Rapport d'avancement

Contexte du sujet :

Les interfaces graphiques traditionnelles sont construites sur la base de composants graphiques agencés selon une disposition fixe, et répondant de manière prédéterminée à chacun des événements qu'ils reçoivent. De ce fait, les interfaces graphiques manquent de souplesse et d'adaptativité par rapport à l'utilisateur qui les manipule. Pour contourner ces limitations, nous allons concevoir un modèle d'interface à base d'agents. Les agents présentent

L'avantage, par rapport aux composants graphiques classiques, d'être autonomes et réactifs, ce qui signifie qu'ils peuvent analyser dynamiquement le contexte d'utilisation de l'interface et y réagir en se réorganisant suivant des critères contrôlés par l'utilisateur.

Objectif du travail:

L'objectif principal de notre travail c'est de développer un générateur d'interface homme machine à base de système multi-agents est qui est compatible avec la méthodologie TOOD (Task Objectect Oriented Description).

Contribution attendue:

L'interface homme machine joue un rôle crucial dans le développement des applications interactives, sa simplicité d'utilisation peut parfois être un critère important d'évaluation de l'application par ailleurs une bonne application qui est représentée par une interface non adéquate peut être jugée comme une application non réussite, d'ou l'utilité des générateur d'interfaces homme machine qui peuvent nous garantir une interface lisible, intuitive et facile à manipuler par n'importe quel utilisateurs.

Aussi nous nous intéressons à résoudre le problème du passage de la spécification des objets IHM à la génération d'interfaces pour la méthodologie TOOD.

Etat de l'art :

Dans ce cadre de ce travail et en ce qui ce qui concerne la partie bibliographie et état de l'art,il existe trois grande partie à étudier :

*partie 1 : interface homme machine.
*partie 2 : agents et système multi-agents.
*partie 3 : générateur d'interfaces homme machine.

Pour la première partie les études faites concernent :

- Etude de la méthodologie TOOD.
- Etude de développement des interfaces homme machine, qui inclue les systèmes homme machine et le processus de développement des IHM : étapes, cycle, outils spécifique et les approches existantes pour le développement des systèmes interactifs.
- Etude des méthodes usuelles de spécification de l'interaction homme machine.
- Etude des architectures logicielles existantes : modèle de base et modèle de référence.

Pour la deuxième partie concernant l'agent et les systèmes multi-agents :
- Etude des modèles multi-agents existants : PAC, MVC, LIM…et les modèles hybrides PAC-AMODEUS.
- Etude des avantages de l'utilisation des systèmes multi-agents dans notre travail : flexibilité, autonomie…
- Etude bien détaillée des agents réactifs.
- Etude des plates-formes multi-agents existantes.

Pour la dernière partie : étude des générateurs existants et leurs caractéristiques

Approche proposée:

L'approche suivie dans notre travail est le passage du modèle de la tâche de la méthodologie TOOD vers le modèle multi-agents et de ce dernier vers la génération de l'interface ou bien la génération du code de l'interface.

Autrement dit, dans un premier lieu nous nous intéressons au passage de l'arborescence de la tâche spécifiant l'interface homme machine vers une arborescence d'agents et de cette dernière vers l'interface homme machine.

Rapport d'avancement détaillé

Introduction générale :

La conception des interfaces homme machine (IHM) constitue un domaine de recherche en pleine expansion. Cependant, malgré les différents travaux de recherche dans ce domaine, nous remarquons toujours l'insuffisance des méthodes et d'outils couvrant les différentes étapes de développement.

Si des solutions on été apporté, aujourd'hui, pour les phases de conception, de spécification, pour les systèmes interactifs, il reste encore des pistes à creuser notamment en ce qui concerne la générations des IHM. Dans ce courant de recherche, notre travail dans ce mémoire vise à générer l'IHM à partir de la méthodologie TOOD (Task Object Oriented Design) en se basant sur les systèmes multi-agents. TOOD étant une méthode de conception et de développement des systèmes interactifs.

J'ai décomposé ce rapport en trois grandes parties :

1) Interaction homme machine.
2) Agents et systèmes multi-agents.
3) Génération des interfaces.

Partie I _ :

Les interfaces homme machine :

1/ *Développement des systèmes interactifs :*

Pour un bon développement d'une application interactive, il faut bien étudier les phénomènes humains, logiciels et matériels qui sont mis en jeu dans cette application, aussi il faut bien résoudre quelques autres problèmes tel que :

a. Le cycle de développement à suivre ?
b. Les modèles à prendre en considération ?
c. L'architecture à suivre pour une implémentation facile et réutilisable ?

1-1/ *cycle de développement :*

Il existe plusieurs modèles à suivre pour développer un système interactif.

i. Le modèle en cascade :

Son principe c'est que chaque phase se termine à une date précise et qu'on ne passe à la phase suivante qu'après sa validation.

+ Ce modèle est utilisé pour les petites applications.
+ Il a comme avantage de stabiliser rapidement le produit.
- Mais il ne favorise pas les changements.
- Il ne prend pas en compte les exigences de l'utilisateur dans les différentes phases de développement de l'application.

ii. Le modèle en V :

Dans ce modèle l'enchaînement des phases prend la forme d'un V ;une phase descendante pour la spécification, conception et codage et une phase ascendante pour les validations et les tests.

+ Il a comme avantage de prévoir en amant l'évaluation du système et de surmonter rapidement les erreurs commises.
+ Il est conçu pour les applications hautement interactives.
- L'analyse et la modélisation des tâches humaines ne sont pas positionnées lors du développement de l'application.

iii. Le modèle spiral :

Contrairement au deux autres modèles, le modèle spiral constitue un processus itératif.

+ Ce modèle favorise l'expression de spécification pour le développement des logiciels fortement interactifs.
- Par contre il néglige l'analyse et la modélisation des utilisateurs.

⇨ Ces trois modèles de base proposés en génie logiciel sont trop génériques si le système visé est hautement interactif. D'où l'utilité des modèles conçue pour les systèmes hautement interactifs :

iv. Modèle Nabla :

C'est un double cycle en V, il différencie l'interface des modules d'aide et il se base sur une confrontation progressive entre un modèle réel et u modèle de référence.

v. Modèle en U :

Ce modèle est constituer de deux phases : une descendante pour la modélisation et une ascendante pour l'évolution du système globale. La méthodologie TOOD se base sur ce modèle.

Les nouveaux modèles adaptés pour les systèmes interactifs assurent une meilleure prise en compte de l'utilisateur, et ce ci dès les premières étapes de spécification, cependant le problème induit, dans la majorité des cas, des problèmes de coopération avec l'utilisateur.

Une question très importante est qui a elle seule constitue un domaine de recherche en génie logiciel s'impose : comment doit – on combiner les différents modèles afin de définir une démarche méthodologique qui regroupe les différents étapes de développement d'une application interactive ? Une réponse à cette question est à l'origine des approches orientées modèles.

2/ *Les approches orientées modèles :*

Les approches orientées modèles sont inspirées des systèmes UIMS, qui se basent sur des langages de spécification, le modèle est le composant central des MBD. Ce modèle lui-même est constituer d'un ensemble de modèles (Tâche, Utilisateur, Interface, Dialogue…).

Il existe de nombreuses approches orientées modèles :

a) **MASTERMIND** (Models Allowing Tools and Explicit Representation to Make Interfaces Natural to Develop): il est base sur trios modèles : modèle de la tâche, modèle de l'application et le modèle de l'interface. Ces trois modèles sont stockés puis compilés en C++, mais reste toujours consultable dynamiquement.

b) **TRIDENT** (Tools FoR an Interactive Developpment EnvironmeNT) c'est une méthodologie qui fournit un environnement pour le développement des applications interactives. Il se base sur un ensemble de modèles et suit un processus de cinq étapes : l'établissement d'un graphe d'enchaînement de fonction, la conception de la présentation, la dérivation de l'architecture, la spécification de dialogue et la réalisation du prototype statique de l'application.

c) **DIANE+** : elle vise à résoudre l'absence de spécification et d'intégration de l'utilisateur dans le développement des systèmes interactifs. Elle s'appuie sur trois modèles pour assurer la génération : le modèle de la tâche, le modèle des objets naturels PAC, le modèle de dialogue et d'aide. Dans DIANE+, la génération est centrée sur deux axes : la génération élémentaire qui assure la validation des choix de conception ; et la génération complète destinée à être utiliser pour l'application finale.

d) **FUSE** (Formal User Interface Spécification Environment) : elle est à la fois une méthodologie et un environnement pour la génération automatique d'interface utilisateur. Dans FUSE, la génération s'appuie sur un modèle de la tâche, un modèle du domaine du problème et un modèle de l'utilisateur.

L'inconvénient majeur des approches de types MBD est la complexité des modèles et des notations qui sont généralement difficiles à appréhender et à manipuler.

3/ *Modèle D'architecture :*

Tous les modèles architecturaux de référence applicables aux systèmes interactifs répondent au requis de modifiabilité : en l'état des connaissances et des pratiques, la mise au point itérative d'une IHM est la seule stratégie effective. Mais le succès de cette approche repose soit sur l'existence de bons outils de construction de maquettes, soit sur la possibilité de retoucher "à la main" une IHM sans mettre en cause la fiabilité, ni atteindre des coûts prohibitifs de mise à jour. Tous les modèles de référence répondent à cette exigence de modifiabilité avec le principe de séparation fonctionnelle. Un premier grain de séparation est la distinction entre les services d'une application et les fonctions chargées d'assurer l'interaction avec l'utilisateur. L'application, appelée aussi noyau fonctionnel, regroupe les concepts du domaine et les opérations qui leur sont applicables. L'IHM a la charge de présenter à l'utilisateur concepts et fonctions et de lui permettre de les manipuler selon un enchaînement, reflet logiciel du modèle de tâche. Les modèles de référence Seeheim et Arch sont des affinements de cette décomposition bipartite. Nous présentons une variante du Seeheim original puis sa version révisée : Arch.

3-1/ *MODELE DE SEEHEIM*

Le modèle de Seeheim, du nom de la ville dans laquelle il fût présenté pour la première fois, raffine la structuration de l'interface en trois composantes:

- la présentation,
- le contrôle de dialogue,
- l'interface avec l'application.

La présentation est responsable de l'image du système qui est présentée à l'usager. Elle est le réceptacle des interactions élémentaires que peut effectuer l'usager.

Le contrôleur du dialogue assume le rôle de serveur syntaxique du modèle "langage". Il est l'intermédiaire entre la présentation et l'interface de l'application. Il assemble les actions élémentaires des présentations en phrases syntaxiques valides qu'il associe, via l'interface avec l'application, à des concepts sémantiques de l'application. Inversement, il traduit les concepts sémantiques de l'application en ensembles d'éléments présentables à l'usager.

Le contrôleur du dialogue est par ailleurs en charge du suivi du dialogue qui s'instaure entre l'usager et l'application. L'application est dépositaire du modèle interne du système. Elle manipule la sémantique du domaine d'activité du système. L'interface de l'application définit les capacités et les moyens d'échanges du contrôleur de dialogue avec l'application. L'intérêt principal du modèle langage est son aspect modulaire qui sépare clairement l'application et l'interface et ainsi permet une mise au point par des ajustements itératifs. La maintenabilité et l'évolutivité du code IHM en sont favorisés. De fait, et en théorie, la modification de la

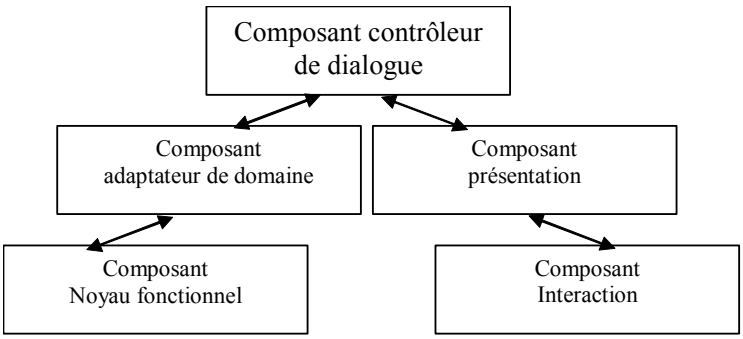
présentation peut être effectuée sans remettre en cause les fonctions de l'application et réciproquement. Ce principe n'est pas nouveau en soi, mais faute d'outils forçant cette séparation, ce principe n'est pas toujours respecté (ou s'il l'est au début du développement de logiciel, il ne l'est plus au gré des modifications et évolutions qu'il subit).

3-2/ *MODELE ARCH :*

Comme le montre la figure au dessous, le modèle Arch s'appuie sur les composants fonctionnels de Seeheim. On y retrouve le Noyau Fonctionnel (NF), le Contrôleur de dialogue et la Présentation. Les pieds de l'arche constituent les composants imposés par la réalité : le NF réalise les concepts du domaine ; le Composant d'Interaction, en contact direct avec l'utilisateur, est mis en œuvre au moyen des interacteurs (*widgets*) d'une boîte à outils. En clé de voûte, le Contrôleur de Dialogue gère l'enchaînement des tâches ainsi que les liens entre les objets regroupés dans ses deux composants voisins : l'Adaptateur de Domaine et la Présentation. L'Adaptateur de Domaine sert, pour l'essentiel, à ajuster les différences de modélisation des objets conceptuels entre le NF et le Contrôleur de Dialogue. Le composant de Présentation joue un rôle similaire : il permet au Contrôleur de Dialogue de s'affranchir du fonctionnement de la boîte à outils du niveau interaction. La Présentation peut se voir comme une boîte à outils virtuelle qui implémente des objets de présentation concrétisés en fin de compte par les interacteurs d'une boîte à outil réelle.

La figure doit se voir comme une représentation canonique. En réalité, l'importance relative des cinq composants de l'arche varie en fonction du cas à traiter. Cette migration des fonctions entre les composants s'exprime métaphoriquement au moyen du métamodèle slinky, du nom de ce jouet qui, une fois mis en mouvement, voit sa masse se déplacer dynamiquement. Le choix de la répartition des fonctions entre les composants relève du savoir-faire en ingénierie du logiciel. L'équilibre doit être le résultat d'un dosage analytique des facteurs qualité comme l'efficacité et la portabilité. On appellera instance d'Arch, un exemplaire de modèle Arch produit en appliquant slinky à un système interactif particulier.

Seeheim et Arch fournissent des structures fonctionnelles canoniques à gros grain. Ces modèles sont utiles comme cadres structurants pour une conception ou une analyse grossière de la décomposition fonctionnelle d'un système interactif. Les modèles de référence multi-agent visent une décomposition fonctionnelle plus fine avec, en filigrane, le support du parallélisme.



Partie II _ :

L'agent et le système multi-agents :

1/ *Qu'est ce qu'un agent?*

Le concept d'agent a été l'objet d'études pour plusieurs décennies dans différentes disciplines. Il a été non seulement utilisé dans les systèmes à base de connaissances, la robotique, le langage naturel et d'autres domaines de l'intelligence artificielle, mais aussi dans des disciplines comme la philosophie et la psychologie. Aujourd'hui, avec l'avènement de nouvelles technologies et l'expansion de l'Internet, ce concept est encore associé à plusieurs nouvelles applications comme *agent ressource, agent courtier, assistant personnel, agent interface, agent ontologique,* etc. Dans la littérature, nous trouvons une multitude de définitions d'agents. Elles se ressemblent toutes, mais diffèrent selon le type d'application pour laquelle est conçu l'agent. À titre d'exemple, voici l'une des premières définitions de l'agent: « Un agent est une entité autonome, réelle ou abstraite, qui est capable d'agir sur elle-même et sur son environnement, qui, dans un univers multi agents, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de ses connaissances et des interactions avec les autres agents ». Il ressort de cette définition des propriétés clés comme *l'autonomie*, *l'action*, la *perception* et la *communication*. D'autres propriétés peuvent être attribuées aux agents. Nous citons en particulier la *réactivité*, la *rationalité*, l'*engagement* et l'*intention*. Comme nous pouvons définir l'agent comme suit : « Un agent est un système informatique, *situé* dans un environnement, et qui agit d'une façon *autonome* et *flexible* pour atteindre les objectifs pour lesquels il a été conçu ». Les notions "situé", "autonomie" et "flexible" sont définies comme suit:

– *situé*: l'agent est capable d'agir sur son environnement à partir des entrées sensorielles qu'il reçoit de ce même environnement. Exemples: systèmes de contrôle de processus, systèmes embarqués.

– *autonome*: l'agent est capable d'agir sans l'intervention d'un tiers (humain ou agent) et contrôle ses propres actions ainsi que son état interne;

– *flexible*: l'agent dans ce cas est *capable de répondre à temps*: l'agent doit être capable de percevoir son environnement et élaborer une réponse dans les temps requis.

– *proactif*: l'agent doit exhiber un comportement proactif et opportuniste, tout en étant capable de prendre l'initiative au "bon" moment;

– *social*: l'agent doit être capable d'interagir avec les autres agents (logiciels et humains) quand la situation l'exige afin de compléter ses tâches ou aider ces agents à accomplir les leurs.

2/ *Les systèmes multi agents :*

Un système multi agents est un système distribué composé d'un ensemble d'agents. Contrairement aux systèmes d'IA, qui simulent dans une certaine mesure les capacités du raisonnement humain, les SMA sont conçus et implantés idéalement comme un ensemble d'agents interagissant, le plus souvent, selon des modes de coopération, de concurrence ou de coexistence.

Un SMA est généralement caractérisé par:
– chaque agent a des informations ou des capacités de résolution de problèmes limitées, ainsi chaque agent a un point de vue partiel.
– il n'y a aucun contrôle global du système multi agent.
– les donnés sont décentralisées.
– le calcul est asynchrone.

Les SMA sont des systèmes idéaux pour représenter des problèmes possédant de multiples méthodes de résolution, de multiples perspectives et/ou de multiples résolveurs. Ces systèmes possèdent les avantages traditionnels de la résolution distribuée et concurrente de problèmes comme la modularité, la vitesse (avec le parallélisme), et la fiabilité (due à la redondance). Ils héritent aussi des bénéfices envisageable de l'Intelligence Artificielle comme le traitement symbolique (au niveau des connaissances), la facilité de maintenance, la réutilisation et la portabilité mais surtout, ils ont l'avantage de faire intervenir des schémas d'interaction sophistiqués. Les types courants d'interaction incluent la coopération (travailler ensemble à la résolution d'un but commun) ; la coordination (organiser la résolution d'un problème de telle sorte que les interactions nuisibles soient évitées ou que les interactions bénéfiques soient exploitées) ; et la négociation (parvenir à un accord acceptable pour toutes les parties concernées).

Les SMA sont à l'intersection de plusieurs domaines scientifiques: informatique répartie et génie logiciel, intelligence artificielle, vie artificielle. Ils s'inspirent également d'études issues d'autres disciplines connexes notamment les sociologies, la psychologie sociale, les sciences cognitives et bien d'autres. C'est ainsi qu'on les trouve parfois à la base des:
– bases de données et bases de connaissances distribuées coopératives.
– systèmes pour la compréhension du langage naturel.
– protocoles de communication et réseaux de télécommunications.
– programmation orientée agents et génie logiciel.
– robotique cognitive et coopération entre robots.
–systèmes distribués.
–interface homme machines.

3/ *MODELE Multi-Agents :*

Un agent est un système de traitement de l'information : il se distingue par un jeu d'opérations, des mécanismes d'entrée/sortie et une capacité à représenter un état que l'on appelle vecteur d'état. Un agent est un acteur communicant : il assume un rôle (c'est un acteur) et se manifeste par un comportement observable via l'acquisition et la production d'informations (il communique). L'acquisition et la production d'information sont des actions dont la réalisation se traduit par des événements. Les informations proviennent ou sont destinées à d'autres agents : l'agent vit en communauté. Il conduit ses activités en parallèle avec celles de ses confrères. Il faut l'opposer à milieu d'activités séquentielles et à entité passive et isolée.

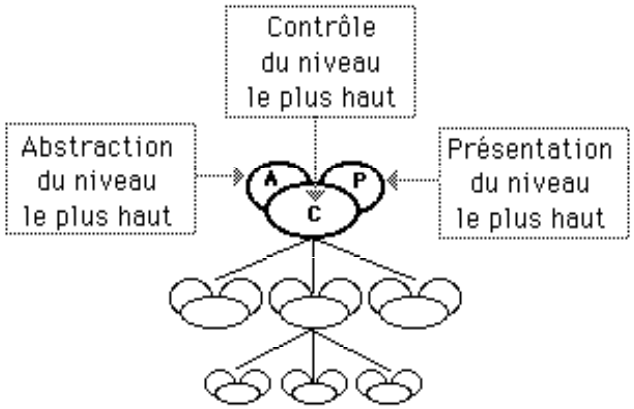
En Intelligence Artificielle Distribuée, on convient de distinguer les agents réactifs des agents intelligents. Les premiers, du type stimuli-réponse, ont un comportement câblé. Les seconds, dotés de mécanisme de planification, poursuivent des buts de manière adaptative. En Interface Homme-Machine, on parle implicitement d'agents réactifs.

3-1/ *Modèle PAC (Présentation, Abstraction, Contrôle) :*

Un agent PAC est constitué de trois facettes : la facette Présentation, la facette Abstraction et la facette Contrôle.

- La facette Présentation définit l'image de l'agent, c'est-à-dire son comportement en entrée comme en sortie vis-à-vis de l'utilisateur.
- La facette Abstraction regroupe les aspects conceptuels de l'agent.
- La facette Contrôle maintient la cohérence entre les facettes Présentation et Abstraction. Présentation et Abstraction ne sont jamais en contact direct. Par ailleurs, cette facette prend en charge les échanges avec les autres agents.

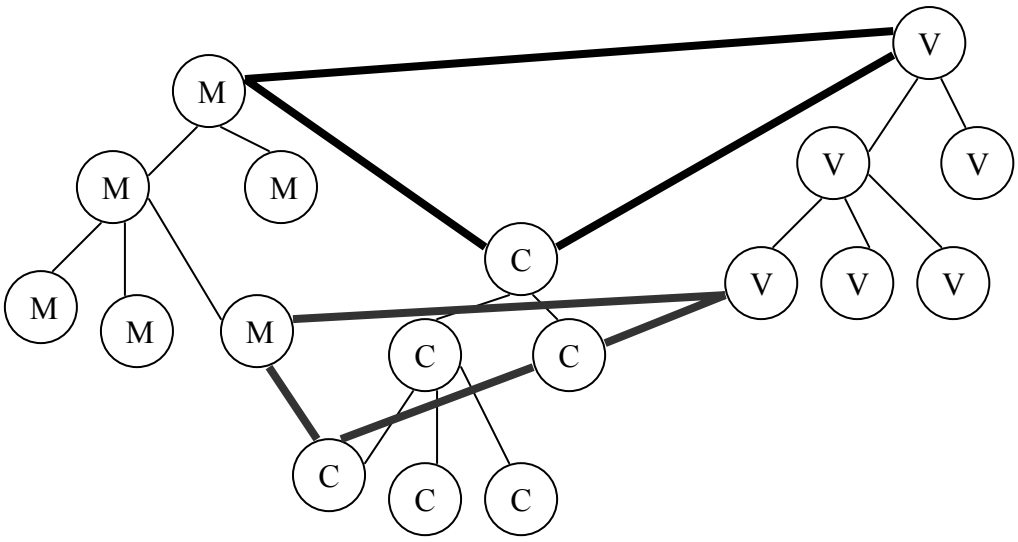
Un système interactif modélisé par PAC est organisé en agents PAC à différents niveaux d'abstraction comme le montre la figure si dessous.



Le modèle PAC est avant tout une structuration conceptuelle, non pas logicielle. PAC n'est associé à aucun environnement particulier. Dans un langage à objets, par exemple, un agent pourrait être constitué d'un unique objet comme de trois objets différents, un par facette. Ou encore, un agent pourrait correspondre à un module de code.

3-2/ *Modèle MVC:(Modèle, Vue, Contrôle) :*

MVC est issu du langage objet Smalltalk. Un agent MVC est constitué de trois facettes : le Modèle (Model), la Vue (View) et le Contrôleur (Controller). Le Modèle est lié au noyau fonctionnel de l'application. Cette facette correspond à l'Abstraction de l'agent PAC. La Vue est liée à la présentation de sortie, typiquement, ce que le système restitue à l'écran. Le Contrôleur est lié à l'interprétation des entrées en provenance de l'utilisateur. En environnement Smalltalk, un agent MVC est constitué de trois objets Smalltalk, un par facette. Trois hiérarchies de classes cohabitent donc : les hiérarchies de Modèles, de Vues et de Contrôleurs.

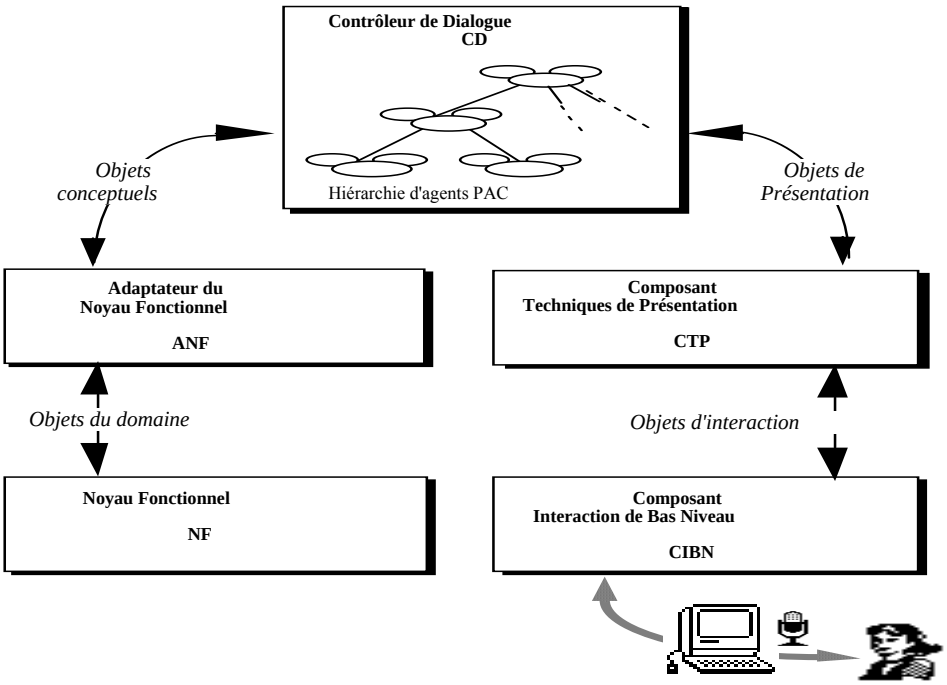


Les objets Modèle peuvent être de toute classe s'avérant appropriée pour la représentation des données manipulées. Les objets Vue et Contrôleur, au contraire, héritent tous des racines uniques Vue et Contrôleur. Une Vue est constituée d'une vue principale à laquelle peuvent être associées plusieurs sous-vues. Notons que les objets Vue possèdent l'information décrivant la façon dont les données doivent être affichées. Par ailleurs, l'une de leur responsabilité est de maintenir la cohérence des informations dans la triade MVC. La Vue et le Contrôleur d'un agent MVC possèdent tous les deux un lien vers le Modèle qui leur est associé, mais aussi des liens de l'un vers l'autre.

3-3/ *MODELE Hybride :*

Modèle PAC-Amodeus :

Comme le montre la figure si dessous, PAC-Amodeus reprend le découpage fonctionnel de Arch et structure le contrôleur de dialogue en agents PAC. Il allie les concepts architecturaux en couche du modèle Arch avec l'approche multi-agents de décomposition du contrôle de dialogue de PAC.



De la même manière que le modèle Arch, le modèle PAC-AMODEUS décompose le système interactif en cinq éléments:

- Le composant d'Interaction.
- Le composant Présentation,
- Le composant Noyau Fonctionnel,
- Le composant Adaptateur de Domaine,
- Le composant Contrôleur de Dialogue.

Le composant d'Interaction :

Le composant d'Interaction et le composant présentation forment une chaîne qui est responsable de la mise en œuvre de l'image du système, au même titre que la présentation du modèle de Seeheim. Le composant d'interaction est la boîte à outils liée à la plate-forme et au système d'exploitation sur lesquels se fera l'implémentation de l'interface de l'application. Les objets d'interaction sont les objets constituant cette boîte à outils. La définition de ces objets est orientée par leur représentation graphique.

Le composant présentation :

PAC-AMODEUS place au-dessus du composant d'interaction une boîte à outils dont les objets, toujours définis par leur représentation graphique, sont indépendants de la plate-forme hôte.

Le composant présentation est une boîte à outils couvrant le composant d'interaction et assurant ainsi l'indépendance de l'IHM vis-à-vis de la plate-forme sur laquelle se fera l'implémentation.

Les objets de présentation sont les objets constituant cette boîte à outils. Ils sont liés directement à un ou plusieurs objets d'interaction qui les implémentent sur différentes plates-formes. Ce raffinement de la partie présentation du système interactif apporte la portabilité et la réutilisabilité que l'on peut attendre d'une indépendance vis-à-vis de la plate-forme hôte.

Le composant Noyau Fonctionnel:

Il réunit les concepts et traitements propres à l'accomplissement des tâches. Il forme un serveur sémantique fournissant des informations au contrôleur de dialogue, à son initiative ou à celle de ce dernier.

Le composant Adaptateur de Domaine :

L'expérience montre la nécessité de la présence, entre le contrôleur de dialogue et le noyau fonctionnel, d'un ensemble d'objets spécifiques assurant l'interface entre les objets manipulés par le contrôleur de dialogue et le noyau fonctionnel, garantissant ainsi l'indépendance du contrôleur de dialogue vis-à-vis du noyau fonctionnel. PAC-AMODEUS réunit ces objets au sein de l'adaptateur de domaine. Ce composant a un rôle symétrique à celui du composant présentation.

Le composant Contrôleur de Dialogue :

Le contrôleur de dialogue joue le rôle de médiateur entre la présentation et le noyau fonctionnel. A ce rôle de médiateur s'ajoute celui de gestionnaire de l'état de l'interaction entre l'usager et la machine, l'ensemble des états, leurs relations et leur dynamique définissant ainsi la structure du dialogue entre l'usager et la machine.

PAC-AMODEUS tire profit de l'approche multi-agents en architecturant le composant contrôleur de dialogue autour d'une hiérarchie d'agents PAC. Le contrôleur de dialogue assure le séquencement des tâches, la traduction du formalisme et l'adaptation des données. Ces traitements sont effectués à différents niveaux d'abstraction et distribués entre de multiples agents. Dans ce cadre, la facette présentation d'un agent PAC du contrôleur de dialogue est en relation avec des objets de présentation du composant présentation. De même, la facette abstraction de l'agent peut être associé à certains objets du domaine du noyau fonctionnel.

4/*Analyse critique de PAC :*

PAC fournit un cadre de construction systématique applicable à tous les niveaux d'abstraction d'un système interactif. Cette approche récursive permet de concevoir une architecture progressivement de façon ascendante ou descendante. PAC permet de traduire à la fois l'encapsulation/l'affinement et la dépendance fonctionnelle, deux activités de conception logicielle intimement reliées. PAC permet de distinguer les services abstraits des techniques d'interaction en introduisant un intermédiaire explicite : le contrôle. Cette propriété d'indépendance présente plusieurs avantages :

– La satisfaction du critère qualité de réutilisabilité des constituants de l'interface et ceci de manière systématique à tous les niveaux d'abstraction.

– La modification de l'interface à moindre coût. PAC indique qu'il est possible de mettre au point une interface de façon itérative et de la faire évoluer facilement.

PAC encourage la répartition des traitements sémantiques et syntaxiques. Cette propriété présente plusieurs retombées intéressantes :

– Les constituants de l'interface communiquent au niveau d'abstraction voulu. En particulier, le noyau fonctionnel a la possibilité de s'exprimer dans son formalisme, à son niveau d'abstraction, ce qui augmente son indépendance vis-à-vis des constituants perceptibles de l'interface.

– Une part de la connaissance du noyau fonctionnel peut être déportée dans l'interface. Cette forme de délégation combine performance et qualité sémantique des retours d'information sans mettre en cause la distinction entre le noyau fonctionnel et l'interface.

5/*Modèles multi-agent et style architectural*

Tous les modèles multi-agent pour système interactif répondent à des préoccupations communes : modularité, distinction explicite entre présentation et abstraction, encapsulation, parallélisme. Tous définissent un vocabulaire d'éléments conceptuels (par exemple, pour PAC, le concept d'agent et ses facettes fonctionnelles P, A, C, et la communication par

événement) ; tous imposent des contraintes entre ces éléments conceptuels (dans PAC, les agents ne communiquent que via leur facette C ; de même, les facettes P et A d'un agent ne communiquent que par la facette C de l'agent) ; tous ont une sémantique – certes plus ou moins formelle : ce sont des styles. Ou, pour reprendre une expression courante, chaque modèle multi-agent a son style. Voyons comment.

Dans MVC, les agents sont, comme dans PAC, structurés en facettes fonctionnelles. Le "Model", qui définit la compétence de l'agent MVC, correspond à la composante Abstraction de PAC. La "Vue" recouvre la fonction de restitution de l'agent et le "Contrôle", sa fonction d'acquisition des actions utilisateur. Le couple "Vue, Contrôle" est équivalent à la facette Présentation de PAC.

Considérons en quelques lignes le niveau implémentationnel des modèles multi-agents :

– Les facettes fonctionnelles d'un agent ne sont pas nécessairement regroupées en un composant. Par exemple, dans MVC, un agent est réalisé par une instance de Modèle, une instance de Contrôle et une instance de Vue reliées par des appels de méthodes. Chaque instance de Modèle (Contrôle, Vue) est une classe d'une arborescence de classes de Modèle (Contrôle, Vue). Inversement, l'implémentation d'un agent PAC en langage C peut se faire sous forme d'un seul module

Les modèles multi-agent relèvent de styles homogènes. Un style homogène convient si aucun composant logiciel n'est réutilisé comme c'était le cas avant l'apparition des boîtes à outils et des générateurs d'interfaces. A contraire, si des composants logiciels sont réutilisés dans le système, que ce soit des outils de développement d'IHM ou un noyau fonctionnel à rendre interactif, il convient alors de transformer ces composants afin de respecter le style adopté. Cette transformation ne se fait pas à moindre coût. Les modèles hybrides répondent à la contrainte d'hétérogénéité.

6/ *Les plates-formes multi-agents :*

Nous donnons ici une brève description des outils et de leurs caractéristiques.

AgentTool : Cet outil se base sur une méthodologie qui se veut une extension au modèle OO : la méthodologie MaSE. Celle-ci comporte sept phases : trouver les buts, appliquer les cas d'utilisation, raffiner les buts, créer les classes d'agents, construire les conversations, assembler les classes d'agents et l'implémentation. Cette méthode met l'accent sur l'analyse et le développement. L'outil permet la vérification et la validation des conversations. Le déploiement (partiel) se fait directement à l'intérieur de l'environnement. La génération automatique du code (en Java) des conversations est disponible. Cet outil est intéressant pour effectuer les premières étapes du développement d'un SMA.

AgentBuilder : AgentBuilder est un environnement de développement complet. Une modélisation orientée-objet avec OMT constitue la base de la conception des systèmes à laquelle on ajoute une partie « ontologie ». L'élaboration du comportement des agents se fait à partir du modèle BDI et du langage AGENT-0. KQML est utilisé comme langage de communication entre les agents. L'exécution du système se fait à partir de l'engin d'exécution d'AgentBuilder. Par contre, on peut créer des fichiers « .class » et les exécuter sur une JVM standard. AgentBuilder est un outil complexe qui demande des efforts d'apprentissage importants et de bonnes connaissances dans le domaine des systèmes multiagents pour être utilisé de façon performante. Il est limité au niveau de l'extensibilité, du déploiement et de la réutilisabilité.

DECAF : DECAF est un environnement de développement de plans. L'outil fourni quelques utilitaires pour l'élaboration de plans et pour la coordination des tâches. Un planificateur

applique des heuristiques pour trouver un ordonnancement aux tâches. Une interface permet la construction de celles-ci. DECAF fourni aussi un éditeur d'agent qui est utile pour le « débuggage ». Aucune méthodologie n'est spécifiée pour la conception.

Jack : L'environnement Jack est constitué d'un éditeur gestionnaire de projet, d'un langage de programmation JAL (Jack Agent Language) et d'un compilateur. Le gestionnaire de projet est une interface qui possède un éditeur de textes où se fait l'implémentation du système. La compilation (passage de JAL à Java) et l'exécution du système se font aussi à l'intérieur de cette interface. Le langage JAL est une extension à Java. Aucune méthodologie n'est proposée. Les agents sont basés sur un modèle BDI. Aucun éditeur n'est disponible pour le développement ou le déploiement des systèmes. Jack est très long à maîtriser, il faut apprendre le langage JAL et connaître le modèle BDI de dMars (d'Iverno, 1997). De plus, le manque de support graphique complique l'implémentation et le déploiement des systèmes.

Jade : Jade est un outil qui répond aux normes FIPA97. Aucune méthodologie n'est spécifiée pour le développement. Jade fourni des classes qui implémentent

8 JFIADSMA 2002, 28-30 octobre, Lille, France « JESS » pour la définition du comportement des agents. L'outil possède trois modules principaux (nécessaire aux normes FIPA). Le DF « director facilitor » fourni un service de pages jaunes à la plateforme. Le ACC « agent communication chanel » gère la communication entre les agents. Le AMS « agent management system » supervise l'enregistrement des agents, leur authentification, leur accès et utilisation du système. Les agents communiquent par le langage FIPA ACL. Un éditeur est disponible pour l'enregistrement et la gestion des agents. Aucune autre interface n'est disponible pour le développement ou l'implémentation. À cause de cette lacune, l'implémentation demande beaucoup d'efforts. Elle nécessite une bonne connaissance des classes et des différents services offerts.

JAFMAS et JiVE : JAFMAS met l'emphase sur les protocoles de communications, l'interaction entre les agents, la coordination et la cohérence à l'intérieur du système. Il propose une méthodologie en cinq phases : identifier les agents, identifier les conversations, identifier les règles de conversation, analyser le modèle des conversations et l'implémentation. L'éditeur graphique (JiVE) est un outil de support pour le développement qui propose une interface qui aide l'utilisateur dans sa démarche. Une particularité de JiVE est la possibilité de travailler en groupe sur un projet. Les réseaux de Pétri et l'utilisation de COOL rendent la création de conversations et la coordination très complexe. Aucun support pour le déploiement n'est disponible.

MadKit : Madkit est un environnement basé sur la méthodologie Aalaadin ou AGR (agent / groupe / rôle). L'outil fourni un éditeur permettant le déploiement et la gestion des SMA (G-box). La gestion faite via cet éditeur offre plusieurs possibilités intéressantes. L'outil offre aussi un utilitaire pour effectuer des simulations.

Zeus : Zeus est un environnement complet qui utilise une méthodologie appelée « role modeling » pour le développement de systèmes collaboratifs. Les agents possèdent trois couches. La première couche est celle de la définition où l'agent est vu comme une entité autonome capable de raisonner en termes de ses croyances, ses ressources et de ses préférences. La seconde couche est celle de l'organisation. Dans celle-ci, il faut déterminer les relations entre les agents. La dernière couche est celle de la coordination. Dans celle-ci, on décide des modes de communication entre les agents, protocoles, coordination et autres mécanismes d'interactions. L'outil est un des plus complets. Les différentes étapes du développement se font à l'intérieur de plusieurs éditeurs : ontologie, description des tâches, organisation, définition des agents, coordination, faits et variables ainsi que les contraintes. Le développement de SMA avec Zeus est cependant conditionnel à l'utilisation de l'approche « role modeling ». L'outil est assez complexe et sa maîtrise nécessite beaucoup de temps.

Partie III _ :

Génération des interfaces

L'interface homme machine joue un rôle crucial dans le développement des applications interactives, sa simplicité d'utilisation peut parfois être un critère d'évaluation de l'application, par ailleurs une bonne application qui est représentée par une interface non adéquate peut être jugée comme une application non réussite.

L'une des solutions pour résoudre cette problématique peut être la génération automatique de l'interface homme machine.

1/*caractéristiques des générateurs d'interface :*

Les générateurs de l'interface ont pour but d'automatiser la conception de l'interface (la parie graphique) ; ils se basent essentiellement sur un ensemble de règles issues du domaine de l'ergonomie, plusieurs générateurs d'interface sont proposés, mais ils ne génèrent qu'une partie de l'interface, et même si dans certains cas la génération est totale, des retouches liées au bon sens de concepteur sont nécessaires.

4/ *Le concept de générateur orienté modèle* (GOM)

Le concept de générateur orienté modèle (GOM) s'appuie sur l'expression déclarative de la sémantique de l'application et des connaissances nécessaires à la spécification de l'apparence et du comportement d'un système interactif. Le but d'un GOM est de faciliter le développement, d'identifier les éléments réutilisables de l'interface et d'encapsuler le plus d'informations possibles dans des modèles. Cette approche est assez sophistiquée puisqu'elle a la prétention de couvrir non seulement tous les composants du modèle Arch mais aussi d'injecter d'autres connaissances spécifiées dans les modèles suivants :

• Le modèle de l'utilisateur : recouvre les caractéristiques des utilisateurs visés (age, profession, expérience ...). C'est un modèle important puisqu'il permet de créer des interfaces adaptées ou adaptables aux futurs utilisateurs ;

• Le modèle d'environnement : définit le contexte d'utilisation du logiciel (par exemple, au bureau, chez soi ou sur le terrain) ;

• Le modèle de tâches : ce terme est utilisé différemment selon les sensibilités scientifiques.
Ici, il s'agit de la spécification des tâches que l'utilisateur désire accomplir. Ce modèle se déduit de l'analyse de l'activité ;

• Le modèle du domaine : spécifie les fonctions et les objets réalisés dans le noyau fonctionnel

• Le modèle de dialogue : définit la structure du dialogue entre l'utilisateur et la machine. Il est directement dérivé du modèle de tâches et du modèle du domaine. Ce modèle définit la structure opératoire de réalisation des tâches avec le système informatique ;

• Le modèle de l'application : décrit la sémantique du noyau fonctionnel et les services qu'il assure ;

• Le modèle d'ergonomie : définit les règles d'ergonomie pour la construction d'interface.
Ces règles peuvent être utilisées au cours de la génération ou lors d'une évaluation après la génération ;

• Le modèle de présentation : représente le composant d'Interaction du modèle Arch. Il peut être écrit avec un langage de haut niveau ou en TK par exemple ;
• Le modèle de comportement : décrit le comportement des entrées. Il se situe à différents niveaux. Il peut décrire de manière très formelle le comportement des objets d'interaction ou spécifier le positionnement relatif des fenêtres ;
• Le modèle de plate-forme : définit les caractéristiques du système ou des systèmes visés (spécification des dispositifs d'entrées et de sorties).
La liste ci-dessus se veut exhaustive et idéale. En pratique, aucun GOM n'inclut tous les modèles de la liste. Chaque GOM pioche dans cette liste, ceux qui correspondent aux objectifs visés.

3/ *Gestion de placement :*

Il existe trois mécanismes de placement des composants d'une interface :

- Le placement statique : consiste à positionner les différents composants de l'interface manuellement à l'aide d'outil de développement
- Le placement implicite : encapsule les différents composants dans des conteneurs, qui sont ensuite positionnées dans l'interface finale.
- Le placement sous contrainte qui sont ensuite positionnés les différents composants les uns par rapport aux autres, comme par exemple : le haut de la barre de menu est égale au haut de la fenêtre.

4/ *Cycle de construction d'une interface :*

Le cycle de construction d'une interface comprend trois phases :

4.1/ *La phase de conception :*

La phase de conception repose en général sur l'analyse des tâches effectuées par l'utilisateur (ou qu'il aura à effectuer), sur l'identification de ses besoins et sur une bonne connaissance des contraintes éventuelles humaines, matérielles ou autres (environnement de travail par exemple). Elle consiste à rationaliser et à organiser les résultats ainsi obtenus pour décrire la façon dont un utilisateur du système final pourrait se représenter son fonctionnement. Une bonne conception repose d'une part sur une bonne conduite des analyses précédentes (ce qui nécessite obligatoirement l'implication de l'utilisateur dans ces analyses) et d'autre part sur un bon modèle d'interaction qui doit comporter des principes d'interaction simples et peu nombreux et qui doit être cohérent.

4.2/ *La phase d'implémentation :*

Elle consiste à mettre en oeuvre sur machine les idées dégagées de la phase précédente. L'utilisation d'outils d'interfaces est recommandée et même nécessaire, car ceux-ci allègent la tâche de programmation, réduisent le nombre de "bugs", assurent une certaine cohérence à l'interface en imposant de respecter des guides de style et facilitent la maintenance.

4.3/ *La phase d'évaluation :*

Elle consiste à demander à des sujets représentatifs des futurs utilisateurs, d'effectuer un certain nombre de tâches en utilisant le système interactif développé. Il s'agira ensuite d'enregistrer, de classer et d'analyser toutes les données ainsi recueillies afin de tirer des conclusions quant aux imperfections et défauts du système et les améliorations qui devront y être apportées. Il est également primordial de prendre en considération les remarques et les suggestions formulées par les sujets. Cette méthode d'évaluation est en général très coûteuse.

5/ *Les générateurs existants :*

Il existe deux familles de générateurs d'interface : les générateurs ascendants et les générateurs descendants.

5-1/ *Les générateurs descendants :*

Les générateurs descendants tels UIDE, SIROCO et ADEPT, sont des outils de spécification de haut niveau d'abstraction. Ils produisent automatiquement le code exécutable de l'IHM à partir d'une description formelle des concepts et des tâches du domaine : ils procèdent de manière descendante depuis les concepts jusqu'aux objets de présentation. Ces générateurs masquent les étapes de conception ergonomiques et logicielles. Ce faisant, ils augmentent les chances de conformité du code exécutable à la spécification des besoins et sont très efficaces pour la production de prototypes.

UIDE propose un langage de spécification des informations relatives au noyau fonctionnel sans considération pour leurs présentations. Le langage est donc fonctionnel. Par exemple, la d"finition d'une fonction du noyau fonctionnel se décrit par :

- Ses conditions d'activation et de terminaison,
- Ses paramètres décorés d'attributs tels que optionnel, par défaut,
- Sa fonction inverse,
- Les fonctions non disponibles au même instant.

Dans SIROCO, la spécification conceptuelle d'un système comprend deux volets : le premier consiste à décrire le fonctionnement du système en termes d'objets conceptuels et de fonctions. Le second définit l'utilisation du système sous forme d'espaces de travail et de perspectives. Un espace de travail regroupe des tâches et des concepts logiquement connectés. Une perspective sur un concept définit les constituants pertinents du concept qu'il convient de présenter à l'utilisateur.

5-2/ *Les générateurs ascendants :*

Les générateurs d'interfaces ascendants produisent automatiquement le code d'une interface à partir d'une spécification externe de l'interface. Alors que les générateurs descendants partent d'une description abstraite des concepts, les générateurs ascendants reposent sur une description concrète des objets de présentation. Les premiers couvrent toute la branche descendante du cycle en V, les seconds viennent en prolongement de l'analyse des besoins mais vont plus loin que les outils de spécifications externes présentés au paragraphe précédent.

Les générateurs d'interface ascendants sont, en général, fondés sur un modèle d'architecture qui d"finit les points d'ancrage du code g"n"r" entre le noyau fonctionnel et les objets de présentation d'une boîte à outils. Pour certains générateurs les protocoles d'accès au noyau fonctionnel et aux boîtes à outils sont clairement définis. C'est le cas de SERPENT qui satisfait au modèle d'architecture ARCH.

Les générateurs ascendants interactifs semblent très séduisants car ils réduisent la phase d'apprentissage. Cependant il est souvent nécessaire de revenir au niveau de la boîte à outils pour des besoins spécifiques de l'interface. De plus, il est rare que la spécification entière puisse se faire par manipulation directe. Généralement la partie statique de l'interface graphique est spécifiée par manipulation directe comme les fenêtres, leurs tailles, leurs positions etc. Au contraire la partie dynamique de l'interface, comme les liens d'ouverture entre deux composants graphiques, nécessite souvent de programmer. C'est le cas du générateur Egéria et de bien d'autres. Interface Builder, qui permet une spécification interactive des liens dynamiques est une exception.

6/ *Outils de développement :*

Il existe deux types d'outil de développement : les boîtes à outils et les squelettes d'application.

6-1/ *Les boîtes à outils :*

Une boîte à outils est une bibliothèque de composants logiciels accessibles au programme client via des appels procéduraux. Ces composants prennent généralement la forme d'objets graphiques organisés en hiérarchies de niveaux d'abstraction extensibles.

De manière générale, le niveau d'abstraction offert par une boîte à outils conditionne L'effort de développement d'une interface utilisateur et fixe les styles d'interface et d'interaction. Plus il est élevé, plus le temps d'implémentation est réduit et plus l'interface produite est normalisée. Il n'en reste pas moins vrai que l'apprentissage de la programmation pour une boîte à outils donnée reste élevé. Les squelettes d'application visent à réduire ce temps de formation.

6-2/ *Les squelettes d'application :*

Au contraire des boîtes à outils, les squelettes d'application réalisent les fonctions générales d'une interface sous forme d'un logiciel réutilisable et extensible. L'utilisation d'un squelette consiste ‹ greffer des composants spécifiques au système à implémenter. En général, un squelette est construit en corrélation avec une boîte à outils.