que nous définissons un peu plus loin. Notons l'existence et le positionnement systématique de trois menus : le menu "fichier", qui comporte une entrée permettant de "quitter" la fenêtre, c'est-à-dire mettre fin à l'instance d'Espace représentée ; le menu d'édition, dont les entrées dénotent les commandes d'édition, et le menu d'aide, dont les entrées présentent les commandes de demande d'aide.

Présentation des données de l'Espace :

On considère toute Perspective comme une entité indivisible, détentrice d'une propriété de localité de présentation. Une Perspective comportant des données est présentée dans une *sous-fenêtre* de la zone de travail de la fenêtre de l'Espace - une Perspective ne comportant pas de données ne donne pas lieu à présentation. Les Perspectives intervenant dans un Espace de Travail sont présentées comme une colonne verticale de sous-fenêtres, dans l'ordre de leur spécification.

Les attributs de données d'une Perspective sont également présentés de façon verticale, dans l'ordre de leur spécification. Les éléments graphiques intervenant dans la présentation d'un attribut sont dépendants du type de cet attribut. Une donnée de type simple non construit apparaît comme un champ de formulaire : champ de texte, fenêtre de texte avec défilement, curseur numérique, etc. Une donnée de type construit est présentée au moyen d'un afficheur spécifique au constructeur de ce type : liste textuelle ou iconique, arbre, etc. Une donnée de type complexe donne lieu, de façon récursive, à la présentation des Perspectives qui lui sont associées.

Paramètres du style de présentation

C'est sur la présentation des commandes que porte le système de paramétrage. Il s'agit de définir le mode d'organisation de la barre de menus : à quoi correspond la structure des menus et sous-menus. Plusieurs types d'organisations sont définis, selon que la structuration à objets est conservée ou non, et selon le critère de regroupement choisi :

- organisation *"par nature"* : le générateur ne conserve pas la structuration en objets, mais axe l'organisation sur les valeurs de la propriété de nature des opérations apparaissant dans l'Espace de Travail.
  L'utilisateur dispose d'un menu par valeur de propriété, dont les entrées présentent les opérations comportant cette valeur.
- organisation *"par trait"* : le générateur ne conserve pas la structuration en objets, mais axe l'organisation sur les valeurs de la propriété de traits sémantiques des opérations.
- organisation *"par Concept"* : le générateur conserve la structuration en objets, et axe l'organisation sur la structuration des attributs de l'Espace.
  Chaque Concept muni de Perspectives comportant des opérations fait l'objet d'un menu dont les entrées présentent ces opérations ; la présence

d'attributs complexes dans ces Perspectives donne lieu à la définition de sous-menus.

- organisation *"par Perspective"* : le générateur conserve la structuration en objets, et axe l'organisation sur les Perspectives utilisées dans l'Espace de Travail.
  L'utilisateur dispose d'un menu par Perspective comportant des opérations ; la présence d'attributs complexes dans une Perspective peut donner lieu à la définition de sous-menus.

Afin de préserver l'homogénéité de la présentation, le choix d'un type d'organisation est général pour l'ensemble de l'interface du système. Nous reviendrons dans la section suivante sur la portée de ce choix, qui dépasse les simples aspects de présentation, et peut affecter les procédures d'activation des commandes.

3.1.3 Définition du style d'interaction de l'interface

La notion de style d'interaction fait référence à la façon dont l'utilisateur interagit avec l'application, c'est-à-dire aux modes d'activation des commandes et de saisie des données qui lui sont proposés.

En prélude à la présentation des règles et système de paramétrage relatifs au style d'interaction, nous précisons les différents aspects du processus d'activation d'une commande : les étapes en jeu et leurs modes de réalisation possibles.

Etapes de l'activation d'une commande

L'activation d'une commande suppose l'obtention auprès de l'utilisateur des informations suivantes :

- l'identification de la commande,
- l'identification de l'instance d'objet sur lequel s'applique la commande,
- la valeur de chacun des paramètres éventuels de la commande.

On appelle *désambiguïsation* d'une commande le processus d'identification de l'instance d'objet sur laquelle porte la commande. La désambiguïsation s'avère nécessaire lorsque l'identification de la commande ne détermine pas l'instance d'objet ; dans le cas contraire, elle peut être considérée comme superflue.

Mode d'activation d'une commande

Le style d'interaction visé est axé sur la désignation directe, voire la manipulation directe lorsque le cas s'y prête.

Désignation directe :

Le mode de désignation directe signifie que certains éléments de présentation peuvent participer au langage d'entrée de l'interface, à travers leur désignation par un clic souris. L'identification d'une commande par désignation directe traduit l'utilisation de systèmes de menus et de boutons. L'identification d'un objet par désignation directe nécessite de disposer d'éléments de présentation désignables - généralement une icône de représentation de l'objet.

Manipulation directe :

Le principe de manipulation directe, dont [Nanard 90] fournit une étude approfondie, permet la mise en œuvre d'un style d'interaction la plupart du temps séduisant, mais qui ne peut être appliqué à l'activation de toute commande du système interactif. Dans le cadre de nos travaux, nous nous intéressons aux aspects génériques de la manipulation directe. Nous distinguons trois grands types de manipulations directes réalisant l'activation de commandes :

- manipulations de type "modifier",
- manipulation de type "passer sur",
- manipulation de type "mettre sur".

Une **manipulation de type "modifier"** ne met (a priori) en jeu qu'un seul objet, le but de la manipulation étant de modifier un ou plusieurs attributs de l'objet.

La modification est effectuée par action sur la présentation même de l'objet : modification de la présentation entière : agrandissement, réduction, déformation, etc., ou bien modification d'un élément de la présentation : déplacement d'un curseur, etc.

La sémantique d'une telle manipulation peut être très variée : dans le cas d'un éditeur graphique, il peut s'agir de simples modifications d'attributs graphiques ; dans le cas où les objets graphiques sont le support de réalisation d'une métaphore d'interaction, la modification d'un objet peut avoir une signification très variable, dépendante de la métaphore, et sur laquelle il est difficile de faire des hypothèses.

Une **manipulation de type "passer sur"** met en jeu au moins deux objets. Ce type de manipulation est a priori dépendant des mouvements de la souris, et non uniquement des positions initiale et finale de la souris.

La sémantique d'une telle manipulation peut être très variée, dépendante de la métaphore en jeu. (Ex : effacer un écran avec un chiffon, indiquer un chemin à suivre, etc.).

Une **manipulation de type "mettre sur"** met en jeu deux objets : l'objet déplacé et l'objet réceptacle. Seules les positions initiale et finale de la souris importent dans ce type de manipulation.

La sémantique d'une telle manipulation peut être diverse, dépendante de l'application et de la métaphore utilisée dans l'interface. Il est cependant possible d'isoler comme cas le plus courant celui de commandes dont le résultat est l'ajout d'un élément dans une liste, ou l'affectation d'une valeur à une variable.

On le voit, l'introduction de mécanismes d'activation d'une opération par manipulation directe dans une interface est très dépendante de l'application, et ne peut la plupart du temps être décrite en termes génériques. Seules certaines manipulations de type "mettre sur" offrent prise à une description générique, les opérations en jeu étant assimilées à des opérations d'ajout et de suppression de valeurs. C'est dans ce domaine très précis, et relativement restreint que se situent nos efforts en matière de manipulation directe.

Règles générales

De façon générale, l'activation d'une commande s'effectue par *désignation directe* des différents éléments la décrivant.

L'identification de la commande passe par la désignation d'une entrée d'un menu de la barre de menus.

L'identification de l'objet sur lequel porte la commande n'intervient que lorsque la désambiguïsation est nécessaire, c'est-à-dire dans les situations suivantes :

- lorsque l'instance d'objet en question n'est pas directement identifiable comme valeur d'un attribut d'Espace ou de Concept.
  L'objet est un élément d'un Concept de type construit (par exemple, un élément d'une liste d'objets).
- Dans tous les cas, si le style de présentation des commandes ne reflète pas la structure des attributs (mode "par Concept"), et que la commande désignée est équivoque dans le sens où elle peut s'appliquer à plus d'un objet présent dans l'Espace.

La désambiguïsation d'une commande s'effectue par désignation directe de l'objet sur lequel s'applique la commande.

La saisie des paramètres d'une commande s'effectue de deux façons. Les paramètres de type simple sont saisis au moyen d'un formulaire. Les paramètres de type complexe (instance d'objet) sont saisis par désignation directe.

Paramètres du style d'interaction

On propose un système de paramétrage portant d'une part sur l'ordre des étapes d'activation d'une commande, et d'autre part sur l'introduction de la manipulation directe comme mode d'activation de certaines commandes.

Ordonnancement des étapes d'activation :

De façon classique, nous proposons trois schémas possibles pour l'activation d'une commande, selon une notation préfixée, postfixée, ou non contrainte.

La **notation préfixée** se traduit par les schémas d'activation d'une commande suivants :
("désig." signifie "désignation" ; [a] signifie "au plus un a", a* signifie "un nombre quelconque de a")

1. *[<désig. objet>] <désig. commande> <désig. param. complexe>* [<saisie formulaire>]*
2. *[<désig. objet>] <désig. param. complexe>* <désig. commande> [<saisie formulaire>]*

La **notation postfixée** impose le schéma d'activation d'une commande suivant :

1. *<désig. commande> [<désig. objet>] <désig. param. complexe>* [<saisie formulaire>]*

Le choix entre la notation préfixée, postfixée ou non contrainte est un paramètre de la génération.

Manipulation directe :

L'introduction de mécanismes de manipulation directe dans le style d'interaction généré est un paramètre de la génération : le développeur doit spécifier s'il souhaite ou non l'utilisation de ces mécanismes dans son application. La manipulation directe est une manipulation de type "mettre sur", et concerne deux grands ensembles d'opérations :

- les opérations élémentaires d'affectation de valeurs complexes,
- les opérations d'un Concept de nature adéquate.

Les opérations d'affectation de valeurs complexes (c'est-à-dire des instances d'objets) sont les suivantes : affectation d'une valeur complexe à un attribut modifiable de l'Espace, ou à un attribut complexe modifiable d'un Concept, ajout d'une valeur dans une liste.

La sémantique de ces opérations est simple : l'élément déplacé n'est ni une copie ni une référence, mais directement une instance d'objet ; l'objet déplacé est ôté de son réceptacle source, et affecté à son réceptacle destination ; dans le cas d'un réceptacle destination de type construit, c'est l'opération d'ajout qui est appelée.

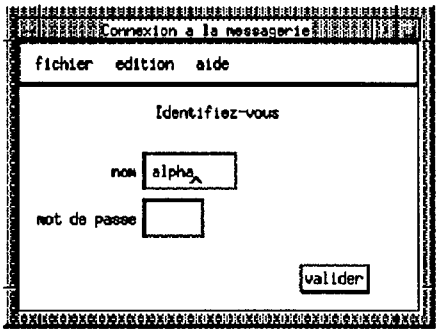
En dehors des opérations d'affectation élémentaires, la manipulation directe est utilisée dans le cas de Perspectives de nature "container" (cf. , Chapitre IV) comportant des opérations dont la propriété de nature dénote une opération d'ajout ou de retrait.

3.1.4 Exemple de génération des aspects externes

Nous complétons notre présentation des règles et paramètres de la génération des aspects externes d'une interface en reprenant l'exemple de l'application Messagerie spécifiée au Chapitre IV. Les copies d'écran fournies dans cette section présentent les principaux aspects de l'interface générée pour cette application par le générateur SIROCO-Guide. Ces copies d'écran n'ont d'autre ambition que de donner un aperçu concret des images que les règles précédemment évoquées permettent de déduire d'une spécification SIROCO.

L'exemple de génération présenté dans cette section correspond à un style d'organisation des commandes "par Concept", et à un mode d'activation préfixé.

L'Ecran 1 présente l'image de l'Espace de Travail de connexion au système, spécifié sous le nom de Login_WS. On peut reconnaître dans la barre de menus les trois menus dits "fixes". L'Espace de Travail Login_WS a la caractéristique de pas comporter de données (la Perspective associée à la variable de l'espace, Login_P, filtre les attributs), et d'offrir une seule opération (opération GetUser) ; cette caractéristique déclenche une exception aux règles générales de présentation des commandes, et conduit à présenter l'opération GetUser et la saisie de ses paramètres dans la zone de travail de la fenêtre de Login_WS ; le bouton "valider" lance l'activation effective de l'opération. Les chaînes de caractères apparaissant dans la présentation proviennent des valeurs des propriétés d'identification fournies dans la spécification.

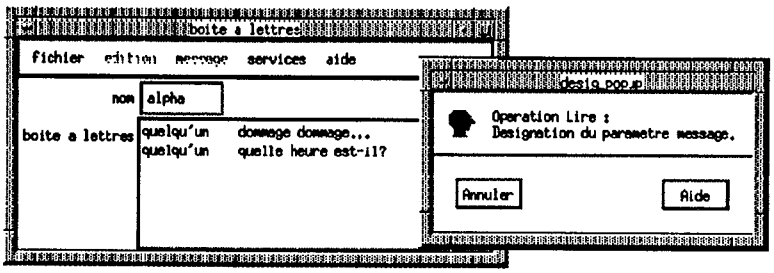


Ecran 1 : *La présentation de l'Espace de Travail de connexion.*

L'Ecran 2 montre l'Espace de Travail de traitement des messages, spécifié sous le nom de HandleMess_WS. La barre de menus reflète le style d'organisation "par Concept" en offrant deux menus non "fixes" : le menu "message" et le menu "services".

- Le menu "message" correspond au Concept de Message, élément de la boîte-à-lettres : Message est le seul Concept de l'Espace dont la Perspective comporte une opération. Le menu ne comporte qu'une entrée, présentant l'opération de destruction, seule opération de la Perspective sur le Concept.
- Le menu "services" présente les opérations de navigation définies sur l'Espace. Il comporte trois entrées, correspondant aux opérations d'écriture, de lecture, et de réponse.

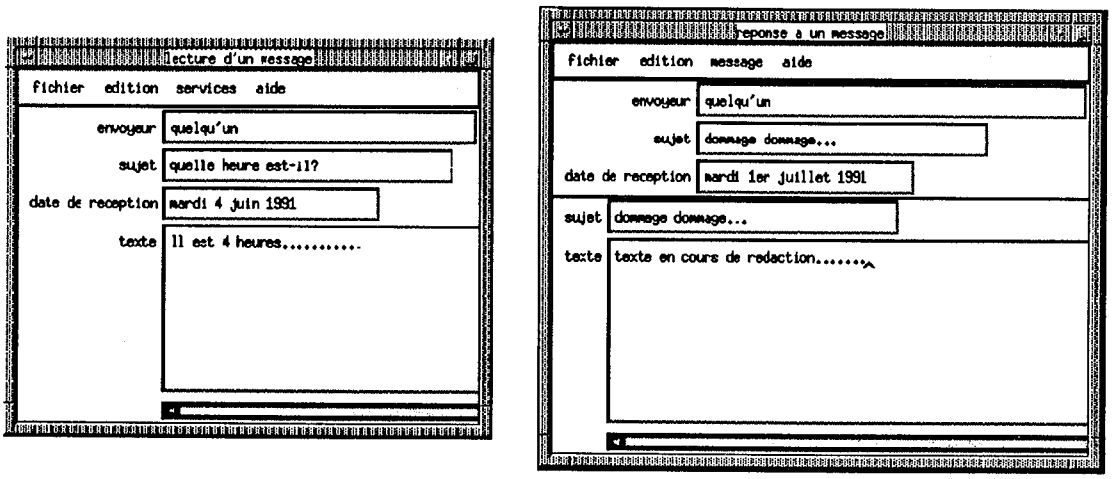
La zone de travail de la fenêtre de l'Espace présente les deux attributs de la Perspective User_P : le nom de l'utilisateur courant et la boîte-à-lettres de ce dernier. C'est un afficheur de liste textuel qui est utilisé pour présenté le contenu du Concept de boîte-à-lettres.



Ecran 2 : *La présentation de l'Espace de Travail de traitement.*

L'Ecran 2 reflète également la gestion du mode préfixé de l'activation d'une commande : le menu "messages" est en grisé (invalidé) car aucun message n'est sélectionné ; le menu "édition" est également invalidé, car aucune sélection n'a été effectuée. La fenêtre de droite résulte de l'activation de l'opération de lecture d'un message par l'utilisateur ; elle requiert la désignation par l'utilisateur du message paramètre de cette opération.

L'Ecran 3 représente l'image des Espaces de Travail de lecture et de réponse, spécifiés respectivement sous les noms de Read_WS et Reply_WS.



Ecran 3 : *La présentation des Espaces de Travail de lecture et de réponse.*

La fenêtre de l'Espace de lecture reflète les caractéristiques de la Perspective utilisées sur l'unique attribut de l'Espace, le message à lire. Aucune opération n'est offerte, d'où l'absence de menu spécifique dans la barre de menus. Les quatre attributs figurant dans la Perspective sont présentés à la verticale, dans l'ordre de leur spécification à l'intérieur de la Perspective. Ces attributs étant de type chaîne de caractères, ce sont des champs de texte qui sont affichés. Notons l'utilisation d'une fenêtre de texte munie d'une barre de défilement pour l'attribut texte du message, induite par la taille spécifiée pour cet attribut.

La fenêtre de l'Espace de réponse a la particularité de comporter deux sous-fenêtres correspondant aux deux attributs de donnée de cet Espace : le message sur lequel porte la réponse, et le message de réponse.

3.2 Une fabrique d'objets

Le générateur de code que nous proposons est une fabrique d'objets.

Comme le représente la Figure V-5, le générateur prend en entrée un fichier contenant une description de l'interface d'une application en langage SIROCO, ainsi qu'un jeu de paramètres. Il produit en sortie un ensemble de fichiers contenant la définition de types et de classes d'objets réalisant l'interface de l'application. C'est en rattachant ces objets au code du composant fonctionnel produit par le développeur que l'on obtient un prototype complet de l'application interactive.

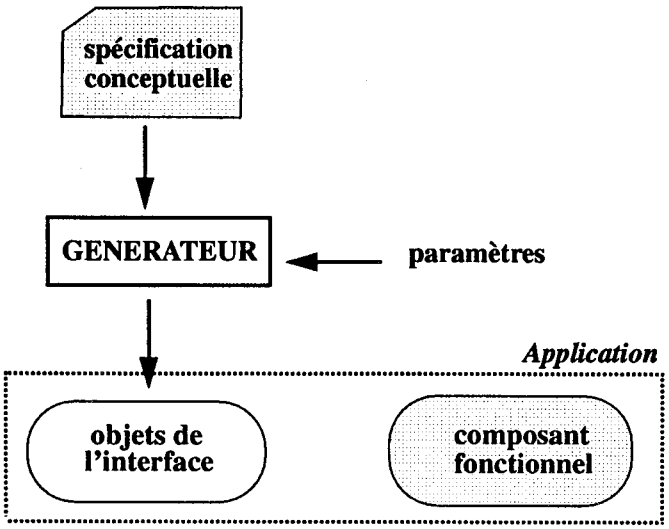


Figure V-5 : *Présentation générale du générateur.*

Après une rapide description de l'approche générale, nous donnons les grands principes de fabrication des différents ensembles d'objets. Une description plus technique de la réalisation du générateur SIROCO-Guide est proposée en Annexe B.

3.2.1 Approche générale

La conception et la réalisation de la fabrique d'objets se situe dans le prolongement de nos travaux sur les modèles d'architecture : les objets produits se conforment au modèle d'architecture étendu décrit en tête de ce chapitre. La fabrique d'objets peut ainsi être perçue comme une automatisation du processus de transcription des éléments conceptuels en objets de réalisation.

Les objets produits se répartissent ainsi en trois ensembles : les objets de gestion de la présentation et de l'interaction, les objets de contrôle, et les objets de l'adaptateur du composant fonctionnel. C'est à partir de ces derniers objets que s'effectue le rattachement de l'interface générée au composant fonctionnel délivré par le développeur, de façon à produire un prototype complet du système interactif.

Nous avons déjà indiqué que le langage cible de cette fabrique est le langage Guide, dans l'environnement graphique X-Window et Motif. Plus précisément, la réalisation de cette fabrique s'effectue dans l'environnement du squelette d'application résultant de nos premiers travaux, introduit au Chapitre III. Ce squelette est repris et étendu de façon à constituer la plateforme d'accueil des objets générés.

3.2.2 Fabrication des objets de l'adaptateur

Ces objets mettent en œuvre l'interface entre le code généré et le composant fonctionnel de l'application. Ils constituent la première "poignée" que le développeur utilisera pour exploiter le code généré en le reliant à son application.

Les objets de l'adaptateur sont directement générés à partir de la spécification des Concepts du système interactif. Un Concept est traité comme la description d'un type d'objet à laquelle vient s'ajouter un ensemble d'informations sous la forme d'étiquettes accolées à chaque élément de ce type. Le travail du générateur est ici très simple, puisqu'il s'agit simplement d'extraire la structure du type sous-jacente à la spécification du Concept.

Le générateur produit une classe "vide" pour chacun des types. Il s'agit en quelque sorte de squelettes de classes destinés à être complétés par le développeur afin de faire le lien avec le composant fonctionnel de l'application interactive. Ce travail pourra être effectué de deux façons :

- soit en définissant le composant fonctionnel directement à partir de ces squelettes de classe.
  (Possible lorsque le composant fonctionnel n'est pas pré-existant au code généré).
- soit en instanciant les éléments du composant fonctionnel à l'intérieur des classes générées.

3.2.3 Fabrication des objets de contrôle

Ces objets mettent en œuvre la dynamique du dialogue. La hiérarchie de contrôle produite reflète les principes du modèle d'architecture étendu, en les figeant de la façon suivante :

- au plus haut niveau, l'objet session_co.
- un objet workspace_co par Espace de Travail.
- un objet data_co par attribut d'un Espace.
- un objet operation_co par Espace.
- subordonnés à l'objet operation_co, un objet dialog-thread_co par opération complexe de l'Espace.

Objet session_co

La génération du type et de la classe de l'objet session_co s'effectue à partir de la spécification de l'objet Session en ce qui concerne la définition du contexte de données, et l'instanciation des Espaces initiaux du système interactif. La génération du code de gestion de la dynamique des Espaces prend en entrée les caractéristiques des différents Espaces spécifiés (unicité ou non, caractère principal ou non, etc.), ainsi que les définitions de transitions entre Espaces figurant dans la spécification.

Objet workspace_co

L'objet workspace_co a un rôle actuellement limité à celui de stockage des données d'un Espace, d'initialisation et de terminaison d'un objet de gestion de l'image et des objets de contrôle réalisant cet Espace : un objet operation_co, gestionnaire des opérations, et plusieurs objets de contrôle des données data_co.

Seule une classe est générée, répondant à un type prédéfini. La génération de cette classe s'effectue à partir de la spécification de l'Espace.

Objet data_co

Un objet data_co a pour rôle de gérer un attribut intervenant dans la spécification d'un Espace.

Seule une classe est générée, implémentant un type prédéfini. La génération de cette classe s'effectue à partir de la spécification de l'attribut de l'Espace.

Objet operation_co

On génère un unique objet operation_co par Espace. Cet objet a pour rôle de gérer d'une part la présentation des opérations de l'Espace (en fonction de leur statut activable ou non, dépendant du contexte, du style d'interaction choisi, et d'éventuelles préconditions), au travers d'un ensemble d'objets de gestion de l'image, et d'autre part de mettre en œuvre l'activation d'une opération par l'utilisateur (saisie des paramètres éventuels et transmission au composant fonctionnel, exécution des post-actions). Seule l'activation des opérations sans paramètres est entièrement prise en charge par l'objet operation_co ; dans le cas d'une opération nécessitant l'entrée de paramètres, l'objet operation_co instancie un objet dialog-thread_co et lui passe le relais.

Seule une classe est générée, répondant à un type prédéfini. La génération de cette classe s'effectue à partir de la spécification de l'Espace : les opérations figurant dans les Perspectives mises en jeu dans l'Espace, ainsi que les opérations d'accès à d'autres Espaces.

Objet dialog-thread_co

L'objet dialog-thread_co a pour fonction, comme son nom l'indique, de gérer un fil du dialogue entre l'utilisateur et le système. Ce fil de dialogue correspond au dialogue nécessaire à la mise en œuvre d'un appel à une opération : désignation et/ou saisie des paramètres de l'opération, prise en compte des cas d'erreur, et demande de confirmation dans le cas d'une opération "sensible". L'exécution effective de l'opération (transmission au composant fonctionnel, exécution des post-actions) est également prise en charge par l'objet dialog-thread_co, qui signale ensuite à l'objet operation_co l'accomplissement de sa mission. Seules les opérations "complexes", c'est-à-dire comportant des paramètres ou nécessitant une confirmation de l'utilisateur donnent lieu à la génération d'un objet dialog-thread_co ; les autres opérations sont directement gérées par l'objet operation_co.

Seule une classe est générée, répondant à un type prédéfini. La génération de cette classe s'effectue à partir de la spécification d'une opération "complexe" apparaissant dans une Perspective.

3.2.4 Fabrication des objets de présentation et d'interaction

L'image de l'interface utilisateur est mise en œuvre par un ensemble d'objets dont seuls les objets de présentation des Concepts de l'Application sont le résultat d'une génération. En effet, de nombreux objets de gestion de l'image sont des objets prédéfinis utilisés par les objets de contrôle au travers de mécanismes de paramétrisation. Ces objets prédéfinis sont les objets *workspace_view* (fenêtre d'un Espace), *operation_view* (mise en œuvre générique d'une barre de menus ou d'une fenêtre de commandes), *param_view* (mise en œuvre générique d'un formulaire de saisie de paramètres), *list_view* (mise en œuvre générique de la présentation d'une liste de données), etc.

Un objet de présentation de données correspond à la présentation d'une Perspective sur un Concept. Le code généré donne lieu à la création d'objets graphiques permettant la visualisation et la modification éventuelle de chaque attribut intervenant dans la Perspective.

La génération du type et de la classe d'un objet de gestion de l'image s'effectue à partir de la spécification d'une Perspective : les attributs figurant dans la description de la Perspective, et leur caractéristiques.

3.2.5 Exemple de génération

Nous reprenons l'exemple du système de messagerie, et achevons notre démonstration. L'exemple que nous décrivons correspond à une génération effectuée avec les paramètres de style d'interaction suivants :

- style de désambiguïsation pré-fixé : l'utilisateur désigne d'abord l'objet puis l'opération à appliquer à l'objet,
- style de saisie des paramètres complexes par désignation, et sans contraintes : l'utilisateur spécifie un paramètre de type objet par désignation directe, avant ou après le choix de l'opération.

L'interface générée permet ainsi les schémas suivants pour l'activation d'une commande (selon que la commande est une opération portant sur un objet ambigu ou non) :

1. *[<désig. objet>] <désig. commande> <désig. param. complexe>* [<saisie formulaire>]*
2. *<désig. param. complexe>* <désig. commande> [<saisie formulaire>]*

Nous limitons notre présentation aux fonctions de l'Espace de Travail de traitement, HandleMess_WS. La Figure V-6 représente l'architecture du code généré pour la mise en œuvre de ces fonctions.

N.B. : seules les classes d'objets effectivement générées apparaissent dans cette figure. Nombreux sont les objets entrant dans la mise en œuvre du système qui ne relèvent pas de la génération, mais de la réutilisation paramétrée. C'est notamment le cas des objets vue attachés aux Espaces de Travail, aux opérations, etc. Le lecteur intéressé trouvera en Annexe B une description de la politique adoptée quant à la génération et à la réutilisation de code.

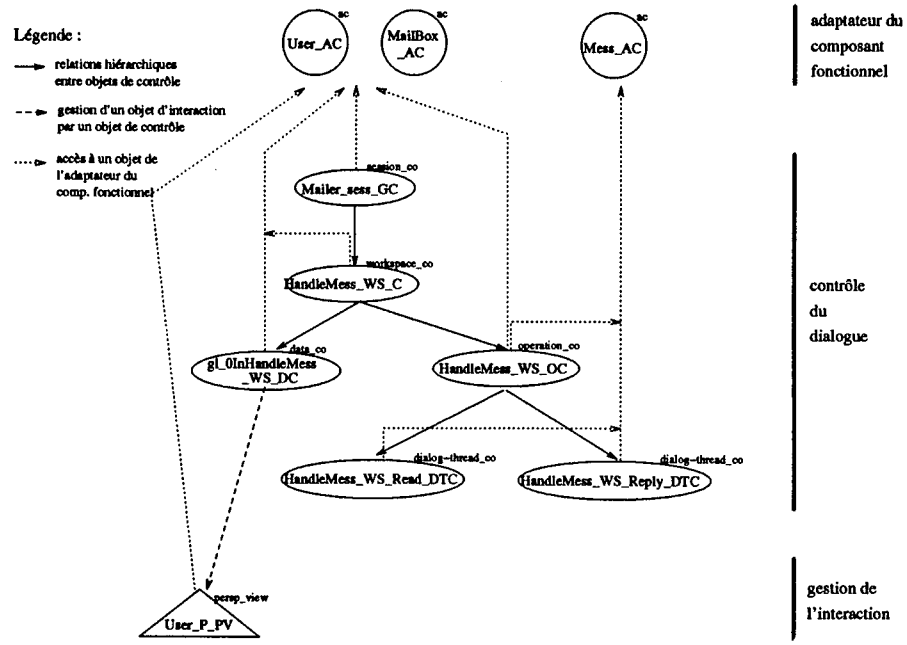


Figure V-6 : *L'architecture du code généré pour l'application Messagerie (extrait).*

Objets de l'adaptateur : un type et une classe sont générés pour chaque Concept de l'Application : utilisateur (User_AC[1]), boîte-à-lettres (Mailbox_AC), message (Mess_AC).

Objets de contrôle : on retrouve l'objet session_co (classe Mailer_sess_GC), et l'objet de gestion de l'Espace de Travail considéré (classe HandleMess_WS_C).

Un objet data_co est généré pour gérer les données dépendant de l'unique attribut de l'Espace de Travail (classe gl_0InHandleMess_WS_DC).

L'objet operation_co (classe HandleMess_WS_OC) met en œuvre les commandes de l'Espace de Travail : il gère la désambiguïsation des opérations de façon pré-fixée, et met en œuvre directement l'opération de destruction d'un message, et l'accès à l'Espace de Travail Write_WS (lecture d'un message) - deux opérations dépourvues de paramètres ; cet objet gère de plus l'instanciation de deux objets dialog_thread_co chargés de mettre en œuvre les opérations dotées de paramètres.

1. Signification des suffixes des types et classes d'objets générés :
   - _AC : Application Concept
   - _SC : Session Controller
   - _C : (Workspace) Controller
   - _DC : Data Controller
   - _OC : Operation Controller
   - _DTC : Dialog Thread Controller
   - _PV : Perspective View

La classe HandleMess_WS_Read_DTC (resp. HandleMess_WS_Reply_DTC) dénote un objet dialog_thread_co qui met en œuvre l'accès à un Espace de Travail Read_WS de lecture d'un message (resp. Reply_WS, de réponse à un message).

Objets de gestion de la présentation et de l'interaction : un unique objet est généré (classe User_P_PV) pour mettre en œuvre la présentation des données contenues dans l'unique attribut de HandleMess_WS, et déduite de la Perspective User_P.

Le lecteur intéressé trouvera en Annexe C le code généré pour les objets de l'application Messagerie qui viennent d'être présentés.

3.3 Services des interfaces générées

Au-delà de la simple mise en œuvre de l'interface décrite dans une spécification SIROCO, le générateur de code offre un cadre privilégié pour la fourniture de ce que nous avons appelé services d'interface au Chapitre III. Ces services - nous citons ici le presse-papier et l'aide statique, constituent une part générique des éléments fonctionnels d'une interface en général, et à ce titre n'apparaissent pas explicitement dans une spécification SIROCO. Au contraire, ces services sont mis en œuvre de façon automatique, partie intégrante du code généré. Ils apparaissent dans l'interface au travers des commandes dites "fixes", précédemment évoquées lors de la description de la barre de menus d'une fenêtre.

3.3.1 Presse-papier

Le presse-papier est un "tampon" susceptible de recevoir des données (au travers des opérations copier et couper), et de restituer des données (opération coller). Les données contenues dans le presse-papier peuvent être de type simple ou complexe. Le presse-papier est réalisé comme un objet partagé entre toutes les applications interactives s'exécutant sur une station de travail, permettant ainsi le transfert d'informations d'une application à une autre.

Le concept de presse-papier véhicule les notions de source et de destination, ainsi que la notion de transformation de type. La source d'une opération copier est une donnée désignable ; toute donnée désignable est susceptible d'être sélectionnée comme source, quel que soit son type. La destination d'une opération coller est un attribut modifiable appartenant à un Espace ou bien à un Concept.

La mise en œuvre de l'opération coller nécessite d'assurer une éventuelle transformation de type afin d'ajuster la donnée contenue dans le tampon au type du réceptacle destination. Les transformations de type assurées par SIROCO sont les suivantes :

- transformations entre types simples :
  texte -> nombre entier,
  nombre entier -> texte.
- transformations entre type complexe et type simple
  instance de Concept -> texte (le texte extrait est celui des attributs visibles dans la Perspective de l'objet désigné, s'ils existent ; dans le cas contraire, le texte extrait est celui des attributs clés de l'objet désigné).
- transformations entre types complexes
  instance de Concept -> instance de Concept, en respectant la règle de conformité des types Guide[1].

Note : les concepts de presse-papier, et d'opérations couper, copier, coller ont été développés lors de nos précédents travaux sur le squelette d'application.

3.3.2 Aide statique

Le service d'aide statique exploite les informations contenues dans une spécification SIROCO, et fournit à l'utilisateur des réponses à ses demandes d'aide concernant les Espaces, objets, attributs, variables, opérations apparaissant dans l'interface.

Les informations d'aide contenues dans une spécification sont doubles : d'une part, les textes définis comme valeurs des propriétés "descriptif court" et "descriptif long" sur chaque élément de la spécification ; d'autre part, les informations non textuelles de structure et de relation apparaissant dans la spécification.

Informations textuelles et non textuelles permettent la mise en œuvre de trois niveaux d'aide : aide courte, aide longue, et aide étendue.

Un message d'aide courte est produit à partir du texte donné comme valeur de la propriété "descriptif court" de l'élément courant.

Un message d'aide longue est produit à partir du texte donné comme valeur de la propriété "descriptif long" de l'élément courant.

L'aide étendue ne concerne que les éléments complexes d'une interface : objets Concepts, Espaces, et opérations. Un message d'aide étendue explicite la structure de l'élément complexe courant ; ce message est généré à partir des descriptions structurelles spécifiées, et des textes de description courte des éléments de la structure.

1. Considérons l'affectation d'un objet de type T1 à une variable définie comme étant de type T2. Cette affectation nécessite que T1 soit syntaxiquement conforme à T2, c'est-à-dire que tout contrôle syntaxique vérifié par T2 soit valide pour T1.

Cette relation est notamment vérifiée entre type et sous-type : un sous-type est conforme à un type puisque, par définition, le sous-type offre au moins l'interface offerte dans le type.

Chapitre VI

Evaluation

1 Introduction

Le moment est venu de faire le point sur les travaux présentés aux chapitres IV et V. Ces travaux se traduisent en un langage de spécification d'un système interactif, SIROCO ; un modèle d'architecture argumenté ; un générateur de code SIROCO-Guide.

Après quelques indications quant à l'état de ces travaux, nous décrivons la démarche d'évaluation adoptée pour la suite de ce chapitre.

1.1 Etat des travaux

Le *langage SIROCO* offre aujourd'hui une forme relativement figée ; le modèle de représentation d'un système interactif ainsi que les principes de la syntaxe SIROCO sont stabilisés, sans pour autant présenter un caractère achevé - l'extension du langage SIROCO est une des perspectives que nous évoquerons lors de la conclusion générale.

Le *modèle d'architecture* défini, fortement ancré dans le modèle de représentation SIROCO, se concrétise dans les objets logiciels d'un squelette d'application qui constitue la plateforme utilisée par SIROCO-Guide.

SIROCO-Guide est une maquette d'outil permettant la génération du code de réalisation de l'interface d'un système décrit en langage SIROCO ; le langage cible de ce générateur est le langage Guide. En développant SIROCO-Guide, notre objectif est la réalisation d'un prototype démontrant l'intérêt et la faisabilité de nos idées. Faute de temps et de moyens - du fait, notamment, de la nature prototypique de l'environnement de développement Guide lui-même, il ne peut être question de produire un outil complet. Les limitations portent principalement sur la version du langage SIROCO exploitée, les contrôles mis en œuvre, la présentation générée.

SIROCO-Guide exploite une version restreinte du langage SIROCO. Les restrictions concernent principalement les aspects du langage relatifs à la spécification du modèle de données, c'est-à-dire des Concepts :

- Héritage et restriction de type ne sont pas mis en œuvre dans la version actuelle de SIROCO-Guide, de même que les notions de composants et de structure d'un Concept.
- La version actuelle de SIROCO-Guide ne traite que les Concepts (et Perspectives) dont la définition est non récursive.

Les contrôles de cohérence d'une spécification en langage SIROCO sont loin d'être complets. Un gros travail est à réaliser afin de consolider SIROCO-Guide : la version actuelle du prototype offre de nombreux "trous" qu'il est nécessaire de combler afin de rendre ce dernier réellement utilisable par un développeur externe au projet.

En ce qui concerne l'image de présentation générée, le seul afficheur complexe intégré à SIROCO-Guide est un afficheur de liste (textuelle ou iconique). Le développement d'afficheurs complexes de type graphe, arbre, etc. est nécessaire afin notamment de prendre en compte les types récursifs d'une spécification.

1.2 Mode d'évaluation

Du fait de la nature prototypique des travaux réalisés, l'expérience d'utilisation de SIROCO est limitée. Les seules expériences d'utilisation dont nous disposons sont en effet

issues des travaux de conception et de réalisation autour de SIROCO : spécification d'applications diverses lors de la mise au point des caractéristiques du modèle et du langage SIROCO ; mise en œuvre d'applications de test de la maquette SIROCO-Guide, et notamment l'application de Messagerie qui illustre la présentation de nos travaux dans les chapitres précédents. Ce manque d'expérience d'utilisation prévient une évaluation directe des fonctions de SIROCO, des performances du code généré, etc. C'est par conséquent une évaluation vis-à-vis de l'existant et des problèmes traités que nous esquissons ici.

Nous avons tenu, lors de l'exposé de nos travaux, à distinguer le modèle et le langage SIROCO de l'outil de développement que représente le générateur de code SIROCO-Guide. Cette dualité est naturellement reflétée dans cette section. Dans un premier temps, nous évaluons l'originalité et l'intérêt de SIROCO-Guide par rapport aux outils de nature voisine que décrit la littérature. Dans un second temps, notre réflexion porte sur nos travaux de modélisation seule ; nous faisons le point sur l'intérêt et les potentialités de SIROCO relativement au processus général de conception et de développement des interfaces utilisateur.

2 SIROCO-Guide et les outils existants

Nous avons au Chapitre III souligné la rareté des outils d'aide au développement axés sur les concepts du domaine de l'application.

Parmi les outils précurseurs des travaux de cette nature, citons Cousin [Hayes 85], et Domain/Dialogue [Schulert 85], ainsi que les travaux de Olsen : MIKE [Olsen 86], puis Mickey [Olsen 89]. Ces différents outils ont pour nature commune de reposer sur un langage déclaratif permettant la description des entités fonctionnelles présentes dans l'interface, et l'indication de la forme de cette présentation ; la compilation de cette description produit une version exécutable de l'interface ainsi décrite.

A notre connaissance, seuls deux outils se fondent sur une spécification purement fonctionnelle, dénuée d'indications concernant la présentation : UIDE, et MacIDA UIMS. Ce sont là les outils de référence de notre évaluation.

2.1 SIROCO-Guide et UIDE

UIDE (User Interface Design Environment) [Foley 88] est un outil de spécification et de génération des interfaces utilisateur relativement complet - dans sa catégorie, l'on peut considérer qu'UIDE est un des outils les plus ambitieux et les plus accomplis, même si les réalisations décrites ne sont pas sans lacunes.

Caractéristiques principales

Comme son nom l'indique, UIDE se veut avant tout un outil d'aide à la conception (au "design") d'une interface. La spécification des données de description d'une interface, l'analyse, l'évaluation, et la transformation de ces données sont les principaux éléments de UIDE. La génération d'une interface exécutable vient ensuite, et ne semble pas le premier intérêt des auteurs de UIDE - il suffit pour s'en persuader de relever l'abondante littérature consacrée aux aspects conceptuels : par exemple, [Foley 87a] et [Foley 87b].

UIDE propose une *spécification* reposant sur un modèle de représentation à objets. Sont spécifiés :

- les classes des objets exportés par le système à destination de l'interface,
- les actions pouvant être effectuées sur les objets,
- les paramètres de ces actions,
- les pré- et post-conditions de ces actions.

La spécification, bien que fortement fonctionnelle, met également en jeu des notions propres à la description du dialogue et du style d'interaction ; par exemple, les notions de commande courante, de valeur courante de paramètre, d'élément sélectionné, etc.

UIDE offre un outil de saisie d'une spécification au travers de menus et de messages d'invite, et propose *un ensemble d'outils* prenant en entrée la base de données résultant d'une spécification :

- deux outils d'assistance à la conception : transformation de l'interface, et évaluation.
- deux outils dédiés à la réalisation : génération d'une interface exécutable, et d'un système d'aide à l'utilisateur.

UIDE offre des fonctions permettant la *transformation d'une spécification* donnée en une spécification fonctionnellement équivalente. Un jeu d'algorithmes de transformation est fourni afin de permettre l'exploration par le concepteur de plusieurs stratégies de structuration des données de spécification de son interface. Parmi les transformations réalisables, la factorisation des objets ou des commandes, par l'introduction de la notion de sélection courante, ou de commande courante ; la mise en œuvre de valeurs par défaut pour les paramètres des commandes ; la spécialisation ou la généralisation des commandes, à partir des hiérarchies d'objets et de commandes spécifiées par le concepteur.

UIDE offre un outil d'*évaluation* non pas de la qualité mais de la rapidité d'utilisation de l'interface spécifiée, par une technique de type "Key-stroke" (cf. section 2.3, Chapitre II).

La *compilation* d'une spécification UIDE produit une version exécutable de l'interface spécifiée. UIDE offre également un système d'*aide contextuelle* générant des messages d'information sur les commandes d'un système, et donnant par exemple les raisons pour lesquelles une commande est invalide.

SIROCO-Guide et UIDE présentent les points communs suivants : ces deux outils reposent sur une représentation conceptuelle d'un système interactif, et le modèle de représentation utilisé par chaque outil est orienté objet. SIROCO-Guide et UIDE offrent cependant de notables différences, tant au niveau du modèle de représentation que des fonctions et services de génération d'interface.

Modèle de représentation conceptuelle

L'élément de base de la spécification UIDE est l'objet, que l'on peut assimiler à notre notion de Concept de l'Application.

Un premier point à souligner est la grande pauvreté du modèle de données de UIDE, tel que décrit dans la littérature : les seules relations possibles entre deux objets proviennent des relations d'héritage ; notamment, un objet ne peut en référencer un autre (c'est-à-dire qu'un attribut d'objet ne peut être que de type simple : chaîne, entier, etc.).

Tout comme dans SIROCO, c'est sur un mécanisme de pré- et de post- conditions que repose l'expression des contraintes de fonctionnement du système.

Contrairement aux choix que nous avons adoptés pour SIROCO, UIDE mêle spécification fonctionnelle et indications sur le dialogue : une spécification UIDE peut faire appel aux notions du niveau du dialogue que sont la sélection courante, la commande courante, etc. L'*intérêt* de ces notions est une description fine du dialogue d'activation d'une commande, au travers de la spécification des pré- et post- conditions de ces commandes : UIDE permet de spécifier si l'activation d'une commande nécessite la sélection d'un objet de type donné, ou bien l'existence d'une valeur d'attribut donné, etc. La finesse de cette description est exploitée par les algorithmes de transformation précédemment évoqués. L'*inconvénient* de l'approche UIDE est l'enchevêtrement de spécifications de niveaux différents : l'on pourra trouver dans une pré-condition à la fois des contraintes fonctionnelles et des contraintes relatives au style du dialogue.

SIROCO fait une distinction très nette entre la spécification du contenu d'une interface et l'indication d'un style de dialogue. Les notions de description du dialogue n'apparaissent pas

dans la spécification, et restent implicites à celle de style d'interaction : par exemple, un style préfixé nécessite la sélection d'un objet avant une commande.

Par ailleurs, UIDE ne comporte pas de notions équivalentes à celles de Perspective et d'Espace de Travail. L'organisation de l'utilisation des données et opérations d'un système n'apparaît pas dans une spécification UIDE. Sans développer la notion de dimension d'utilisation d'un système qui caractérise nos travaux, la génération d'une interface nécessite de disposer d'informations minimales quant au regroupement des données, et à l'existence de fenêtres de partitionnement de l'interface. La littérature n'est pas explicite quant au mode de recueil de ces informations dans UIDE. Il semblerait qu'elles soient fournies explicitement par le concepteur dans l'environnement d'exécution de l'interface (appelé SUIMS), lors de la modification de la présentation - étape que nous décrivons ci-après.

Style d'interface

Les styles d'interfaces mis en œuvre par SIROCO-Guide et UIDE sont assez différents : notamment, il n'est pas clair que UIDE autorise le multifenêtrage dans la version décrite dans [Foley 88] ; la présentation des commandes dans UIDE n'est pas explicite non plus. SIROCO-Guide et UIDE favorisent cependant tous deux la désignation directe. UIDE autorise la désignation de positions, un type de paramètre non réalisé dans la version actuelle de SIROCO-Guide.

En ce qui concerne la paramétrisation du style d'interface à mettre en œuvre, SIROCO et UIDE offrent des services différents, et en quelque sorte complémentaires : UIDE offre des outils de transformation du style d'interaction, tandis que SIROCO définit des paramètres de style de présentation et d'interaction. Les algorithmes de transformation de style sont un des principaux atouts d'UIDE, permettant la mise en œuvre de puissantes transformations sur la "géométrie" des commandes et de leurs paramètres. SIROCO ne traite pas cette question, mais offre un mécanisme de paramétrisation relativement simple portant sur le style d'interaction (il est à noter que le choix d'un style pré ou postfixé est équivalent à l'application d'un algorithme de transformation de UIDE), ou bien sur la présentation, aspect non traité par UIDE.

Mise en œuvre de l'interface

SIROCO-Guide et UIDE semblent adopter deux stratégies très différentes en ce qui concerne la mise en œuvre de l'interface. SIROCO-Guide génère du code qui sera intégré au code du composant fonctionnel de l'application, le lien s'effectuant au travers des classes d'objets dits "de l'adaptateur du composant fonctionnel" ; le code est compilé. UIDE ne génère pas du code mais produit deux bases de données décrivant d'une part les données conceptuelles de l'interface et d'autre part les données de description concrète de l'interface ; l'exécution d'une application interactive passe par l'exécution d'un outil appelé SUIMS, noyau générique exploitant les deux bases de données.

La base des données de description concrète de l'interface est directement modifiable lors de l'exécution de l'interface à l'intérieur de SUIMS, ce qui permet au concepteur de contrôler et modifier les détails de présentation et de techniques d'interaction.

UIDE interdit très clairement la modification de l'interface en dehors des outils fournis dans cet environnement, tandis que SIROCO-Guide considère la possibilité de modification/ surcharge du code généré comme une échappatoire aux limitations de la génération

(principalement en ce qui concerne la présentation générée). Le choix effectué par UIDE est ambitieux car il suppose la mise à disposition d'outils puissants de spécification/modification des paramètres concrets de l'interface. Les limites de la spécification sont cependant claires sur au moins un point, celui de la présentation de données complexes nécessitant des afficheurs spécifiques (par exemple, graphes, diagrammes, etc.) ; la réponse d'UIDE sur ce point n'est pas explicite.

Conclusion

UIDE et SIROCO sont deux projets de tailles très inégales ; UIDE est un véritable environnement de développement des interfaces, SIROCO-Guide n'est qu'un prototype aux fonctions limitées. Toutes proportions gardées, ces deux outils possèdent des bases communes, mais diffèrent profondément dans leurs approches de la spécification et de la réalisation d'une interface.

UIDE mêle spécification des Concepts de l'Application et description des détails du dialogue. SIROCO au contraire isole la spécification conceptuelle de la spécification du style de dialogue, réduisant cette dernière au positionnement de paramètres globaux à l'interface. La première solution favorise la souplesse au détriment de la clarté des spécifications et de l'homogénéité du dialogue ; la seconde assure au contraire un partage rigoureux des spécifications et l'obtention d'un style de dialogue homogène, au détriment d'une finesse d'expression du dialogue.

Les éléments de la dimension d'utilisation développés dans SIROCO n'ont pas d'équivalent dans UIDE. C'est lors de l'intervention du concepteur sur la présentation générée que cette dimension trouve implicitement une expression, de façon partielle et diffuse puisque relevant des détails de présentation uniquement. A notre sens, il s'agit là d'une lacune importante dans le modèle de spécification de UIDE.

En ce qui concerne la mise en œuvre de l'interface, SIROCO-Guide et UIDE adoptent des stratégies différentes, sans doute liées aux environnements particuliers aux deux projets : la stratégie de SIROCO-Guide est parfaitement intégrée à l'environnement de programmation à objets Guide, celle d'UIDE étant axée sur la mise en œuvre de bases de données.

2.2 SIROCO-Guide et MacIDA UIMS

MacIDA UIMS [Petoud 90] est un outil de développement d'interface réalisé dans le contexte des applications de gestion des systèmes d'information. En dépit de cette dimension "bases de données", les outils tel celui-ci réalisent des fonctions analogues à celles d'outils de développement d'interfaces plus généraux. C'est pourquoi la comparaison de SIROCO-Guide et de MacIDA UIMS nous semble intéressante ; on s'attachera à distinguer les spécificités issues du domaine des bases de données.

Les principes de MacIDA UIMS sont les suivants :

- les applications visées sont des applications de consultation et/ou de modification d'une base de données.
- MacIDA prend en entrée la spécification fonctionnelle d'une application, exprimée en langage IDA/DSL selon un modèle entité-relation.

MacIDA UIMS offre les outils suivants :

- un générateur d'interface mettant en œuvre un système d'aide à l'utilisateur sous la forme d'un "tableau de bord" indiquant notamment l'état d'avancement des tâches de l'utilisateur.
- un éditeur interactif permettant l'adaptation par manipulation directe de l'interface résultant de la génération.

Modèle de représentation conceptuelle

Le modèle de représentation adopté par MacIDA est celui de IDA [Bodart 89a] (cité dans [Petoud 90]), une méthodologie de conception des systèmes d'information. Ce modèle est très significatif de l'environnement "bases de données" ; il s'agit d'un modèle de spécification du composant fonctionnel de l'application, et non pas d'un modèle de représentation conceptuelle de l'interface. Notamment, le modèle IDA privilégie l'expression de contraintes d'enchaînement des fonctions caractéristiques des applications de gestion, et reflétant l'aspect fortement structuré de l'arbre des tâches de ce type d'applications.

La notion de "phase" dans IDA, décrite comme "une unité spatio-temporelle d'exécution", semble correspondre à notre notion d'Espace de Travail, à ceci près que deux phases ne peuvent s'exécuter en parallèle. Une phase permet l'exécution d'un ensemble de fonctions sur les données du système. Le modèle des données est un schéma entité-relation. Les contraintes d'enchaînement des fonctions sont exprimées sous la forme de "messages d'entrée" et "messages de sortie" d'une fonction. Un message peut représenter soit des données à saisir par l'utilisateur, soit une erreur d'exécution, soit un résultat interne transmis entre deux fonctions ou deux phases.

La notion de Perspective SIROCO ne possède pas d'équivalent dans IDA. Notons simplement que les messages représentant des données à saisir par l'utilisateur dénotent une notion de Perspective sur une entité ou une relation du modèle des données - Perspective limitée à la spécification du mode d'accès (lecture ou écriture) aux attributs de cette entité ou relation. Lors de la génération de l'interface, toutes les entités et les relations définies dans la phase sont présentées, munies de l'ensemble de leurs attributs.

Style d'interface

Les styles d'interface mis en œuvre par SIROCO-Guide et MacIDA sont assez différents : les interfaces MacIDA mettent en jeu principalement des formulaires de saisie ; désignation directe, manipulation directe ne sont pas mises en œuvre, hormis les services offerts par la Toolbox du Macintosh.

MacIDA ne semble offrir aucun support à la paramétrisation automatique du style d'interface à mettre en œuvre ; toute modification de l'interface générée passe par l'utilisation de l'éditeur interactif d'interface.

Mise en œuvre de l'interface

SIROCO-Guide et MacIDA ont en commun de s'inscrire dans un environnement de développement à objets (Object Pascal pour MacIDA), et d'utiliser un squelette d'application (MacApp pour MacIDA) lors de la mise en œuvre de l'interface.

Conclusion

En conclusion, la principale différence entre MacIDA et SIROCO-Guide concerne le modèle de spécification de l'interface. MacIDA choisit de mettre en œuvre une spécification fonctionnelle calquée sur les schémas de données mis en œuvre dans l'application ; SIROCO définit un modèle de spécification conceptuelle de l'interface qui notamment distingue les concepts du niveau "utilisation" et ceux du niveau "fonctionnement".

L'inconvénient majeur d'une spécification strictement fonctionnelle telle que celle utilisée par MacIDA est de refléter dans l'interface utilisateur la structuration du composant fonctionnel du système : les schémas de données internes au système se trouvent directement exposés dans l'interface. Le soin est laissé à l'utilisateur d'adapter son modèle mental au modèle de réalisation du système... Une telle approche nous semble à l'opposé des pratiques ergonomiques axées sur l'utilisateur, visant à définir un modèle conceptuel d'un système en accord avec les représentations mentales, l'environnement de travail et les besoins de ce dernier.

3 SIROCO et le processus général de conception des IHMs

En concevant et en réalisant SIROCO, nous répondions à un double objectif : un outil de support au développement des IHMs axé sur des concepts de niveau d'abstraction élevé, et une recherche sur la transition entre conception et réalisation des interfaces. Nous proposons de replacer nos résultats dans le contexte général des méthodes et outils décrits aux chapitres II et III afin de déterminer l'apport et les potentialités de SIROCO relativement au processus général de conception des interfaces utilisateur.

3.1 SIROCO et la conception

Notre démarche repose sur une décomposition des choix de conception d'un système interactif en trois ensembles : les choix conceptuels, les choix relatifs au style d'interaction mis en œuvre, et les choix concernant les détails de présentation de l'interface. Cette décomposition déborde le cadre linguistique que proposent les méthodes globales rapportées en section 2.1, Chapitre II : si interaction et présentation portent sur les niveaux syntaxique et lexical, les choix conceptuels relèvent des niveaux sémantique et conceptuel, mais recouvrent également la partie la plus abstraite du niveau syntaxique du modèle linguistique.

3.1.1 Choix conceptuels

SIROCO propose un modèle de représentation des choix de la spécification conceptuelle. Ce modèle fait intervenir deux types de concepts : d'une part des éléments représentant la sémantique du système (Concepts de l'Application), et d'autre part des abstractions liées à l'organisation de l'interface du système, dans le contexte de l'utilisation des éléments sémantiques (Perspectives et Espaces de Travail). L'intérêt de cette double dimension est de différencier les éléments et contraintes d'un système selon qu'ils se rattachent au mode de fonctionnement de ce dernier, ou bien à l'utilisation qui en est proposée.

En ce qui concerne la représentation des données et opérations du système, le concepteur définira sous la forme de Concepts de l'Application les objets dont est porteur le système ; au travers de Perspectives, les différents modes de perception et d'utilisation de ces objets, dans le contexte des Espaces de Travail qui organiseront l'activité de l'utilisateur.

La logique des traitements s'exprime au travers des opérations de navigation entre Espaces de Travail, et des pré-conditions et post-actions définies sur les opérations des Concepts de l'Application et des Perspectives - ce double niveau de contraintes permet de faire la part entre les contraintes de fonctionnement et les contraintes liées à l'utilisation proposée dans l'interface.

Perspectives et Espaces de Travail sont des notions qui ne sont jamais explicitement représentées dans les méthodes de conception décrites au Chapitre II. La notion de Perspective peut être déduite des procédures de réalisation des tâches - bien souvent explicitées dans ces méthodes, en observant les éléments de données intervenant dans ces procédures. La notion d'Espace de Travail n'est pas représentée en dehors des notions de fenêtre ou zone d'affichage (cf les "display area" de CLG) offertes au niveau syntaxique de la description d'une interface, concernant l'organisation spatiale de la présentation de l'interface. Les choix de SIROCO dénotent une volonté d'abstraction visant à capter et à formaliser *la structure fondamentale d'une interface*. Outre une description abstraite de l'interface, cette structure permet de mettre en lumière un ensemble d'opérations de transition qui ne relèvent pas de la sémantique de l'application : les opérations de navigation entre Espaces de Travail[1]. Cette structure est la

charpente à partir de laquelle les tâches de spécification de la présentation et de la syntaxe du dialogue sont menées.

3.1.2 Choix de présentation et d'interaction

SIROCO adopte une approche simplificatrice quant à la spécification des aspects externes de l'interface : la marge décisionnelle du concepteur est limitée au choix de quelques paramètres fixant de façon générique le style de l'interaction et de la présentation. En dehors de ces paramètres, la représentation externe de l'interface résulte d'un habillage automatique des éléments de la représentation conceptuelle.

L'intérêt de la stratégie adoptée est de garantir l'homogénéité des interfaces, par le respect d'un guide de style étendu. Il est clair cependant que tous les aspects externes d'une interface ne peuvent être définis de façon pleinement satisfaisante par un automate. Nous distinguons les choix de conception selon qu'ils s'accommodent ou non de la démarche d'automatisation.

En ce qui concerne la présentation générale des fenêtres et des commandes, nous considérons notre approche comme légitime : cette présentation fait l'objet des règles les plus précises dans les guides de style - c'est sur la présentation des commandes que porte principalement la recherche de l'homogénéité entre interfaces. L'adoption de règles de présentation modulables par un système de paramétrage permet la spécification des choix de conception en "intension" plutôt qu'en "extension" ; outre la compacité de cette spécification, l'homogénéité des choix est naturellement garantie.

L'insuffisance de l'automatisation porte principalement sur la présentation des données, et sur les syntaxes d'activation des commandes. L'attribution automatique d'une présentation aux données figurant dans un Espace de Travail s'effectue en sélectionnant les techniques d'interaction (au sens du Chapitre III) générales les plus appropriées ; il en résulte des interfaces offrant formulaires et icônes. Ce type d'interface convient à un ensemble d'applications de présentation de données (notamment, l'application Messagerie qui illustrre les chapitres précédents). Dans le cas d'applications plus complexes, ou d'interfaces moins banalisées, en dehors d'un contexte de prototypage rapide, cet habillage automatique est insuffisant. La spécification de choix de présentation pourra alors s'effectuer à partir de la notion de Perspective.

De même, la syntaxe d'activation des commandes automatiquement définie est une syntaxe normalisée, qui référence la présentation de ces commandes (l'utilisateur doit sélectionner l'élément de menu correspondant à la commande désirée). Si l'introduction de séquences syntaxiques additionnelles peut être rendue automatique (notamment, les raccourcis clavier exploitant les initiales des noms de commandes), il est des syntaxes spécifiques qui sortent du champ d'action de l'automate. Ces syntaxes sont typiquement celles de la manipulation directe. Elles seront spécifiées à partir de la notion de Perspective.

---

1. Le modèle supporte d'autres opérations de transition, non représentées dans la version actuelle du langage de spécification. Ces extensions au langage seront évoquées en conclusion générale de cette thèse, lors de la définition des perspectives de nos travaux.

3.2 SIROCO et la transition vers le développement

SIROCO aborde la transition entre conception et développement[1] en exploitant le "paradigme" unificateur qu'est l'*objet*. La modélisation à objets s'applique à la représentation de concepts abstraits tout comme à la définition de modules de code. La transition conception/ développement peut alors se voir comme la traduction d'objets conceptuels en objets de programmation.

L'unification entre l'analyse d'un système et sa réalisation est un thème sous-jacent à de récents travaux portant sur les méthodes de conception à objets. Ces travaux (tels ceux rapportés dans [de Champeaux 91], ou surtout [Jacobson 87]) proposent des méthodes générales d'analyse et de conception des systèmes axées sur la notion d'objet ; l'objectif étant une réalisation orientée objet, ces travaux proposent des méthodes d'analyse également orientées objet[2], mieux adaptées que les méthodes d'analyse traditionnelles, et susceptibles de simplifier la transition entre l'analyse et la spécification interne d'un système. Cette transition ne peut être complètement explicite cependant car trop de paramètres entrent en jeu, qu'une méthode générale ne peut intégrer. Nos travaux se placent dans un contexte différent : il ne s'agit pas de traiter du problème général de la transition entre l'analyse d'un système et sa spécification interne, mais d'étudier la transition entre la spécification d'une interface utilisateur d'un type donné et sa réalisation. C'est cette spécificité qui, à notre sens, permet d'apporter une réponse plus précise à la question traitée.

Nous proposons d'aborder cette transition en exploitant pour la spécification interne du code d'une interface utilisateur un modèle d'architecture dont les composants sont définis relativement aux éléments de la modélisation SIROCO de cette interface. Ce modèle d'architecture "argumenté" étend un modèle général (présenté en section 3.3 du Chapitre II) défini dans le contexte d'outils de programmation (squelette d'application évoqué au Chapitre III), et validé par la mise en œuvre de plusieurs applications. Le modèle "argumenté" est lui-même validé par l'outil de génération de code SIROCO-Guide décrit au Chapitre V, puisque la structuration du code généré résulte de l'application des principes de ce modèle.

Les limites de cette approche sont principalement celles du modèle de description SIROCO, décrites à la section précédente : dans le cas de présentations et de syntaxes non normalisées, la mise en œuvre de l'interface pourra nécessiter l'introduction de composants supplémentaires au niveau des modules de code se rapportant aux Perspectives. Il sera alors nécessaire d'étendre la partie "présentation" du modèle pour l'adapter aux besoins spécifiques rencontrés.

3.3 Conclusion

SIROCO est un modèle de représentation conceptuelle d'une interface qui s'attache à la description des éléments sémantiques présentés, ainsi que de la structure fondamentale de cette interface. Si les aspects de présentation sont traités de façon simplificatrice, nous avons dans les sections précédentes montré qu'une véritable spécification externe peut être envisagée avec pour point d'ancrage les éléments de notre modèle. Au-delà de la représentation

1. Plus précisément, entre l'analyse d'un système et sa spécification interne.

2. Le lecteur intéressé trouvera dans [Normand 91] une comparaison entre les modèles de représentation adoptés dans les travaux précédemment cités et le modèle SIROCO.

conceptuelle, SIROCO possède des ramifications dans un modèle d'architecture logicielle et offre ainsi un support à la transition entre la conception d'une interface utilisateur et la spécification interne du code de réalisation de cette interface.

Le modèle SIROCO ne fournit pas une méthode, mais un modèle, un cadre de pensée pour la conception et le développement des interfaces utilisateur. Ce cadre est, à ce stade de nos travaux, incomplet. SIROCO en constitue à nos yeux le noyau. Le complément de ce noyau passe par la construction de techniques d'évaluation de l'interface, et, en tout premier lieu, par une liaison explicite avec une méthode d'analyse de la tâche. Ces deux objectifs seront plus largement décrits dans la conclusion générale de cette thèse, lors de l'énoncé des perspectives de nos travaux.

Conclusion générale

La contribution de la thèse

Ce travail contribue au domaine de l'ingénierie des interfaces homme-ordinateur sous trois formes : une analyse critique de l'existant mettant en lumière les forces et faiblesses des résultats de la recherche ; un travail de modélisation portant à la fois sur la représentation conceptuelle d'une interface et sur la structuration du code la réalisant ; la conception et la réalisation d'un outil de génération de code à partir de la spécification conceptuelle des interfaces utilisateur.

Nous avons tout d'abord mené un travail d'analyse et de synthèse de l'existant en matière d'aide à la conception et au développement des systèmes interactifs : méthodes et modèles d'une part (au Chapitre II), outils informatiques d'autre part (au Chapitre III) sont l'objet d'une analyse critique visant à en déterminer les caractéristiques, apports et faiblesses. Les principales conclusions de cette étude fournissent le point de départ et la motivation de cette thèse ; elles sont les suivantes :

- les modèles de conception des interfaces utilisateur sont coupés des modèles de réalisation (modèles d'architecture logicielle).
- les modèles de réalisation sont bien souvent trop généraux, et dénués de références pratiques permettant de guider leur utilisation.
- les outils d'aide au développement offrent la plupart du temps des services dont le niveau d'abstraction ne concerne que la représentation externe des interfaces, ignorant les aspects conceptuels de ces dernières.

Ce travail d'analyse débouche sur une démarche originale - puisqu'à notre connaissance non explorée : l'utilisation d'un modèle de représentation conceptuelle d'une interface utilisateur à la fois comme cadre de référence d'un modèle d'architecture, et comme entrée

d'un générateur de code réalisant l'interface. Cette démarche se traduit par une double contribution : d'une part, des modèles et un langage de spécification, et d'autre part un générateur de code.

Nous avons conçu un modèle de représentation conceptuelle des interfaces utilisateur mettant en œuvre une distinction entre dimension de fonctionnement et dimension d'utilisation : le modèle SIROCO. Ce modèle est muni d'un langage de spécification permettant la description d'une interface selon ses principes. La présentation du modèle et du langage SIROCO fait l'objet du Chapitre IV.

Nous avons défini un modèle d'architecture étendu utilisant le contenu d'une spécification SIROCO comme argument, apportant ainsi un support à la transition entre les tâches de conception et de développement d'une interface (cf Chapitre V).

Enfin, nous avons développé un prototype de générateur de code qui, à partir d'une spécification en langage SIROCO, génère automatiquement une présentation, et produit en langage Guide le code nécessaire à la réalisation de l'interface spécifiée. Cet outil de développement est présenté au Chapitre V. Un premier intérêt du générateur de code SIROCO-Guide est de valider notre modèle d'architecture étendu, puisque le code généré met ce modèle en application. Le second intérêt est bien sûr la réalisation automatique d'une interface. Au-delà du prototypage de système interactif, notre générateur est une fabrique d'objets Guide intégrables au système interactif final.

L'apport, les limitations et la spécificité des résultats de nos travaux ont été évalués au Chapitre VI. Le générateur SIROCO-Guide se distingue des outils de nature voisine (nommément, UIDE et MacIDA UIMS) principalement par le modèle de représentation des interfaces qu'il emploie - la spécificité du modèle SIROCO concerne la double dimension fonctionnement/utilisation. Au-delà du générateur SIROCO-Guide, l'ambition du modèle SIROCO est de dessiner un cadre de pensée pour la conception et le développement des interfaces utilisateur.

Perspectives de développement

L'évaluation effectuée au Chapitre VI laisse entrevoir pour nos travaux de multiples perspectives. Une fois encore, la dualité de nos travaux transparaît dans nos propositions : nous distinguons l'évolution de l'outil SIROCO-Guide et le prolongement de notre activité de modélisation.

Extensions de SIROCO-Guide

En dehors du complément des services réalisés par le prototype actuel, évoqués en section 1.1, Chapitre VI, plusieurs extensions peuvent être envisagées pour SIROCO-Guide ; l'objectif est, dans l'absolu, la constitution d'un atelier logiciel complet assurant la saisie d'une spécification SIROCO, l'adaptation de l'interface générée, et la gestion des versions de l'interface, en tenant notamment compte des éventuelles modifications du code par le programmeur.

Spécification interactive

L'écriture d'une spécification d'interface SIROCO est une tâche qui peut être malaisée si le seul outil dont dispose le concepteur est un simple éditeur de texte. Les problèmes auxquels se heurte le concepteur découlent de la grande modularité du modèle de représentation SIROCO : il est difficile de maintenir la cohésion d'un ensemble de modules, et d'assurer la cohérence des relations entre les modules.

La complexité engendrée par la modularité est un problème bien connu dans le domaine de la programmation à objets ; les réponses apportées dans ce domaine sont des outils de type "browser" permettant de visualiser les liens d'héritage et de référence entre objets. [Adar 91] par exemple offre un outil permettant la définition et la génération dynamique de vues partielles et sélectives sur les composants d'un programme à objets. Nous envisageons pour SIROCO un environnement de spécification interactive relevant de principes analogues.

Adaptation et gestion du produit de la génération :

Nous avons évoqué la modification par le développeur du code généré comme un recours pour pallier certaines insuffisances des présentations automatiques (cf section 3.1.1 du Chapitre V). Une modification directe n'est bien entendu pas souhaitable, car elle mettrait en péril l'intégrité du système. Le contrôle de ces modifications passe par un outil d'aide à l'adaptation des aspects externes de l'interface générée. A la manière d'outils tels MacIDA UIMS, nous envisageons pour SIROCO un outil d'édition interactive de la présentation résultant d'une génération automatique. Une étude approfondie est nécessaire afin de déterminer le champ d'action possible pour cet éditeur ; au-delà de la modification des détails de la présentation, de l'ajout de syntaxes d'activation des commandes, un sujet de recherche est la substitution de techniques d'interaction et, à un niveau de granularité moins fin, de gestionnaires de la présentation d'une Perspective.

Enfin, un outil de gestion de versions est nécessaire dans la perspective de la construction cyclique d'une interface, afin de gérer en parallèle les versions successives d'une spécification et les éventuelles modifications du code généré correspondant.

Prolongements au modèle SIROCO

Une première perspective, la plus immédiate, concerne le développement du modèle SIROCO lui-même : notre sentiment est que, dans l'état actuel de notre recherche, toutes les potentialités de nos choix de modélisations ne sont pas complètement exploitées. Nous souhaitons approfondir certains aspects du modèle, et enrichir le langage de spécification en conséquence. Cet approfondissement concerne notamment la notion de Perspective, que nous souhaitons compléter en introduisant celle de Perspective conditionnelle, et celle d'opération de transition entre Perspectives.

Au-delà d'un simple développement du modèle SIROCO existant, nous choisissons de mettre l'accent sur deux directions de recherche plus fondamentales : l'évaluation d'une spécification, et le rattachement de SIROCO à une méthode d'analyse des tâches.

Evaluation d'une interface :

En tant que formalisme, le langage SIROCO offre un support potentiel à l'évaluation d'une interface par l'application de fonctions de détection systématique d'anomalies ou de situations comportant un risque ergonomique. Restent à délimiter précisément les anomalies détectables de façon systématique ; la tâche est loin d'être simple. Deux directions sont à étudier : l'évaluation d'une interface "dans l'absolu", et l'évaluation d'une interface relativement à un arbre des tâches.

L'évaluation d'une interface "dans l'absolu", c'est-à-dire sans autres données que celles d'une spécification SIROCO nous semble a priori limitée à la détection de problèmes grossiers. Une telle évaluation repose sur la mesure de la complexité de navigation à travers les Espaces de Travail d'une interface, la complexité de navigation à l'intérieur d'un même Espace, la cohérence des Perspectives définies sur un même Concept, etc.

L'évaluation d'une interface relativement à un arbre des tâches permet la détection d'une bonne ou mauvaise couverture de l'ensemble des tâches de l'arbre, et des mesures de complexité de nature analogue à celles évoquées en section 2.3.1 du Chapitre II. Ce type d'évaluation nécessite de disposer d'un formalisme de description d'un arbre des tâches, et des méthodes de réalisation de ces tâches à l'intérieur de l'interface décrite en langage SIROCO. L'on rejoint ici la seconde direction de recherche mentionnée.

Prolongement vers un formalisme de représentation des tâches :

Lors de la description d'un mode d'emploi de SIROCO, en section 5 du Chapitre IV, nous soulignions le caractère délicat de l'étape du choix des tâches figurant dans un Espace de Travail, et l'absence de méthode permettant d'extraire ces dernières de l'arbre général des tâches. L'utilisation conjointe d'un outil de recueil et d'expression des données de description des tâches, tel MAD [Pierret-Golbreich 89], offre d'intéressantes perspectives, tant au niveau de l'élaboration de règles de passage de l'arbre des tâches à la spécification des Espaces de Travail que sur le plan de l'évaluation d'une spécification SIROCO relativement à l'arbre des tâches. Le prolongement de SIROCO vers un formalisme de représentation des tâches offre le précieux complément évoqué au Chapitre VI, dans la perspective d'un cadre de pensée pour la conception et le développement des interfaces utilisateur.

Annexe A

Grammaire du langage SIROCO

Conventions d'écriture des éléments de syntaxe :

- les terminaux sont notés en majuscules ou entre guillemets,
- les non-terminaux sont notés entre < et >,
- un nombre quelconque de x s'écrit x*,
- au moins un x s'écrit x+,
- au plus un x s'écrit [x],
- exactement un x s'écrit (x) ou x,
- l'alternative s'écrit |.

1 La spécification

<spécification> = <concept>*
<perspective>*
<workspace>+
< session >

2 Un Concept

<concept> = <def_concept>
| <restrict_concept>

<def_concept> = <en-tête_def_concept>
<propriété_concept>*
<attribut>*
<opération>*
<fin_concept>

<en-tête_def_concept> = CONCEPT <ident_concept>
[IS_A <ident_concept>] IS
<ident_concept> = <ident>
<ident> = <letter> (<letter> | <digit>| '_')*
<letter> = ('A'..'Z' | 'a'..'z')
<digit> = ('0'..'9')

<fin_concept> = END CONCEPT <ident_concept>

<restrict_concept> =<en-tête_restrict_concept>
<propriété_concept>*
<restrict_attribut>*
<restrict_opération>*
<fin_concept>

<en-tête_restrict_concept> = CONCEPT <ident_concept>
RESTRICTS <ident_concept> IS

2.1 Propriété d'un Concept

<propriété_concept> = <propriété_fixe_concept>
| <propriété_contextuelle_concept>

<propriété_fixe_concept> = <sem_prop>
| <cohérence_prop>

<sem_prop> = _SEM <sem_prop_val>* <end_mark>
<sem_prop_val> = <ident>
<end_mark> = ';'

<cohérence_prop> = _ATTR_CHECK <expression_Guide>
<expression_Guide> = G_BEGIN <G_expression[1]> G_END

1. <G_expression> est une expression en langage Guide (cf le manuel du langage Guide).

<propriété_contextuelle_concept> = <identification_prop>

<identification_prop> = <nom_prop>
| <description_courte_prop>
| <description_longue_prop>
<nom_prop> = _NAME "" <texte[1]> ""
<description_courte_prop> = _SHORT_DESCR "" <texte> ""
<description_longue_prop> = _HELP_INFO "" <texte> ""

2.2 Attribut d'un Concept

<attribut> = <entête_attribut> <propriété_attribut>*

<entête_attribut> = (ATTRIBUTE | R_ATTRIBUTE) <ident_attribut> <type>
<ident_attribut> = <ident>

<type> = <type_simple> | <type complexe>

<type_simple> = <type_prédéfini> | <constructeur> <type_prédéfini>
<type_prédéfini> = STRING | CHAR | INTEGER | BOOLEAN
<constructeur> = LIST OF

<type_complexe> = <ident_concept> | <constructeur> <ident_concept>

<restrict_attribut> = <en-tête_restrict_attribut> <propriété_attribut>*

<entête_restrict_attribut> = (ATTRIBUTE | R_ATTRIBUTE) <ident_attribut>

<propriété_attribut> = <propriété_fixe_attribut>
| <propriété_contextuelle_attribut>

<propriété_fixe_attribut> = <kind_prop_attribut>
| <sem_prop>
| <size_prop>
| <valeurs_possibles_prop>
| <composition_prop>

<kind_prop_attribut> = _KIND <kind_prop_attribut_val>* <end_mark>
<kind_prop_attribut_val> = KEY | KEY_PART | MAIN

<size_prop> = _SIZE <entier>

1. <texte> est une chaîne de caractères quelconques, différents de "".

<valeurs_possibles_prop> = _POSS_VALUES <expression_Guide> <end_mark>

<composition_prop> = _COMPOSITION <comp_prop_val>
<comp_prop_val> = REFERENCE | COMPONENT

<propriété_contextuelle_attribut> = <identification_prop>
| <valeur_initiale_prop>

<valeur_initiale_prop> = _INIT_VALUE <expression_Guide>

2.3 Opération d'un Concept

<opération> = <entête_opération>
<propriété_opération>*
<paramètre>*

<entête_opération> = OPERATION <ident_opération>
<ident_opération> = <ident>

<restrict_opération> = <en-tête_opération>
<propriété_opération>*
<restrict_paramètre>*

<propriété_opération> = <propriété_fixe_opération>
| <propriété_contextuelle_opération>

<propriété_fixe_opération> = <kind_prop_opération>
| <sem_prop>
| <valid-condition_prop>
| <pre-action_prop>
| <post-action_prop>
| <param_check_prop>

<kind_prop_opération> = _KIND <kind_prop_opération_val>* <end_mark>
<kind_prop_opération_val> = DESTROY | VALIDATE | EXTERN

<valid-condition_prop> = _VALID_COND <expression_Guide>

<pre_action_prop> = _PRE_ACTION <expression_Guide>

<post_action_prop> = _POST_ACTION <expression_Guide>

<param_check_prop> = _PARAM_CHECK <expression_Guide>

<propriété_contextuelle_opération> = <identification_prop>

<paramètre> = <entête_paramètre> <propriété_paramètre>*

<entête_paramètre> = <mode_passage_paramètre> <ident_paramètre> <type>
<mode_passage_paramètre> = IN | OUT | IN_OUT
<ident_paramètre> = <ident>

<restrict_paramètre> = <entête_restrict_paramètre> <propriété_paramètre>*

<entête_restrict_paramètre> = <mode_passage_paramètre> <ident_paramètre>

<propriété_paramètre> = <propriété_fixe_paramètre>
| <propriété_contextuelle_paramètre>

<propriété_fixe_paramètre> = <sem_prop>
| <size_prop>
| <valeurs_possibles_prop>

<propriété_contextuelle_paramètre> = <identification_prop>
| <valeur_initiale_prop>
| <lien_out_prop>

<lien_out_prop> = _LINK <expression_Guide>

3 Perspective

<perspective> = <en-tête_perspective>
< corps_perspective>
<fin_perspective>

<en-tête_perspective> = PERSPECTIVE <ident_perspective>
ON <concept_ident> IS
<ident_perspective> = <ident>

<corps_perspective> =<propriété_perspective>*
<attribut_ds_persp>*
<opération_ds_persp>*

<fin_perspective> = END PERSPECTIVE <ident_perspective>

3.1 Propriété d'une Perspective

<propriété_perspective> = <propriété_niveau_perspective>
| <propriété_contextuelle_concept>

<propriété_niveau_perspective> = <kind_prop_perspective>
| <sem_prop>
| <input_valid_mode_prop>

<kind_prop_perspective> = _KIND <kind_prop_perspective_val>* <end_mark>
<kind_prop_perspective_val> = PROTOTYPE | CONTAINER | SELECTABLE

<input_valid_mode_prop> = _INPUT_VALIDATION_MODE <input_mode_val>
<input_mode_val> = IMPLICIT | EXPLICIT

3.2 Attribut dans une Perspective

<attribut_ds_persp> = <entête_attribut_ds_persp>
<propriété_attribut_ds_persp>*

<entête_attribut_ds_persp> = (ATTRIBUTE | R_ATTRIBUTE) <ident_attribut>
[<vue_sur_concept>]

<vue_sur_concept> = WITH PERSPECTIVE <ident_perspective>
| WITH PERSPECTIVE <ident_perspective> IS
<corps_perspective>
<fin_perspective>

<propriété_attribut_ds_persp> = <propriété_attribut_niveau_perspective>

| <propriété_contextuelle_attribut>

<propriété_attribut_niveau_perspective> = <kind_prop_attribut>
| <sem_prop>
| <valeurs_possibles_prop>

3.3 Opération dans une Perspective

<opération_ds_persp> = <entête_opération_ds_persp>
<propriété_opération_ds_persp>*
<paramètre_ds_persp>*
<pré-action>*
<post-action>*

<entête_opération_ds_persp> = OPERATION <ident_opération>

<propriété_opération_ds_persp> = <propriété_op_niveau_perspective>
| <propriété_contextuelle_opération>

<propriété_op_niveau_perspective> = <kind_prop_opération>
| <sem_prop>
| <valid-condition_prop>
| <param_check_prop>

<paramètre_ds_persp> = <entête_paramètre_ds_persp>
<propriété_paramètre_ds_persp>*

<entête_paramètre_ds_persp> =<mode_passage_paramètre> <ident_paramètre

<propriété_paramètre_ds_persp> = <propriété_param_niveau_perspective>
|<propriété_contextuelle_paramètre>

<propriété_param_niveau_perspective> = <sem_prop>
| <valeurs_possibles_prop>

<pré-action> = PRE_ACTION <action>* END

<action> = <simple_action> | <ws_access_trigger> | <expression_Guide>

<simple_action> = NEW_INSTANCE
| RESET
| RESET_PROTO
| SCAN_INPUT

<ws_access_trigger> = WS_ACCESS TO <ident_workspace>
[<transmission_data_prop>]
[<relation_prop_accès_ws>]

<post-action> = <incond_post-action> | <cond_post-action>

<incond_post-action> = POST_ACTION <action>* END

<cond_post-action> = ON <condition> DO <action>* END

4 Espace de Travail

<workspace> = <en-tête_workspace>
<propriété_workspace>*
<variable_activité>*
<accès_workspace>*
<fin_workspace>

<en-tête_workspace> = WORKSPACE <ident_workspace> IS
<ident_workspace> = <ident>

<fin_workspace> = END WORKSPACE <ident_workspace>

4.1 Propriété d'un Espace de Travail

<propriété_workspace> = <identification_prop>
| <unicité_prop_workspace>
| <instances_prop>
| <sem_prop>
| <kind_prop_workspace>

<unicité_prop_workspace> = _UNIQUE_IDEM? <booléen>
<booléen> = TRUE | FALSE

<instances_prop> = _INSTANCES <instances_prop_val>
<instances_prop_val> = UNIQUE | MULTIPLE

<kind_prop_workspace> = _KIND <kind_prop_workspace_val>* <end_mark>
<kind_prop_workspace_val> = MAIN | EXIT_GATE

4.2 Variable d'un Espace de Travail

<variable_workspace> = <mode_initialisation> <ident_variable_ws>
<type_variable_ws> <vue_sur_variable_ws>

<mode_initialisation> = IN | GLOBAL | LOCAL

<ident_variable_ws> = <ident_variable_locale>
| <ident_session>.<ident_variable_globale>
<ident_variable_locale> = <ident>
<ident_variable_globale> = <ident>

<type_variable_ws> = <ident_concept>

<vue_sur_variable_ws> = <vue_sur_concept>

4.3 Opération de navigation d'un Espace de Travail

<accès_workspace> = <entête_accès_ws>
<propriété_accès_ws>*
<paramètre_accès_ws>*
<pré-action>*
<post-action>*

<entête_accès_ws> = WS_ACCESS <ident_accès_ws> TO <ident_ws>
<ident_accès_ws> = <ident>

<propriété_accès_ws> = <identification_prop>
| <sem_prop>
| <kind_prop_accès_ws>
| <transmission_data_prop>
| <relation_prop_accès_ws>

<kind_prop_accès_ws> = _KIND <kind_prop_accès_ws_val>* <end_mark>
<kind_prop_accès_ws_val> = <kind_prop_opération_val>

<transmission_data_prop> = _DATA_TRANSMISSION
(<ident_variable_locale> <expression_Guide>) * <end_mark>

<relation_prop_accès_ws> = _RELATION <relation_prop_accès_ws_val>*
<end_mark>

<relation_prop_accès_ws_val> = [SEQUENCE
| PARALLEL
| SUB_WORKSPACE
| SUSPEND]

<paramètre_accès_ws> = <paramètre>

5 Session

<session> = <en-tête_session>
<propriété_session>*
[<variables_globales>]
<workspaces_initial>
<fin_session>

<en-tête_session> = WORKSPACE <ident_session> IS
<ident_session> = <ident>

<fin_session> = END SESSION <ident_session>

5.1 Propriété d'une Session

<propriété_session> = <identification_prop>
| <sem_prop>

5.2 Données globales d'une Session

<variables_globales> = GLOBALS <variable_session>*

<variable_session> = <ident_variable_locale> <ident_concept>

5.3 Espaces de Travail initiaux d'une Session

<workspaces_initial> = WORKSPACES <workspace_session>*

<workspace_session> = <déclaration_workspace_session>
[<transmission_data>]

<déclaration_workspace_session> = <ident_variable_locale>
<ident_workspace>

<transmission_data> = WITH (<ident_variable_locale> <expression_Guide>)*
END

Annexe B

Le générateur de code SIROCO-Guide

Le générateur SIROCO-Guide est écrit en langage C, et utilise les outils Unix LEX et YACC. Sont présentés successivement la structure du générateur et son schéma d'exécution, et les caractéristiques du code généré.

1 Structure et schéma d'exécution du générateur

La structure du générateur SIROCO-Guide peut être déduite de son schéma d'exécution, à chaque étape de l'exécution correspondant un module du générateur.

Le schéma général d'exécution du générateur est représenté dans la Figure B-1. Il comporte trois grandes étapes : analyse grammaticale, contrôles de complétude et de cohérence, et phase de génération proprement dite.

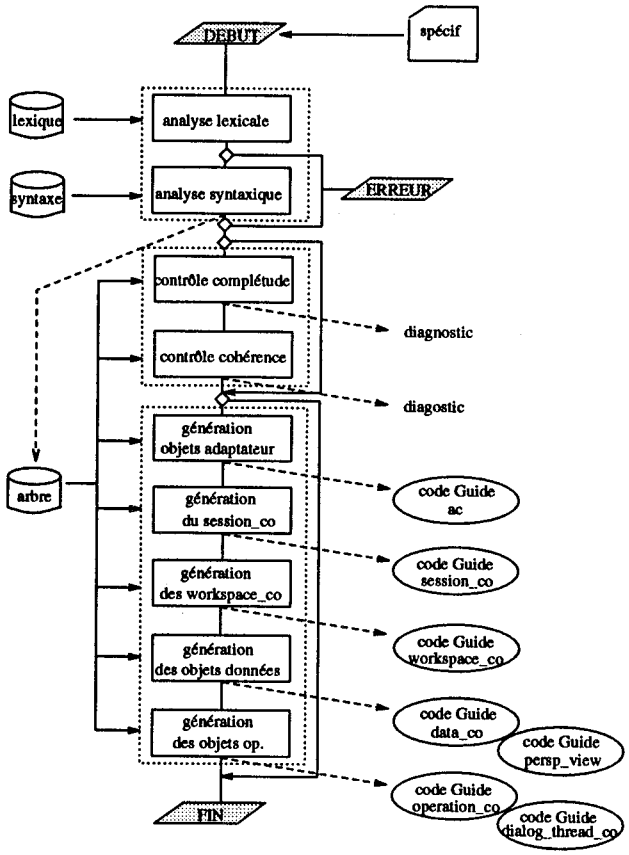


Figure B-1 *: schéma général d'exécution du générateur*

1.1 Analyse grammaticale

L'analyse grammaticale prend en entrée le fichier de spécification et produit une structure d'arbre, représentation interne des données de spécification. L'analyse grammaticale s'effectue en deux temps : l'*analyse lexicale* exploite le fichier de spécifications et produit un flot de "tokens" correspondant à des mots clefs du langage SIROCO, et de "tokens" n'appartenant pas au langage (chaînes de caractères) ; l'*analyse syntaxique* exploite le flot de "tokens" en

fonction des règles de la grammaire syntaxique du langage SIROCO, et construit l'arbre au fur et à mesure de l'analyse. La détection d'une erreur lexicale ou syntaxique met fin à l'exécution du générateur[1].

1.2 Contrôles

Selon les paramètres d'appel du générateur, des contrôles de complétude et de cohérence sont effectués ou non. *Contrôle de complétude* et *contrôle de cohérence* de la spécification sont menés en parcourant l'arbre de représentation interne ; ils produisent un diagnostic indiquant sous la forme d'avertissements d'éventuelles anomalies (cf 4.2, Chapitre IV).

1.3 Génération de code

Selon les paramètres d'appel du générateur, la génération de code proprement dite est effectuée ou bien l'exécution prend fin. La génération est menée en parcourant l'arbre de représentation interne ; elle se décompose en cinq étapes successives, correspondant à la génération des types et classes Guide (cf Chapitre V) des objets de l'adaptateur du composant fonctionnel, des objets de contrôle des Espaces : objet session_co et objets workspace_co, des objets relatifs aux données : data_co et persp_view, et enfin des objets de contrôle relatifs aux opérations : operation_co et dialog_thread_co.

La génération du type (éventuel) et de la classe d'un objet s'effectue en parallèle, type et classe étant en principe générés linéairement, en suivant les règles du langage Guide ; pour un type : entête de déclaration, variables visibles, signatures de méthodes, terminaison de la déclaration ; pour une classe : entête de déclaration, variables, méthodes et procédures, terminaison de la déclaration. Le principe de linéarité n'est pas toujours respecté cependant car il serait trop coûteux dans certains cas, nécessitant de multiples parcours de l'arbre à l'identique ; l'exécution du générateur met alors en œuvre un ou plusieurs des mécanismes suivants :

- génération parallèle de plusieurs morceaux de code, dans un jeu de tampons distincts.
- génération "avancée" d'un morceau de code dans un tampon, dont un sous-produit est une structure de données interne qui sera exploitée pour générer le code courant.

Sans détailler chacune des étapes de la génération, nous présentons à titre d'exemple l'étape de génération des objets operation_co. Chaque Espace donne lieu à la génération d'un objet operation_co. La Figure B-2 explicite le contenu partiel de la classe d'un objet operation_co en général, en numérotant chacun des blocs composant cette classe, dans l'ordre linéaire de leur disposition dans le fichier généré. La Figure B-3 représente le schéma d'exécution du module du générateur produisant un objet operation_co, en explicitant l'ordre de génération effective de chaque bloc de la classe.

1. La version actuelle de l'analyseur grammatical ne comporte pas de fonction de reprise après erreur ; en conséquence, l'exécution du générateur est interrompue dès qu'une erreur est détectée, en phase lexicale comme en phase syntaxique.

```
CLASS ... SUBCLASS OF OpCtrl IMPLEMENTS OpCtrl IS [1]

        <variables de contrôle>                          [2]
        <variables de travail>                           [3]
        <variables de gestion des dialog_thread_co>      [4]

        METHOD InitDisplay_MenuBar;                      [5]
              ... /* initialisation de menubar_view */
        END InitDisplay_MenuBar;

        METHOD CallOp(...);                              [6]
              ... /* appel effectif aux opérations simples */
        END CallOp;

        METHOD CallAA(...);                              [7]
              ... /* lancement transition vers un WS */
        END CallAA;

        METHOD Init(...);                                [8]
              ... /* initialisations internes */
        END Init;

        METHOD AskInvalid(...);                          [9]
              ... /* gestion validité des opérations */
        END AskInvalid;

        <procédures d'activation des dialog_thread_co>   [10]

        METHOD DtCtrlExit(...);                          [11]
              ... /* gestion terminaison des dialog_thread */
        END DtCtrlExit;

        METHOD SelRefModified(...);                      [12]
              ... /* action sur modification de la sélection
                     (gestion validité des opérations) */
         END SelRefModified;

        METHOD ManageAmbigOp(...);                       [13]
              ... /* mise en œuvre désambiguïsation */
        END ManageAmbigOp;

        ...

END ... .
```

Figure B-2 : *classe d'un objet operation_co*

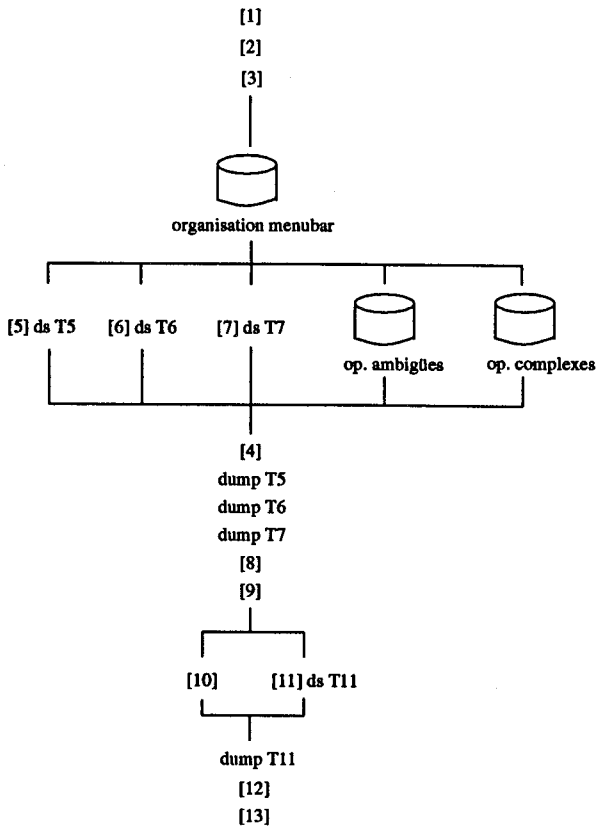


Figure B-3 *: schéma de génération d'un objet operation_co*

La version actuelle du prototype de SIROCO-Guide a été réalisée en privilégiant clarté et modularité des programmes au détriment éventuel des performances d'exécution du générateur : les parcours de l'arbre ne sont pas toujours optimisés, dans le sens où l'on ne cherche pas à tout prix à minimiser le nombre de parcours de chaque branche de l'arbre. Cet objectif de clarté est cohérent avec la notion de prototype ; les performances d'exécution étant acceptables, une ré-organisation optimisée du code ne nous semble pas indispensable.

2 Caractéristiques de la génération

La réalisation d'un générateur de code peut a priori mettre en œuvre deux grands types de politique : soit l'on fait générer à chaque fois l'ensemble du code répondant aux spécifications, soit l'on isole des composants généraux ou génériques, et l'on ne fait générer que la partie du code spécifique aux spécifications, le code généré réutilisant les composants isolés. Le choix de la réutilisation est dans notre cas immédiat, de par le volume de code nécessaire à la mise en œuvre d'une interface.

2.1 Techniques de réutilisation

Une fois le choix de la réutilisation effectué, se pose dans un environnement à objets le problème du choix entre deux grandes techniques de réutilisation : d'une part la technique consistant à sous-classer une classe existante, et d'autre part la technique de réutilisation par instanciation/paramétrage d'une classe existante. Dans le premier cas, la classe existante offre un ensemble de fonctions destinées à être héritées ; le plus souvent (cf les classes du squelette utilisées par héritage), elle n'a pas d'autonomie propre dans le sens où elle n'est pas destinée à être directement instanciée. Dans le second cas, la classe existante encapsule un ensemble de fonctions autonomes dont le comportement dépend de paramètres fournis à l'initialisation.

Le choix entre les deux techniques de réutilisation dépend de plusieurs facteurs, le principal concernant le type de réutilisation désiré : utilisation ou héritage de comportement. On fait ici la distinction entre l'utilisation de services par un objet et l'annexation de fonctions par un objet (avec particularisation ou non). Dans le premier cas, la réutilisation sert à la mise en œuvre du comportement de l'objet ; dans le second cas, la réutilisation augmente le comportement de l'objet. La réutilisation par instanciation signifie l'utilisation de services ; la réutilisation par héritage peut mettre en œuvre les deux types de réutilisation : les fonctions héritées sont soit simplement "annexées" par la sous-classe, et fournies dans l'interface de cette dernière (augmentation du comportement), soit appelées dans les méthodes définies dans la sous-classe (utilisation de services).

2.2 Générer ou réutiliser ?

La mise en œuvre d'un générateur de code nécessite de faire la part entre le code à générer et le code à factoriser et réutiliser. Cette distinction n'est pas toujours évidente ; trois facteurs l'influencent :

- facteur de complexité.
  La factorisation d'un comportement ne peut s'effectuer que dans des limites raisonnables de complexité de sa réutilisation (cf le système de paramètrage associé au code réutilisable).
- facteur de performance.
  Les performances à l'exécution de code généré répondant à un besoin précis sont bien entendu meilleures que celles d'un morceau de code générique paramétré.
- facteur de lisibilité.
  La réutilisation de code favorise la lisibilité des programmes générés - ne serait-ce qu'en raison de la limitation du volume de ces programmes.

Le facteur de complexité conduit à choisir la solution de génération lorsque l'algorithme à mettre en œuvre est trop "fin", trop dépendant des particularités de la spécification. Le facteur de performances guide les choix relatifs aux morceaux de code "sensibles", exécutés fréquemment ou bien requérant des temps de réponse courts. Le facteur de lisibilité est particulièrement important lorsque le code produit est susceptible d'être modifié par le programmeur.

2.3 Choix effectués pour SIROCO-Guide

Les choix effectués lors de la réalisation de SIROCO-Guide sont contrastés : si la réutilisation de code est un principe constant, elle est plus ou moins accentuée selon les classes du code généré.

Le code produit par le générateur SIROCO-Guide utilise un ensemble de bibliothèques schématisé en Figure B-4 :

- bibliothèques graphiques (Motif, Xt, Xlib)
- squelette d'application
- extension au squelette
- bibliothèque SIROCO

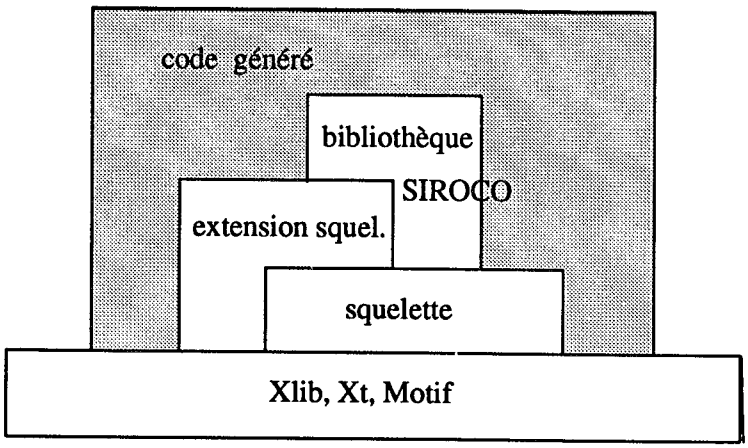


Figure B-4 : *code généré et code réutilisé*

La bibliothèque de base est l'ensemble des objets d'interfaçage avec les bibliothèques Motif, Xt et Xlib. L'utilisation s'effectue par instanciation.

Le squelette d'application développé lors de nos précédents travaux est au centre de la génération : d'une part un certain nombre de services du squelette sont directement utilisés par le code généré (notamment le presse-papier, la connexion au serveur graphique, les mécanismes de propagation des modifications des données de l'application) ; d'autre part, le squelette sert de plate-forme à deux ensembles d'objets développés pour les besoins de SIROCO-Guide : une extension au squelette, et une bibliothèque d'objets SIROCO. La réutilisation du squelette s'effectue par héritage pour les classes des objets du modèle d'architecture sous-jacent, et par instanciation pour un ensemble de classes réalisant des services.

La réalisation du générateur nous a amené à affiner le modèle de décomposition architecturale élaboré pour le squelette, notamment en ce qui concerne les objets de contrôle : des rôles bien précis ont été identifiés, qu'un outil général de programmation tel le squelette ne pouvait intégrer. D'où le développement d'un ensemble de types et de classes supports des objets de contrôle identifiés, cet ensemble constituant une couche d'extension du squelette original. La réutilisation s'effectue également par héritage et par instanciation.

Les objets de contrôle de l'interface (session_co, workspace_co, operation_co, data_co, dialog_thread_co) sont générés par sous-classage des classes de l'extension du squelette. Le code hérité est la plupart du temps minimal : héritage de variables, de mécanismes élémentaires, etc. La raison de cette réutilisation minimale repose sur les facteurs de complexité et de performance.

En ce qui concerne les objets de présentation de l'interface, la tactique est différente : l'on a privilégié la réutilisation de code, et ce pour deux raisons : d'une part le facteur de complexité (il est relativement simple d'isoler des fonctions de présentation réutilisables), et d'autre part le facteur de lisibilité (le code de présentation est la partie du code généré la plus susceptible d'être surchargée par le programmeur). Les objets de présentation des variables d'un Espace (data_view) sont générés par sous-classage d'une classe de l'extension du squelette. Un certain nombre de classes génériques ont été définies, réutilisables directement par instanciation, et paramétrables au moyen, par exemple, de structures d'information ; ces classes constituent la bibliothèque SIROCO, et mettent en œuvre la présentation de formulaires de paramètres, d'afficheurs de listes, de barres de menus, etc.

2.4 Quelques chiffres...

A titre indicatif, nous donnons ici quelques chiffres bruts caractérisant la taille des logiciels et spécifications se rapportant au générateur SIROCO-Guide.

Le prototype réalisé compte environ 20000 lignes de code source C.

Le squelette, son extension, et la bibliothèque SIROCO - c'est-à-dire l'ensemble du code réutilisable par le code généré - totalisent environ 15000 lignes de code source Guide.

La spécification de l'application Messagerie présentée au Chapitre IV compte 230 lignes en langage SIROCO. Le code généré pour cette application représente environ 4000 lignes de code source en langage Guide (non commenté).

Annexe C

Extrait du code généré pour le système Messagerie

A titre indicatif, nous fournissons dans cette Annexe un extrait du code produit par le générateur SIROCO-Guide pour le système de Messagerie spécifié au Chapitre IV, section 5. *Cet extrait correspond à la mise en œuvre des fonctions de l'Espace de Travail de traitement, HandleMess_WS, dont l'architecture est décrite en section 3.2.5 du Chapitre V* - le lecteur se référera à cette section.

Sont successivement fournis les objets :

- *HandleMess_WS_C :* l'objet de gestion de l'Espace de Travail considéré.
- *gl_0InHandleMess_WS_DC*: objet de contrôle gérant les données dépendant de l'unique attribut de l'Espace de Travail considéré.
- *HandleMess_WS_OC*: objet de contrôle mettant en œuvre les commandes de l'Espace de Travail.
- *HandleMess_WS_Read_DTC* : objet de contrôle de l'accès à un Espace de Travail Read_WS de lecture d'un message. (N.B. : l'objet de contrôle de l'accès à un Espace Reply_WS, analogue à l'objet HandleMess_WS_Read_DTC, n'est pas fourni).
- *User_P_PV*: objet de présentation des données contenues dans l'unique attribut de HandleMess_WS, et déduite de la Perspective User_P.

1 L'objet HandleMess_WS_C

```
TYPE HandleMess_WS_C SUBTYPE OF ActCtrl  IS
 gl_0 :  REF User = NIL;
 dc_gl_0 : REF DataCtrl;
 METHOD InitActCtrl(IN REF Mailer_sess_GC);
END HandleMess_WS_C.


CLASS HandleMess_WS_C SUBCLASS OF ActCtrl
 IMPLEMENTS HandleMess_WS_C IS

Mailer_sess : REF Mailer_sess_GC;

METHOD InitActCtrl(IN g : REF Mailer_sess_GC);
BEGIN
  Mailer_sess := g;
  SUPER.Init(g);
  SUPER.Init_internals("HandleMess_WS", "boite a lettres");
END InitActCtrl;

METHOD InitDisplay;
BEGIN
  opCtrl := HandleMess_WS_OC.New;
  opCtrl.Init(Mailer_sess, SELF, actView, selCtrl, helpCtrl);
  SUPER.InitDisplay;
  dc_gl_0 := gl_0InHandleMess_WS_DC.New;
 dc_gl_0.Init(Mailer_sess, SELF, actView, opCtrl, selCtrl, helpCtrl);
  SELF.AssessGlobalVars;
  actView.DisplayImage;
END InitDisplay;

METHOD AssessGlobalVars;
BEGIN
  IF gl_0 # Mailer_sess.user
  THEN
      gl_0 := Mailer_sess.user;
      dc_gl_0.ChangeVarValue(gl_0);
  END;
END AssessGlobalVars;

METHOD Activate;
BEGIN
  SELF.AssessGlobalVars;
  dc_gl_0.Activate;
  opCtrl.Activate;
  actView.DisplayImage;
 END Activate;

METHOD Exit : Integer;
BEGIN
   actView.UndisplayImage;
   Mailer_sess.ActExit(SELF);
   RETURN(SUCCESS);
END Exit;

END HandleMess_WS_C.
```

2 L'objet gl_0InHandleMess_WS_DC

```
TYPE User_DC SUBTYPE OF DataCtrl IS

var : REF User = NIL;
END User_DC.


CLASS gl_0InHandleMess_WS_DC SUBCLASS OF DataCtrl
  IMPLEMENTS User_DC IS

actCtrl : REF HandleMess_WS_C;

sess : REF Mailer_sess_GC;
  pv_User_P : REF User_P_PV;

METHOD Init(IN geneC : REF GeneralCtrl; actC : REF ActCtrl ; actV : REF ActView;opC : REF
OpCtrl; selC : REF SelCtrl; helpC : REF HelpCtrl);
BEGIN
  actCtrl := ASSERTYPE(actC);
  sess := ASSERTYPE(geneC);
  SUPER.Init(geneC, actC, actV, opC, selC, helpC);
END Init;

METHOD InitDisplay;
BEGIN
  pv_User_P := User_P_PV.New;
  pv_User_P.Init(actCtrl, SELF, map_data, var, "boite-a-lettre", actView);
  pv_User_P.DisplayImage;
END InitDisplay;

METHOD ChangeVarValue(IN v : REF Displayable);
BEGIN
  IF var # NIL THEN var.SubCOAccess(SELF); END;
  var := ASSERTYPE(v);
  IF var # NIL THEN var.AddCOAccess(SELF); END;
  pv_User_P.ChangeVarValue(var);
  SELF.DispDataModifed(var, "", NIL);

END ChangeVarValue;

METHOD DispDataModifed(IN c : REF Top; elem_id : String; d : REF Top);
concept_path : String[PATH_LENGTH];
concept, o : REF Displayable;
BEGIN
  concept := ASSERTYPE(c);
  IF concept = var
THEN
      IF elem_id = "name" OR elem_id = ""
      THEN
        pv_User_P.UpdateVarImage(ASSERTYPE(concept), "name", ASSERTYPE(d));
      END;
    IF elem_id = "mbox" OR elem_id = ""
    THEN
      map_data.GetOldConcept(concept, "mbox", o);
      IF var = NIL THEN
        IF o # NIL THEN
          IF o # NIL THEN o.SubCOAccess(SELF); END;
          map_data.InitNilConceptSon("", "MailBox_P", "mbox", NIL);
        END;
        pv_User_P.UpdEl_MailBox_P("mbox", NIL, "", ASSERTYPE(d));
      ELSE
        IF o # var.mbox THEN
          IF o # NIL THEN o.SubCOAccess(SELF); END;
          IF var.mbox # NIL THEN var.mbox.AddCOAccess(SELF); END;
          map_data.UpdateConceptSon(concept, "mbox", var.mbox);
        END;
        pv_User_P.UpdEl_MailBox_P("mbox", var.mbox, "", ASSERTYPE(d));
      END;
    END;
  RETURN;
END;

map_data.GetConceptPath(concept, concept_path);
WHILE (concept_path # "NULL") DO
  IF concept_path = "mbox"
  THENpv_User_P.UpdEl_MailBox_P(concept_path, ASSERTYPE(concept), elem_id , ASSERTYPE(d));
  END;
  IF concept_path = "mbox.messages"
```

```
  THENpv_User_P.UpdConstrEl_HeaderList_P(concept_path, ASSERTYPE(concept), elem_id ,
ASSERTYPE(d));
  END;
  map_data.GetConceptNextPath(concept_path);
END;
END DispDataModifed;

METHOD RetrieveUserModifs;
i, j, k, l, m, n : Integer ; values_cpt : Integer; values : Array[ATTRIBUTE_NB] OF REF
AttributeValue;
BEGIN

END RetrieveUserModifs;

METHOD ResetImage;
BEGIN
  SELF.DispDataModifed(var, "", NIL);

END ResetImage;

METHOD NewInstance;
BEGIN
  IF var # NIL THEN var.SubCOAccess(SELF); END;
  actCtrl.gl_0 := User.New;actCtrl.gl_0.Init; var := actCtrl.gl_0;
  var.AddCOAccess(SELF);
  map_data.InitNilConceptSon("", "User_P", "", var);
END NewInstance;

METHOD ExecuteOnDestroy;
BEGIN
  map_data.Destroy;
  pv_User_P.Destroy;
END ExecuteOnDestroy;
END gl_0InHandleMess_WS_DC.
```

3 L'objet HandleMess_WS_OC

```
CLASS HandleMess_WS_OC SUBCLASS OF OpCtrl
  IMPLEMENTS OpCtrl IS

Mailer_sess : REF Mailer_sess_GC;

actCtrl : REF HandleMess_WS_C;
mb_descr : Array[DESCR_LENGTH] OF Record
                                    ind : Integer;
                                    id : String[80];
                                    title : String[80];
                                    level : Integer;
                                    END;
wa_commands : Array[DESCR_LENGTH] OF Record
                                    ind : Integer;
                                    id : String[80];
                                    title : String[80];
                                    END;
param_descr : Array[DESCR_LENGTH] OF PARAM_DESCR;
da_Mess_AC : REF Mess_AC;

act_HandleMess_WS_Read_DTC : Array[MAX_NB_CONCUR_OP] OF REF DTCtrl;
act_HandleMess_WS_Read_DTC_cpt : Integer = -1;

non_act_HandleMess_WS_Read_DTC : Array[MAX_NB_CONCUR_OP] OF REF DTCtrl;
non_act_HandleMess_WS_Read_DTC_cpt : Integer = -1;

act_HandleMess_WS_Reply_DTC : Array[MAX_NB_CONCUR_OP] OF REF DTCtrl;
act_HandleMess_WS_Reply_DTC_cpt : Integer = -1;

non_act_HandleMess_WS_Reply_DTC : Array[MAX_NB_CONCUR_OP] OF REF DTCtrl;
non_act_HandleMess_WS_Reply_DTC_cpt : Integer = -1;

METHOD InitDisplay_MenuBar;
i : Integer = -1;
p_count : Integer;
BEGIN
 i := -1;
 data.AddGop(0, NON_AMBIG_OP, "internal_actCtrl", NIL, "SpecOpTp", VALID_ST);
 data.AddOp(0,0, NON_AMBIG_OP, "internal_actCtrl", "Exit", NIL, "", NO_PARAMS, VALID_ST);
 i := i + 1;
 mb_descr[i].ind := 0; mb_descr[i].title := "exit";
 mb_descr[i].id := "Exit"; mb_descr[i].level := 1;
 mbView.InitMenuBarItem(0,"fle", "fchier",  mb_descr, i+1);
 i := -1;
 data.AddGop(1, AMBIG_OP, "clipboard", NIL, "SpecOpTp", VALID_ST);
 data.AddOp(1, 1, AMBIG_OP, "clipboard", "CopyRequested", NIL, "SpecOpTp", NO_PARAMS,
VALID_ST);
 i := i + 1;
 mb_descr[i].ind := 1; mb_descr[i].title := "copier";
 mb_descr[i].id := "CopyRequested"; mb_descr[i].level := 1;
 data.AddOp(1, 2, AMBIG_OP, "clipboard", "CutRequested", NIL, "SpecOpTp", NO_PARAMS,
VALID_ST);
 i := i + 1;
 mb_descr[i].ind := 2; mb_descr[i].title := "couper";
 mb_descr[i].id := "CutRequested"; mb_descr[i].level := 1;
 data.AddOp(1, 3, AMBIG_OP, "clipboard", "PasteRequested", NIL, "SpecOpTp", NO_PARAMS,
VALID_ST);
 i := i + 1;
 mb_descr[i].ind := 3; mb_descr[i].title := "coller";
 mb_descr[i].id := "PasteRequested"; mb_descr[i].level := 1;
 mbView.InitMenuBarItem(1,"edit", "edition",  mb_descr, i+1);
 i := -1;
 data.AddGop(2, AMBIG_OP, "", NIL, "Mess_AC", VALID_ST);
 data.AddOp(2, 4, AMBIG_OP, "", "delete", NIL, "Mess_AC", NO_PARAMS, VALID_ST);
 i := i + 1;
 mb_descr[i].ind := 4; mb_descr[i].title := "destruction";
 mb_descr[i].id := "delete"; mb_descr[i].level := 1;
 mbView.InitMenuBarItem(2,"gop2", "message",  mb_descr, i+1);
 i := -1;
 data.AddGop(3, NON_AMBIG_OP, "", NIL, "SpecOpTp", VALID_ST);
 data.AddActAccess(5, "Read", NO_PARAMS, VALID_ST);
 i := i + 1;
 mb_descr[i].ind := 5; mb_descr[i].title := "Lire";
 mb_descr[i].id := "Read"; mb_descr[i].level := 1;
 data.AddActAccess(6, "Reply", NO_PARAMS, VALID_ST);
 i := i + 1;
 mb_descr[i].ind := 6; mb_descr[i].title := "Repondre";
 mb_descr[i].id := "Reply"; mb_descr[i].level := 1;
```

```
 data.AddActAccess(7, "Write", NO_PARAMS, VALID_ST);
 i := i + 1;
 mb_descr[i].ind := 7; mb_descr[i].title := "Ecrire";
 mb_descr[i].id := "Write"; mb_descr[i].level := 1;
 mbView.InitMenuBarItem(3,"activites", "services",  mb_descr, i+1);
 i := -1;
 data.AddGop(4, NON_AMBIG_OP, "helpCtrl", NIL, "SpecOpTp", VALID_ST);
 data.AddOp(4,8, NON_AMBIG_OP, "helpCtrl", "GeneHelp", NIL, "", NO_PARAMS, VALID_ST);
 i := i + 1;
 mb_descr[i].ind := 8; mb_descr[i].title := "aide generale";
 mb_descr[i].id := "GeneHelp"; mb_descr[i].level := 1;
 data.AddOp(4, 9, AMBIG_OP, "helpCtrl", "SelHelp", NIL, "SpecOpTp", NO_PARAMS, VALID_ST);
 i := i + 1;
 mb_descr[i].ind := 9; mb_descr[i].title := "aide sur la selection";
 mb_descr[i].id := "SelHelp"; mb_descr[i].level := 1;
 mbView.InitMenuBarItem(4,"help", "aide",  mb_descr, i+1);

END InitDisplay_MenuBar;


METHOD CallOp(IN path : String ; id : String; values : Array OF REF DataItemValue);
result : Integer;
BEGIN
 IF (path = "internal_actCtrl") AND (id = "Exit") THEN   result := internal_actCtrl.Exit ;
END;
 IF (path = "clipboard") AND (id = "CopyRequested") THEN   result :=
clipboard.CopyRequested(actView) ; END;
 IF (path = "clipboard") AND (id = "CutRequested") THEN   result :=
clipboard.CutRequested(actView) ; END;
 IF (path = "clipboard") AND (id = "PasteRequested") THEN   result :=
clipboard.PasteRequested(actView) ; END;
 IF (path = "da_Mess_AC") AND (id = "delete") THEN   result := da_Mess_AC.delete;  END;
 IF (path = "helpCtrl") AND (id = "GeneHelp") THEN   result := helpCtrl.GeneHelp ; END;
 IF (path = "helpCtrl") AND (id = "SelHelp") THEN   result := helpCtrl.SelHelp ; END;
END CallOp;

METHOD CallAA(IN id : String; values : Array OF REF DataItemValue);
props : ACT_INST_PROP;
BEGIN
 IF id = "Read"
 THEN
      Activate_HandleMess_WS_Read_DTC;
 END;
 IF id = "Reply"
 THEN
      Activate_HandleMess_WS_Reply_DTC;
 END;
 IF id = "Write"
 THEN
   props.is_main := FALSE;
   props.is_exit_gate := FALSE;
   props.relation := PARALLEL_RELATION;
   props.is_replaced := FALSE;
   Mailer_sess.ActivateAct_Write_WS(props, actCtrl);
  END;
END CallAA;


METHOD Init(IN geneC:REF GeneralCtrl;actC:REF ActCtrl;actV:REF ActView;selC:REF
SelCtrl;helpC:REF HelpCtrl);
BEGIN
  Mailer_sess := ASSERTYPE(geneC);
  actCtrl := ASSERTYPE(actC);
    da_Mess_AC := Mess_AC.New;

  SUPER.Init(geneC,actC,actV,selC,helpC);
  clipboard.AddCOAccess(SELF);
END Init;

METHOD AskInvalid(IN gop_ind : Integer);
BEGIN
  IF gop_ind = 1
  THEN
    IF sel_ref = NIL
    THEN
      IF (data.PutOpLocalStatus(1, INVALID_ST)) THEN mbView.UpdateOpStatus(1,
data.GetOpStatus(1)); END;
    ELSE
      IF NOT sel_ref.Conform("Top")
      THEN
```

```
      IF (data.PutOpLocalStatus(1, INVALID_ST)) THEN mbView.UpdateOpStatus(1,
data.GetOpStatus(1)); END;
      ELSE
      IF (data.PutOpLocalStatus(1, VALID_ST)) THEN mbView.UpdateOpStatus(1,
data.GetOpStatus(1)); END;
      END;
    END;
    IF sel_ref = NIL
    THEN
      IF (data.PutOpLocalStatus(2, INVALID_ST)) THEN mbView.UpdateOpStatus(2,
data.GetOpStatus(2)); END;
    ELSE
      IF NOT sel_ref.Conform("Top")
      THEN
      IF (data.PutOpLocalStatus(2, INVALID_ST)) THEN mbView.UpdateOpStatus(2,
data.GetOpStatus(2)); END;
      ELSE
      IF SELF.CutRequested_CheckValid THEN
        IF (data.PutOpLocalStatus(2, VALID_ST)) THEN mbView.UpdateOpStatus(2,
data.GetOpStatus(2)); END;
      ELSE
        IF (data.PutOpLocalStatus(2, INVALID_ST)) THEN mbView.UpdateOpStatus(2,
data.GetOpStatus(2)); END;
      END;
      END;
    END;
    IF sel_ref = NIL
    THEN
      IF (data.PutOpLocalStatus(3, INVALID_ST)) THEN mbView.UpdateOpStatus(3,
data.GetOpStatus(3)); END;
    ELSE
      IF NOT sel_ref.Conform("Top")
      THEN
      IF (data.PutOpLocalStatus(3, INVALID_ST)) THEN mbView.UpdateOpStatus(3,
data.GetOpStatus(3)); END;
      ELSE
      IF SELF.PasteRequested_CheckValid THEN
        IF (data.PutOpLocalStatus(3, VALID_ST)) THEN mbView.UpdateOpStatus(3,
data.GetOpStatus(3)); END;
      ELSE
        IF (data.PutOpLocalStatus(3, INVALID_ST)) THEN mbView.UpdateOpStatus(3,
data.GetOpStatus(3)); END;
      END;
      END;
    END;
  END;
  IF gop_ind = 2
  THEN
    IF sel_ref = NIL
    THEN
      IF (data.PutOpLocalStatus(4, INVALID_ST)) THEN mbView.UpdateOpStatus(4,
data.GetOpStatus(4)); END;
    ELSE
      IF NOT sel_ref.Conform("Mess_AC")
      THEN
      IF (data.PutOpLocalStatus(4, INVALID_ST)) THEN mbView.UpdateOpStatus(4,
data.GetOpStatus(4)); END;
      ELSE
      IF (data.PutOpLocalStatus(4, VALID_ST)) THEN mbView.UpdateOpStatus(4,
data.GetOpStatus(4)); END;
      END;
    END;
  END;
  IF gop_ind = 4
  THEN
    IF sel_ref = NIL
    THEN
      IF (data.PutOpLocalStatus(9, INVALID_ST)) THEN mbView.UpdateOpStatus(9,
data.GetOpStatus(9)); END;
    ELSE
      IF NOT sel_ref.Conform("Top")
      THEN
      IF (data.PutOpLocalStatus(9, INVALID_ST)) THEN mbView.UpdateOpStatus(9,
data.GetOpStatus(9)); END;
      ELSE
      IF (data.PutOpLocalStatus(9, VALID_ST)) THEN mbView.UpdateOpStatus(9,
data.GetOpStatus(9)); END;
      END;
    END;
  END;
```

```
END AskInvalid;

PROCEDURE Activate_HandleMess_WS_Read_DTC;
BEGIN
  IF non_act_HandleMess_WS_Read_DTC_cpt > -1
  THEN
    act_HandleMess_WS_Read_DTC_cpt := act_HandleMess_WS_Read_DTC_cpt + 1;
    act_HandleMess_WS_Read_DTC[act_HandleMess_WS_Read_DTC_cpt] :=
non_act_HandleMess_WS_Read_DTC[non_act_HandleMess_WS_Read_DTC_cpt];
    non_act_HandleMess_WS_Read_DTC_cpt := non_act_HandleMess_WS_Read_DTC_cpt - 1;
  ELSE
  act_HandleMess_WS_Read_DTC[act_HandleMess_WS_Read_DTC_cpt] := HandleMess_WS_Read_DTC.New;
  act_HandleMess_WS_Read_DTC[act_HandleMess_WS_Read_DTC_cpt].Init(Mailer_sess, actCtrl,
SELF, actView, selCtrl, helpCtrl, data, ddView);
  END;
  act_HandleMess_WS_Read_DTC[act_HandleMess_WS_Read_DTC_cpt].Activate;
END Activate_HandleMess_WS_Read_DTC;

PROCEDURE Activate_HandleMess_WS_Reply_DTC;
BEGIN
  IF non_act_HandleMess_WS_Reply_DTC_cpt > -1
  THEN
    act_HandleMess_WS_Reply_DTC_cpt := act_HandleMess_WS_Reply_DTC_cpt + 1;
    act_HandleMess_WS_Reply_DTC[act_HandleMess_WS_Reply_DTC_cpt] :=
non_act_HandleMess_WS_Reply_DTC[non_act_HandleMess_WS_Reply_DTC_cpt];
    non_act_HandleMess_WS_Reply_DTC_cpt := non_act_HandleMess_WS_Reply_DTC_cpt - 1;
  ELSE
  act_HandleMess_WS_Reply_DTC[act_HandleMess_WS_Reply_DTC_cpt] :=
HandleMess_WS_Reply_DTC.New;
  act_HandleMess_WS_Reply_DTC[act_HandleMess_WS_Reply_DTC_cpt].Init(Mailer_sess, actCtrl,
SELF, actView, selCtrl, helpCtrl, data, ddView);
  END;
  act_HandleMess_WS_Reply_DTC[act_HandleMess_WS_Reply_DTC_cpt].Activate;
END Activate_HandleMess_WS_Reply_DTC;

METHOD DtCtrlExit(IN dtCtrl : REF DTCtrl);
i, j : Integer;
BEGIN
  IF dtCtrl.ObjectTypeName = "HandleMess_WS_Read_DTC"
  THEN
    i := 0;
    WHILE i<= act_HandleMess_WS_Read_DTC_cpt DO
      IF act_HandleMess_WS_Read_DTC[i] = dtCtrl
      THEN
        non_act_HandleMess_WS_Read_DTC_cpt := non_act_HandleMess_WS_Read_DTC_cpt + 1;
        non_act_HandleMess_WS_Read_DTC[non_act_HandleMess_WS_Read_DTC_cpt] :=
act_HandleMess_WS_Read_DTC[act_HandleMess_WS_Read_DTC_cpt];
        FOR j := i+1 TO act_HandleMess_WS_Read_DTC_cpt DO
          act_HandleMess_WS_Read_DTC[j - 1] := act_HandleMess_WS_Read_DTC[j];
        END;
        act_HandleMess_WS_Read_DTC_cpt := act_HandleMess_WS_Read_DTC_cpt - 1;
        RETURN;
      END;
    END;
    RETURN;
  END;
  IF dtCtrl.ObjectTypeName = "HandleMess_WS_Reply_DTC"
  THEN
    i := 0;
    WHILE i<= act_HandleMess_WS_Reply_DTC_cpt DO
      IF act_HandleMess_WS_Reply_DTC[i] = dtCtrl
      THEN
        non_act_HandleMess_WS_Reply_DTC_cpt := non_act_HandleMess_WS_Reply_DTC_cpt + 1;
        non_act_HandleMess_WS_Reply_DTC[non_act_HandleMess_WS_Reply_DTC_cpt] :=
act_HandleMess_WS_Reply_DTC[act_HandleMess_WS_Reply_DTC_cpt];
        FOR j := i+1 TO act_HandleMess_WS_Reply_DTC_cpt DO
          act_HandleMess_WS_Reply_DTC[j - 1] := act_HandleMess_WS_Reply_DTC[j];
        END;
        act_HandleMess_WS_Reply_DTC_cpt := act_HandleMess_WS_Reply_DTC_cpt - 1;
        RETURN;
      END;
    END;
    RETURN;
  END;
END DtCtrlExit;


METHOD SelRefModifed(IN r : REF Top; tp_ch : Boolean);
BEGIN
  sel_ref := r;
  IF r = NIL THEN ResetAmbigEls; RETURN; END;
```

```
  IF tp_ch
  THEN
    IF r.Conform("Top")
    THEN
    IF (data.PutGopLocalStatus(1, VALID_ST)) THEN mbView.UpdateGOpStatus(1,
data.GetGopStatus(1)); END;
    ELSE
    IF (data.PutGopLocalStatus(1, INVALID_ST)) THEN mbView.UpdateGOpStatus(1,
data.GetGopStatus(1)); END;
    END;
    IF r.Conform("Mess_AC")
    THEN
    IF (data.PutGopLocalStatus(2, VALID_ST)) THEN mbView.UpdateGOpStatus(2,
data.GetGopStatus(2)); END;
    ELSE
    IF (data.PutGopLocalStatus(2, INVALID_ST)) THEN mbView.UpdateGOpStatus(2,
data.GetGopStatus(2)); END;
    END;
  END;
END SelRefModifed;

PROCEDURE ResetAmbigEls;
BEGIN
  IF (data.PutGopLocalStatus(1, INVALID_ST)) THEN mbView.UpdateGOpStatus(1,
data.GetGopStatus(1)); END;
  IF (data.PutGopLocalStatus(2, INVALID_ST)) THEN mbView.UpdateGOpStatus(2,
data.GetGopStatus(2)); END;
END ResetAmbigEls;

METHOD ManageAmbigOp(IN op_descr : REF OpTableEntry);
BEGIN
 IF (op_descr.obj_type = "SpecOpTp") AND (op_descr.id = "CopyRequested") THEN
SELF.CallOp(op_descr.path, op_descr.id, NIL); RETURN;
  END;
 IF (op_descr.obj_type = "SpecOpTp") AND (op_descr.id = "CutRequested") THEN
SELF.CallOp(op_descr.path, op_descr.id, NIL); RETURN;
  END;
 IF (op_descr.obj_type = "SpecOpTp") AND (op_descr.id = "PasteRequested") THEN
SELF.CallOp(op_descr.path, op_descr.id, NIL); RETURN;
  END;
 IF (op_descr.obj_type = "Mess_AC") THEN    da_Mess_AC := ASSERTYPE(sel_ref);
    SELF.CallOp("da_Mess_AC", op_descr.id, NIL); RETURN;
  END;
 IF (op_descr.obj_type = "SpecOpTp") AND (op_descr.id = "SelHelp") THEN
SELF.CallOp(op_descr.path, op_descr.id, NIL); RETURN;
  END;
END ManageAmbigOp;
END HandleMess_WS_OC.
```

4 L'objet HandleMess_WS_Read_DTC

```
CLASS HandleMess_WS_Read_DTC SUBCLASS OF DTCtrl
  IMPLEMENTS AppliChannel IS

Mailer_sess : REF Mailer_sess_GC;

actCtrl : REF HandleMess_WS_C;
desigP0 : REF Mess_AC;
desigPind : Integer = -1;

METHOD Init(IN geneC:REF GeneralCtrl;actC:REF ActCtrl; opC : REF OpCtrl; actV:REF
ActView;selC:REF SelCtrl;helpC:REF HelpCtrl; d : REF OpCtrlData; ddV : REF DDView);
BEGIN
  Mailer_sess := ASSERTYPE(geneC);
  actCtrl := ASSERTYPE(actC);
  op_id := "Read"; title := "Lire"; object_name := "";
  SUPER.Init(geneC,actC,opC,actV,selC,helpC, d, ddV);
  END Init;

METHOD CallOp;
result : Integer;
props : ACT_INST_PROP;
BEGIN
   props.is_main := FALSE;
   props.is_exit_gate := FALSE;
   props.relation := PARALLEL_RELATION;
   props.is_replaced := FALSE;
  Mailer_sess.ActivateAct_Read_WS(props, actCtrl,ASSERTYPE(desigP0));
  SELF.Exit;
END CallOp;

METHOD SelRefModifed(IN r : REF Top; tp_ch : Boolean);
BEGIN
  sel_ref := r;
  IF tp_ch
  THEN
      IF r = NIL THEN SELF.RequestRefPDesig; RETURN; END;
      IF desigPind = 0
      THEN
      IF NOT r.Conform("Mess_AC") THEN SELF.RequestCorrectPDesig; RETURN; END;
      desigP0 := ASSERTYPE(r);
      ddView.DismissM(desambig_mess_ind);
      SELF.RequestDesigParams;  RETURN;
      END;
  END;
END SelRefModifed;

METHOD Activate;
BEGIN
  clipboard.AddCOAccess(SELF);
  IF sel_ref # NIL THEN IF sel_ref.Conform("Mess_AC") THEN desigP0 :=
ASSERTYPE(sel_ref);desigPind := 0; SELF.RequestDesigParams; RETURN; END;
 END;
  desigPind := -1;
  SELF.RequestDesigParams;
END Activate;

METHOD RequestDesigParams;
BEGIN
  desigPind := desigPind + 1;
  IF desigPind = 0
  THEN
  param_name := "message";
  param_object_name := "message";
  SELF.RequestPDesig;
  RETURN;
  END;
   desigPind := -1;
  SELF.CallOp;
END RequestDesigParams;

METHOD Exit;
BEGIN
sel_ref := NIL;
  clipboard.SubCOAccess(SELF);
  opCtrl.DtCtrlExit(SELF);
END Exit;
END HandleMess_WS_Read_DTC.
```

5 L'objet User_P_PV

```
TYPE User_P_PV SUBTYPE OF PerspView IS

METHOD CrEl_MailBox_P(IN englob_cont : REF Widget_X; f_concept_ind : Integer ; path :
String);

METHOD UpdEl_MailBox_P(IN path : String; concept : REF MailBox_AC; elem_id : String ; d :
REF ModifDescr);

METHOD CrConstrEl_HeaderList_P(IN englob_cont : REF Widget_X; f_concept_ind : Integer ;
path : String);

METHOD UpdConstrEl_HeaderList_P(IN path : String; concept : REF DisplayableList; elem_id :
String; d : REF ModifDescr);
END User_P_PV.


CLASS User_P_PV SUBCLASS OF PerspView
  IMPLEMENTS User_P_PV IS

persp_id : String[ID_LENGTH] = "User_P";
Mailer_sess : REF Mailer_sess_GC;
actCtrl : REF ActCtrl;
dataCtrl : REF User_DC = NIL;
var : REF User_AC = NIL;

data_item_descr : REF DataItemDescr = NIL;
list_data_item_descr : REF ListDataItemDescr = NIL;

METHOD Init(IN actC : REF ActCtrl; perspC : REF DataCtrl; data : REF PerspViewData; v :REF
Displayable ; name : String[80]; fv : REF View);
BEGIN
 dataCtrl := ASSERTYPE(perspC); actCtrl := ASSERTYPE(actC); var := ASSERTYPE(v);
  data_item_descr := DataItemDescr.New; list_data_item_descr := ListDataItemDescr.New;
  SUPER.Init(actC, perspC, data, v, name, fv);
END Init;

METHOD CreateImage;
cont : REF Widget_X;
desig_w : REF Xm_Widget; w : REF Widget_X;
concept_ind : Integer;
BEGIN
  SUPER.CreateImage;
  cont := container;
  w := NIL;
  desig_w := NIL;
  concept_ind := map_data.AddConcept(-1, var, "", persp_id, CONCEPT_TP, VALID_ST, NIL, w,
desig_w, -1);
  data_item_descr.id := "name";
  data_item_descr.title := "nom";
  data_item_descr.create_label := TRUE;
  data_item_descr.visible_box := TRUE;
  data_item_descr.label_length := 16;
  data_item_descr.size := 12;
  data_item_descr.tp := STRING_TP;
  data_item_descr.is_initialized := FALSE;
  data_item_descr.is_modifable := FALSE;
  data_item_descr.has_value_limits := FALSE;
  data_item_descr.poss_nb := 0;
  w := SUPER.CreateStringField(cont, data_item_descr);
  map_data.AddData(concept_ind, "name", STRING_TP, VALID_ST, w, -1);
SELF.CrEl_MailBox_P(cont, concept_ind, "mbox");
END CreateImage;

METHOD CrEl_MailBox_P(IN englob_cont : REF Widget_X; f_concept_ind : Integer ; path :
String);
cont : REF Widget_X;desig_w : REF Xm_Widget; w : REF Widget_X; concept_ind : Integer;
local_persp_id : String[ID_LENGTH] = "MailBox_P";tmpw0, tmpw1 : REF Widget_X;
BEGIN
  args_cpt := 0; SELF.SetArgsForSon(englob_cont);
  hci.XtSetArg(args[args_cpt], XmNbottomAttachment, XmATTACH_FORM);
  args_cpt := args_cpt + 1;
  tmpw0 := englob_cont.XmCreateForm("mbox", args, args_cpt);
  SELF.AddFormSon(englob_cont, tmpw0);
  hci.XtSetArgXmString(args[0], XmNlabelString, xmutil.XmStringCreate("boite a lettre",
XmSTRING_DEFAULT_CHARSET));
  hci.XtSetArg(args[1], XmNwidth, 16 * CHAR_WIDTH);
  hci.XtSetArg(args[2], XmNleftAttachment, XmATTACH_FORM);
```

```
  hci.XtSetArg(args[3], XmNmarginTop, 7);
  tmpw1 := tmpw0.XmCreateLabelGadget("mbox_label", args, 4);
  xmutil.XtManageChild(tmpw1);
  hci.XtSetArg(args[0], XmNleftAttachment, XmATTACH_WIDGET);
  hci.XtSetArg(args[1], XmNleftWidget, tmpw1.widget);
  hci.XtSetArg(args[2], XmNtopAttachment, XmATTACH_FORM);
  hci.XtSetArg(args[3], XmNrightAttachment, XmATTACH_FORM);
  hci.XtSetArg(args[4], XmNbottomAttachment, XmATTACH_FORM);
  cont := tmpw0.XmCreateForm("mbox_frame_rc", args, 5);
  xmutil.XtManageChild(cont);
  xmutil.XtManageChild(tmpw0);
  w := NIL;
  desig_w := NIL;
  concept_ind := map_data.AddConcept(f_concept_ind, NIL, path, local_persp_id, CONCEPT_TP,
VALID_ST, NIL, w, desig_w, -1);
SELF.CrConstrEl_HeaderList_P(cont, concept_ind, path + "." + "messages");
END CrEl_MailBox_P;

METHOD CrConstrEl_HeaderList_P(IN englob_cont : REF Widget_X; f_concept_ind : Integer ;
path : String);
cont : REF Widget_X;desig_w : REF Xm_Widget; w : REF Widget_X; concept_ind : Integer; v :
REF ListView;
local_persp_id : String[ID_LENGTH] = "HeaderList_P";
BEGIN
  desig_w := NIL;
  concept_ind := map_data.AddConcept(f_concept_ind, NIL, path, local_persp_id, CONCEPT_TP,
VALID_ST, cont, NIL, desig_w, -1);
  list_data_item_descr.id := "messages";
  list_data_item_descr.title := "messages";
  list_data_item_descr.create_label := FALSE;
  list_data_item_descr.desig := 1;
  list_data_item_descr.visible_elem_nb := 10;
  list_data_item_descr.item[0] := DataItemDescr.New;
  list_data_item_descr.item[0].id := "name";
  list_data_item_descr.item[0].title := "nom";
  list_data_item_descr.item[0].create_label := FALSE;
  list_data_item_descr.item[0].visible_box := TRUE;
  list_data_item_descr.item[0].label_length := 0;
  list_data_item_descr.item[0].size := 12;
  list_data_item_descr.item[0].tp := STRING_TP;
  list_data_item_descr.item[0].is_initialized := FALSE;
  list_data_item_descr.item[0].is_modifable := FALSE;
  list_data_item_descr.item[0].has_value_limits := FALSE;
  list_data_item_descr.item[0].poss_nb := 0;
  list_data_item_descr.item[1] := DataItemDescr.New;
  list_data_item_descr.item[1].id := "subject";
  list_data_item_descr.item[1].title := "sujet";
  list_data_item_descr.item[1].create_label := FALSE;
  list_data_item_descr.item[1].visible_box := FALSE;
  list_data_item_descr.item[1].label_length := 0;
  list_data_item_descr.item[1].size := 40;
  list_data_item_descr.item[1].tp := STRING_TP;
  list_data_item_descr.item[1].is_initialized := FALSE;
  list_data_item_descr.item[1].is_modifable := FALSE;
  list_data_item_descr.item[1].has_value_limits := FALSE;
  list_data_item_descr.item[1].poss_nb := 0;
  list_data_item_descr.item[2] := DataItemDescr.New;
  list_data_item_descr.item[2].id := "time";
  list_data_item_descr.item[2].title := "date de reception";
  list_data_item_descr.item[2].create_label := FALSE;
  list_data_item_descr.item[2].visible_box := FALSE;
  list_data_item_descr.item[2].label_length := 0;
  list_data_item_descr.item[2].size := 30;
  list_data_item_descr.item[2].tp := STRING_TP;
  list_data_item_descr.item[2].is_initialized := FALSE;
  list_data_item_descr.item[2].is_modifable := FALSE;
  list_data_item_descr.item[2].has_value_limits := FALSE;
  list_data_item_descr.item[2].poss_nb := 0;
  list_data_item_descr.item_elem_nb := 2;
  v := IListView.New;
  v.Init(dataCtrl, SELF, actView, list_data_item_descr);
  v.CreateImage(englob_cont);
  map_data.UpdV_ConceptEntry(concept_ind, v, -1);
END CrConstrEl_HeaderList_P;

METHOD ChangeVarValue(IN v : REF Displayable);
w : REF Widget_X;
BEGIN
  var := ASSERTYPE(v);
  map_data.InitNilConceptSon("", persp_id, "", var);
END ChangeVarValue;
```

```
METHOD UpdateVarImage(IN concept : REF Displayable; elem_id : String; d : REF ModifDescr);
w : REF Widget_X;
o : REF Displayable;
BEGIN
IF concept = var
THEN
  IF elem_id = "name" OR elem_id = ""
  THEN
    map_data.GetDataWidget_X(concept, "", persp_id, "name", w);
    IF var = NIL THEN SUPER.UpdateStringFieldValue(w, "");
    ELSE SUPER.UpdateStringFieldValue(w, var.name);
    END;
  END;
  RETURN;
END;
END UpdateVarImage;

METHOD UpdEl_MailBox_P(IN path : String; concept : REF MailBox_AC; elem_id : String ; d :
REF ModifDescr);
w : REF Widget_X;
o : REF Displayable;
new_o : REF Displayable;
local_persp_id : String[ID_LENGTH] = "MailBox_P";
BEGIN
  IF elem_id = "messages" OR elem_id = ""
  THEN
      map_data.GetOldConcept(concept,  "messages", o);
      IF concept = NIL THEN new_o := NIL; ELSE new_o := concept.messages; END;
      IF (new_o = NIL) AND (new_o # o) THEN new_o := NIL; END;
      IF (new_o # o) THEN
        map_data.UpdateConceptSon(concept, "messages", ASSERTYPE(new_o));
      END;
      SELF.UpdConstrEl_HeaderList_P(path + "." + "messages", ASSERTYPE(new_o), "" ,
ASSERTYPE(d));
  END;
END UpdEl_MailBox_P;

METHOD UpdConstrEl_HeaderList_P(IN path : String; concept : REF DisplayableList; elem_id :
String; d : REF ModifDescr);
v : REF ListView;
o : REF Displayable;
new_o : REF Displayable;
t : REF Mess_AC;
list_item_value : REF ListDataItemValue;
a : Array[ITEMS_NB] OF REF ListDataItemValue;
i : Integer;
local_persp_id : String[ID_LENGTH] = "HeaderList_P";
BEGIN
  v := ASSERTYPE(map_data.GetConceptView(path, local_persp_id));
  IF d = NIL THEN
    IF concept = NIL THEN v.Reset; RETURN; END;
    FOR i := 1 TO concept.NbItems DO
      t := ASSERTYPE(concept.Go(i));
      list_item_value := ListDataItemValue.New;
      list_item_value.cpt_value := ASSERTYPE(t);
      list_item_value.item[0] := DataItemValue.New;
      list_item_value.item[0].cpt_value := t;
      list_item_value.item[0].tp := STRING_TP;
      list_item_value.item[0].string_value := t.sender.name;
      list_item_value.item[1] := DataItemValue.New;
      list_item_value.item[1].cpt_value := t;
      list_item_value.item[1].tp := STRING_TP;
      list_item_value.item[1].string_value := t.subject;
      list_item_value.item[2] := DataItemValue.New;
      list_item_value.item[2].cpt_value := t;
      list_item_value.item[2].tp := STRING_TP;
      list_item_value.item[2].string_value := t.time;
      a[i-1] := list_item_value;
    END;
    v.Reset(a, concept.NbItems -1);
    RETURN;
  END;
  IF d.tp = "add" THEN
      t := ASSERTYPE(concept.Go(d.pos + 1));
      list_item_value := ListDataItemValue.New;
      list_item_value.cpt_value := ASSERTYPE(t);
      list_item_value.item[0] := DataItemValue.New;
      list_item_value.item[0].cpt_value := t;
      list_item_value.item[0].tp := STRING_TP;
      list_item_value.item[0].string_value := t.sender.name;
```

```
        list_item_value.item[1] := DataItemValue.New;
        list_item_value.item[1].cpt_value := t;
        list_item_value.item[1].tp := STRING_TP;
        list_item_value.item[1].string_value := t.subject;
        list_item_value.item[2] := DataItemValue.New;
        list_item_value.item[2].cpt_value := t;
        list_item_value.item[2].tp := STRING_TP;
        list_item_value.item[2].string_value := t.time;
        v.AddItem(d.pos, list_item_value);
    END;
    IF d.tp = "sub" THEN
        v.SubItem(d.pos);
    END;
    IF d.tp = "upd" THEN
        t := ASSERTYPE(concept.Go(d.pos + 1));
        list_item_value := ListDataItemValue.New;
        list_item_value.cpt_value := ASSERTYPE(t);
        list_item_value.item[0] := DataItemValue.New;
        list_item_value.item[0].cpt_value := t;
        list_item_value.item[0].tp := STRING_TP;
        list_item_value.item[0].string_value := t.sender.name;
        list_item_value.item[1] := DataItemValue.New;
        list_item_value.item[1].cpt_value := t;
        list_item_value.item[1].tp := STRING_TP;
        list_item_value.item[1].string_value := t.subject;
        list_item_value.item[2] := DataItemValue.New;
        list_item_value.item[2].cpt_value := t;
        list_item_value.item[2].tp := STRING_TP;
        list_item_value.item[2].string_value := t.time;
        v.UpdateItem(d.pos, list_item_value);
    END;
END UpdConstrEl_HeaderList_P;

METHOD GetImageValues(IN concept : REF Displayable; OUT values_cpt : Integer; values :
Array OF REF AttributeValue);
w : REF Widget_X; modif : Boolean; obj : REF Top;
BEGIN
 values_cpt := -1;
END GetImageValues;
END User_P_PV.
```

Références bibliographiques

[Adar 91]
M. Adar, E. Kantorowitz, E. Bar-On, A system supporting design implementation and maintenance of object-oriented programs, *Proceedings of the Fifth Israel Conference on Computer Systems and Software Engineering*, 1991.

[Adobe 85]
Adobe, *PostScript Language Reference Manual*, Addison Wesley, Reading, Mass., 1985.

[Arch 91]
The Arch Model: Seeheim Revisited, User Interface Developers' Workshop, April 26, 1991.

[Barthet 88]
M. F. Barthet, Logiciels Interactifs et Ergonomie, Dunod, 1988.

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

[Bellamy 90]
R. K. E. Bellamy, J. M. Carroll, Redesign by Design, *Proceedings of INTERACT'90*, 1990, p. 199-205.

[Boehm 82]
Boehm B. W., Software Engineering Economics, Prentice Hall, Englewood Cliffs (N.J.), 1982.

[Boies 88]
Stephen J. Boies, William E. Bennett, John D. Gould, Sharon L. Greene, Charles Wiecha, *The Interactive Transaction System (ITS): Tools for Application Development*, Technical Report, IBM T. J. Watson Research Center, New York, 1988.

[Borenstein 85]
N. S. Borenstein, The Evaluation of Text Editors: a Critical Review of the Roberts and Moran Methodology Based on New Experiments, *Proceedings of CHI'85*, 1985, p. 99-105.

[Caelen 91]
J. Caelen, J. Coutaz, Interaction homme-machine multimodale : problèmes généraux, *IHM'91*, Troisièmes Journées sur l'Ingénierie des Interfaces Homme-Machine, décembre 1991.

[Card 83]
S. Card, T. Moran, A. Newell, *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Publ., 1983.

[Cardelli 85]
L. Cardelli, R. Pike, Squeak: a language for communicating with Mice, *Computer Graphics*, July 1985, p. 199-204.

[Carroll 85]
J. M. Carroll et M. B. Rosson, Usability Specifications as a Tool in Iterative Development, *Advances in Human Computer Interaction*, vol. 1, éd. H. R. Hartson, Ablex Publishing Corporation, 1985.

[Carroll 89]
J. M. Carroll, W. A. Kellogg, Artifact as Theory-Nexus: Hermeneutics meets Theory-Based Design, *Proceedings of CHI'89*, 1989, p. 7-14.

[Coutaz 87]
Joëlle Coutaz, PAC, an Implementation Model for Dialog Design, Proceedings of INTERACT'87, Septembre 1987, p. 431-436.

[Coutaz 90]
Joëlle Coutaz, *Interface homme-ordinateur : conception et réalisation*, Dunod Publ., 1990.

[Coutaz 91a]
Joëlle Coutaz, Architectural design for user interfaces, *Proceedings of ESEC'91*, European Software Engineering Conference, octobre 1991.

[Coutaz 91b]
Joëlle Coutaz, Laurence Nigay, Seeheim et architecture multi-agent, *IHM'91*, Troisièmes Journées sur l'Ingénierie des Interfaces Homme-Machine, décembre 1991.

[Cox 86]
B. Cox, *Object-Oriented Programming, An Evolutionary Approach*, Addison & Wesley, 1986.

[de Champeaux 91]
D. de Champeaux, Object-Oriented Analysis and Top-Down Software Development, *ECOOP'91*, July 1991, p. 360-376.

[Decouchant 91]
D. Decouchant, V. Normand, G. Vandôme, Application design using the Comandos distributed object oriented system, *Proceedings of HCI'91*, 1991.

[Dix 87]
A. J. Dix, M. D. Harrison, Formalising models of interaction in the design of a display editor, *Proceedings of Human-Computer Interaction - INTERACT'87*, 1987, p. 409-414.

[Draper 86]
Stephen W. Draper, Display managers as the basis for user-machine communication, in [Norman 86].

[Egeria 90]
Egeria Manual, BULL, Centre de recherche de Sophia Antipolis, 1990.

[Flecchia 87]
M. A. Flecchia, R. D. Bergeron, Specifying complex dialogs in Algae, *Proceedings of CHI+GI'87 Conference*, 1987, p. 229-234.

[Foley 84]
J. D. Foley, A. Van Dam, Fundamentals of interactive computer graphics, Addison Wesley, 1984.

[Foley 87a]
J. Foley, Transformations on a Formal Specification of User-Computer Interfaces, *Computer Graphics*, 21(2), avril 1987, p. 109-113.

[Foley 87b]
J. Foley, W. C. Kim, C. Gibbs, Algorithms to Transform the Formal Specification of User-Computer Interface, *Proceedings of INTERACT'87*, 1987, p. 1001-1006.

[Foley 88]
J. Foley, W. C. Kim, S. Kovacevic, K. Murray, *The User Interface Design Environment*, Report GWU-IIST-88-4, George Washington University, January 88.

[Goldberg 84]
A. Goldberg, *Smalltalk-80: The Interactive Programming Environment*, Addison-Wesley Publ., 1984.

[Green 85]
Mark Green, *The Design of Graphical User Interfaces*, Ph. D. Thesis, CSRI-170-85, Computer Systems research Institute, University of Toronto, 1985.

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

[Guindon 87]
R. Guindon, H. Krasner, B. Curtis, Cognitive processes in software design: activities in early, upstream design, *Proceedings of INTERACT'87*, 1987, p. 383-388.

[Harrison 91]
M. Harrison, G. Abowd, Formal Methods in Human Computer Interaction: a Tutorial, *CHI'91 Tutorial*, 1991.

[Hayes 85]
P. J. Hayes, P. Szekely, R. Lerner : Design Alternatives for User Interface Management Systems Based on Experience with Cousin, *Proceedings of CHI'85*, April 1985, p. 169-175.

[Henderson 86]
D. A. Jr Henderson, S. K. Card, Rooms : The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window Based Graphical User Interface, *ACM Transactions on Graphics*, (5)3, July 1986, p. 211-243.

[Hill 86]
R. D. Hill, Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction: the Sassafras UIMS, *ACM Transactions on Graphics*, (5)3, July 1986.

[Hill 87]
R. D. Hill, Event Response Systems - a technique for specifying multi-thread dialogs, *Proceedings of the CHI+GI'87 Conference*, 1987, p. 241-248.

[Hutchins 86]
E. L. Hutchins, J. D. Hollan, D. A. Norman, Direct Manipulation Interfaces, User *Centered System Design*, édité par D. A. Norman, S. W. Draper, Lawrence Erlbaum Associates, 1986.

[ISO 85]
International Organisation for Standardization, Information processing systems - Computer graphics - *Graphical Kernel System (GKS) functional description*, ISO IS 7942, juillet 1985.

[Jacobson 87]
I. Jacobson, Object Oriented Development in an Industrial Environment, *Proceedings of OOPSLA'87*, October 87, p. 183-191.

[John 90]
B. E. John, Extensions of GOMS analyses to expert performance requiring perception of dynamic visual and auditory information, *Proceedings of CHI'90*, 1990, p. 107-115.

[Kieras 85]
D. Kieras, P. G. Polson, An approach to the formal analysis of user complexity, *International Journal of Man-Machine Studies*, 22, 1985, p. 365-394.

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

[Krakowiak 90]
S. Krakowiak, M. Meysembourg, H. Nguyen Van, M. Riveill, C. Roisin, Design and implementation of an object-oriented, strongly typed language for distributed applications, *Journal of Object-Oriented Programming*, September 1990.

[Krasner 88]
G. E. Krasner, S. T. Pope, A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80, Journal of Object-Oriented Programming, août-septembre 1988, p. 26-49.

[Laurel 86]
Brenda K. Laurel, Interface as mimesis, *User Centered System Design*, édité par D. A. Norman, S. W. Draper, Lawrence Erlbaum Associates, 1986.

[Lisbon 90]
User Interface Management and Design, *Proceedings of the Workshop on User Interface Management Systems and Environments*, Lisbon, Portugal, 4-6 June 1990.

[Lunati 91]
J.-M. Lunati, A. I. Rudnicky, Spoken language interfaces: The OM system, *Proceedings of CHI'91*, video program,1991, p. 453-454.

[MacLean 89]
A. MacLean, R. M. Young, T. P. Moran, Design rationale: the argument behind the artifact, *Proceedings of CHI'89*, 1989, p. 247-252.

[MacLean 90]
A. MacLean, V. Bellotti, R. Young, What rationale is there in design?, *Proceedings of INTERACT'90*, 1990, p. 207-212.

[Marchionini 91]
G. Marchionini, J. Sibert (éditeurs), An Agenda for Human-Computer Interaction: Science and Engineering Serving Human Needs, Report of an Invitational Workshop Sponsored by the National Science Foundation, *SIGCHI Bulletin*, 23(4), octobre 1991.

[Meyer 88]
B. Meyer, *Object-Oriented Software Construction*, Prentice Hall, 1988.

[MIT 87]
The Xlib Reference Manual, X Protocol Version 11, 1987.

[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),1981, p. 3-50.

[Myers 89]
B. A. Myers, Encapsulating Interactive Behaviors, *Proceedings of CHI'89*, 1989, p. 319-324.

[Nanard 90]
Jocelyne Nanard, *La manipulation directe en interface homme-machine*, Thèse de Doctorat d'Etat, Université de Montpellier II, décembre 1990.

[Newell 89]
A. Newell, *Unified Theories of Cognition : The 1987 William James Lectures*, Harvard University Press, 1989.

[Nguyen Van 90]
H. Nguyen Van, M. Riveill, C. Roisin, *Manuel du langage Guide (V1.5)*, Rapport Technique 3-90, Bull-IMAG Systèmes, décembre 1990.

[Norman 83]
Donald A. Norman, Some observations on mental models, *Mental models*, édité par D. Gentner et L. A. Stevens, Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.

[Norman 86]
Donald A. Norman, S. W. Draper, *User Centered System Design*, Lawrence Erlbaum Associates, 1986.

[Normand 90a]
Véronique Normand, *Construction des applications interactives Guide : état des travaux et objectifs*, note interne, Bull-IMAG Systèmes, 1990.

[Normand 90b]
Véronique Normand, A Practical Framework for Interactive Applications in GUIDE, an Object-Oriented Distributed System, *Proceedings of TOOLS'90*, July 1990, p. 657-768.

[Normand 90c]
Véronique Normand, *Manuel utilisateur de l'UIMS de Guide*, Note interne, Bull-IMAG Systèmes, juin 1990.

[Normand 90d]
Véronique Normand, Serge Lacourte, Evaluation de Guide-1 par l'utilisateur, Note interne, Bull-IMAG Systèmes, 1990.

[Normand 91a]
Véronique Normand, *Spoken language translation in a travel information domain : Combining voice and graphic interaction*, Technical Report CMU-CS-91-104, School of Computer Science, Carnegie Mellon University, January 1991.

[Normand 91b]
Véronique Normand, Développement d'une application multimodale : une expérience, *Actes des 2èmes Journées Nationales du GRECO-PRC Communication Homme-Machine*, Janvier 1991.

[Normand 91c]
Véronique Normand, *Modélisation et génération des interfaces utilisateur dans un environnement à objets*, Rapport 11-91, Bull-IMAG Systèmes, août 1991.

[Normand 91d]
Véronique Normand, *Manuel du langage SIROCO*, Note interne, Bull-IMAG Systèmes, août 1991.

[Normand 91e]
Véronique Normand, Spécification conceptuelle des interfaces et outils de développement : une approche, *IHM'91*, 3èmes Journées sur l'Ingénierie des Interfaces Homme-Machine, décembre 1991.

[Normand 92]
Véronique Normand, Joëlle Coutaz, Unifying the Design and Implementation of User Interfaces Through the Object Paradigm, *Proceedings of ECOOP'92*, 1992.

[OSF 89a]
OSF/Motif Programmer's Reference Manual, Open Software Foundation, Cambridge, MA, 1989.

[OSF 89b]
OSF/Motif Style Guide, Open Software Foundation, Cambridge, MA, 1989.

[Olsen 83]
D. R. Olsen, E. P. Dempsey : Syngraph: a graphical user interface generator, *Computer Graphics*, July 83, p. 43-50..

[Olsen 84]
D. R. Olsen Jr, W. Buxton, R. Ehrich, A Context for User Interface Management, *IEEE Computer Graphics and Applications*, 4(12), décembre 1984, p. 33-42.

[Olsen 86]
D. R. Olsen, MIKE: The Menu Interaction Kontrol Environment, *ACM Transactions on Graphics*, 5(4), October 1986.

[Olsen 89]
D. R. Olsen, A Programming Language Basis for User Interface Management, *Proceedings of CHI'89*, mai 1989, p. 171-176.

[Pellaumail 86]
P. Pellaumail, *Guide d'utilisation d'AXIAL*, Editions d'Organisation, 1986.

[Petoud 90]
Isabelle Petoud, Génération automatique de l'interface homme-machine d'une application de gestion hautement interactive, Thèse de Doctorat Université de Lausanne, 1990.

[Pfaff 85]
User Interface Management Systems, G. E. Pfaff ed., Eurographics Seminars, Springer-Verlag, 1985.

[Pierret-Golbreich 89]
C. Pierret-Golbreich, I. Delouis, D. Scapin, *Un outil d'acquisition et de représentation des tâches orienté objet*, Rapport 1063, INRIA, Rocquencourt, France, août 1989.

[Polson 85]
P. G. Polson, D. E. Kieras, A Quantitative Model of the Learning and Performance of Text Editing Knowledge, *Proceedings of CHI'85*, 1985.

[Quint 85]
V. Quint, I. Vatton, Grif :un éditeur interactif structuré, Rapport de Recherche TIGRE no 27, IMAG-LGI, 1985.

[Roberts 83]
T. L. Roberts, T. P. Moran, The Evaluation of Text Editors: Methodology and Empirical Results, *Communications of the ACM*, April 1983.

[Scapin 88]
D. L. Scapin, P. Reynard et A. Pollier, *La conception ergonomique d'interfaces : problèmes de méthode*, Rapport de Recherche n° 957, INRIA-Rocquencourt, décembre 1988.

[Scheifler 86]
R. W. Scheifler, J. Gettys, The X Window System, ACM Transactions on Graphics, 5(2), avril 1986, p. 79-109.

[Schmucker 86]
K. Schmucker, MacApp: An Application Framework, *Byte*, 11(8), 1986, p. 189-193.

[Schulert 85]
A. J. Schulert, G. T. Rogers, J. A. Hamilton : ADM - A Dialog Manager, *Proceedings of CHI'85*, April 1985, p. 177-183.

[Senach 90]
B. Senach, Evaluation de l'ergonomie des interfaces homme-machine : du prototypage aux systèmes experts, *Actes du colloque ERGO IA'90*, Septembre 1990, p. 85-106.

[Shneiderman 82]
B. Schneiderman, Multiparty grammars and related features for defining interactive systems, *IEEE Transactions on Systems, Man and Cybernetics*, number 2, 1982.

[Sibert 86]
J. L. Sibert, W. D. Hurley, T. W. Bleser, An Object-Oriented User Interface Management System, *Proceedings of SIGGRAPH'86*, Dallas, 1986.

[Siochi 89]
A. C. Siochi, H. R. Hartson, Task-Oriented Representation of Asynchronous User Interfaces, Proceedings of CHI'89, 1989, p. 183-188.

[SUN 87]
SUN Microsystems Inc., *NeWS Manual*, 1987.

[Tanner 83]
P. Tanner, W. Buxton, Some Issues in Future User Interface Management Systems (UIMS) Development, *IFIP Working Group 5.2 Workshop on User Interface Management*, Seeheim, November 1983.

[Tarby 91]
J.-C. Tarby, M. F. Barthet, Production d'interfaces : vers la génération automatique de contrôleur de dialogue, *IHM'91*, Troisièmes Journées sur l'Ingénierie des Interfaces Homme-Machine, décembre 1991.

[Tardieu 83]
H. Tardieu, S. Rochfeld, R. Coletti, *La méthode Merise,* Editions d'Organisation, 1983.

[Tauber 90]
M. J. Tauber, On mental models and the user interface, *Human-computer interaction: selected readings: a reader,* édité par J. Preece, L. Keller et H. Stolk, Prentice Hall, UK, 1990, pp. 309-324.

[Vander Zanden 91]
B. Vander Zanden, The Lapidary Graphical Interface Design Tool, *Proceedings of CHI'91*, video program,1991, p. 465-466.

[Webster 89]
B. F. Webster, *The NeXT Book*, Addison Wesley publisher, 1989.

[Young 89]
D. A. Young, *X Window Systems, programming and applications with Xt*, Prentice Hall, 1989.

[Young 90]
R. M.Young, J. Whittington, Using a Knowledge Analysis to Predict Conceptual Errors in Text-Editor Usage, Proceedings of CHI'90, 1990, p. 91-97.

Index des auteurs

Table des matières