الجمهوريـــة الجزائريـــة الديمقراطيـــة الشعبيـــة

وزارة التعليـم العالـي و البحث العلمي
جامعـة حسيبـة بن بوعلـي بالشلف
كليـة العلـوم الدقيقة و الاعلام الالـي
قسـم الاعلام الآلـي

High Education and Research Ministry
Hassiba Benbouali University of Chlef
Faculty of Exact Sciences and Informatics
Computer Science Department

Polycopié de cours

Interface Homme-Machine

Pour la M1 Ingénierie des logiciels (IL)

Arrété 1352 du 09 aout 2016

Par : BECHAR Rachid

Maitre de conférences classe A

Année universitaire 2024/2025

Contenu du cours

Chapitre I: Introduction

I.1 Quelques définitions

L'ergonomie : Le terme "Ergonomie" vient du grec ergon (travail) et nomos (lois ou règles). L'ergonomie peut donc être définie comme " la science du travail ". Elle comprend différentes disciplines (physiologie, psychologie, sociologie, médecine,…) qui s'associent pour accéder à une connaissance scientifique de l'homme au travail.

L'ergonomie est une discipline, une démarche, un point de vue que l'on peut appliquer à tout ce qui nous entoure. Au sens le plus général, elle concerne les outils utilisés par l'être humain. Dans les années 50, Alain Wisner, un des pionniers de l'ergonomie en France, a donné la définition suivante :

« L'ensemble des connaissances scientifiques relatives à l'homme nécessaires pour concevoir des outils, des machines et des dispositifs qui puissent être utilisés avec le maximum de confort, de sécurité et d'efficacité ».

L'Interaction Homme-Machine : C'est une discipline qui correspond à l'étude de l'Homme et des technologies informatiques et aussi les liens entre ces deux disciplines. L'Interaction Homme-Machine (IHM) est la discipline consacrée à la conception, la mise en œuvre et à l'évaluation des systèmes informatiques interactifs destinés à des utilisateurs humains ainsi qu'à l'étude des principaux phénomènes qui les entourent.

Remarque : *Une IHM est en général l'outil permettant à l'utilisateur d'interagir, interférer ou communiquer avec un système complexe et interactif que nous pouvons assimiler à une boite noir. Pour notre cours, on s'intéresse seulement aux IHM destinées à des systèmes informatiques.*

L'interaction homme/machine peut être représentée par le schéma suivant :

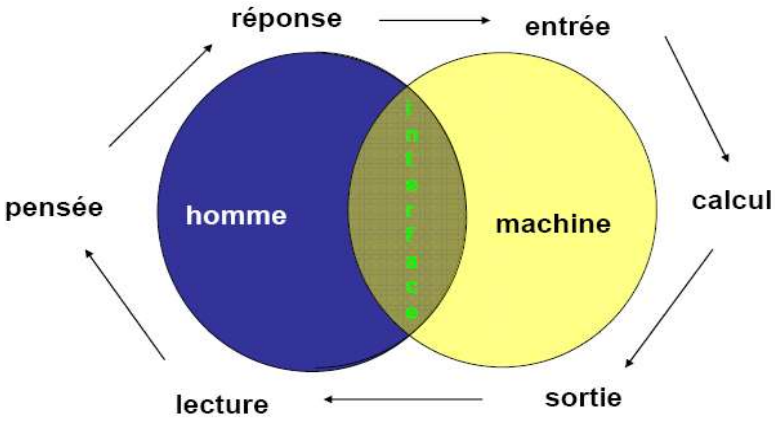


Figure 1: Interaction Homme/Machine

Un système interactif : est un système dont le fonctionnement dépend d'informations fournies par un environnement externe qu'il ne contrôle pas [*Wegner, 1997*]. Les systèmes interactifs sont également appelés ouverts, par opposition aux systèmes fermés –ou autonomes–dont le fonctionnement peut être entièrement décrit par des algorithmes.

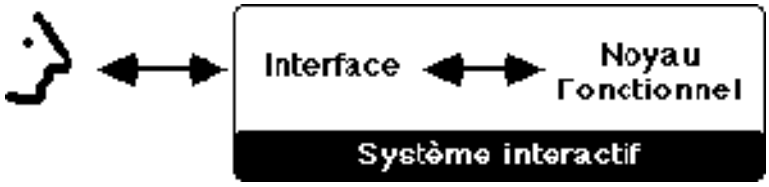


Figure 2: Schéma général d'un système interactif

Une interface : est l'ensemble des dispositifs matériels et logiciels qui permettent à un utilisateur de commander, contrôler ou superviser un système interactif.

I.2 Les enjeux des IHM

Le concepteur et/ou développeur d'une interface homme-machine doit prendre en considération les aspects ergonomiques qui facilitent l'utilisation de son système et son acceptation par l'utilisateur final qui ne voit pas les autres contraintes dictées par les technologies utilisées. Ceci présente un certain nombre d'enjeux et de difficultés telles que :

- Mettre l'utilisateur au centre de la démarche de conception.
- Passer d'une vision du progrès centrée sur le développement des technologies à une démarche centrée sur les utilisateurs.
- Créer des objets et des systèmes faciles à utiliser avec confort et sécurité.

I.3 Les risques d'une mauvaise interface

Si l'utilisateur n'est pas bien pris en considération lors de la conception/développement d'une interface, cela conduit à un mauvais produit présentant les risques suivants :

- Coût d'apprentissage élevé (par formation),
- Perte de productivité de l'utilisateur,
- Utilisation incomplète du système,
- Coût de maintenance du aux mauvaises manipulations,
- Perte de crédibilité,
- Rejet par les utilisateurs.

I.4 IHM : Un domaine pluridisciplinaire

Il conclut des éléments évoqués précédemment que le domaine de l'interaction homme-machine doit prendre en considération plusieurs aspects techniques, artistiques, ergonomiques, commerciaux et humains aussi. De ce fait, l'IHM fait appel à plusieurs disciplines et qui sont :

- Facteurs humains : psychologie, ergonomie, sociologie,
- Aspects informatiques : génie logiciel, langages, système, réseau, base de données, dispositifs d'entrée-sortie, …
- Conception (design): art graphique, design industriel, …

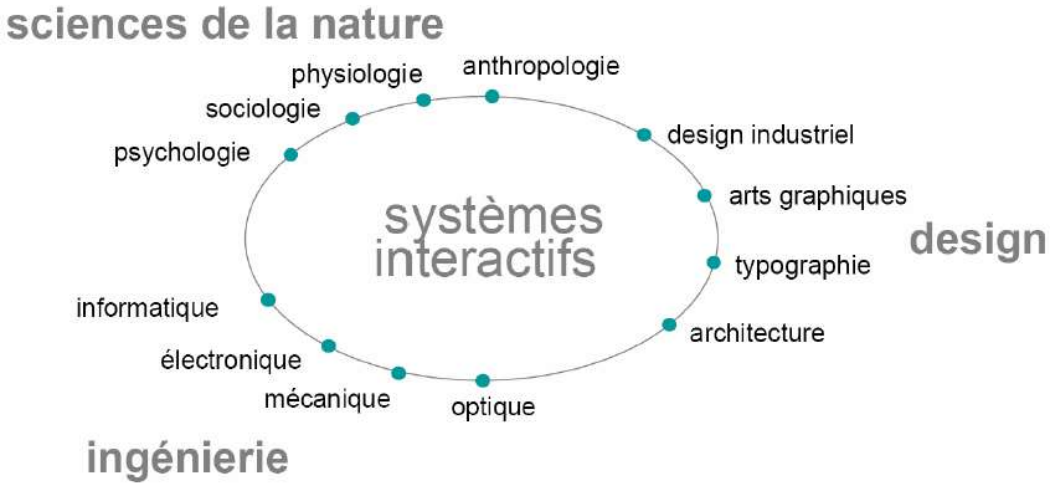


Figure 3: L'IHM est un domaine pluridisciplinaire

I.5 Organisation du cours

Il résulte de toutes ces définitions et remarques que l'étude et la réalisation d'une IHM est un travail qui a plusieurs axes. En plus des notions données dans le cours IHM de la 3eme année Licence informatique, nous verrons dans ce cours plus de détails sur les différents modèles et outils, pour cela, le présent cours sera organisé en chapitres suivants :

1. Introduction
2. Cycle de vie du logiciel interactif
3. Modèles d'architecture pour les IHM (Seeheim, PAC, PAC-Amodeus, un peu de MVC)
4. Catégories d'outils pour la construction des IHM (Logiciels graphiques de base, boîtes à outils / frameworks, générateurs interactifs)
5. Prise en compte des utilisateurs dans le processus de conception des IHM
6. Présentation de l'API Swing de java
7. Méthode d'implémentation efficace du modèle PAC basée sur l'utilisation de design patterns

Chapitre II: Cycle de vie du logiciel interactif

II.1 Modèles de cycle de vie en IHM

Il est clair qu'une IHM est un produit logiciel comme tout autre logiciel et qui passe par les étapes connues en génie logiciel qui sont exprimés en différents modèles (cascade, incrémental, V, …). Généralement ces étapes se résument comme suit :

1. Préparation du cahier de charges
2. Analyse des besoins
3. Conception
4. Implémentation ou développement
5. Déploiement et tests
6. Maintenance

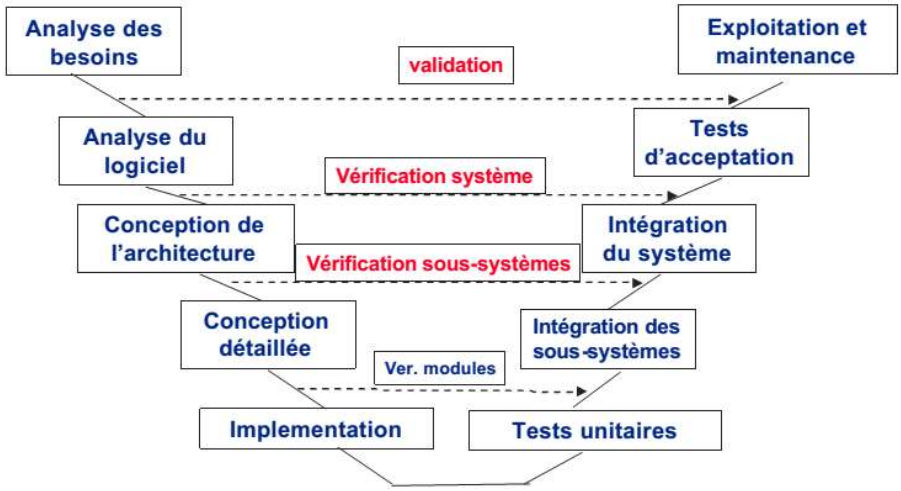


Figure 4: Cycle de vie d'un produit logiciel selon le modèle en V

Quand il s'agit de produire une IHM, il suffit de préciser à chaque étape quels sont les points à cibler. Il est clair que la conception d'une IHM doit être basée sur l'utilisateur et ses actions, les éléments techniques de l'interaction et aussi l'utilisabilité du produit sans oublier les aspects de la qualité ergonomique, ce qui nécessite une certaine validation.

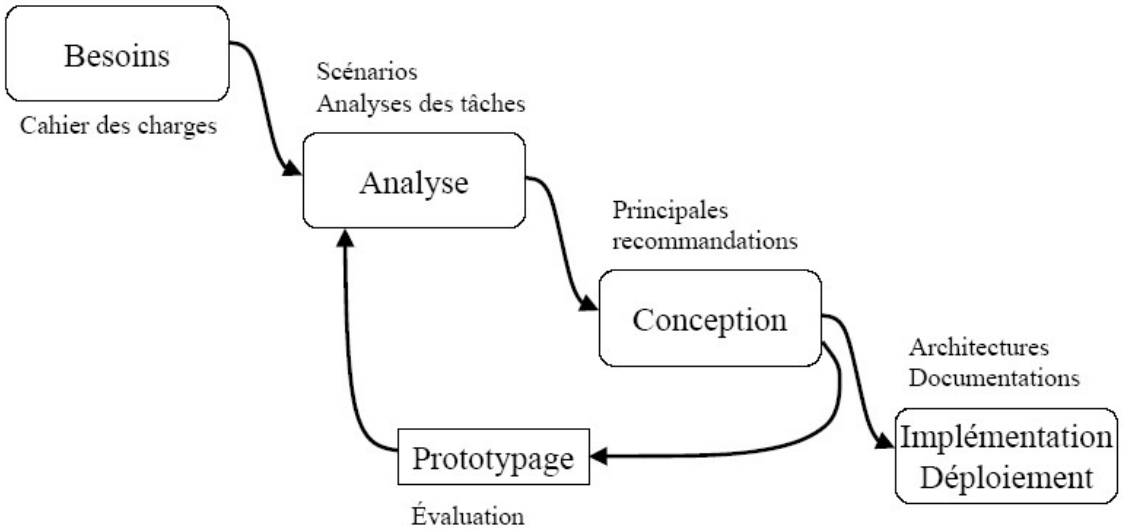


Figure 5: Première schématisation du cycle de vie d'une IHM

Pour arriver à un tel but, le développeur d'une IHM doit prendre en considération tous les principes et recommandations connus dans ce domaine surtout en termes de règles ergonomiques portant sur l'utilité et l'utilisabilité du système. Il faut donc se rappeler des critères ergonomiques tels que le guidage, la concision, la gestion d'erreurs … etc (voir les critères de Bastien et Scapin par exemple). Pratiquement, cela fait appel aux guides de style qui doivent être au centre des préoccupations.

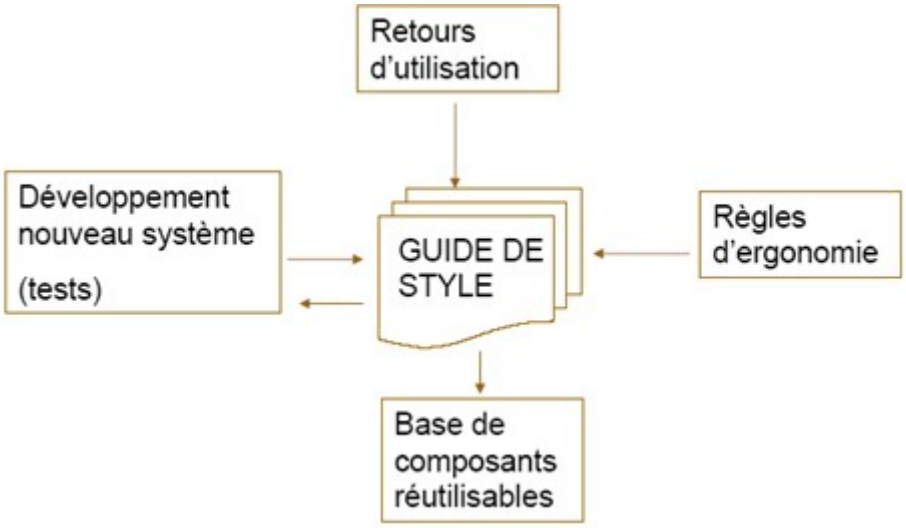


Figure 6: Apports des guides de style et des règles ergonomiques

En fait, il n y a pas de réelle méthode pour la prise en compte des problèmes ergonomiques en dehors des cycles de production d'un produit logiciel (notions du génie logiciel). Mais quand il s'agit d'une IHM, le concepteur/développeur doit se concentrer sur les aspects qui concernent l'interaction entre l'utilisateur et la machine. C'est-à-dire, il doit préciser et fixer ses choix dans chaque étape du modèle en fonction des règles et des paramètres ergonomiques. Si on considère le modèle basique en cascade ou itératif par exemple, il devient comme suit :

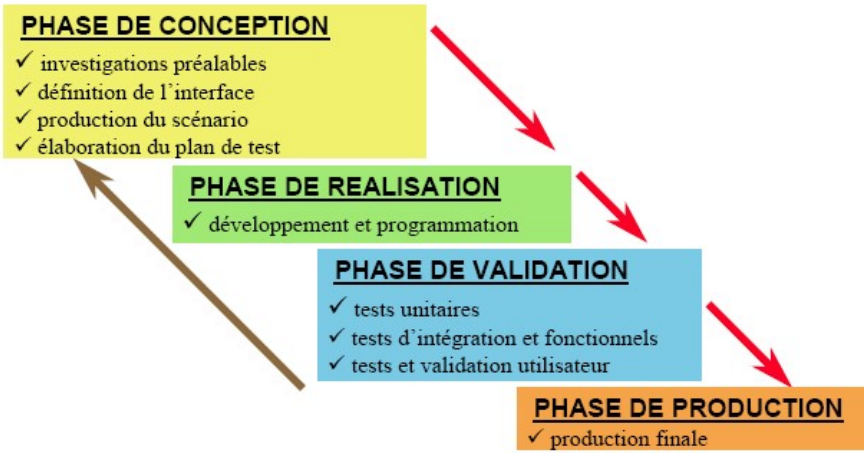


Figure 7: Cycle de production d'une IHM (modèle itératif)

Ce modèle est connu par son mode itératif qui est considéré comme inconvénient en génie logiciel. Le modèle en V est un remède pour cela. Quand il s'agit d'une IHM, le modèle en V devient le suivant :

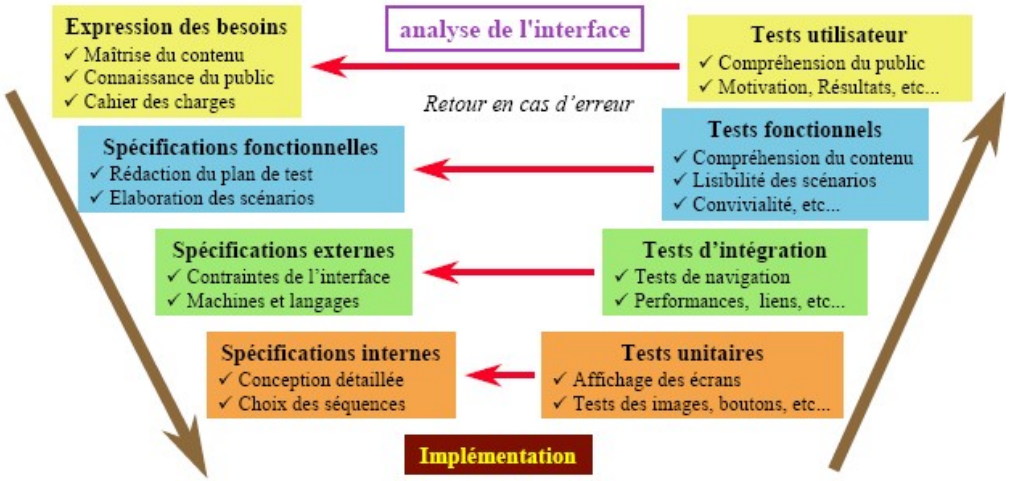


Figure 8: Cycle de production d'une IHM (modèle en V)

Pour arriver au produit final, le concepteur d'une interface homme-machine aura besoin des outils suivants :

- ***Maquette*** : ensemble d'objets graphiques donnant une image de l'écran utilisateur, mais sans les fonctionnalités.
- ***Prototype*** : développement d'un produit qui est soumis à des éventuelles améliorations par raffinement/extension pour arriver au produit final.

Après avoir établit le cahier des charges d'une IHM, les étapes d'analyse des besoins et de conception sont primordiales. En effet, lors de l'analyse, le concepteur doit fréquenter l'utilisateur final en essayant de collecter toutes les informations sur ces besoins, fonctionnalités souhaitées, préférences, profils, … etc. par la suite, le concepteur doit les interpréter, les modéliser et les mettre en œuvre selon des choix de conception adaptés. Dans ce qui reste de ce cours, nous allons détailler ces deux étapes importantes dans le processus de développement d'une interface logicielle.

II.2 Analyse des besoins

Un projet d'IHM consiste à concevoir un nouveau système mais parfois il s'agit à remplacer ou à mettre à jour un système existant. Quelle que soit la situation initiale et quel que soit l'objectif du projet, les besoins, les exigences et les attentes de l'utilisateur doivent être discutés, affinés, clarifiés et probablement redéfinis. Cela nécessite une compréhension des aspects suivants:

- Les utilisateurs, leurs profils et leurs capacités.
- Leurs tâches et objectifs.
- Les conditions dans lesquelles le produit sera utilisé.
- Les contraintes sur les performances du produit.

Un besoin est une déclaration concernant un produit prévu qui précise ce qu'elle devrait faire ou comment il devrait fonctionner. L'un des objectifs de l'analyse des besoins est de rendre les exigences précises, sans ambiguïté et aussi clair que possible. Par exemple pour une application GPS d'un smartphone pourrait sur le temps de charger une carte qui doit être minimal.

Il existe plusieurs techniques appliquées pour l'analyse des besoins en IHM telles que :

- L'interview ou l'entretien
- Le questionnaire
- Observation directe et indirecte
- Groupes de discusion
- Etude de la documentation
- Analyse des projets similaire

Par exemple, l'entretien pour cibler des groupes d'utilisateurs spécifiques, L'observation pour comprendre le contexte d'exécution des tâches, le questionnaire pour atteindre une population plus large et des groupes de discussion pour parvenir à un consensus.

On donne ici quelques lignes directrices lors de la collecte de données concernant les besoins des utilisateurs :

1. Se concentrer sur l'identification des besoins des parties prenantes.
2. Impliquer tous les groupes de parties prenantes.
3. Impliquer un seul représentant de chaque groupe de parties prenante n'est pas suffisant, surtout pour les groupes volumineux.
4. Soutenir les sessions de collecte de données avec des accessoires appropriés, tels que le prototypage si possible.

Le processus d'analyse des besoins prend en considération le fait que l'utilisateur va faire une tâche d'interaction dans un contexte précis. Donc la tâche est le point essentiel dans tout ça dont il est nécessaire d'avoir des outils ou des modèles pour représenter et analyser ces tâches.

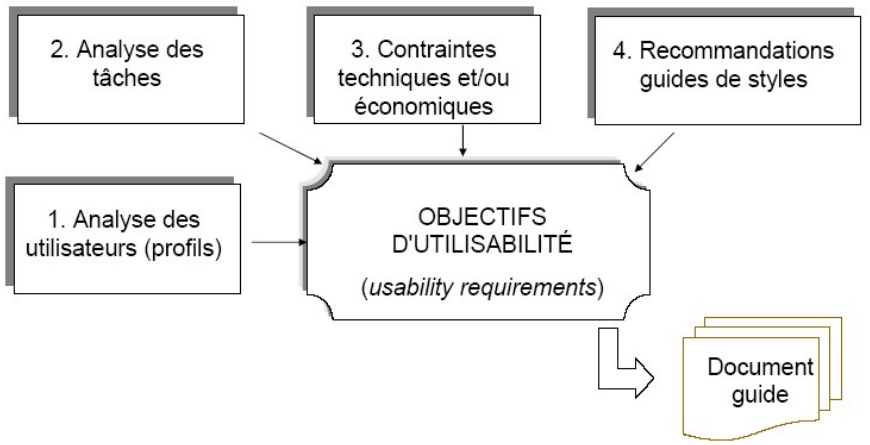


Figure 9: Principaux aspects de l'analyse des besoins en IHM

La description des tâches

La description des tâches a été utilisée dans le développement de logiciels pour de nombreuses années via des modèles et des méthodes différentes dont les plus courantes sont:

1. Utilisation des scénarios et personas : basée sur l'étude de séquencement des tâches (scénarios) et des utilisateurs particuliers (personas).
2. Les cas d'utilisation : expression des acteurs et des fonctionnalités connue en conception UML
3. Cas d'utilisation essentiels (connues par task cases) : une sorte d'abstraction des scénarios.

II.3 Conception, développement et tests

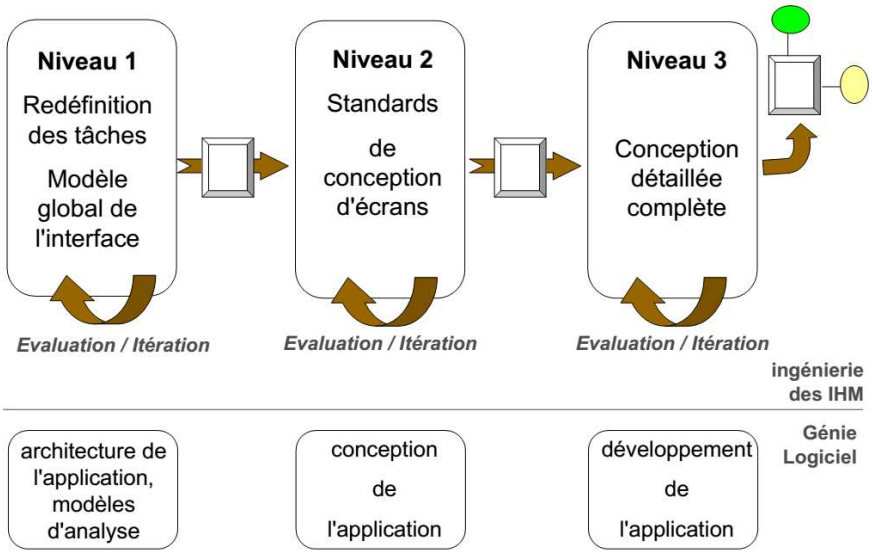


Figure 10: Niveaux de conception d'IHM

La conception d'une IHM passe par des étapes de différents niveaux de détails commençant par le modèle global jusqu'à l'élaboration d'un modèle conceptuel détaillé qui permettra la mise en oeuvre du projet par la suite.

Niveau1 : ça concerne la redéfinition des tâches, Modèle global de l'interface, Architecture et présentation générale de l'interface. Ce niveau est orienté produit / processus. Les aspects à déterminer dans cette étape sont :

1. définition et représentation des objets ou processus métiers,
2. types de fenêtres utilisés selon la tâche,
3. principaux affichages (style et non pas contenu),
4. navigation (structuration des menus, ...).

Niveau2 : C'est le modèle conceptuel de l'interface dont lequel on doit spécifier les standards de conception d'écran afin d'assurer une certaine cohérence. Ces standards concernent en premier lieu :

1. Utilisation des différents contrôles,
2. Position et format des composants,
3. Polices et couleurs,
4. Terminologie et messages,
5. Sémantique des événements d'interaction (clics, raccourcis, ...).

Niveau3 : c'est la conception détaillée et le développement de l'interface avec une simple instanciation des niveaux précédents tout en respectant les recommandations ergonomiques spécifiques, en plus de :

1. Identification des chemins de navigation entre écrans,
2. Implantation des barres de menus et autres contrôles,
3. Implantation du contenu de toutes les fenêtres.

II.4 Modèles enrichis pour IHM

L'approche traditionnelle de génie logiciel distingue généralement six phases dans la vie d'un logiciel : étude préalable, spécification, conception, implémentation, test et la phase de maintenance. Cela offre un cadre général très utile pour les concepteurs. Cependant, les IHM sont caractérisées par certains aspects qui ne sont pas cités directement dans ces modèles. L'analyse et la modélisation des utilisateurs et des tâches humaines ne sont pas préconisées et

sont laissées à l'appréciation des concepteurs. Afin de remédier à ces lacunes, d'autres modèles enrichis pour les IHM ont été mis au point pour le développement des applications hautement interactives.

L'idée d'enrichissement des cycles de vie pour les IHM, repose sur l'intégration, d'un point de vue méthodologique, des aspects fondamentaux de l'interaction homme-machine comme la modélisation des tâches humaines, la réalisation itérative des prototypes ou l'évaluation des systèmes interactifs. Parmi ces modèles, il y a: le modèle de Long, le modèle en étoile, et le modèle ∇(Nabla).

II.3.1 Modèle de Long

Proposé en 1990, ce modèle est proche du modèle classique en cascade . L'IHM est positionnée dans l'étape de conception tout en insistant sur l'importance de l'évaluation et des itérations lors du projet.

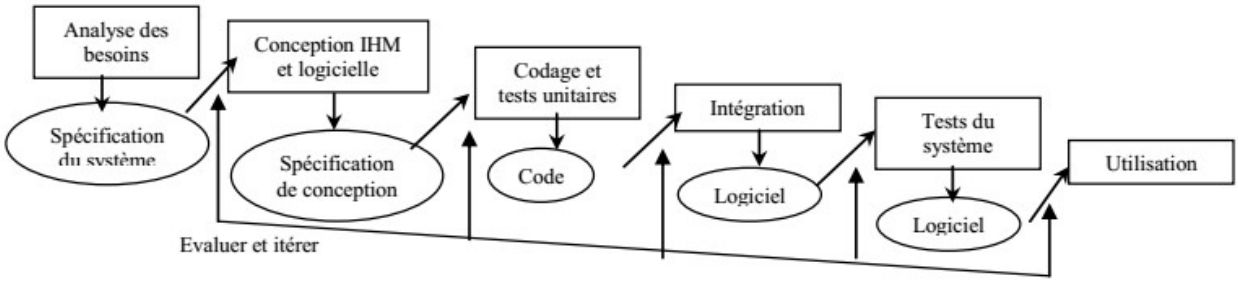


Figure 11: Modèle de Long

Ce modèle est loin d'être parfait. Il a toutefois le mérite d'être incitateur sous l'angle des interactions homme-machine.

II.3.2 Modèle en étoile

Le modèle proposé par Hartson en 1989, appelé aussi modèle en étoile situe l'évaluation au centre même du cycle complet. Il préconise des interactions/itérations entre l'évaluation et chacune des autres étapes. L'étape d'évaluation est vue comme une étape intermédiaire permettant de protéger l'équipe de développement d'un rejet terminal.

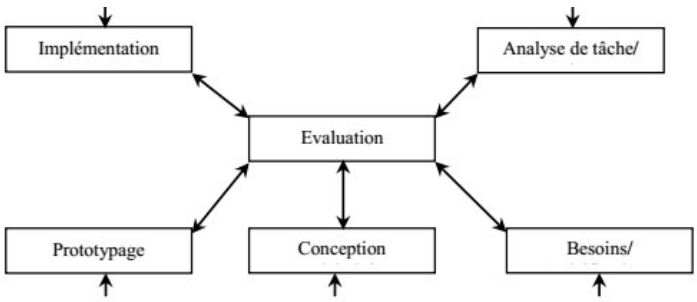


Figure 12: Modèle en étoile

Le modèle en étoile n'impose pas d'ordre a priori dans l'accomplissement des étapes du processus, bien qu'en pratique, les activités de développement soient reportées en fin de cycle. Il sous-entend une conception participative visant la détection précoce de problèmes d'utilisabilité, requérant une forte adhésion de l'utilisateur par cette implication centrale.

II.3.3 Modèle ∇(Nabla)

Le modèle ∇ proposé par Kolski en 1997 (figure ci-dessous) a pour objectif de situer les différentes étapes du génie logiciel nécessaires pour développer un système interactif, tout en différenciant l'interface proprement dite (partie gauche du modèle) des modules applicatifs ou d'aide éventuellement accessibles à partir de ceux-ci (partie droite). Une des caractéristiques marquantes du modèle est de positionner des étapes, inexistantes dans les modèles classiques, où les facteurs humains doivent être considérés par l'équipe de développement.

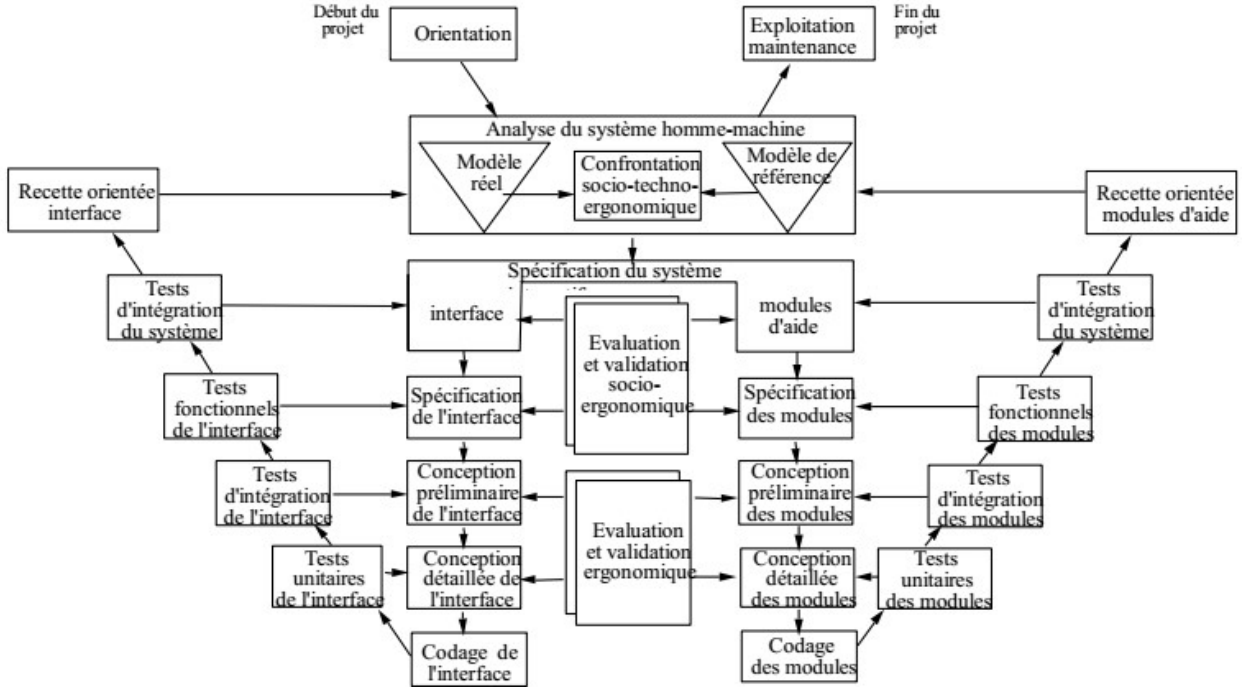


Figure 13: Modèle Nabla

II.5 Méthodes de conception adaptées pour IHM

Par la suite, de nouvelles méthodes de conception logicielle sont apparues et qui offrent une certaine souplesse avec une prise en compte des problèmes liés à l'interaction homme-machine en particulier l'utilisateur de l'interface. Ces méthodes sont caractérisées par:

- ✓ Centrées sur l'utilisateur : pour prendre en considération le client mais aussi les utilisateurs.
- ✓ Avec prototype: pour faire apparaitre les idées, et les résultats des projets.
- ✓ Evaluation précoce: le retours utilisateur est bien pris en compte avec les éventuelles erreurs.
- ✓ Progressives: afin d'améliorer les propositions, le processus est itérative et incrémentale.

Parmi ces méthodes qui sont mieux adaptées à la conception des IHM, il y a la méthodes Agile et le Design thinking.

II.4.1 Méthodes Agiles

L'approche des méthodes Agiles a été créée par regroupement de plusieurs méthodes existantes partageant des valeurs communes telles que:

- ✓ Un développement est itératif, incrémental et adaptatif;

- ✓ Une collaboration forte avec le client ou l'utilisateur;
- ✓ Avoir un produit logiciel opérationnel;

Comme exemples des méthodes agiles de développement logiciel, on peut citer:

- ✓ Rapid Application Development (RAD, 1991);
- ✓ Rational Unified Process (RUP, 1995);
- ✓ Dynamic Systems Development Method (DSDM, 1995);"
- ✓ Scrum (1996);
- ✓ Feature Driven Development (FDD, 1999);"
- ✓ Extreme Programming (XP, 1999);
- ✓ Adaptive Software Development (ASD, 2000);
- ✓ Crystal Clear (2004);

Le principe général en Agile est basé sur les aspects suivants:

- ➢ Conception flexible, itérative et incrémentale; □
- ➢ Projet fragmenté en plusieurs sous-parties avec possibilité de livraison continue; □
- ➢ Adaptation au changements par définition d'objectifs à court terme et ré-ajustables;
- ➢ Evaluations précoces;
- ➢ Le client ou l'utilisateur a un rôle pivot au cœur du processus avec des tests;

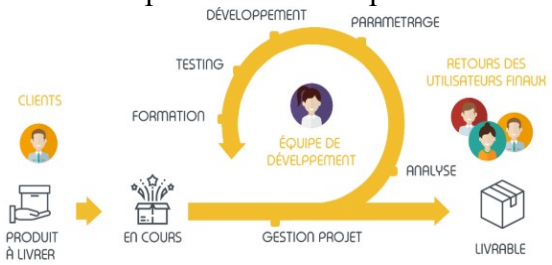


Figure 14: Pricipe générale d'une méthode Agile

Pour la conception des IHM, la méthode SCRUM semble plus adaptée comme méthode Agile. Elle est basée sur un découpage du projet en boites temporelles appelées «Sprints» de durées fixes mais différentes dont chaque sprint commence par une planification et se termine par une démonstration ou évaluation.

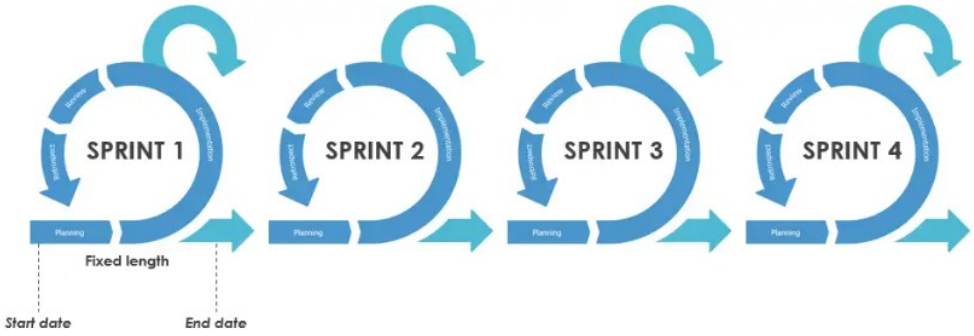


Figure 15: Découpage en Sprints (méthode SCRUM)

II.4.2 Le Design Thinking

Le Design Thinking ou pensée design est une démarche d'innovation qui qui incite à redéfinir les problèmes, identifier des solutions et proposer des alternatives. C'est une sorte de conception créative an passant par les étapes suivantes: empathie, définition, idéation, prototype et test.

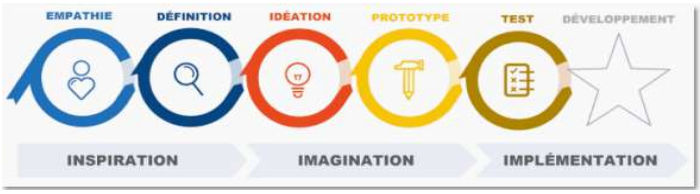


Figure 16: Le design thinking

Comme processus de développement qui utilise cette méthodologie d'innovation, Donald Norman a créé le processus dit «double diamant» basé sur quatre étapes à savoir la découverte, la définition, le développement et enfin la livraison formant une carte visuelle que les concepteurs peuvent utiliser pour organiser leurs pensées et améliorer le processus créatif.

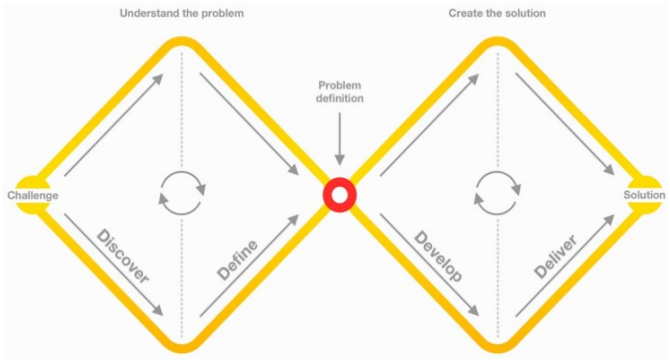


Figure 17: Principe et étapes du processus double diamant

(Ref: Livre de Donald Norman «The Design of Everyday Things")

Chapitre III: Modèles d'architecture pour les IHM (Seeheim, PAC, PAC-Amodeus, un peu de MVC)

C'est après l'invention de la souris en 1964 et l'apparition du graphisme et du multifenêtrage en 1969 que la recherche dans le domaine des IHM s'est lancée. En 1983, la ville de Seeheim a accueilli un workshop sur le rôle, le modèle, la structure et la construction des systèmes de gestion des interfaces usagers (User Interface Management Systèmes : UIMS). Lors de ce workshop les intervenants ont défini un modèle d'architecture pour les IHM qui porte le nom de la ville : modèle de Seeheim. Par la suite, ce modèle a servi de base pour la naissance de nouveaux modèles plus élaborés et influencés par le paradigme orienté objet tels que les modèles PAC et MVC.

Les premiers modèles étaient des modèles conceptuels qui avaient pour but l'identification des différentes parties d'une application interactive, les modèles les plus récents se sont dirigés plus vers l'implantation. Dans ce qui suit, nous présenterons cinq modèles d'architecture pour les applications interactives. Tous ces modèles ont eu un impact important dans le domaine des IHM.

III.1 Modèle Seeheim

Seeheim est le premier modèle d'architecture largement accepté qui part d'une approche linguistique. Celui-ci est issu d'un groupe de travail sur les systèmes interactifs ayant eu lieu à Seeheim (une ville allemande) en 1985. C'est un modèle qui est destiné principalement au traitement lexical des entrées et sorties dans les interfaces textuelles. S'il est peu utile aujourd'hui pour décrire les systèmes interactifs hautement graphiques, il a servi de base à beaucoup d'autres modèles.

Seeheim divise l'interface en trois couches logicielles selon une approche linguistique :

1. **Présentation** : représente la partie de l'application directement liée aux périphériques d'entrées/sorties. Il gère l'affichage à l'écran et les périphériques d'entrée (clavier, souris, etc.), et il est responsable de l'apparence de l'interface.
2. **Contrôleur de dialogue** : gère le séquencement de l'interaction en entrée et en sortie. Il maintient un état lui permettant de gérer les modes d'interaction et les enchaînements d'écrans, c'est la partie qui décide de l'accès ou du refus des interactions de l'utilisateur.
3. **L'interface du noyau fonctionnel** : est la couche adaptative entre le système interactif et le noyau fonctionnel. Elle convertit les entrées en appels du noyau fonctionnel et les données abstraites de l'application à des objets présentables à l'utilisateur.

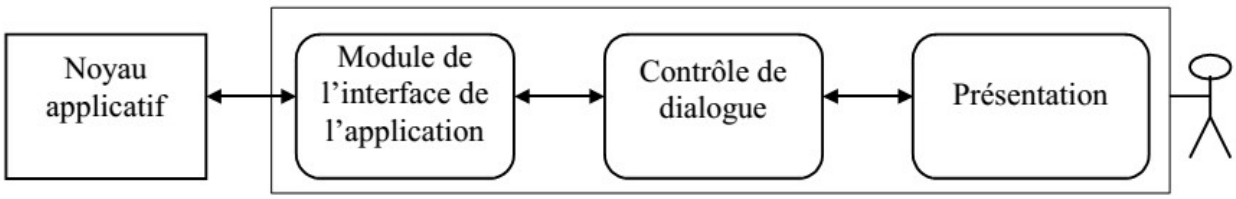


Figure 18: Composantes du modèle Seeheim

Le noyau applicatif regroupe toutes les fonctions réalisées par l'application, ce module est la partie non interactif de l'application.

Le modèle de Seeheim définit clairement les rôles des quatre modules d'une application interactive au niveau conceptuel. Malheureusement, son implantation présente des difficultés

puisque les rôles des trois modules de l'interface ne sont pas suffisamment explicités. Cette architecture a été longuement critiquée du fait de sa structure en couches, de la centralisation du contrôle et de la passivité de son noyau applicatif.

III.2 Le modèle Arch

Comme révision de ce modèle, un autre élément a été ajouté pour prendre en compte le retour sémantique rapide par exemple, lors d'une opération de glisser-déposer, il peut s'avérer utile de modifier instantanément l'apparence de l'icône-cible pour indiquer si l'opération est valide. Dans ce cas, la couche de présentation doit permettre au contrôleur de dialogue d'accéder directement aux informations sémantiques du noyau fonctionnel.

Le modèle Seeheim révisé est connu aussi par le modèle Arch. Il introduit la notion adaptateur et d'objet pour chaque module pour avoir des objets d'intéraction, de présentation et des objets de domaine afin d'ajouter plus de détails pour la structure surtout avec l'apparition de boites à outils qui ont fait une révolution en développement graphique.

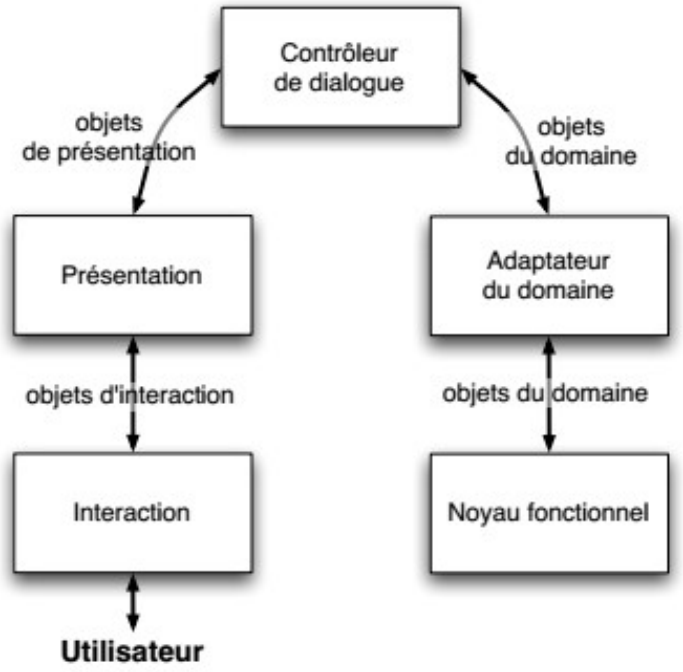


Figure 19: Le modèle Arch

III.3 Le modèle MVC

Avec l'arrivée de la programmation orientée objet et par la suite les systèmes multi-agent, d'aute modèles d'architecture IHM ont connu le jour en se basant sur l'aspect agent. La structure d'un système interactif est vue comme un système de plusieurs agents dont chaque agent est responsable d'une facette différente mais complémentaire.

Avant d'expliquer les composantes du MVC, on examine le problème classique de la représentation et d'affichage de données provenant d'un système d'information d'entreprise sur les écrans de différents utilisateurs de niveau différents (simple opérateur, administrateur, gérant, ...), cela dépend de la technologie matérielle et logicielle utilisée pour l'interface. Donc, il est clair que différentes présentations sont possibles dont chacune est dite «Vue» qui présente le même système d'information dit «Modèle» et les interactions seront gérées par un ou plusieurs «Contrôleur».

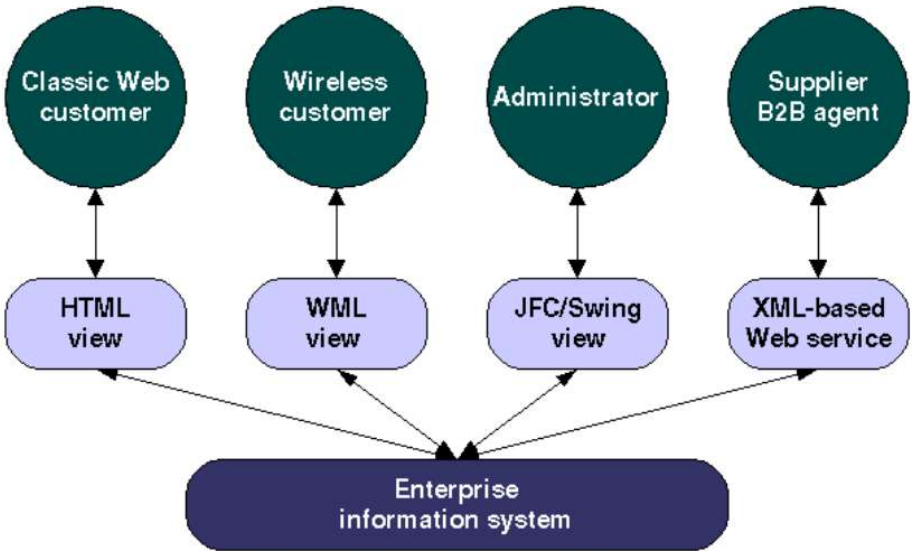


Figure 20: Différentes vues pour un seul SI

Le modèle MVC (Model View Controler) est le plus connu de ce type. Il est composé d'un triplet de composants autonomes qui communiquent entre eux comme le montre la figure suivante:

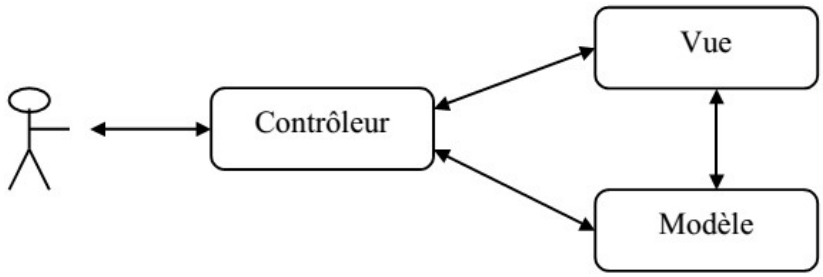


Figure 21: Le modèle MVC

MVC a été introduit comme architecture de référence dans l'implémentation des interfaces utilisateurs de l'environnement Smalltalk. C'est une architecture basée sur l'aspect agent, dont les principales motivations sont la compatibilité avec les langages à objets et la conception itérative.

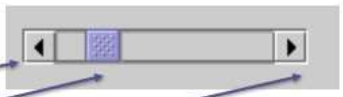
MVC décompose les systèmes interactifs en une hiérarchie d'agents. Un agent MVC consiste en un modèle muni d'une ou plusieurs vues et d'un ou plusieurs contrôleurs:

- **Le modèle** est la structure de données que l'on veut représenter à l'écran et qui est composée d'un certain nombre d'objets.
- **La vue** est la représentation externe du modèle. C'est par elle que l'utilisateur perçoit les objets du système et que le modèle reflète ses changements. Dans une application, à un même modèle peuvent correspondre plusieurs vues.
- **Le contrôleur** régule les interactions entre la vue et le modèle. Il est chargé de gérer les actions de l'utilisateur sur la vue, et il informe le modèle des changements faits sur celle-ci. Le modèle modifie son état et informe la vue du nouvel aspect qu'elle doit prendre.

Chacun des trois composants de la triade MVC est un objet à part entière. Au sens de la programmation orientée objet, une classe de Modèle peut être compatible avec plusieurs classes de vues et de contrôleurs.

Exemple : Le composant JSlider en Java Swing :

Quelles sont les données associées à un slider ?



- *Modèle* :
  – valeur minimale = 0
  – valeur courante = 15
  – valeur maximale = 100

- *Vue :* apparence de l'objet selon les données courantes
- *Contrôleur* :

– Traiter les clics de souris sur les boutons terminaux 

– Gérer les drag*s* de souris sur l'ascenseur 

Enfin, le principe général de fonctionnement du MVC put être donné par le diagramme de séquence suivant:

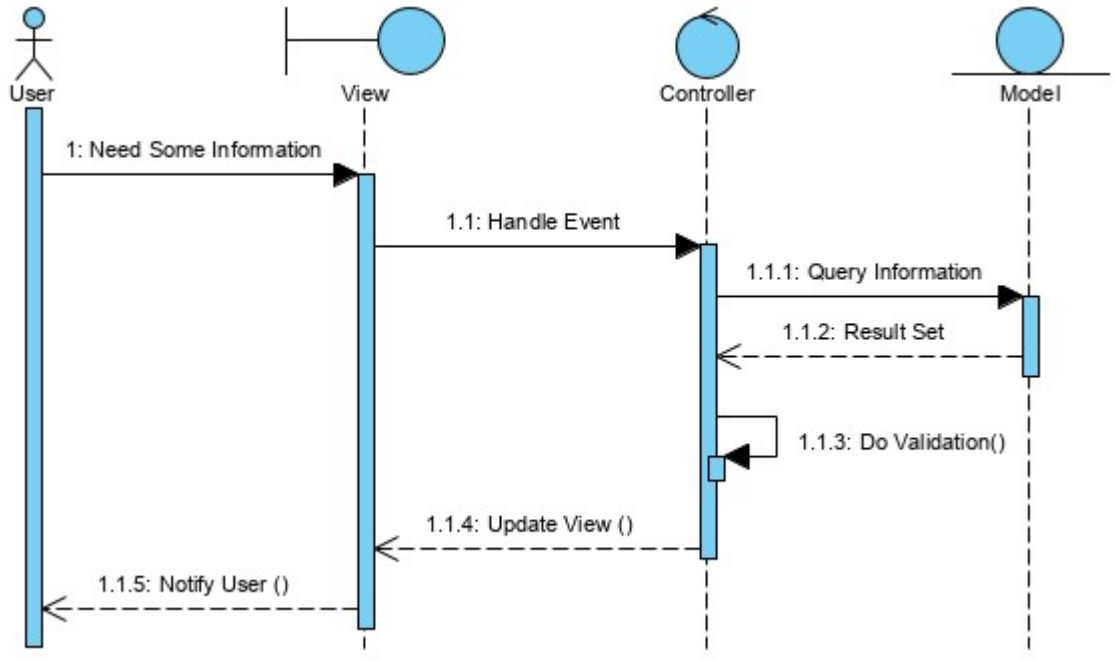


Figure 22: Un scénario pour le MVC

III.4 Le modèle PAC

L'architecture PAC (Présentation, Abstraction et Contrôle) proposée par Coutaz en 1990 (université de Grenoble) se base sur une hiérarchie d'objets pour la modélisation des applications interactives. PAC est un modèle multi-agent qui a comme principes directeurs le concept d'agent à facettes et une décomposition récursive. Un système interactif est modélisé par une hiérarchie d'agents PAC, comme le schématise la figure ci-après.

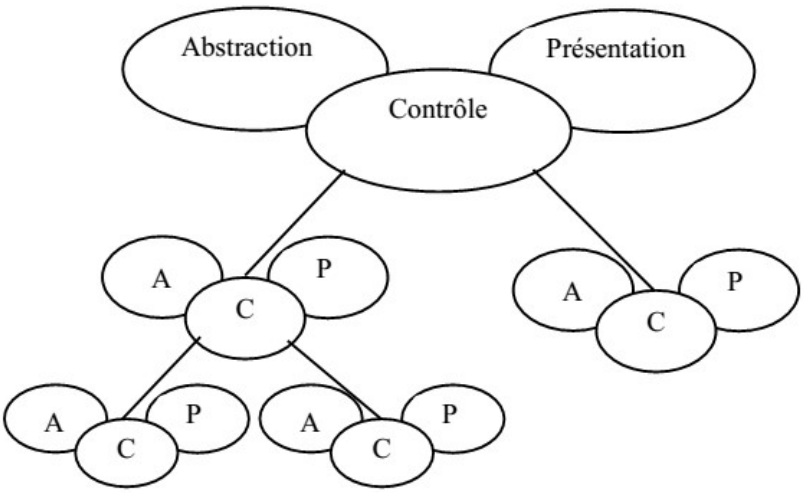


Figure 23: le modèle PAC

Un agent PAC adopte trois perspectives complémentaires : la Présentation, l'Abstraction et le Contrôle.

- **La Présentation** définit le comportement perceptible de l'agent pour un agent humain, elle concerne à la fois les entrées et les sorties c'est-à-dire les modalités d'action accessibles à l'utilisateur et la restitution perceptible.
- **L'Abstraction**, avec ses fonctions et ses attributs internes, définit la compétence de l'agent indépendamment des considérations de présentation ; elle constitue le noyau fonctionnel de l'agent.
- **Le Contrôle** a un double rôle : il sert de pont entre les facettes Présentation et Abstraction de l'agent, et il gère des relations avec d'autres agents PAC.

La figure suivante montre un exemple d'agent PAC pour interroger une base de données.

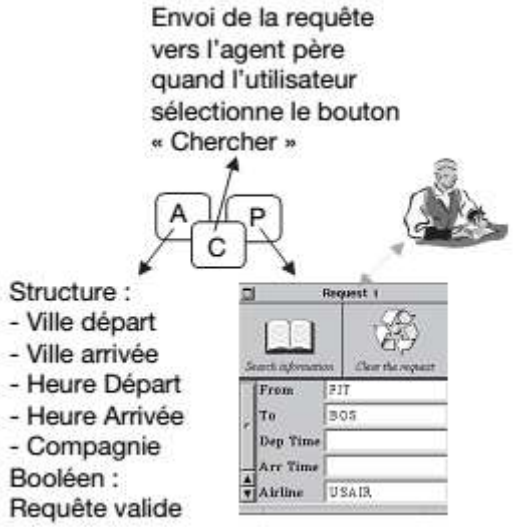


Figure 24: Exemple d'agent PAC : une requête de BDD

Tout échange d'informations entre l'Abstraction et la Présentation s'effectue via le Contrôle. C'est aussi par leur Contrôle que deux agents PAC communiquent.

Le modèle PAC se distingue du modèle MVC par la définition récursive des systèmes interactifs. Mais ici, le rôle de ces modules peut être diffèrent :

- La présentation définit le comportement en entrée et en sortie de l'agent, tel qu'il est perçu par l'utilisateur.
- L'abstraction encapsule la partie sémantique de l'agent.
- Le contrôle maintient la consistance entre la présentation, l'abstraction et communique avec les autres agents.

Cependant cette hiérarchie d'agents PAC peut causer un ralentissement du système à cause des échanges de messages entre les différents contrôleurs de l'architecture.

III.5 Le modèle PAC-Amodeus

PAC-Amodeus (Par Nigay en1993) essaye de regrouper les approches linguistiques et à agents en proposant un modèle hybride. Il réutilise les anciens modèles comme Seeheim et décrit le contrôleur de dialogue avec une hiérarchie d'agents PAC.

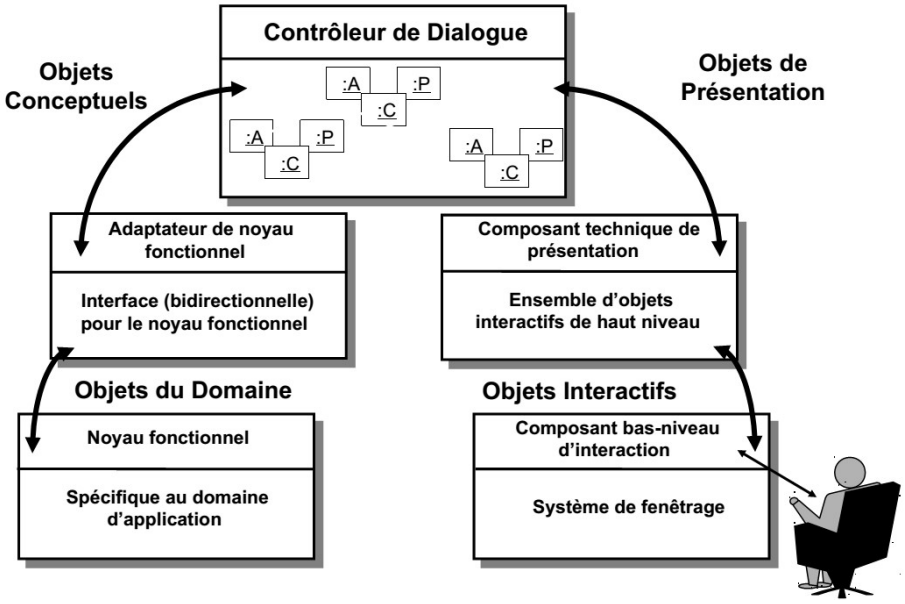


Figure 25 : Le modèle PAC-Amodeus

Le Noyau Fonctionnel (NF) réalise les concepts du domaine. L'Adaptateur du Noyau Fonctionnel (ANF) sert d'interface entre les objets du domaine et les objets conceptuels exportés vers l'utilisateur. Le Contrôleur de Dialogue (CD), clé de voûte du système interactif, prend en charge l'enchaînement des tâches, gère chaque fil de dialogue au moyen d'une grappe d'agents PAC, et assure la correspondance entre objets conceptuels et objets de présentation. Le Composant Technique de Présentation (CTP) définit les règles de correspondance entre les objets de présentation et les objets d'interaction. Par exemple, la classe de présentation "choix multiple" est concrétisée sous forme d'un "menu de boutons radio". Le Composant Bas Niveau d'Interaction (CBNI) désigne la plate-forme d'accueil logicielle et matérielle. Ce niveau regroupe les services d'acquisition, d'estampille et de répartition des événements mais aussi les objets d'interaction des boîtes à outils.

III.6 MVC et Java Swing

III.6.1 Lien entre MVC et Java Swing

En pratique, plusieurs systèmes de visualisation graphiques sont basés sur le modèle d'architecture MVC tels que la boite à outils de Java Swing. Il est à rappeler que le principal objectif de ce modèle d'architecture est la séparation des rôles**,** on a donc :

- Le ***modèle*** qui est l'élément principal du composant, il contient les données.
- Les ***vues*** du composant qui sont des visualisations des données du modèle, une vue s'abonne à
  un modèle, et se met à jour quand les données du modèle évoluent.
- Le ***contrôleur*** assure la synchronisation entre le modèle et les différentes vues à travers des traitements.

Il existe en Swing des composants génériques pour les modèles de données comme par exemple :

Composant	Modèle	Vue
JList	– classe **ListModel** pour les données – classe **ListSelectionModel** pour gérer les sélections	
JTable	– classe **TableModel** pour les données – classe **TableColumnModel** pour définir les colonnes – classe **ListSelectionModel** pour gérer les sélections	

III.6.2 Etude de cas : le composant JList

JList fait partie du package Java Swing. C'est un composant qui affiche un ensemble d'objets et permet à l'utilisateur de sélectionner un ou plusieurs éléments. Comme tous las composants Swing, JList hérite de la classe JComponent et est basé sur ListModel comme le montre la structure suivante :

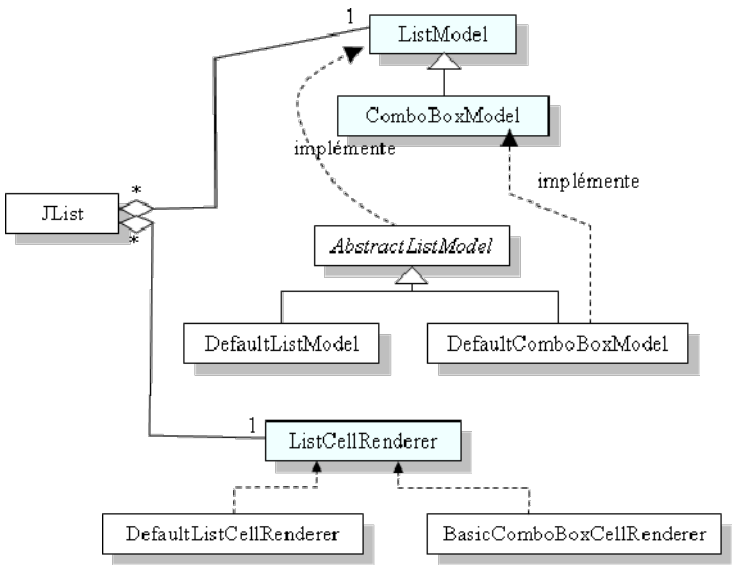


Figure 26: Composition de la classe JList en Java Swing

Les constructeurs de la classe JList sont:

Constructeurs de JList	Description
JList()	Crée une liste vide

JList(E[] L)	Crée une nouvelle liste avec les éléments du tableau
	`String[] Couleurs = {"Red","Green","Blue","White"};`
	`JList liste=new JList(Couleurs);`

JList(ListModel d)
Ou bien Pour une liste modifiable
JList(DefaultListModel d)
	Crée une nouvelle liste avec le modèle de liste spécifié
	`DefaultListModel d = new DefaultListModel<>();`
	`d.addElement("Item1");`
	`d.addElement("Item2");`
	`d.addElement("Item3");`
	`d.addElement("Item4");`
	`JList list = new JList<>(d);`
	`list.setBounds(100,100, 75,75);`

Accès aux éléments du modèle

Opération	Exemple

Récupérer l'élément à la position i	`//récupérer le modèle de la liste`
	`ListModel myModel=liste.getModel();`
	`String premierelement=myModel.getElementAt(0).toString();`
	`System.out.println("Le premier élément est: "+premierelement);`

Récupérer tous les éléments d'une liste par une boucle	`ListModel myModel=liste.getModel();`
	`int size = myModel.getSize();`
	`for (int i = 0 ; i < size ; i++) {`
	`  Object elem = myModel.getElementAt(i);`
	`  System.out.println(elem);`
	`}`

récupérer l'élément sélectionné et sa position	`int index = liste.getSelectedIndex();` `String ch = (String) liste.getSelectedValue();`

Les méthodes couramment utilisées de JList et qui permettent l'accès à ses données (le modèle) sont:

- **getSelectedIndex()** : Renvoie l'index de l'élément sélectionné de la liste
- **getSelectedValue()** : Renvoie la valeur sélectionnée de l'élément de la liste
- **setSelectedIndex(int i)** : Définit l'index sélectionné de la liste sur i
- **setSelectionBackground(Color c)** : Définit la couleur d'arrière-plan de la liste
- **setSelectionForeground(Color c)** : Modifie la couleur de premier plan de la liste
- **setListData(E[] l)** : Remplace les éléments de la liste par les éléments de l.
- **setVisibleRowCount(int v)** : Modifie la propriété visibleRowCount
- **setSelectedValue(Object a, boolean s)** : Sélectionne l'objet spécifié dans la liste.
- **setListData(Vector l)** : Construit un ListModel en lecture seule à partir d'un vecteur spécifié.
- **getSelectedValuesList()** : Renvoie une liste de tous les éléments sélectionnés.
- **getSelectedIndices()** : Renvoie un tableau de tous les indices sélectionnés, dans l'ordre croissant
- **getMinSelectionIndex()** : Renvoie le plus petit index sélectionné, ou -1 si la sélection est vide.
- **getMaxSelectionIndex()** : Renvoie le plus grand index sélectionné, ou -1 si la sélection est vide.
- **isSelectedIndex(int i)** : Renvoie true si l'index spécifié est sélectionné, sinon false.

- **setFixedCellWidth(int w)** : Remplace la largeur de cellule de la liste par la valeur passée en paramètre.
- **setFixedCellHeight(int h)** : Remplace la hauteur de cellule de la liste par la valeur passée en paramètre.

Evénements générés de JList

Les événements générés par une liste sont des événements de sélection de type ListSelectionEvent (pas d'événement de type ActionEvent).

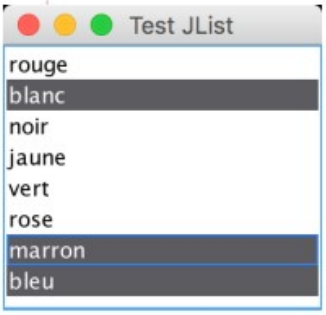
• Implémentation de l'interface: ListSelectionListener qui ne comporte qu'une seule méthode :

public void valueChanged(ListSelectionEvent e)

Voici un exemple simple sur l'utilisation statique de la classe JList avec gestion d'évènements:

```
public class JListStatiqueTest extends JFrame implements ListSelectionListener {

    private JList liste ;
    static final String[] couleurs = {"rouge", "blanc", "noir", "jaune", "vert",
        "rose", "marron", "bleu"};


    public JListStatiqueTest(String t) {
        super (t);

        // definition d'une liste
        liste = new JList(couleurs);
        liste.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
        this.getContentPane().setLayout(new BorderLayout());

        liste.addListSelectionListener(this);
        JScrollPane panneau = new JScrollPane(liste);
        liste.setSelectedIndex(1);

        this.getContentPane().add(panneau, BorderLayout.CENTER);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```

Ajouter un écouteur à la liste



III.6.3 Utilisation des Patterns Observer et Observable

Pour une mise en oeuvre efficace et bien structurée du modèle MVC avec les composants graphiques de Java Swing, l'utilisation des patrons de conception Observer/Observable semble plus adaptée. E effet, Observer est un design pattern comportemental qui permet de mettre en place un mécanisme de souscription pour envoyer des notifications à plusieurs objets, au sujet d'événements concernant les objets qu'ils observent. C'est exactement le même phénomène qui se passe pour avec les composante MVC où chaque changement doit être observé sur la vue qui est la partie visible de l'interface via des listeners puis notifié pour le modèle, la gestion des liseners correspond à la partie contrôleur.

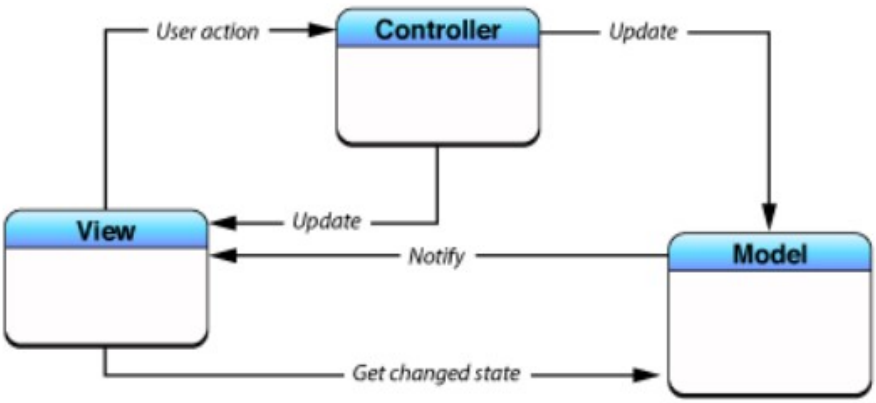


Figure 27: Principe de notification en MVC

Ici la vue joue le role de l'Observer et le modèle comme Observable comme le montre la figure.

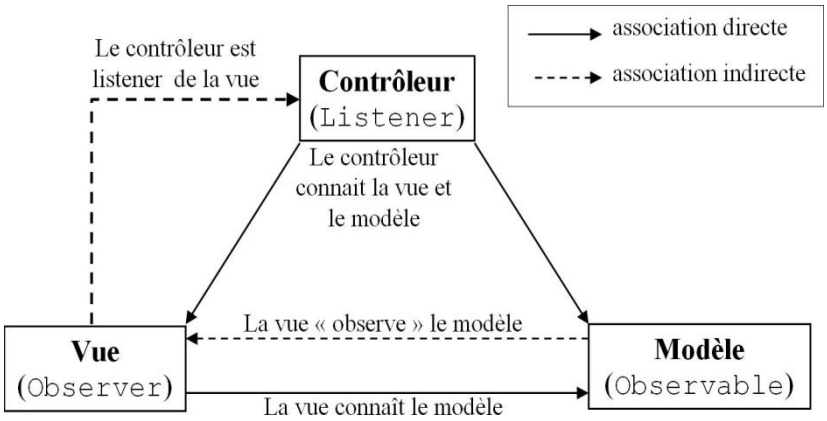


Figure 28: Mise en oeuvre du MVC avec le pattern Observer

***NB:** L'utilisation des patrons de conception sera plus détaillée dans le dernier chapitre de ce cours.*

III.6.4 Un exemple détaillé

On prend exemple l plus connu interaction avec java qui est celui d'une ardoise graphique simple qui permet de tracer une forme sur la surface d'un frame.

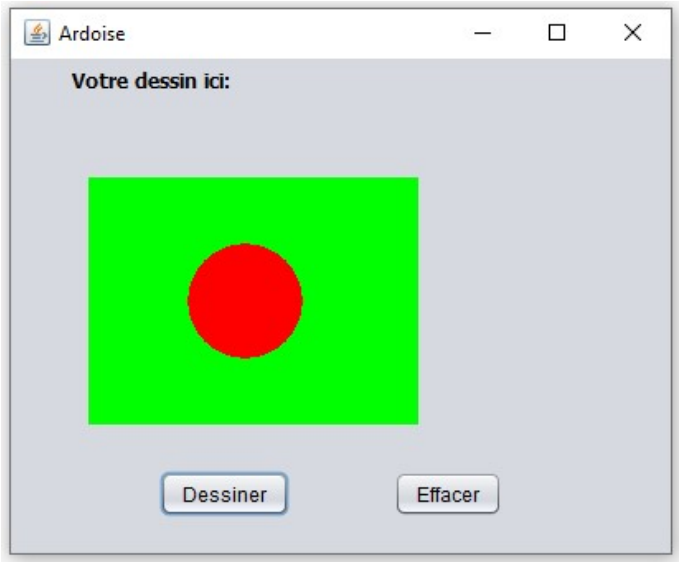


Figure 29: Exemple simple d'une ardoise graphique

Les composantes MVC correspondantes à cet exemples sont les suivantes:

Le modèle: c'est la classe de l'application elle-même, elle contient tous les attributs et les méthodes participant à la tâche principale.

La vue: Les différents objets graphiques visuels.

Le contrôleur: c'est l'ensemble des listeners et qui, lorsque des événements parviennent via la vue, prévient le modèle en conséquence.

En exploitant les patterns Observer et Observable de java (correspondant respectivement aux classes java.util.Observer et java.util.Observable), le fonctionnement est résumé come suit:

Observer possède la méthode *addObserver* qui permet d'inscrire des observateurs. Observable déclare uniquement la méthode :

public void update(Observable o, Object arg);

Le modèle étend Observer, l'interface est inscrite comme observateur du modèle. Quand le modèle est modifié, il faut actualiser l'interface, le modèle n'a pas besoin de connaître l'interface, après une modification, elle utilise les instructions :

setChanged(); notifyObservers();

ce qui a pour conséquence de faire exécuter les méthodes update des observateurs inscrits pour observer le modèle. Ainsi, cette structuration resemble au code suivant:

```
Vue(Modele modele) {
        this.modele = modele;
.....
modele.addObserver(this);
..... }
```

```
public void update(Observable o, Object arg)
{
ardoise.setPosedeDisque(modele.getExiste());
        ardoise.repaint();
```

```
class Modele extends Observable
{
  private boolean existe;
  void setExiste(boolean existe) {
      this.existe = existe;
      setChanged();

      notifyObservers();

  }
```

```
class  Controleur implements ActionListener
{
  Modele modele;
  Vue vue;

  Controleur(Modele modele, Vue vue) {
        this.modele = modele;
        this.vue = vue;
  }
```

Chapitre IV: Catégories d'outils pour la construction des IHM

IV.1 Nécessité des outils

Les applications interactives sont généralement complexes et parfois difficiles à déboguer et à modifier. Il y a des études (comme celle de Myers en 1995) qui ont montré que 50% du code des applications correspond au développement de l'IHM et à son implantation.
Plus les interfaces sont faciles à utiliser plus elles sont difficiles à implanter. Pour faciliter la création des IHM et leur implantation, l'utilisation d'outils dans les phases de développement s'avère nécessaire. Ainsi le développement d'outils pour le support des IHM est devenu une activité primordiale dans les domaines de recherche et le marché du logiciel.
Au cours du temps, plusieurs outils ont apparus par développement successif, chaque nouvelle boite est conçue au-dessus de l'autre formant une structure en couches comme dans la figure suivante :

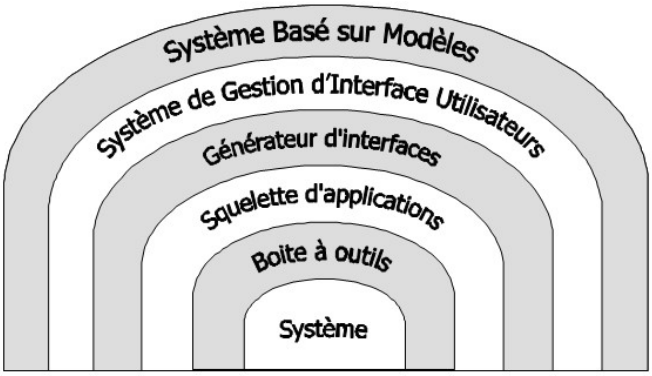


Figure 30: Famille des outils de développement des IHM

L'utilisation d'outils de création d'interfaces peut apporter plusieurs avantages parmi lesquels on peut citer :

- Meilleure qualité, avec moins de défauts de structuration
- Interfaces plus efficaces, en essayant plusieurs styles d'interactions
- Créer des interfaces plus facilement, voire automatiquement…
- Créer des interfaces plus rapidement, en utilisant des prototypages
- Interfaces plus faciles à maintenir, pour une meilleure évolution
- Possibilité de laisser la charge de la conception à des professionnels des IHM, qui ne sont pas informaticiens (artistes, psychologues, ergonomes, …)
- Cohérence des IHM créées avec un même outil
- Code de l'IU mieux structuré et plus modulaire car séparé de l'application

IV.2 Différentes catégories d'outils

Dans le marché des outils de construction des interfaces logicielles, on trouve plusieurs types d'environnement variant d plus simples et basiques, passant par les boites à outils connues et allant jusqu'aux générateurs de haut niveau. Ces outils peuvent être répartis en quatre principales catégories et qui sont:

- Logiciels graphiques de base, systèmes de fenêtrage
- Boîtes à outils

- Applications extensibles, frameworks ou encore squelettes d'applications
- Constructeurs, générateurs d'interfaces (*environnements de développement d'interfaces (SGIU) : interactifs, à manipulation d'objets de dialogue à langage de spécification du dialogue automatiques à partir de spécifications des fonctionnalités d'une application*

IV.3 Logiciels graphiques de base

Un logiciel graphique de base offre un environnement de construction d'interfaces à partir des composantes graphiques basiques telles que les formes, les couleurs, ...etc. En général, le développeur avec ce type d'outil n'a pas d'objets évolués pour l'interaction à part des bibliothèques couvrant tous les niveaux fonctionnels d'un logiciel interactif en se basant sur des mécanismes de:

- Fenêtrage
- Affichage (dessin)
- Rares objets de présentation
- Rares objets de dialogue

Leur utilisation consiste à appeler des primitives graphiques à partir de l'application comme par exemple la primitive qui dessine un rectangle avec une couleur, une taille et une position données sans gestion éventements associés directement. Donc, tout doit être prévu et implémenté par le développeur. Il y a des exemples connus pour cette catégorie comme:

- **GKS** (*Graphical Kernel System*): le premier système graphique de bas niveau adopté par l'ISO en 1977, basé sur les deux standards (CGI: *Computer Graphics Interface* et CGM: *Computer Graphics Metafile*). C'est un outil pour les graphiques 2D basé sur les formes connues comme Polyline, Fillarea et Genaralized Drawing Primitives.
- **HIGZ** (*High level Interface to Graphics and ZEBRA*): est une librairie graphique 2D standard et libre provenant du CERN Program Library. Elle offre des routines graphiques simples à utiliser (functions such as histograms, full graphs, circles).
- P**HICS** (*Programmer's Hierarchical Interactive Graphics System*): est une bibliothèque standard d'environ 400 fonctions dites primitives qui permettent à l'utilisateur d'afficher et d'interagir avec des graphiques 2D et 3D manipulant des attributs graphiques comme location, orientation, color, et appearance tout en cachant à l'utilisateur les détails dépendants du matériel.

- **X11 et X-Lib**

X11 est le système graphique le plus populaire du système d´exploitation *UNIX* basé sur la bibliothèque X-Lib. Sa large distribution, le fait qu'il soit libre de tous droits de distribution et surtout ses qualités techniques exceptionnelles en ont fait un *standard* de l'industrie du logiciel. Ses caractéristiques principales sont:

- l'utilisation d'une architecture *client/serveur* et cela dans une totale hétérogénéité (le serveur et le client peuvent fonctionner sur des architectures totalement différentes). Le dialogue entre le serveur et les clients se fait conformément au *protocole X* ce qui permet d'assurer la *transparence du dialogue*.
- la *portabilité* des librairies et des applications X11 (le même code tourne sur une grosse station de travail ou bien un petit PC sous Linux)

En dépits des avantages de X11, il serait faux de croire que le développement d'une application X soit une chose simple. Les concepts utilisés sont complexes (programmation orientée objet,

gestion d'évènements, ...), le nombre de fonctions à connaitre est important et la documentation titanesque et parfois difficile à aborder...

Structure d'un programme X

Un programme X peut grossièrement se diviser en 3 parties:

- l'ouverture d'une connexion à un *serveur X* (serveur local ou bien terminal).
- les initialisations des fenêtres initiales (cas de la Xlib) ou des objets initiaux (cas d'un toolkit)
- l'attente d'évènements (souris, clavier, ou autres...) dans une boucle infinie dont on ne sortira qu'à la fin du programme.

La réception des évènements déclenchera les actions décrites à la deuxième partie (initialisation).

X est basé sur les trois concepts : **Display** (la connexion de l'application à un serveur X), **l'écran** (unités d´affichage) et **la fenêtre.**

Avantages et inconvénients des logiciels graphiques

Avantage :
- Très puissants, permettent de tout construire !

Inconvénients :
- utilisation très complexe ! (perte de temps)
- il faut savoir programmer (le graphique !)
- exclusion des experts en IHM non informaticiens
- problèmes de portabilité d'une boîte à outils vers une autre
- pas de séparation formelle IU <-> application
- applications à contrôle interne

IV.4 Boîtes à outils

Une boîte à outils (Toolkit en anglais) est une bibliothèque d'objets d'interaction appelés composants graphiques ou widgets. Les toolkits fournissent une interface de programmation orientée (*API: application programming interface*). Elles sont relativement nombreuses, pour ne citer que les principales Java/Swing, MFC (Microsoft Fundation Class), QT et Tk.
La construction d'une interface logicielle est faite en combinant un ensemble de composants graphiques, tels que les fenêtres, les boutons, les menus, etc.

Ces derniers sont capables de réagir aux interactions de l'utilisateur. Dans ce but, les boîtes à outils utilisent le mécanisme d'événements pour gérer le dialogue entre l'utilisateur et la présentation. Quand l'utilisateur interagit avec un composant graphique de la présentation, son action est transformée par la boîte à outils en événement. A la réception de l'événement, le composant graphique réagit à travers les deux comportements suivants:

- **Un comportement interne:** représente une réaction propre d'un composant graphique à un événement. Il n'est généralement pas modifiable, son rôle est essentiellement esthétique et n'a d'importance que dans la présentation. Par exemple, l'état d'un bouton enfoncé ou non est modifié quand l'utilisateur appuie sur le bouton de la souris et que le pointeur se trouve dessus;

- **Un comportement externe :** associé au résultat que le concepteur souhaite obtenir quand le composant graphique subit une action de la part de l'utilisateur. Ce comportement doit être programmé par le concepteur en utilisant un mécanisme de traitement des événements qui diffère selon la boîte à outils employée : boucle d'événement, fonctions de rappels ou abonnements. Par exemple, dans le cas du traitement des événements par abonnement, utilisé dans la boîte à outils Java/Swing, la conception consiste à associer à chaque composant graphique émetteur d'événements un ou plusieurs objets récepteurs d'événements qui se chargeront d'appeler des éléments du noyau fonctionnel. Ainsi, les boîtes à outils permettent la description de la présentation mais aussi assurent le dialogue de l'application.

```
JButton mon_button = new JButton("OK")
mon_button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                ref_nf.incrementeCompteur();
        }
});
```

Figure 31 : Abonnement d'un composant graphique Swing à un écouteur

L'exemple de la figure décrit l'utilisation du traitement des événements par la technique d'abonnement. L'objet *mon_button* est créé sur la première ligne puis il est associé à un objet écouteur *ActionListener* sur la deuxième ligne. Suivant la nature de l'événement émis, la méthode *actionPerformed()* appelle la méthode *incrementeCompteur()* du noyau fonctionnel *ref_nf*.

Une boîte à outils présente de nombreux avantages. Tout d'abord, elle permet de donner un style à l'interface et permet d'obtenir une homogénéité visuelle entre différentes applications. Ensuite, elle permet d'intégrer des critères ergonomiques dans les composants graphiques même si le concepteur ne possède pas de compétences en psychologie cognitive.
En contrepartie, les boîtes à outils souffrent de nombreux inconvénients:

- Difficulté d'utilisation : il est nécessaire d'être informaticien et de connaître des langages de programmation pour utiliser les boîtes à outils ce qui exclut de leur utilisation les personnes issues des domaines liées comme la psychologie, l'ergonomie et aussi l'utilisateur final.
- Le temps d'apprentissage est souvent long.
- Absence de structuration explicite du code : elle aboutit logiquement à des architectures logicielles floues. Il n'y a pas de réelle séparation entre le noyau fonctionnel et la présentation de l'application.
- La maintenance de l'application est donc rendue difficile.

- Difficulté de vérification : afin de juger la notion d'utilisabilité, il est nécessaire de passer par de longues phases de tests exhaustifs. La relecture éventuelle du code de l'application pour vérifier les propriétés du système est difficile puisque le code contient à la fois des éléments de la présentation, du dialogue et du noyau fonctionnel.

IV.5 Applications extensibles (Frameworks)

L'utilisation des boîtes à outils amène le concepteur à programmer certaines séquences de code un grand nombre de fois, dans des applications différentes. Il parait intéressant de coder une fois pour toutes les séquences sous forme d'une structure d'application interactive adaptable : c'est la notion de squelette d'application (application Framework en anglais). Le rôle du concepteur est alors d'adapter les squelettes d'application à son développement en supprimant des portions de code inutiles et modifiant celles inadaptées en ajoutant des nouvelles fonctionnalités.

Les squelettes d'application permettent ainsi la factorisation de parties de code récurrentes et la diminution de l'effort de développement d'un facteur quatre à cinq par rapport aux seules boîtes à outils. Ils facilitent le développement et permettent au concepteur de se concentrer sur des fonctionnalités annexes telle que le dialogue de l'application ou les fonctionnalités. Ils garantissent aussi une homogénéité élevée des systèmes interactifs créés à partir des mêmes squelettes d'applications. Le WinDev est un exemple connu comme Framework pour les interfaces manipulant des bases de données (fenêtres, états, requêtes, ...).

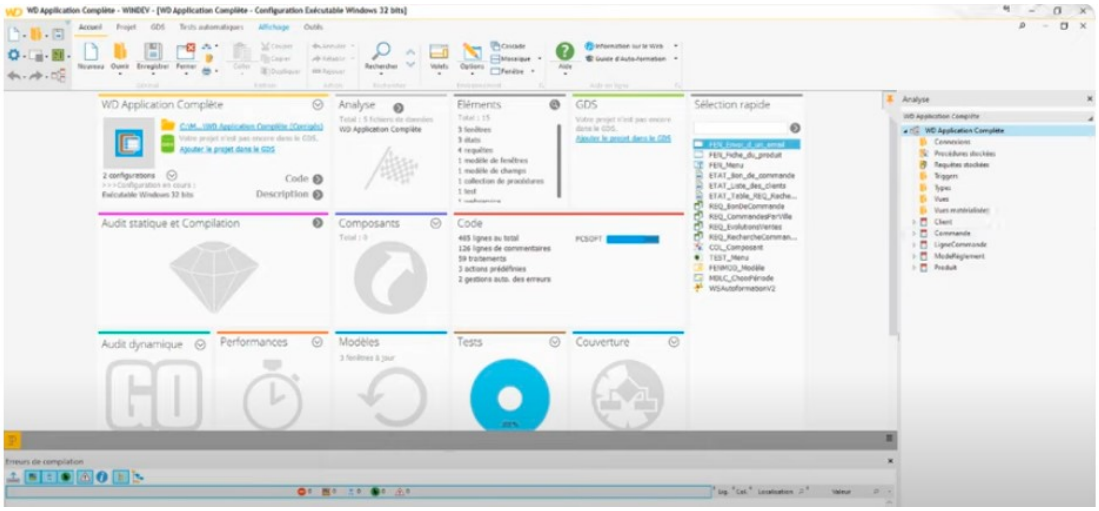


Figure 32 : Aperçu du Framework WinDev

JAVA/Swing propose dans ce sens plusieurs squelettes d'application tant au niveau de la partie interface graphique que sur des aspects purement fonctionnels. Par exemple, cette boîte à outils propose des squelettes d'application pour gérer le mécanisme Défaire/Refaire, de sauvegarde au moyen du mécanisme de la sérialisation, ou pour se connecter à d'autres sous-systèmes (base de données, serveurs HTTP ...). JAVA/Swing propose aussi des squelettes d'applications liés à des éléments de l'interface graphique comme des boîtes de dialogue prédéfinies.
En contrepartie, l'utilisation des squelettes d'applications amène aux remarques suivantes :

- Compréhension du fonctionnement souvent très abstrait, les squelettes d'applications nécessitent de la part du concepteur une bonne connaissance de leur fonctionnement. Ils ne suppriment pas non plus la nécessité de connaître la boîte à outils sous-jacente, ce qui réserve leur utilisation aux informaticiens ;
- Limitation du domaine d'application : les squelettes d'applications sont par nature stéréotypés, ce qui limite leur domaine d'application.

IV.6 Générateurs d'interfaces

Les générateurs d'interfaces ou GUI-Builder (Graphic User Interface Builder) sont des outils qui permettent de créer facilement la couche présentation d'une application de façon graphique en mettant en œuvre un aspect dit No-Code. Pour cela, le concepteur dessine son interface en utilisant des primitives graphiques représentant les widgets de la boîte à outils sous-jacente. Le concepteur visualise instantanément le résultat et, il lui est possible sous certaines limites de tester son interface construite comme par l'exemple dans l'outil GILT.

Employant généralement des techniques de programmation visuelle (manipulation directe des objets graphiques au moyen de la souris, visualisation des informations...), les générateurs de présentations sont faciles à utiliser et permettent d'obtenir rapidement des maquettes. Il n'est pas nécessaire d'être un spécialiste en programmation pour construire la couche présentation.

Les générateurs de présentations produisent un squelette d'application à partir de l'interface construite graphiquement par le concepteur. Il suffit d'ajouter la dynamique du dialogue et les appels aux fonctions du noyau fonctionnel pour construire l'application. Toutefois, cette étape doit être réalisée par un concepteur ayant des connaissances en programmation.

Les générateurs de présentation sont bien au point, et maintenant intégrés dans des produits commerciaux. Par exemple, Visual Studio, NetBeans et Marvel (*Delphi CBuilder et JBuilder de Borland qui sont les plus classiques*). Ils disposent de constructeurs d'interface directement intégrés à leurs environnements de programmation. Ces environnements intègrent de nombreux squelettes d'applications qui facilitent le développement de l'application interactive, par exemple l'utilisation d'assistants (Wizards en anglais).

Cependant, les générateurs de présentation souffrent des inconvénients issus de la boîte à outils sous-jacente. Ils ne se limitent qu'à la couche présentation en permettant la représentation graphique des différents écrans et boîtes de dialogue d'une application, mais sans définir la succession de ces objets à l'écran (dynamique). Dans ce sens, l'outil BeanBuilder11 associé à la technologie JAVA/Beans apporte une contribution dans ce domaine. En effet, il autorise une construction de la couche présentation et une ébauche d'un pseudo dialogue au travers des objets Beans.

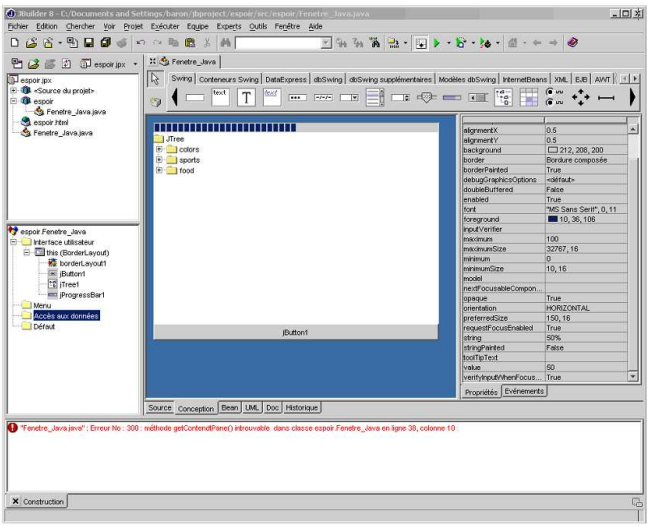


Figure 33: Exemple de GUI-Builder intégré dans l'environnement JBuilder

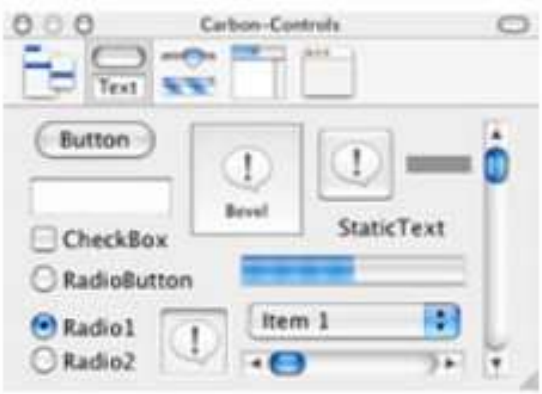


Figure 34 : Exemple d'utilisation d'un générateur d'interface

Principales fonctionnalités des créateurs d'interface utilisateur No-Code

Les créateurs d'interface utilisateur No-code se caractérisent par plusieurs fonctionnalités essentielles qui en font un choix fiable pour les non-concepteurs. Ces fonctionnalités améliorent le processus de création d'applications et garantissent que le résultat final est de qualité professionnelle.

1. **Fonctionnalité glisser-déposer :** les créateurs d'interface utilisateur No-code fournissent une interface intuitive qui permet aux utilisateurs de simplement drag and drop des éléments d'interface utilisateur sur l'écran. Cela élimine le codage manuel et permet aux utilisateurs de créer facilement des mises en page d'application.
2. **Composants d'interface utilisateur prédéfinis :** la plupart des plates no-code proposent une bibliothèque de composants d'interface utilisateur prédéfinis qui servent de blocs de construction pour l'application. Les utilisateurs peuvent choisir parmi différents composants tels que des boutons, des formulaires, des éléments de navigation, etc.
3. **Conception réactive :** les créateurs d'interface utilisateur No-code offrent des options de conception réactive pour répondre aux différents appareils et tailles d'écran disponibles aujourd'hui. Cette fonctionnalité garantit que l'application apparaît et se comporte de manière optimale sur différents appareils et résolutions d'écran.
4. **Collaboration en temps réel :** de nombreux créateurs d'interface utilisateur no-code proposent des fonctionnalités collaboratives qui facilitent une coopération transparente entre les membres de l'équipe et les parties prenantes. Cela permet des boucles de rétroaction en temps réel et aboutit à une conception finale plus cohérente.
5. **Directives de conception intégrées :** les créateurs d'interface utilisateur No-code incluent souvent des directives de conception et des bonnes pratiques qui aident les non-concepteurs à créer des applications professionnelles. Ces directives aident les utilisateurs à adhérer à des principes de conception éprouvés et à éviter les erreurs courantes.

Il y a des générateurs qui ont apparu récement et ont prouvu leur éfficacité tels que:
Bubble, Wix, Webflow, et Appmaster.

Etapes d'utilisation d'un générateur d'interfaces

En général, l'utilisation d'un générateur passe par quatre étapes qui constituent en fait, trois axes de recherche sur les systèmes de génération d'interfaces utilisateur (ces axes sont utilisés dans le modèle de Seeheim):

a. **Description de la présentation de l'interface :** elle se fait par :
   - Création interactive de l'apparence de l'interface
   - Définition de la présentation par « drag and drop »
   - Édition des propriétés des composants de dialogue
   - Inspecteurs (Ilog)
   - Editeurs de propriétés de widgets (UIM/X)
   - Intégration des objets de l'application par programmation en créant de nouveaux objets d'interface (Widget Factory) et aussi en intégrant des bibliothèques d'objets (IlogViews), …

b. **Expression du contrôle du dialogue :** qui permet la génération du squelette produit, il existe plusieurs outils avec les différents langages de programmation comme : C ou C++, UIM/X, XFaceMaker, IlogBuilder, Lisp et Aïda, Java et Swing, NetBeans, IlogJViews, JBuilder, …

c. **Spécification de l'interface avec l'application :** elle consiste à ajouter les appels aux fonctionnalités de l'application dans le squelette généré et on peut aussi:
   - Modifier directement le squelette (DevGuide) :
   - Insérer les fonctionnalités à l'aide d'outils fournis par le générateur (Ilog, UIM/X)

d. **Le contrôle : c'est le fonctionnement de l'application elle même** à travers l'interaction avec l'utilisateur, ce qui n'est pas traité par les premiers générateurs. Cela se fait par programmation directement à l'aide du langage de la boîte à outils sous-jacente:
   - Pour obtenir du dialogue avec rétroaction sémantique
   - Pour avoir un comportement dynamique de l'IU
   - Exploitation de nouvelles possibilités offertes, comme la définition de nouveaux objets de dialogue, avec un comportement propre (Widget Factory), la définition de méthodes associées aux objets de dialogue, la création d'objets (héritage, composition), comme dans les générateurs Builder et Visuel.

Avantages et inconvénients

Avantages des générateurs
- Faciles à utiliser (applications conversationnelles)
- IU faciles à modifier
- Obtention rapide de prototypes
- Interfaces qui se ressemblent (cohérence)
- Excellent rapport gains / prix

Inconvénients des générateurs
- Interfaces qui se ressemblent (pas vraiment d'originalité).
- Interfaces statiques : permettent rarement (sans aucune programmation) de définir des objets de dialogue qui évoluent dynamiquement au cour de la manipulation de l'interface.
- Ne permettent pas encore tout à fait de visualiser facilement les objets complexes des applications (pour applications à manipulation directe).
- Il faut encore connaître les mécanismes de la boîte à outils sous-jacente…

Chapitre V: Prise en compte des utilisateurs dans le processus de conception des IHM

V.1. Introduction

Un produit logiciel est enfin utilisé par des utilisateurs finaux à travers son interface. Ces utilisateurs devraient être au centre de la conception de tout produit. La conception centrée sur l'utilisateur (*UCD: User Centred Design*) est standardisée par la norme ISO 13407 (*elle définit sept ensembles de pratique de base pour mettre en œuvre le processus de conception centrée sur l'humain)* et aussi l'ISO 9241-210 de 2010. C'est une démarche qui renforce le rôle de l'utilisateur et elle intègre ce dernier à toutes les étapes du processus de développement d'une IHM. Cette démarche suppose :

- La prise en considération des besoins d'utilisateurs potentiels ainsi que leurs caractéristiques différentielles lors de la conception d'une IHM (capacités cognitives et physiques, mémoire, perception, expérience, etc.);
- L'implication et la participation active de l'utilisateur final au cours des différentes phases de développement.

Le concepteur doit se centrer sur les dimensions humaines, sociales et cognitives de l'utilisation d'un système et faire participer activement les utilisateurs à la conception en faisant appel à une démarche itérative de conception et en faisant intervenir une équipe de conception multidisciplinaire.

Selon les options théoriques et méthodologiques, la manière dont l'utilisateur final est pris en considération dans un système peut revêtir des formes variées :

- Utilisateur en tant que personne possédant des caractéristiques qui le distinguent d'autres utilisateurs, notamment des handicaps et des déficits sensoriels, moteurs, cognitifs ou sociaux ;
- Utilisateur en interaction avec le contexte dans lequel son activité se déroulera ;
- Utilisateur en tant que personne qui évolue dans un milieu social et interagit avec les autres dans l'exécution de tâches (activités coopératives);
- Utilisateur doté d'un niveau d'expertise plus ou moins élevé et d'une capacité d'adaptation plus ou moins importante par rapport aux tâches (flexibilité) ;

Après cette introduction, on peut conclure à la définition suivante de la conception centrée sur l'utilisateur:

Définition: La conception centrée sur l'utilisateur *ou conception orientée utilisateur (UCD, User-Centered Design en anglais)* est une philosophie et une démarche de conception en particulier présente en ergonomie informatique, où les besoins, les attentes et les caractéristiques propres des utilisateurs finaux sont pris en compte à chaque étape du processus de développement du produit.

V.2. Pourquoi centré utilisateur

Un des défis pour les concepteurs : les humains et les machines fonctionnent fondamentalement de manière assez différente. La vision diffère selon le point de vue adopté. On montre dans le tableau ci-dessous les point de déférence qui laisse penser à une conception centrée sur l'utilisateur dans le cas d'une IHM bien sûr:

Approche	**Les humains sont …**	**Les machines sont …**
Centrée sur la machine (*technocentrée*)	Vagues Désorganisés Distraits Émotifs Peu logiques	Précises Ordonnées Imperturbables Sans émotions Logiques
Centrée sur l'humain (anthropocentrée)	Créatifs Conciliants S'adaptent aux changements Débrouillards Capables de décider en fonction des circonstances	Sottes Rigides Insensibles aux changements Sans imagination Contraintes à prendre des décisions uniformes

Il est connu en IHM que le concepteur utilise des modèles qui sont basés principalement sur l'utilisateur et ses tâches en plus d'autres modèles pour interaction. Mais pour bien utiliser l'interface produite, l'utilisateur doit aussi avoir un moyen pour comprendre et suivre les actions possibles qui mènent à ces objectifs. Le schéma suivant montre cette intersection et met évidence la nécessité d'une conception centrée utilisateur.

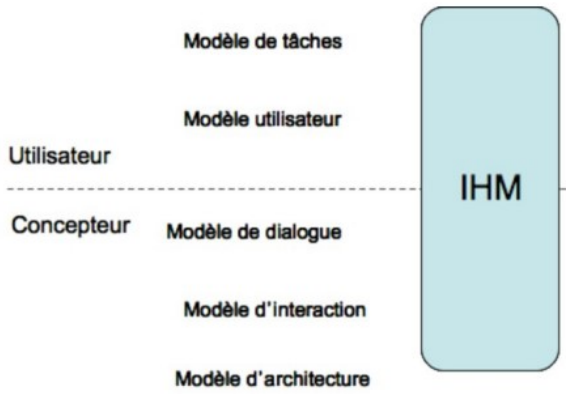


Figure 35: Intersection concepteur/utilisateur

V.3 Principe

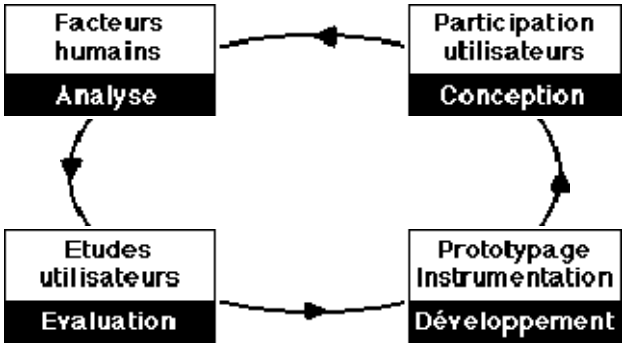


Figure 36: Principe de la démarche UCD

La démarche de conception centrée sur l'utilisateur repose sur l'idée que les utilisateurs finaux sont les mieux positionnés pour évaluer et utiliser le produit. Par conséquent, le développement d'un produit est a priori davantage guidé par les besoins et exigences des utilisateurs finaux, plutôt que par des possibilités techniques ou technologiques. Cependant, l'utilisateur final peut être entendu de deux manières :

- L'utilisateur réel, le plus susceptible d'utiliser le produit répondant à ses exigences et étant peut-être déjà utilisateur d'une version précédente du produit
- L'utilisateur potentiel qui présente des exigences proches ou équivalentes, et dont l'utilisation du produit pourrait intéresser.

La définition et le recueil des besoins, des attentes et des exigences applicables au produit doivent être issus d'une démarche rigoureuse dans le cadre d'une intervention ergonomique, d'une enquête utilisateur, d'un test utilisateur. Ces étapes peuvent être effectuées avec un produit existant ou un prototype.

V.4. Étapes du processus d'une conception centrée utilisateur

Le processus de production d'une interface logicielle qui est une tâche d'informaticien passe bien sûr par les étapes d'analyse, conception, production ou développement et enfin évaluation ou test. Quand il s'agit d'une conception centrée sur l'utilisateur, ces étapes sont expliquées de la façons suivante:

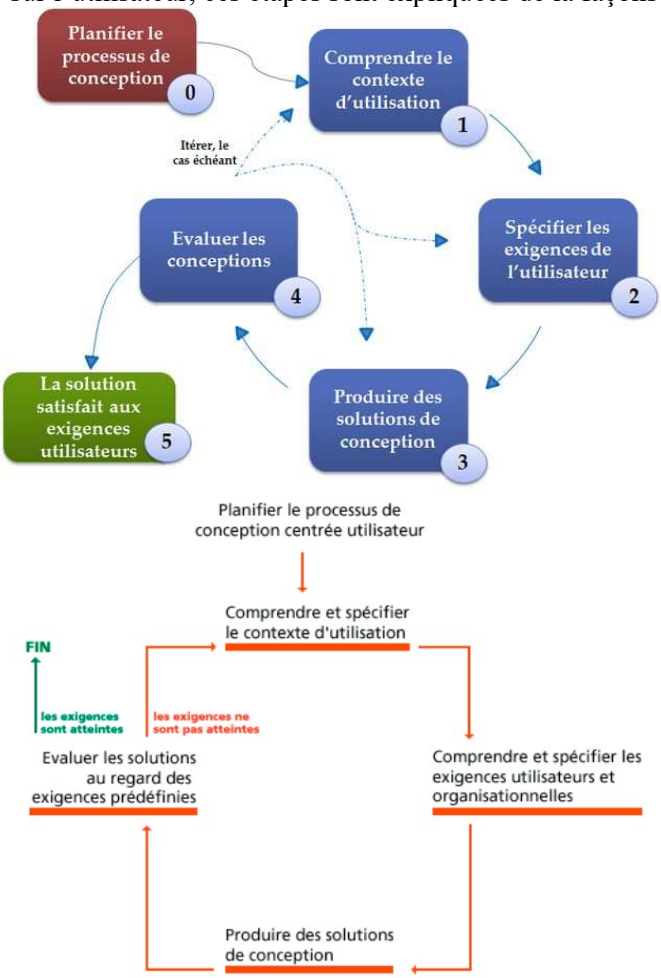


Figure 37: Étapes du processus UCD

En général, ces étapes ne sont pas exhaustives. Les étapes et méthodes utilisées seront adaptées en fonction de facteurs variés tels que temps disponible, marges de manœuvre financières, disponibilité des utilisateurs ou opérateurs, type d'application (experte ou grand public), domaine d'intervention (classiquement, logiciel ou Web), et des compétences des intervenants.
De façon résumée, les moyens d'impliquer les utilisateurs passent par des méthodes de types entretiens ou questionnaires, observation, focus groups et tests utilisateurs. On peut affecter chacune des méthodes à une étape particulière du cycle de UCD.
Nous allons expliquer ici les principales étapes montrées dans la figure.

a. Planifier le processus de conception

Identification des grandes lignes du produit. Il s'agit ici d'établir ce que sera le produit. Il faut aussi définir ce que le produit ne sera pas afin d'éviter les dépassements de délai et de budget.
C'est une cette pré-étape qui consiste à planifier les activités de développement dans une optique de conception centrée utilisateur. L'équipe projet doit donc avoir atteint un consensus concernant la recherche de la satisfaction de la norme ISO 13407 et donc de ses implications sur les plans techniques, méthodologiques, et de conduite de projet. Les avantages doivent être connus de l'équipe, notamment le retour sur investissement, la satisfaction des utilisateurs, l'utilisabilité du système et l'adaptation aux caractéristiques des opérateurs. En cette étapes, plusieurs points sont à éclaircir :

- Les exigences doivent être clairement explicitées.
- Étudier de quelle manière la démarche peut aider à atteindre les objectifs organisationnels.
- Définir précisément le type de conception choisie, les méthodes centrées utilisateur qui seront mises en œuvre, les étapes d'implication des utilisateurs...
- Consulter les documents d'entreprise qui pourraient intéresser le projet et participer à la collaboration inter-spécialistes.
- Dans le cas de la conception d'une application métier, constituer un groupe de travail (composé notamment des opérateurs, futurs utilisateurs du système).

b. Spécifier le contexte d'utilisation

Il s'agit donc de comprendre le public ou l'utilisateur cible et ses caractéristiques, ses buts, ses tâches et ses environnements. On commence d'abord de décrire les environnements technique, physique, ambiants, social, organisationnel et législatif. Les contraintes matérielles doivent être identifiées par la connaissance du parc informatique. On adaptera en effet la conception des dispositifs d'entrée et d'affichage en fonction de ces données.
Il est important à cette étape d'identifier le profil de l'utilisateur (connaissances, compétences, fonctions, tâches à accomplir, niveau d'expérience métier et d'expérience de l'outil informatique, langage, éducation, formation, caractéristiques physiques, psychologiques, habitudes, aptitudes), cela permettra de choisir les méthodes d'évaluation et de sélectionner des participants aux tests utilisateurs.
On doit aussi identifier s'il y a des groupes différenciés d'utilisateurs (experts, opérateurs avec des responsabilités, opérateurs simples, ...) et donc activités et accès à l'information différents, et détailler leurs caractéristiques et besoins respectifs.
A cette étape, les interviews et les questionnaires sont des sources d'information essentielles pour l'ergonome. L'analyse des besoins peut aussi profiter de la méthode des groupes de discussion (focus groups traditionnels ou électroniques, par exemple sous la forme de forums de discussion). Des entretiens semi-directifs permettent de compléter les données d'observation que l'on aura recueillies. On doit veiller à faire valider par les utilisateurs les descriptions de tâches obtenues. Les connaissances extraites lors de l'analyse de tâche permettront de dégager des modèles de tâche.

Pour le développement de services ne correspondant pas à des besoins utilisateur identifiés, cette méthode peut servir à générer des idées de fonctionnalités et à se poser la question de l'utilité. Les méthodes de benchmarking pour le Web (analyse de sites concurrents), ou de revue de systèmes et produits similaires pour les solutions logicielles permettent de connaître l'existant, de se baser sur ces éléments ou d'en extraire des principes négatifs ou positifs.

Le choix des techniques à utiliser est fonction du contexte technique (type et caractéristiques de l'application, niveau d'abstraction des items), des marges temporelles et financières et aussi de la disponibilité des utilisateurs. Ces différentes techniques nécessitent en en effet plus ou moins de matériel, de temps et de compétences de la part de l'expérimentateur.

c. Spécifier les exigences liées à l'utilisateur et à l'organisation

Il s'agit de prendre en compte les informations recueillies dans l'étape précédente (besoins, compétences et l'environnement de travail de tous les intervenants pertinents sur le système). L'identification des buts et tâches des utilisateurs est la base du travail de spécification des exigences. Les documents de spécifications consistent en des descriptions précises des profils d'utilisateurs et des cas d'utilisation.
On utilise ces connaissances pour extraire des exigences précises concernant l'assistance du système à l'exécution des tâches et les objectifs que les utilisateurs pourront atteindre en se servant de l'outil. Ces objectifs sont déterminés du point de vue qualitatif et quantitatif. On peut fixer des exigences à atteindre concernant les critères suivants :
- Taux de succès
- Nombre d'erreurs
- Temps d'exécution des tâches
- Nombre d'étapes nécessaires à la complétion des tâches
- Eventuels recours à une aide interne ou externe au produit
- Rythme d'apprentissage
- Satisfaction des utilisateurs…

Les exigences doivent ensuite être ordonnées selon leur importance. A ce niveau, les objectifs d'utilisabilité vont guider et justifier les choix de conception. Ils fourniront par la suite des critères d'acceptation lors des tests utilisateurs.
On doit définir un modèle conceptuel de l'outil, une représentation des tâches qu'il doit supporter et des résultats qu'il doit permettre d'atteindre en termes d'efficacité, d'efficience, et de satisfaction des utilisateurs.
Il ne faut pas oublier aussi les objectifs opérationnels et financiers. On doit définir la faisabilité du traitement et de la maintenance, la conception du travail, les pratiques et la structure de l'organisation, la conception de l'interface et du poste de travail.
Les exigences organisationnelles peuvent-elles être déterminées en termes de processus et flux d'échanges (exemple taux d'appel d'un centre de télémarketing). C'est précisément lors de cette étape que la mobilisation d'une équipe pluridisciplinaire, aux compétences variées, est nécessaire.

d. Produire des solutions de conception

Cette étape vise à utiliser les connaissances acquises lors des étapes précédentes pour matérialiser les solutions afin de pouvoir les modifier en fonction des feedback utilisateurs. Le choix de solutions potentielles se fait en deux grandes étapes:
- L'ergonome se fonde d'abord sur son expertise et ses connaissances pour déterminer un éventail de choix possibles.

- Ensuite, il teste ces options avec les utilisateurs pour définir la plus adaptée.

Il faut donc faire des choix concernant la navigation, l'architecture de l'information, les styles de dialogue et le design. Cela fait appel aux autres spécialistes en interface homme-machine (notamment designers et architectes de l'information).

On peut par exemple créer des maquettes statiques sur papier et mimer l'enchaînement des écrans en fonctions des réponses de l'utilisateur (c'est le prototypage papier). On peut aussi concevoir des storyboards puisqu'il s'agit de permettre à l'utilisateur de jouer des scénarios d'utilisation en fonction des consignes. On peut construire ces prototypes plus réalistes en html ou en utilisant des outils logiciels tels que PowerPoint ou Flash. La discussion de ces solutions très tôt dans le cycle sont essentiels pour percevoir les problèmes d'utilisabilité avant qu'ils ne soient trop coûteux à résoudre.

e. Évaluer les solutions conçues au regard des exigences

Le pilotage de tests utilisateurs selon un protocole d'évaluation précis permet de détecter facilement les défauts de l'interface selon leur importance en fonctions des objectifs d'utilisabilité définis précédemment (*y a-t-il des choses qui auront peu d'impact sur mes objectifs d'utilisabilité ?*). Pour choisir l'option de conception qui corresponde le mieux aux exigences fonctionnelles et des utilisateurs, on peut aussi procéder à une évaluation experte sur la base des heuristiques, normes, recommandations et critères en ergonomie des interfaces. Des questionnaires de satisfaction, de préférences peuvent être proposés aussi aux utilisateurs.

L'objectif de l'évaluation des solutions est de recueillir un feedback sur la conception développée. Il permettra d'améliorer la conception. C'est une évaluation de la satisfaction des objectifs utilisateur et organisationnels.

V.5. Principales techniques d'analyse en UCD

En adoptant une conception d'IHM centrée sur l'utilisateur, plusieurs techniques sont connues pour recueillir des informations, proposer des solutions et aussi les évaluer. Chaque technique peut être utile pour un ou plusieurs aspects de cette démarche. Les plus connues de ces techniques sont :

- **Tri de carte :** réunir des utilisateurs représentatifs du public-cible et représenter chaque fonctionnalité par une carte. les utilisateurs regroupent les cartes qui vont ensemble puis nomment les groupes (menus) .

- **Brainstorming :** générer un grand nombre d'idées créatives imaginées par les participants à la conception pour résoudre un problème. les idées sont choisies et classées par nombre de votes.

- **Focus group** : réunir des utilisateurs représentatifs du public-cible (dix maximum), échanges dirigés informels sur le sujet étudié, via des activités support puis synthèse des idées exprimées.

- **Entretien critique :** identifier des exemples de problèmes rencontrés concrètement par les utilisateurs dans la situation actuelle avec interview avec l'utilisateur sur son environnement de travail en lui demandant de se souvenir d'un problème particulier déja vécu.

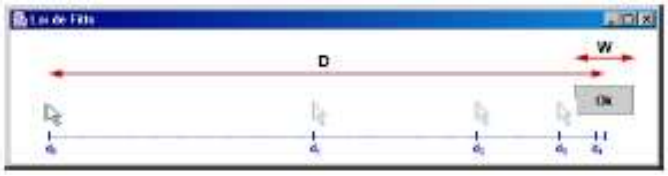
V.5. Quelques lois liées à l'utilisabilité

Lors de l'évaluation des solutions proposées, certaines lois connues en IHM peuvent être utilisées surtout par les experts afin de faire des choix ou d'améliorer des fonctionnalités comme par exemple la loi de Fitts et la loi de Hick.

V.5.1. Loi de Fitts

Dans une interface graphique, le ***temps minimal mis pour atteindre une cible*** est fonction de sa distance ***D*** et de sa taille ***W***.

- Le temps ***T*** croît proportionnellement au logarithme du rapport ***D/W***.
- Un mouvement est décomposable en une suite de micromouvements.



D	10 cm	10 cm	30 cm
W	1 cm	1 mm	2 mm
T	0.43 s	0.78 s	0.82 s

T est donné par la ***loi de Fitts*** par la formule : $T = k.\ log_2(2D/W)$

k : constante qui dépend des temps de cycles ***Tp*** et ***Tm*** *(temps des processeurs perceptif et moteur).* Elle est de l'ordre de ***0.1 sec.***

Différentes formulations de la loi de Fitts ont été proposées suite à un certain nombre d'études théoriques et de vérifications expérimentales (ces nouvelles formulations permettent d'éviter que le résultat de l'expression devienne négatif, ce qui est difficile à interpréter pour un temps).
La variante la plus utilisée actuellement est la formulation de Shannon de la loi de Fitts :

$$T = a + b.\ log_2(D/W + 1)$$

Dont ***a*** et ***b*** sont des constantes déterminées expérimentalement et qui dépendent principalement des propriétés du dispositif de pointage et accessoirement des aptitudes de l'utilisateur.

Le terme logarithmique de l'expression ($log_2(D/W + 1)$) est appelé **indice de difficulté.** Cette formulation ne change pas radicalement le comportement du temps en fonction de l'indice de difficulté.

D	10 cm	10 cm	30 cm
W	1 cm	1 mm	2 mm
T	0.46 s	0.77 s	0.82 s

Par exemple, avec a=0.1 s et b=0.1 s, on obtient les valeurs:

V.5.1. Loi de Hick

La ***loi de Hick*** (ou ***Hick-Hyman***) permet de déterminer le ***temps moyen*** qu'il faut à un utilisateur pour prendre une décision en fonction du nombre de choix qu'il a à disposition.

- Ce temps ***T*** croît proportionnellement au logarithme du nombre de choix (l'utilisateur constitue des catégories et en élimine environ la moitié à chaque étape de son processus de décision).
- Si l'on a ***n*** choix équiprobables, le temps moyen ***T*** mis par l'utilisateur pour prendre sa décision est donné par la formule :

$$T = b.\ log_2(n + 1)$$

Dont ***b*** est une constante qui dépend de l'utilisateur et du contexte (difficulté de la décision). Elle est de l'ordre de 0.15 sec.

Le terme logarithmique de l'expression (***log2(n+1)***) est appelé ***entropie de la décision*** *(*notée ***H****),* c'est-à-dire :

$$T = b.H$$

Le tableau ci-aparés montre quelques valeurs :

n	2	5	10	50
T	0.24 s	0.39 s	0.52 s	0.86 s

Si les ***n*** choix ne sont pas équiprobables, l'entropie ***H***, et donc le temps ***T*** associé, dépendent de la probabilité ***pi*** de chacun des choix. Obtient donc les formules suivantes:

$$H = \sum_{i}^{n} p_i \log_2 \left(\frac{1}{p_i} + 1 \right)$$

$$T = b \cdot \sum_{i}^{n} p_i \log_2 \left(\frac{1}{p_i} + 1 \right)$$

- Des options plus probables que les autres font baisser le temps moyen nécessaire pour prendre la décision.
- D'une manière générale, l'ajout d'une option supplémentaire a plus d'impact dans une liste avec peu d'options que dans une longue liste.
- La loi de Hick peut parfois être utile pour déterminer le nombre d'options à proposer et le nombre de niveaux d'imbrication (dans un menu par exemple).

Chapitre VI: Présentation de l'API Swing de java

VI.1. Introduction

Swing est une boite à outils permettant la création d'interfaces utilisateur en Java. C'est une amélioration de la boite à outils **AWT** (*Abstract Window Toolkit*) qui était la boite à outils originale tout en s'appuyant dessus. Les interfaces graphiques ainsi créées sont indépendantes de l'OS et de l'environnement graphique sous-jacents. Swing fournit les classes pour la représentation des différents éléments d'interfaces graphiques : fenêtres, boutons, menus, etc., et la gestion de l'interaction par événements. Ces classes se trouvent dans le paquetage javax.swing et ses sous-paquetages.

Pour construire une interface graphique avec Swing, on utilise des éléments préfabriqués (boutons, zone de textes, listes déroulantes, etc.) qu'on peut assembler et arranger selon les besoins comme on peut aussi construire de nouveaux éléments plus complexes.

VI.2. Architecture générale de Swing

Java Foundation Class (JFC) est un ensemble regroupant des ajouts à Java 1.1 qui sont désormais intégrés au SDK 1.2 et suivants (Java 2 Platform). Le JFC comprend :

-***les composants Swing*** qui sont des composants avancés complètement écrits en langage Java.

-***Java 2D*** : Utilisation de classes Graphics 2D amenant des manipulations complexes de la couleur, la manipulation simple des transformations affines (rotation, homothétie, …), traitement des textures, …

- ***L'accessibilité*** : la manipulation simple des ordinateurs pour les personnes handicapés moteurs

-Le « drag and drop » : glisser-déposer entre application quelconque (pas forcément Java) sur une plate-forme.

En fait Swing hérite de AWT, c'est-à-dire que les classes Swing s'appuient sur l'arborescence AWT. Les principales classes Swing sont montrées en bas de la figure suivante:

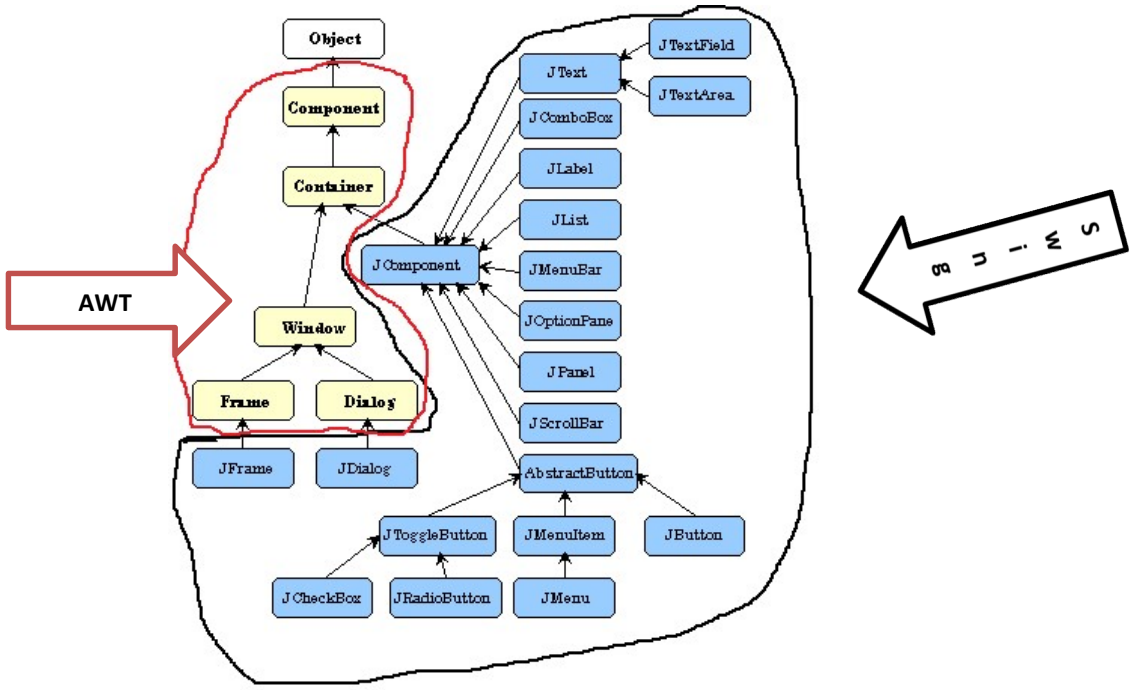


Figure 38: Architecture de Java/Swing

VI.3. Composants lourds / légers

En java, les composants dérivent tous de la classe *java.awt.Component*. Les composants *AWT* sont liés à la plate-forme locale d'exécution, car ils sont implémentés en code natif du système d'exploitation hôte et la Java Machine y fait appel lors de l'interprétation du programme Java. Ceci signifie que dès lors que vous développez une interface AWT sous Windows, lorsque par exemple cette interface s'exécute sous MacOs, l'apparence visuelle et le positionnement des différents composants changent. En effet la fonction système qui dessine un composant sous Windows ne le dessine pas de la même façon sous Mac ou Linux et des chevauchements de composants peuvent apparaître si vous les placez au pixel près (d'où la nécessité des gestionnaires de placement qu'on va voir après). Les composants qui dépendant du système hôte sont appelés en Java des composants lourds. En Java le composant lourd est identique en tant qu'objet Java et il est associé localement lors de l'exécution sur la plateforme hôte à un élément local dépendant du système de l'hôte dit *peer*.

Résultat → Tous les composants du package AWT sont des composants lourds.

Par opposition aux composants lourds utilisant des *peer* de la machine hôte, les composants légers sont entièrement écrits en Java. En outre un tel composant léger n'est pas dessiné visuellement par le système, mais en Java. Ceci apporte une amélioration de portabilité et permet même de changer l'apparence de l'interface sur la même machine grâce à la classe *lookAndFeel* qui permet de déterminer le style d'aspect employé par l'interface utilisateur.

Résultat → La majorité des composants Swing (package javax.swing) sont des composants légers.

En Java on ne peut pas se passer de composants lourds (communiquant avec le système) car la Java Machine doit communiquer avec son système hôte. Par exemple la fenêtre étant l'objet visuel de base dans les systèmes modernes elle est donc essentiellement liée au système d'exploitation et donc ce sera en Java un composant lourd. Dans le package Swing le nombre de composants lourds est réduit au strict minimum soient quatre types de fenêtres (*JFrame, JDialog, JWindow, JApplet*).

Les fonctionnalités des composants se décomposent essentiellement en deux catégories :

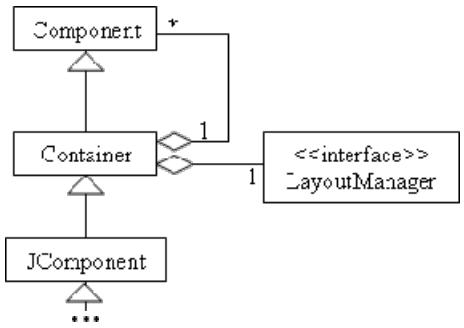
- **Apparence du composant :** il s'agit par exemple de la gestion d'attributs de base comme la visibilité, la taille, la couleur, la police, etc…
  L'apparence d'un objet sur l'écran est assurée par la méthode *paint()* qui est au moins appelée lors du premier affichage du composant et aussi la méthode re*paint()* pour redessiner un composant. Celle-ci demande à Swing de prévoir un appel à *paint()*. Swing se charge ensuite d'organiser et d'optimiser au mieux les différentes requêtes pour redessiner les composants.

- **Comportement du composant :** il s'agit de la réaction du composant aux **événements** contrôlés par l'utilisateur. Un événement est une action sur l'interface qui est susceptible de déclencher une réaction. Ces actions peuvent être un click de souris, le déplacement de la souris, le redimensionnement d'un objet, son masquage, sa sélection, la frappe d'une touche du clavier, etc. Pour ces actions, Swing envoie une notification d'événement aux objets qui se sont **abonnés** en tant que "**listener**" auprès du composant qui est à l'origine de l'événement. Ce listener doit posséder les méthodes de réaction à l'événement et celles-ci sont alors invoquées. C'est le g**estionnaire d'événements** de Swing qui est en charge de ce mécanisme.

VI.4. Les gestionnaires de placement

Les gestionnaires de placement (*layout manager*) permettent de disposer des composants dans un conteneur en fonction de leurs caractéristiques comme par exemple la taille préférée (*preferredSize)*. Plusieurs gestionnaires de placement sont disponibles avec Swing.

Chaque conteneur peut contenir soit des composants atomiques, soit d'autres conteneurs. Le placement des composants dans un conteneur est une stratégie : le conteneur délègue à un *LayoutManager* la responsabilité de placer les composants en fonction de leurs tailles préférées, ainsi que des contraintes du conteneur lui-même. Le diagramme de classes ci-après montre ces relations :

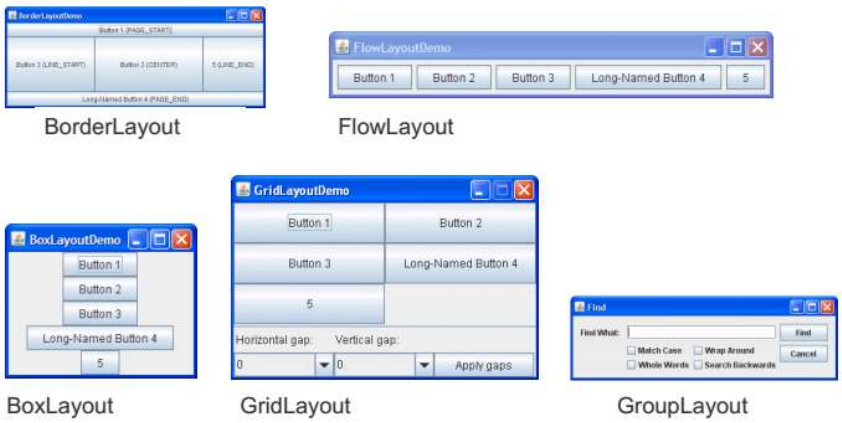


La gestion de placement dans un contenneur est assurée par les méthodes suivantes :

void setLayout(LayoutManager lm)	Affectation d'un gestionnaire de placement à un conteneur. Sinon mettre *null*
LayoutManager getLayout()	Retourne le getsionnaire de placement d'un conteneur.
void doLayout()	La méthode entraîne le replacement de tous les composants contenus dans le conteneur.

void validate()	La méthode entraîne le replacement de tous les composants contenus dans le conteneur, en appelant *doLayout()*.

Les différentes dispositions qu'on peut utiliser avec Java Swing sont montrées dans la figure suivante :



Un exemple récapitulatif

La classe JFrame

Pour utiliser une fenêtre de type JFrame, on doit l'instancier et initialiser les paramètres de la fenêtre (titre, taille, …) puis la rendre visible, comme ceci :

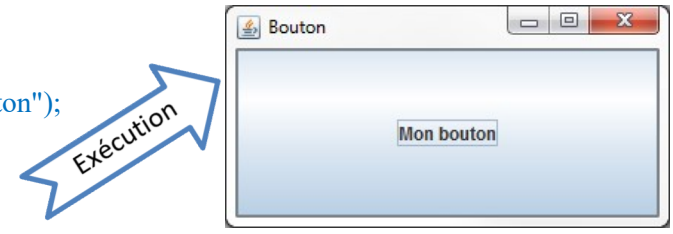
```
import javax.swing.JFrame;

public class Fenetre extends JFrame {

 public Fenetre(){

        this.setTitle("Ma première fenêtre Java");

        this.setSize(400, 500);

        this.setLocationRelativeTo(null);

       this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

       this.setVisible(true);

} }
```

Utiliser la classe JButton

La classe JButton issue du package javax.swing permet d'ajouter des boutons sur la fenêtre. Mais pour mieux gérer la disposition, il vaut mieux utiliser un objet conteneur par instanciation de la classe JPanel. Dans la classe Fenetre, nous allons créer une variable d'instance de type JPanel et une autre de type JButton. Faisons de JPanel le content pane de notre Fenetre, puis définissons l'étiquette puis on le met sur le panel. Le code devient donc:

```
import javax.swing.JButton;
```

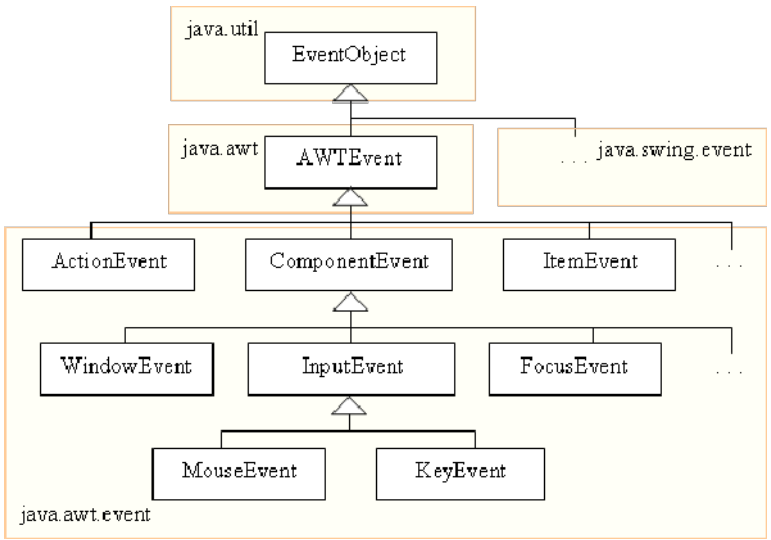
```
import javax.swing.JFrame;
import javax.swing.JPanel;
public class Fenetre extends JFrame {
  private JPanel pan = new JPanel();
  private JButton bouton = new JButton("Mon bouton");
  public Fenetre(){
    this.setTitle("Bouton");
    this.setSize(300, 150);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setLayout(new BorderLayout());
    pan.add(bouton, BorderLayout.CENTER);       //Ajout du bouton au centre de la fenètre
    this.setContentPane(pan);
    this.setVisible(true);
  }      }
```



VI.5. Gestion des évènements

Les composants Swing ont la capacité de gérer des événements qui sont générés soit par programmation soit par une action de l'utilisateur sur le composant. Ces événements peuvent déclencher une action qui doit être exécutée par d'autres composants.

- Un composant qui crée des événements est appelé ***source***. Le composant source délègue le traitement de l'événement au composant ***auditeur***.
- Un composant qui traite un événement est appelé ***auditeur*** ou ***écouteur*** (listener).
  En java, les évènements ont une structure hiérarchique qui est montrée dans la figure ci-après :



Pour les interfaces graphiques, généralement on utilise ActionEvent, MouseEvent et KeyEvent dont voici quelques détails :

a. ActionEvent

Méthode	Utilisation
Object getSource()	Retourne l'objet source de l'événement.
int getID()	Retourne le type d'événement.
String getActionCommand()	Retourne le texte associé au composant source de l'événement (bouton ou menu)
int getModifiers	Retourne un entier indiquant si les touches maj, alt ou ctrl étaient appuyées au moment de la génération de l'événement.
long getWhen()	Retourne la date et l'heure de la génération de l'événement.
String paramString()	Retourne une chaîne de caractères contenant toutes les informations précédentes.

Syntaxe d'utilisation :

```
interface ActionListener { void  actionPerformed(ActionEvent e);  }
```

Les composants sources de ActionEvent sont :

- Boutons : JButton, JRadioButton, JCheckBox, JToggleButton
- Menus : JMenuItem, JMenu, JRadioButtonMenuItem, JCheckBoxMenuItem
- Texte : JTextField

b. MouseEvent

Méthode	Utilisation
Object getSource()	Retourne l'objet source de l'événement.
int getID()	Retourne le type d'événement.
Point getPoint()	Retourne les coordonnées de la souris lors de la génération de l'événement.
int getX()	Retourne la coordonnée en X de la souris lors de la génération de l'événement.
int getY()	Retourne la coordonnée en Y de la souris lors de la génération de l'événement.
int getModifiers	Retourne un entier indiquant si les touches maj, alt ou ctrl étaient appuyées au moment de la génération de l'événement.
long getWhen()	Retourne la date et l'heure de la génération de l'événement.
int getButton())	Retourne quel bouton a été cliqué. les trois valeurs possibles

	sont : *MouseEvent.BUTTON1*, *MouseEvent.BUTTON2*, *MouseEvent.BUTTON3*
int getClickCount()	Retourne le nombre de clics associés à cet événement.

Syntaxe d'utilisation :

```
interface MouseListener  {
  void mouseClicked(MouseEvent  e);
  void mouseEntered(MouseEvent  e);
  void mouseExited(MouseEvent e);
  void mousePressed(MouseEvent e);
  void mouseReleased(MouseEvent e);    }
```

Tous les composants peuvent être sources d'un évenement de type *MouseEvent* . Lors d'un clic les méthodes sont appelées dans l'ordre suivant *: mousePressed*, *mouseReleased* puis *mouseClicked.*

c. KeyEvent

Les événements sont générés dans l'ordre :

1. *keyPressed*
2. *keyTyped* : pour les touches qui ont un effet sur le texte d'un composant texte.
3. *keyReleased*

Object getSource()	Retourne l'objet source de l'événement.
int getID()	Retourne le type d'événement.
char getKeyChar()	Retourne le caractère correspondant à la touche du clavier, ou *java.awt.event.KeyEvent.CHAR_UNDEFINED* s'il n'y a pas de caractère associé à la touche (touche de fonction par exemple).
int getKeyCode()	Retourne le code du caractère correspondant à la touche du clavier.
int getModifiers	Retourne un entier indiquant si les touches maj, alt ou ctrl étaient appuyées au moment de la génération de l'événement.
long getWhen()	Retourne la date et l'heure de la génération de l'événement.
String getKeyModifiersText(int mod) >	Retourne le modifieur sous forme de chaîne de caractères : Maj+Crtl+Alt.

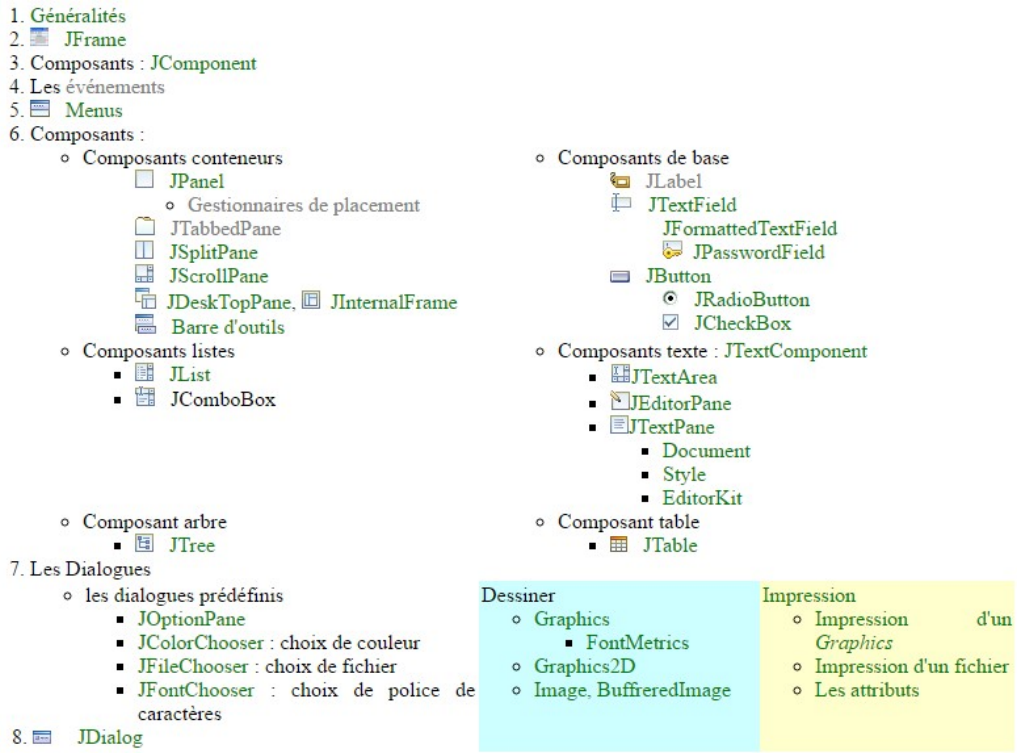
Syntaxe d'utilisation :

```
interface MouseListener {
```

```
void keyTyped(KeyEvent e);

void keyPressed(KeyEvent e);

void KeyReleased(KeyEvent e);   }
```

Tous les composants peuvent être sources de *KeyEvent*.

6. Les principaux composants Swing



Chapitre VII: Méthode d'implémentation efficace du modèle PAC basée sur l'utilisation de design patterns

VII.1 Introduction et historique

Les patrons de conception ont été formellement reconnus en 1994 à la suite de la parution du livre *Design Patterns: Elements of Reusable Software (écrit par* Gamma, Helm, Johnson et Vlissides (*Gang of Four - GoF))*. Il décrit vingt-trois « patrons GoF » et comment s'en servir.
Les patrons de conception sont utilisés dans le développement des IHM, notamment pour faciliter la mise en place de l'approche MVC. D'ailleurs, l'API Swing incorpore plusieurs patrons de conception dans sa structure et son fonctionnement. Ainsi, l'utilisation de classe implémentant des « Listeners » pour la gestion d'événements sur les composantes graphiques correspond aux patrons de conception « Observer ». L'imbrication de composants graphiques entre elles et l'appel en cascade des méthodes correspondent au patron de conception « Composite ». Enfin, un autre exemple est celui du « AbstractTableModel » ou « AbstractListModel » qui implémente le patron de type « Template ».

VII.2 Définitions

Plusieurs définitions ont été données pour un patron de conception (design pattern), dont voici quelques-unes :

- **Coad [1992]** : Une abstraction d'un doublet, triplet ou d'un ensemble de classes qui peut être réutilisé encore et encore pour le développement d'applications
- **Aarsten [1996]** : Un groupe d'objets coopérants liés par des relations et des règles qui expriment les liens entre un contexte, un problème de conception et sa solution.
- **Appleton [1997]** : Une règle tripartite exprimant une relation entre un certain contexte, un certain problème qui apparaît répétitivement dans ce contexte et une certaine configuration logicielle qui permet la résolution de ce problème.

Les patrons de conception sont à la fois des solutions génériques de définition de classes en réponse aux problèmes couramment liés au développement logiciel et des bonnes pratiques de développement. Ils mettent l'accent sur la modularité, la réduction du couplage entre les classes et la réutilisabilité.

But = Proposer de bonnes solutions types à des problèmes de conception récurrents et identifiés.

Les patrons de conception sont divisés en 3 groupes : les patrons de construction (ou création), les patrons structuraux et les patrons comportementaux :

a. Patrons de création : leur utilisation consiste à :

- Description de la manière dont un objet ou un ensemble d'objets peuvent être créés, initialisés, et configurés.
- Isolation du code relatif à la création, à l'initialisation afin de rendre l'application indépendante de ces aspects.

Patrons de ce type: **Abstract Factory, Builder, Factory Method, Prototype, Singleton.**

b. Patrons de structure : leur utilisation consiste à :

- Description de la manière dont doivent être connectés des objets de l'application afin de rendre ces connections indépendantes des évolutions futures de l'application.
- Découplage de l'interface et de l'implémentation de classes et d'objets.

Patrons de ce type : **Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy.**

b. Patrons de comportement : leur utilisation consiste à :

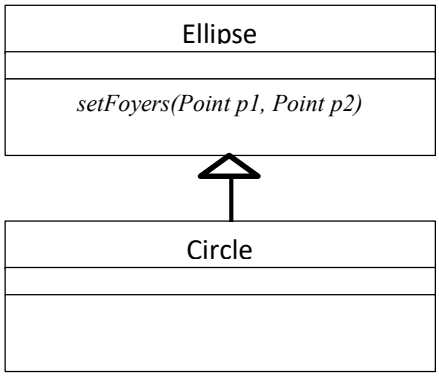
- Description de comportements d'interaction entre objets.
- Gestion des interactions dynamiques entre des classes et des objets.

Patrons de ce type : **Command, Interpreter, Iterator, Mediator, Memento.**

Chaque patron de conception doit être présenté par les éléments suivants :

- **Pattern Name and Classification :** Le nom du pattern, évoque succinctement l'esprit du pattern.
- **Intent** Une brève description qui répond aux questions suivantes :
  a. Qu'est-ce que le design pattern fait ?
  b. Quels sont son objectif et sa justification ?
  c. Quel problème de conception particulier aborde-t-il ?
- **Also Known As :** Autres noms connus pour le pattern, s'il y en a.
- **Motivation** Un scenario qui illustre un problème de conception et comment la structuration des classes et des objets dans ce pattern le résolvent. Le scénario devra aider à comprendre la description plus abstraite du pattern qui suit.
- **Applicability :**
  - Quelles sont les situations dans lesquelles le pattern peut être appliqué ?
  - Quels sont les exemples de mauvaise conception que le pattern peut attaquer ?
  - Comment reconnaitre ces situations ?
- **Structure :** Une représentation graphique des classes dans le pattern utilisant une notation basée sur l'Object Modeling Technique (OMT).
- **Participants :** Les classes et/ou les objets qui participent au design pattern et leurs responsabilités.
- **Collaborations :** Comment les participants collaborent pour tenir leurs responsabilités.
- **Conséquences :**
  - Comment le pattern satisfait-il ses objectifs ?
  - Quelles sont les conséquences et résultats de l'usage du pattern ?
  - Quels points de la structure permet-il de faire varier indépendamment ?

Exemple : Le dilemme cercle/ellipse : les cercles sont-ils des ellipses !?
Ellipse : 2 foyers, Circle : 1 seul centre
problème avec la méthode *setFoyers(Point p1, Point p2)*
Ici, il ne faut pas utiliser l'héritage juste pour factoriser du code, c'est-à-dire pour extension/spécialisation. mais éventuellement, on doit utiliser la composition.



VII.3. Exemple de patrons de création : *Abstract Factory*

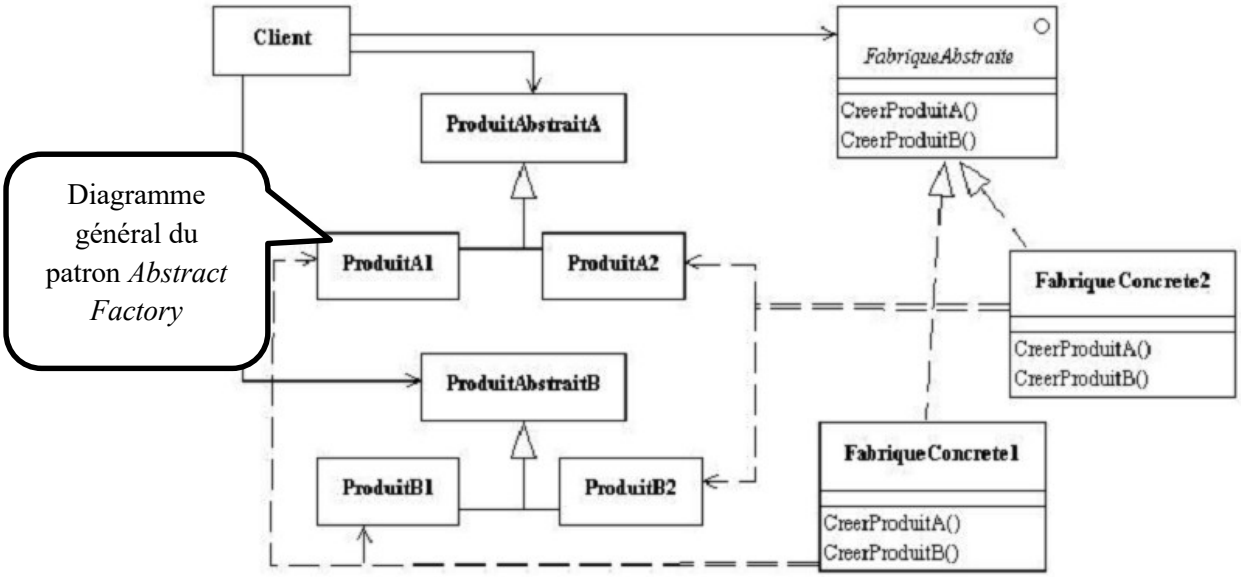
Le but principal d'utiliser un tel patron est de permettre de créer des familles de produits en masquant les mécanismes de choix des classes de mise en œuvre de ces produits. Prenons ici l'exemple de création d'une interface homme-machine indépendante de la plate-forme sur laquelle on exécute cette interface.

Le problème : On veut développer une application graphique multiplateforme donc les problèmes rencontrés sont les suivants :

- Il existe une bibliothèque graphique pour chaque système (Windows, MacOS, Linux, …)
- Les classes de l'IHM sont différentes d'une plate-forme à l'autre

Les solutions possibles sont :

1. Développer quatre applications différentes, donc on a quatre codes sources qui vont vite diverger.
2. Tout écrire dans un seul code source mais avec des structures conditionnelles <*si alors sinon*> ou bien avec des #ifdef #endif (le cas du C++).
3. Utiliser le patron *Abstract Factory,* dans ce cas on peut se baser sur le diagramme de classe suivant (dans le cas général de ce patron) :



Dans cette solution, les classes supérieures ont des rôles importants :

Rôle de la classe **Client** :

- détient une référence sur une Abstract factory
- crée des produits par appel des opérations de cette référence
- ne connaît pas la classe concrète des produits

Rôle de la classe **ProduitAbstrait** :

- Masquer la classe concrète
- Offrir un ensemble d'opérations applicables à toutes les variantes d'un même produit

Rôle de la classe **FabriqueAbstrait** : (veut dire Abstract factory)

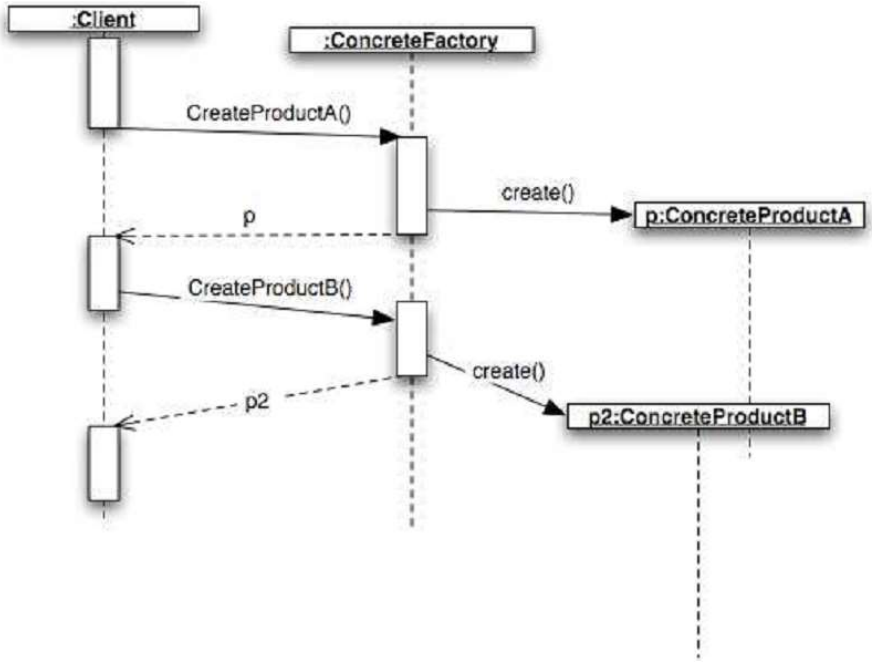
- Comporte une opération de création, pour chaque produit, une opération de création retourne un objet produit
- La classe concrète des produits est masquée.

Rôle de la classe **FabriqueConcrete :**

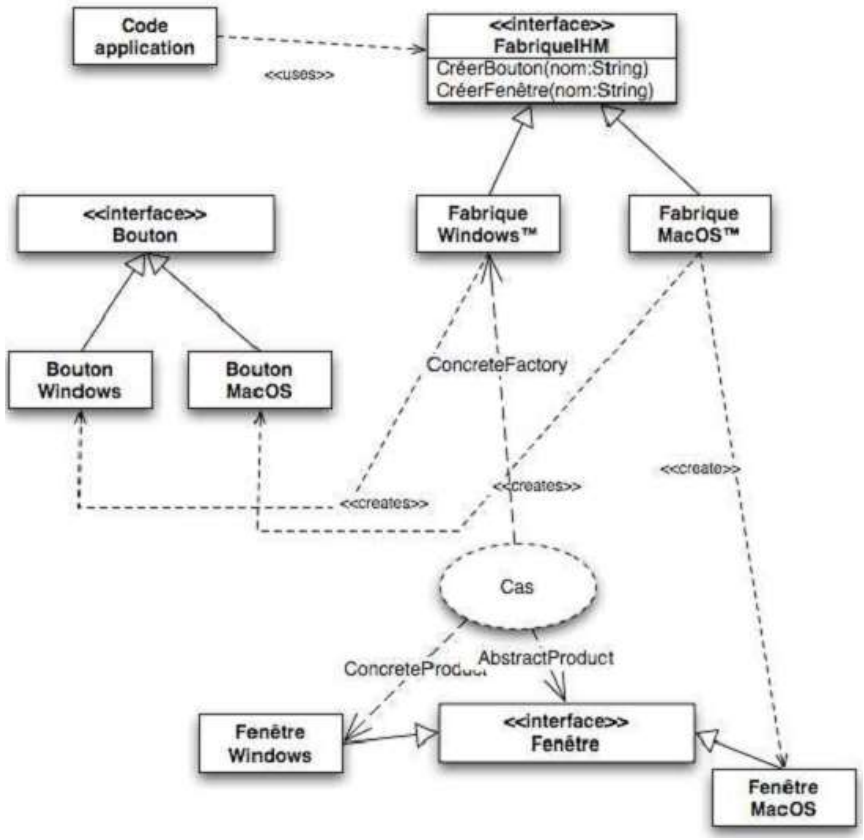
- Contient la mise en œuvre spécifique des opérations

- Non accessible au client
- Peut être amené à jouer un rôle d'adaptateur

Cela peut être modélisé par le diagramme de séquence générique suivant :



Solution typique du problème de l'interface multiplateforme :



VII.4. Rappel sur le modèle PAC

Le modèle PAC définit une structure récursive d'un système interactif sous forme d'une hiérarchie d'agents. Cette hiérarchie permet d'exprimer des relations entre agents et reflète plusieurs niveaux d'abstraction depuis l'application jusqu'aux éléments fins de l'interaction. Un agent PAC définit une compétence à un niveau d'abstraction donné. C'est un acteur à trois facettes: la présentation qui définit le comportement perceptible de l'agent, l'abstraction qui représente son expertise, et le contrôle qui a un double rôle: il sert de lien entre les facettes présentation et abstraction de l'agent et il gère des relations avec d'autres agents PAC de la hiérarchie. En général, l'abstraction du niveau le plus haut correspond à la notion d'application du modèle Seeheim comme le montre la figure ci-après, tandis que le contrôle du niveau le plus haut remplit des fonctions similaires au contrôleur de dialogue.

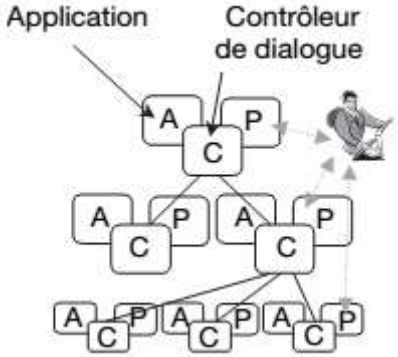


Figure : architecture du modèle PAC

VII.5. Méthodologie de conception basée sur PAC et les patrons de conception

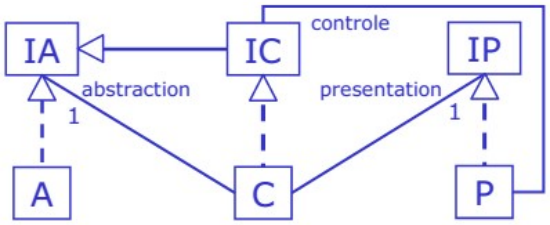
NB : *cette méthodologie a été proposée par des chercheurs français (T.Duval et J.C.Tarby).*
Selon le modèle PAC, une application est décomposée en trois parties principales :

- L'Abstraction correspondante au noyau fonctionnel
- La Présentation qui signifie l'interface elle-même.
- Le Contrôle qui représente le contrôleur de dialogue qui est.

Pour que la méthodologie soit facilement applicable, plusieurs conditions doivent être réunies :

- Le noyau fonctionnel doit être séparé de l'interface.
- L'application doit être codée avec un langage objet proche de Java.
- Le concepteur doit avoir une bonne vue d'abstraction.

Principe de base : Le noyau fonctionnel contient toutes les abstractions qui sont gérées par l'application. L'ensemble de ces abstractions représente l'Abstraction de PAC. Chaque abstraction X du noyau fonctionnel, ainsi que chaque contrôle et chaque présentation associés à ces abstractions, sont représentés par leur interface (au sens UML), respectivement IA_X, IC_X et IP_X. Par ailleurs, l'interface IC de chaque « contrôle » hérite de l'interface IA de « abstraction » associée (voir la figure), ce qui permettra à un composant contrôle d'être un proxy de son composant abstraction associée. Cette caractéristique facilite grandement la structure du code et « soude » chaque abstraction à son contrôle, la présentation venant alors se « greffer » sur cette association abstraction/contrôle. Notons qu'il est interdit de rajouter des méthodes directement dans les classes sans les avoir ajoutées au préalable dans les interfaces associées.



Les classes et les interfaces sont groupées dans des paquetages pour plus de lisibilité. On a donc les paquetages *abstraction, interfaceAbstraction, presentation, interfacePresentation, controle, interfaceControle*,

Si le nombre de classes à développer est très grand, on a intérêt à procéder en plus à un découpage hiérarchique (on aura dans ce cas, des fabriques de fabriques).

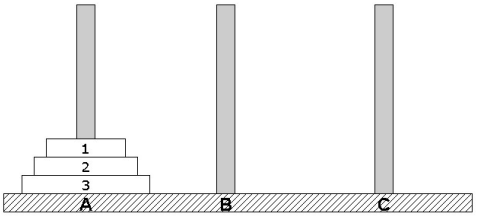
La méthodologie passe par cinq étapes :

Etape 1 : Elaboration du Noyau Fonctionnel (IA et A)

Cette première étape consiste à définir les interfaces des abstractions (IA). Pour chaque abstraction X, on définit dans son interface IA_X les méthodes qui permettent d'accéder aux données associées à l'abstraction. Lorsque l'interface IA_X est définie pour une abstraction X, on crée ensuite la classe abstraction A_X associée.

Quand toutes les interfaces d'abstraction ont été produites, on crée une fabrique A_Factory, en utilisant le patron de conception Fabrique Abstraite, qui se charge de créer toutes les abstractions. Cette fabrique, qui possède également une interface IA_Factory, ne devant être créée qu'à un seul exemplaire, nous utilisons le patron de conception Singleton pour écrire le code qui lui correspond.

VII.6 Exemple : Tour de Hanoi



Etape1 :

Dans le cas du tour de Hanoï, les abstractions sont A_Anneau, A_Tour, et A_Hanoi (qui permet la résolution du problème). Par conséquent, notre méthodologie donnera naissance aux interfaces IA_Anneau, IA_Tour et IA_Hanoi.

Parmi les méthodes, on trouve, par exemple pour IA_Tour, void depiler() et void empiler (IA_Anneau a).

L'interface de fabrique IA_Factory contient quant à elle des méthodes telles que IA_Tour newA_Tour (String nom, int n) ou IA_Anneau newA_Anneau (int v), que la fabrique A_Factory Implémente avec :

```
public IA_Tour newA_Tour (String nom, int n) {
return new A_Tour (nom, n) ;                    }
public IA_Anneau newA_Anneau (int v) {
return new A_Anneau (v) ;                       }
```

Etape 2 : Création des Interfaces de Présentation et de Contrôle (IP et IC)

Pour cette seconde étape, on peut commencer indifféremment par les interfaces de présentation ou de contrôle. Le plus important est de maintenir la cohérence entre les deux composantes.

a. Les interfaces de présentation :

Si l'IHM doit piloter le noyau fonctionnel, cela signifie que des présentations P_X envoient des messages aux abstractions A_X associées en passant par les contrôles C_X. Dans ce cas, d'une part on ajoute aux interfaces de présentation IP_X une méthode IC_X getControle () qui renvoie le contrôle C_Xassocié, et d'autre part, pour éviter à nouveau des appels en boucle infinie, les méthodes que l'on ajoute à la présentation (IP_X et plus tard P_X), s'appelleront *f_in (IP_Y y) ou f_in ()*, suivant que le concepteur décide ou non d'utiliser des paramètres. Ces méthodes *f_in* sont également optionnelles. A ce pilotage direct, via des appels à des méthodes de l'abstraction, peut s'ajouter un pilotage indirect qui nécessite des étapes intermédiaires d'échanges entre la présentation et le contrôle avant de déclencher une méthode de l'abstraction. On peut donc être amené dans ce dernier cas à ajouter de nouvelles méthodes, de type g_*in* et g_*out*, qu'il n'est pas possible de déduire automatiquement des interfaces d'abstraction, dans les interfaces de contrôle et de présentation.

Dans tous les cas, les paramètres des méthodes de l'interface de présentation ne peuvent être que de type « interface de présentation », de type « interface de contrôle » et des types standards (booléen, etc.).
Ces propriétés des interfaces de présentation garantissent que l'IHM est fortement découplée du reste de l'application et peut ainsi être facilement remplacée par une autre IHM respectant ces mêmes contraintes.

b. Les interfaces de contrôle :

Les composants de contrôle seront les proxys de leurs abstractions associées, c'est pourquoi ils devront implémenter la même interface que leur abstraction. Les interfaces de contrôle *IC_X* héritent donc des interfaces *IA_X* et contiennent en plus des méthodes *IP_XgetPresentation ()* qui renvoient les présentations qui leur sont associées. De plus, comme indiqué précédemment, si des méthodes g_in () ou g_in (IP_Y y) (et éventuellement g_out () ou g_out (IP_Y y)) ont été ajoutées au niveau des interfaces de présentation pour piloter le noyau fonctionnel, alors on doit trouver de nouvelles méthodes g () ou g (IC_Y y) au niveau des interfaces de contrôle afin de faire la jonction entre la partie présentation et la partie abstraction. Par exemple, IC_Tour contient des méthodes, appelées par son composant de présentation, dont le but est de traiter les événements graphiques d'interaction de l'utilisateur. C'est le cas de entreeDnD (IC_Anneau a), qui réagit au début du survol d'une tour par un anneau manipulé par l'utilisateur, et qui est appelée par la méthode entreeDnD_in de la présentation.
A la fin de cette étape, on ajoute le code de l'interface de la fabrique, IP_Factory, chargée de créer les interfaces de présentation. Par défaut, les signatures des méthodes de IP_Factory sont directement issues de IA_Factory, même si tout n'est pas nécessaire pour la version finale ; le concepteur pourra ensuite éliminer les méthodes ou les paramètres inutiles. Notons également qu'un paramètre IC_X est automatiquement ajouté dans les paramètres ; à charge du concepteur de l'utiliser ou non.

Exemple : Tour de Hanoi

Etape2 :

Dans notre exemple des tours de Hanoï, parmi les méthodes d'interfaces de présentation, on trouve ainsi dans IP_Anneaula méhode IC_Anneau getControle(), et dans IP_Tour la méthode void empiler_out (IP_Anneau pa). dans IC_Anneau et IC_Tour, on trouve des méthodes getPresentation() qui renvoient respectivement un IP_Anneau et un IP_Tour.
Pour les tours de Hanoï, IP_Factorycontient par défaut :

```
IP_Tour newP_Tour (String nom, int n, IC_Tour controle) ;
```

```
IP_Anneau newP_Anneau (int v, IC_Anneau controle) ;
IP_Hanoi newP_Hanoi (int n, IC_Hanoi controle) ;
```

car IA_Factory contient :

```
IA_Tour newA_Tour (String nom, int n) ;
IA_Anneau newA_Anneau (int v) ;
IA_Hanoi newA_Hanoi (int n) ;
```

Etape 3 : Création des Présentations (P)

Après avoir défini les interfaces de présentation dans l'étape 2, il reste à écrire le code de chaque méthode pour ces interfaces. Par défaut, le constructeur de chaque présentation est du type :

```
public P_X (IC_X controle) { this.controle = controle ; }
```

A la fin de cette étape, en utilisant le patron de conception Singleton, on ajoute le code de la fabrique chargée de créer les présentations, c'est-à-dire P_Factory. Le code de cette fabrique est entièrement déduit, y compris les import, d'une part de son abstraction IP_Factory, et d'autre part des constructeurs des présentations P_X. Par exemple, pour le cas de Hanoï, la P_Factory contient :

Exemple : Tour de Hanoi

Etape 3 :

```
package presentation;
import interfaceControle.IC_Anneau;        (... autres « import »...)
public class P_Factory implements IP_Factory {
public IP_Tour newP_Tour (String nom, int n, IC_Tour controle) {return (new P_Tour (n, controle)); }
public IP_Anneau newP_Anneau (int v, IC_Anneau controle) {return (new P_Anneau (v, controle)); }
public IP_Hanoi newP_Hanoi (int n) { return (new P_Hanoi (n)) ; }
protected P_Factory () {  }
private static IP_Factory instance = new P_Factory () ;
public static IP_Factory getInstance () { return instance ; }
}
```

Etape 4 : Création des Contrôles (C)

De même que les présentations, pour la création des contrôles, il reste à écrire le code de chaque méthode présente dans les interfaces de contrôle. Par défaut, un constructeur de contrôle est du type:

```
public C_X (UnTypeY y) {
abstraction = ConcreteFactory.getAFactory().newA_X(y) ;
presentation = ConcreteFactory.getPFactory().newP_X(y, this) ;
}
```

La fabrique concrète (ConcreteFactory) sera abordée à l'étape 5. Etant donné le lien d'héritage entre IA et

IC , les paramètres du constructeur du contrôle sont identiques avec ceux du constructeur de l'abstraction associée.

Pour l'exemple :de Tour de Hanoi :

Etape 4 :

Pour C_Anneau:

```
public C_Anneau (int v) {
abstraction = ConcreteFactory.getAFactory().newA_Anneau(v) ;
presentation = ConcreteFactory.getPFactory().
newP_Anneau(v,this) ;
}
```

Pour terminer cette étape, on ajoute le code de la fabrique chargée de créer les contrôles, c'est-à-dire C_Factory. Avec l'exemple de Hanoï, *C_Factory* contient par défaut la méthode *newA_Anneau(int v)*car *A_Anneau*a un constructeur A_Anneau(int v), la méthode newA_Tour (String nom, int n)car le constructeur de *A_Tourest A_Tour (String nom, int n)*, etc. Le code par défaut de ces deux méthodes est donc:

```
public IA_Anneau newA_Anneau (int v) { return (new C_Anneau (v)) ; }
public IA_Tour newA_Tour (String nom, int nbAnneauxMax) { return (new C_Tour (nom,
nbAnneauxMax)) ; }
```

Etape 5 : Finalisation du code

A cette dernière étape, le code produit est complété par l'écriture du paquetage correspondant à la concreteFactory qui aura la charge d'instancier les classes définies lors de ces quatre étapes précédentes. Pour notre exemple, le code de cette « concrete factory » est :

```
package concreteFactory;
(...import ...)
public class ConcreteFactory {
protected static IA_Factory aFactory ;
public static void setAFactory(IA_Factory f) { aFactory = f ; }
public static IA_Factory getAFactory() { return (aFactory) ; }
(... idem pour C_Factory, et P_Factory...)
public static void setFactory(IA_Factory f) { aFactory = f ;}
public static IA_Factory getFactory() { IA_Factory factory = aFactory ;
if (cFactory != null) { factory = cFactory ; }
return (factory) ; }
}
```

Références bibliographiques

- *David Benyon, Designing Interactive Systems: A Comprehensive Guide to HCI, UX and Interaction Design,Pearson; 3 edition, 2013*
- *Yvonne Rogers, Helen Sharp & Jenny Preece, Interaction Design: beyond human-computer interaction (3rd edition), Wiley, 2011*
- *Norman DA, The Design of Everyday Things, Basic Books, 2002. Serengul Smith-Atakan The FastTrack to Human-Computer Interaction, (Paperback)Thomson Learning, 2006.*
- *Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, design Patterns, catalogue de modèles de conception réutilisables - International Thomson Publishing 1996*
- *Nathalie Lopez, Jorge Migueis, Emmanuel Pichon - Intégrer UML dans vos projets Eyrolles*
- *Bertrand Meyer - Conception et programmation orientées objet - Eyrolles*
- *Joëlle COUTAZ et Laurence NIGAY, Architecture logicielle conceptuelle des systèmes interactifs, Chapitre 7 du livre Analyse et Conception de l'Interaction Homme-Machine dans les systèmes d'information de Christophe Kolski, Eyorolles 2001.*
- *William Germain Dimbisoa, Thèse de doctorat «Génération automatique des IHM à partir de modèles conceptuels selon l'approche MDE», Université de Fianarantsoa, 2020.*
- *Sophie Dupuy-Chessa , Modélisation en Interaction Homme-Machine et en Système d'Information : A la croisée des chemins, Habilitation a diriger des recherches en informatique de l'université de grenoble, 2011.*
- *Fares Zaidi, Processus, méthodes et outils de développement centré utilisateur : contributions à la réduction des gaps recherche-pratique en conception et évaluation des IHM automobiles embarquées, Thèse de doctorat en ergonomie à l'université de Loraines 2020.*
- *Thierry Duval, Jean-Claude Tarby, "Améliorer la conception des applications interactives par l'utilisation conjointe du modèle PAC et des patrons de conception", publié dans "IHM 2006 (2006) 225-232" DOI : 10.1145/1132736.1132773.*

Annexe A: Travaux pratiques à réaliser à la salle avec l'enseignant

TP1 : Utilisation des listes

On considère l'ensemble des listes suivantes:
Element_Type = {Colors, Days} : une liste déroulante qui exprime le type d'élément à afficher dans une deuxième liste.
Colors = {Red, Green, Blue, Yellow, Orange, White, Black}: Une liste simple avec les couleurs citées.
Days = {Sunday, ……., Friday, Saturday} : Une liste simple avec les jours de la semaine.
En utilisant seulement une liste déroulante et une liste simple en java swing, on vous demande de réaliser une interface qui permet de :

- Choisir le type d'éléments de la liste dans Element_Type ;
- Changer automatiquement le contenu de la deuxième liste selon ce choix, ça sera les éléments de Colors ou bien Days.
- Afficher l'élément sélectionné de la deuxième liste dans une étiquette.

TP2 : Utilisation des tables

1. Ecrire le code java qui permet de créer, remplir et afficher la table suivante sans utiliser la palette d'objets Swing (seulement pour le composant table).

Nom	Prénom	Note1	Note2	Décision
Ghanem	Reda	12	14	Admis
Zidane	Mohamed	08	10	Ajourné
…	…	…	…	…

2. En se basant maintenant sur les composants graphique Swing (utiliser la palette), réaliser une interface qui permet de lire les dimensions N et M d'une matrice A et ces éléments puis la réafficher dans une autre matrice B.

TP3 : Modèle MVC

On veut créer une interface permettant le contrôle de la température en degrés Celsius. Cette interface comporte deux vues représentant la même température avec des vue différentes. La modification d'une vue doit mettre automatiquement à jour l'autre.

1. Proposer une modélisation MVC.
2. Réaliser cette interface.

TP4 : Utilisation des dialogues

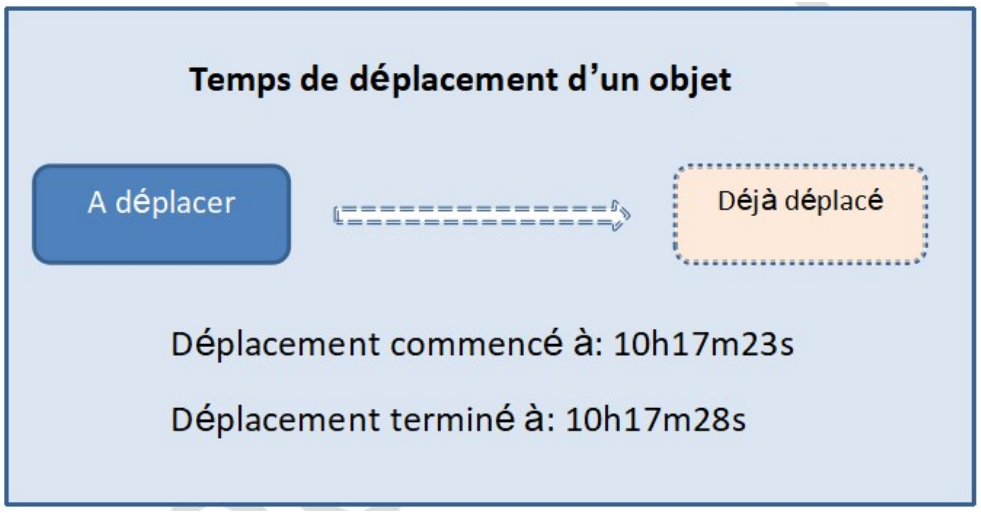
- Une interface de selection de fichier (file chooser) ou de couleur (colourchooser).
- Récupération de paramètres saisis dans une boite de dialogue.

TP5 : Utilisation des graphiques

Réalisation d'un traceur de courbes

TP6 : Gestion éventements en Java Swing

Réaliser une interface qui permet de déplacer un objet (Bouton on Label par exemple) par la souris puis affiche le temps de début et de fin de l'opération de déplacement.



Annexe B: Mini projet à réaliser par binôme d'étudiants

1. Objectif du travail

Le travail consiste à développer une application java pour la simulation de stationnement d'un véhicule dans un parking dont voici le cahier des charges :

1.1. Les paramètres nécessaires

Le parking dispose de 10 places connues par leur numéro (de 1 à 10).
Une place est à l'état libre ou occupée.
Chaque opération de placement d'un véhicule à un temps calculé par le système.

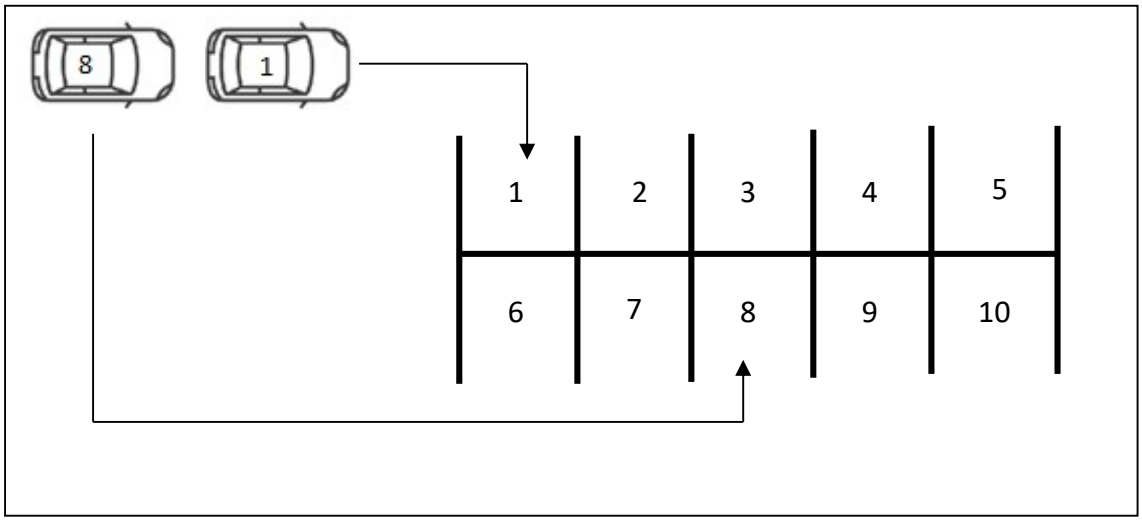
1.2. Les fonctionnalités demandées

L'interface à développer doit fournir à l'utilisateur les fonctionnalités suivantes:

- Réinitialiser : permet de vider les places et placer le(s) véhicule(s) au point de départ.
- Entrer le nombre de véhicules à stationner (de 0 à 10).
- Choisir un numéro de place pour chaque véhicule à stationner.
- Faire bouger le véhicule pour le placer dans la place choisie.
- Suivre et signaler le bon stationnement d'un véhicule dans une place.
- Calculer le temps pris par l'utilisateur pour placer les véhicules.
- Signaler toute erreur ou fausse manipulation(place inexistante, place occupée, ...)

1.3. Coté ergonomique

- Le placement d'un véhicule dans une place est une opération qui doit être visible à l'utilisateur avec un compteur de temps. Le déplacement ce fait par les touches de direction du clavier et avec la souris aussi.
- Le bon stationnement n'est signalé que si le véhicule est placé dans la place précisée.



2. Travail demandé

2.1. Partie conception (Rapport)

Comme une conception simplifiée du système, il est demandé de :

-Expliquer la structure logicielle et le fonctionnement en utilisant quelques diagrammes UML (diagramme des cas d'utilisation et diagramme de classes).

-Expliquer les composantes du modèle MVC qui correspond à un véhicule.

-Citer les objets graphiques utilisés et les évènements associés pour assurer l'interaction.

-Justifier le choix des styles et rappeler les recommandations ergonomiques suivies.

2.2. Partie pratique (Logiciel)

Il est demandé de développer l'application conformément à la conception faite et en utilisant un toolkit graphique java (Swing ou bien Fx).

3. Evaluation

Le travail sera évalué en prenant en considération les points suivants :

- Qualité de la conception UML.
- La modélisation en MVC.
- Correspondance entre la conception et la réalisation.
- Fonctionnement et ergonomie.

4. Remise des travaux

Le travail se fait en binôme et sera remis et évalué deux semaines avant la fin du semestre (remise du rapport, consultation de l'application, signature sur une liste de présence en une date à préciser par la suite).

// Cette fiche est disponible en ligne avec le cours.