Méthodes d'analyse et de conception orientées objet : présentation d'UML (Unified Modeling Language)

Bessai *F.Z.*
CEntre de Recherche sur ***l'Information*** *Scientifique et Technique*
Laboratoire Base de Données et Système d'Information
E-Mail *ZBessai@mail. cerist. dz*

INTRODUCTION

L'approche objet a pour but une modélisation des propriétés statiques et dynamiques de l'environnement dans lequel sont définis les besoins. Il est appelé le domaine de problème. Elle formalise notre perception du monde et des phénomènes qui s'y déroulent et met en correspondance l'espace de problème et l'espace de la solution, en préservant la structure et le comportement du système analysé.

L'approche objet est loin d'être une idée récente. Simula, premier langage de programmation a implémenté le concept de type abstrait à l'aide de classes, date de 1967. En 1976, Smalltalk implémenté les concepts fondateurs de l'approche objet: encapsulation, agrégation et héritage.

L'approche objet est donc devenue une réalité. Utiliser l'approche objet n'est plus une mode, mais un réflexe quasi-automatique dès lors qu'on cherche à concevoir des logiciels complexes qui doivent *résister* à des évolutions incessantes.

Bien que les racines de l'objet soient solidement ancrées dans les années 60, ce n'est que vers les années 80 que les méthodes objet ont commencé à émerger. Par la suite, l'approche objet est devenue incontournable pour le développement de systèmes, qu'ils soient industriels ou de gestion, de simulation ou de contrôle de processus. Devant cet engouement pour les techniques orientées objet, il est naturel de s'attendre à une augmentation du nombre de méthodes.

Après une définition des concepts de l'approche objet, une étude comparative des cinq méthodes, les plus diffusées à l'heure actuelle à savoir OOA (Oriented Object Analysis), OOD (Oriented Object Design), OOA/OOD, OMT (Object Modeling Technic) et OOSE (Oriented Object Software Engineering) sera faite, qui sera suivit d'une présentation du langage UML et nous terminerons pas une conclusion sur la modélisation et la conception orientée objet.

I- LES CONCEPTS DE BASE DE L'APPROCHE OBJET

Trois points de vue ont conduit à la notion d'objet : le point de vue structurel, point de vue conceptuel et le point de vue acteur.

- Le point de vue structurel : l'objet est perçu comme une instance d'un type de données (classe) caractérisé par une structure cachée par des opérations.

- Le point de vue conceptuel : l'objet correspond à un concept du monde réel qui peut être spécialisé.

- Le point de vue acteur : l'objet est une entité autonome et active qui répond à des messages.

I.1- Objet

I.1.1- Définition

L'objet est une abstraction d'une occurrence d'entité du monde réel ou virtuel caractérisée par un identificateur unique et invariant, une classe d'appartenance et un état représenté par une valeur simple ou structurée, il peut être vu :

Objet = identité + comportement + état.

- **L'identité** : caractérise l'existence propre d'un objet, elle permet de le distinguer de façon non ambiguë, et cela indépendamment de son état.

- **Le comportement** : regroupe toutes les compétences d'un objet et décrit les actions et les réactions de cet objet. Chaque atome de comportement est appelé opération ou méthode. Les opérations d'un objet sont déclenchées suite à l'envoi de message par un autre objet.

- **L'état** : l'état d'un objet, à un moment donné, correspond à une sélection de valeurs, parmi toutes les valeurs possibles des différents attributs d'un de cet objet, sachant qu'un attribut est une information qui qualifie l'objet.

Exemple d'objet :
L'employé, Tata Omar, n°0001, âgé de 32 ans, travaille en tant que directeur de l'agence B.M.R

N°employer : 0001
Nom : Tata
Prénom : Omar
Age : 32
Grade : directeur
Agence : B.M.R

Le comportement de l'objet Employé est caractérisé par les opérations qu'il peut exécuter. Dans notre cas, nous pouvons avoir les opérations suivantes :

- Nom en majuscule ;
- Changer âge ;
- Changer agence ;
- Changer grade.

I.1.2- Communication entre objets

Le comportement global d'une application repose sur la communication entre les objets qui la composent. La communication entre objets du domaine d'application est de première importance dans la modélisation objet, elle est réalisée à l'aide des envois de messages.

- **Les messages** : le message est l'unité de communication entre objets : l'envoi de message est le seul moyen d'accéder à un objet, les messages permettent l'activation des méthodes des objets. Envoyer un message à un objet c'est lui signifier ce qu'il doit faire.

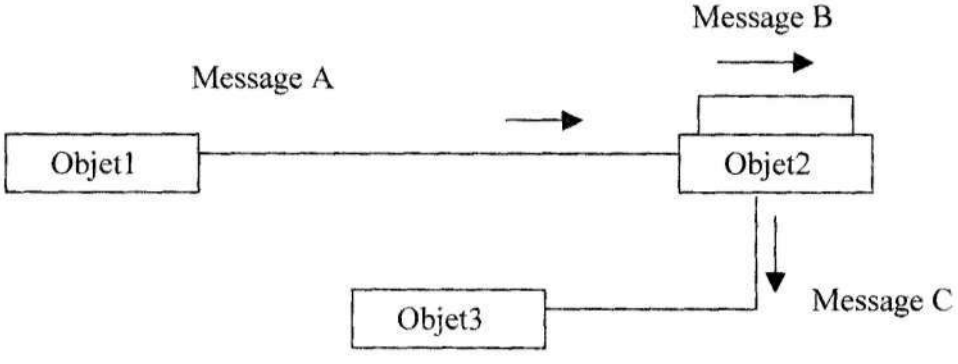


Figure 1 : Exemple de communication entre objets

- **Catégories de messages :** Il existe cinq catégories principales de messages :
  -Les constructeurs qui créent les objets,
  -Les destructeurs qui détruisent les objets,
  -Les sélecteurs qui renvoient tout ou une partie de l'état d'un objet,
  -Les modificateurs qui changent tout ou une partie de l'état d'un objet,
  -Les itérateurs qui visitent l'état d'un objet ou le contenu d'une structure de données qui contient plusieurs objets.

I.2- Classe

I.2.1- Définition

C'est une abstraction d'un ensemble d'objets qui possèdent une structure identique (liste des attributs) et même comportement (liste des opérations). Un objet est une instance d'une et une seule classe [MUL 98]. Une classe peut être vue ainsi :

Classe = attributs + méthodes + instanciations.

- **Attribut :** Une structure de données (identificateur et domaine de valeur) caractérisant un aspect (ou une caractéristique) de l'objet.

- **Méthode :** une opération (fonction ou procédure) représentant les services offerts par l'objet et qui se traduisent par des applications sur les même objets. Chaque méthode est définie par un nom, appelé sélecteur de la méthode, une liste de paramètres (arguments de la méthode) et un corps de la fonction (ou de la procédure).

- **Instanciation :** Une classe est l'entité conceptuelle qui décrit l'objet. Sa définition sert de modèle pour construire ses représentants physiques appelés instances. Une instance est ainsi, un objet particulier crée pour sa classe, qui joue le rôle de moule permettant de produire plusieurs exemplaires.

Exemple de classe :

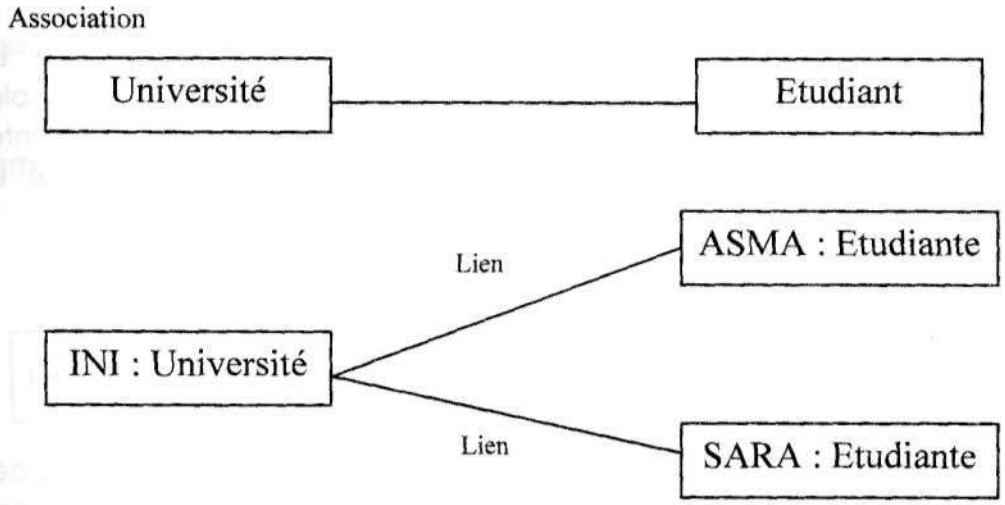
Nom de la classe :	Employé
Attributs :	N°Employé
	Nom
	Prénom
	Age
	Grade
	Agence
Méthodes :	Nom en majuscule
	Changer âge
	Changer grade
	Changer agence.

I.2.2- Les relations entre les classes

Les liens particuliers qui relient les objets peuvent être vus de manière abstraite dans le monde des classes : à chaque famille de liens entre objets correspond une relation entre les classes de ces même objets. De même que les objets sont instances de classes, les liens entre objets sont instances de relations entre classes [MUL 98].

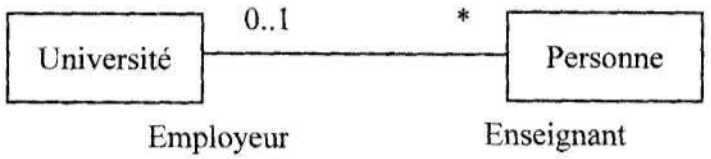
- **Association :** Une association est une abstraction des liens qui existent entre les objets instances des classes associées.

Exemple :



L'association peut être décorée par une forme verbale active ou passive comme il est possible de préciser le rôle d'une classe au sein d'une association, le rôle est écrit du coté de la classe qui le joue.

Exemple :



La classe Université joue le rôle de l'employeur pour les enseignants de la classe personne, et la classe personne joue le rôle d'enseignant pour la classe université.

Les rôles portent également une information de multiplicité qui précise le nombre d'instances qui participent à la relation.

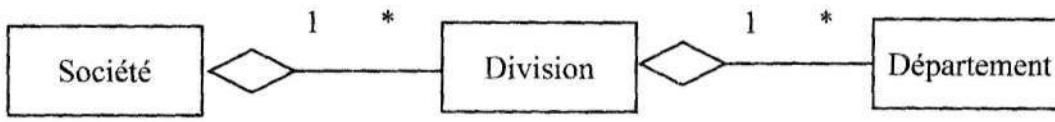
Le tableau suivant résume les valeurs de multiplicité les plus courantes.

1	Un et un seul
0.1	Zéro ou un
M..N	De M à N (entiers naturels
*	De zéro à plusieurs
0..*	De zéro à plusieurs
1..*	De un à plusieurs

- **L'agrégation**

L'agrégation est une forme particulière d'association qui exprime un couplage plus fort entre les classes. L'une des classes joue un rôle plus important que l'autre dans la relation. L'agrégation permet de représenter des relations de type maître et esclave, tout et partie ou composé et composant.

Exemple :



La société représente l'agrégat pour la classe division qui représente l'agrégé pour société et l'agrégat pour département. Lorsque la multiplicité de l'agrégat (comme pour notre exemple) vaut 1, la destruction de l'agrégat entraîne la destruction des composants (agrégés).

I.2.3- Les hiérarchies de classe

Les hiérarchies de classe ou classification permettent de gérer la complexité en ordonnant les objets au sein d'arborescence de classes d'abstraction croissante.

- **La généralisation**

C'est un point de vue porté sur les hiérarchies de classe. Elle consiste à factoriser les éléments communs (attributs, opérations et contraintes) d'un ensemble de classes dans une classe plus générale appelée super-classe [MUL 98].

- **La spécialisation**

C'est un autre point de vue porté sur les hiérarchies de classe, elle permet de capturer les particularités d'un ensemble d'objets non discriminés par les classes identifiées. Les nouvelles caractéristiques sont représentées par une nouvelle classe, sous-classe des classes existantes.

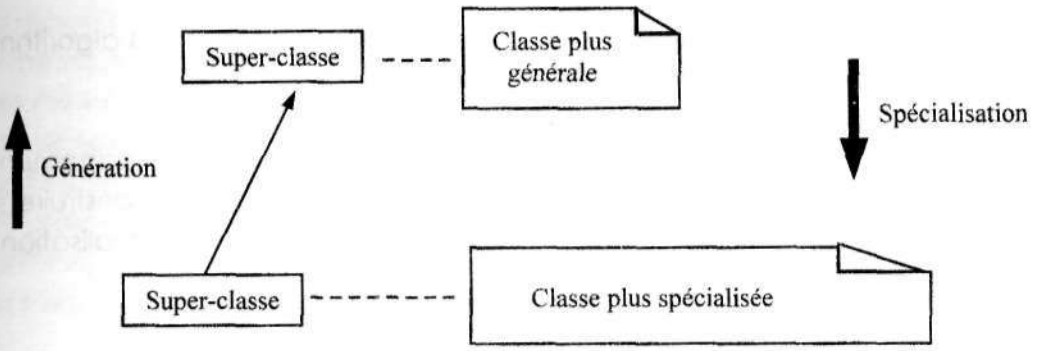


Figure 2 : La généralisation et la spécialisation offrent deux points de vue antagoniste sur une hiérarchie de classe

Exemple de généralisation :

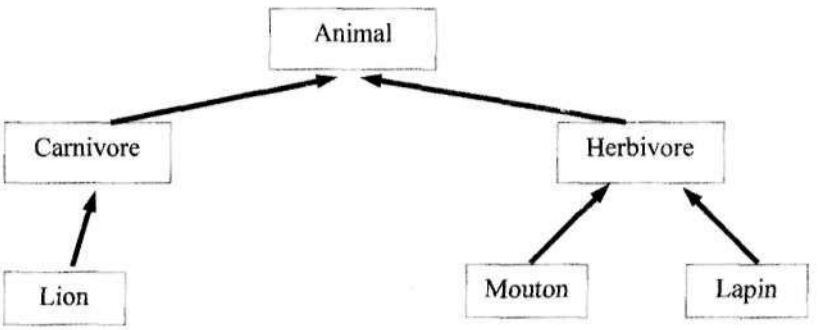


Figure 3 : La généralisation ne porte ni nom particulier ni valeur de multiplicité

I.2.4- L'héritage

C'est une technique offerte par les langages de programmation pour construire une classe à partir d'une ou plusieurs autres classes, en partageant les attributs, des opérations et parfois des contraintes, au sein d'une hiérarchie de classe. L'héritage est donc le mécanisme de transmission des propriétés d'une classe vers une sous-classe.

Il existe deux types d'héritage :

- **Héritage simple :** une sous-classe hérite des propriétés d'une classe (super-classe) au maximum.
- **Héritage multiple :** une sous-classe hérite des propriétés de plus d'une super-classe, ce type d'héritage n'est pas supporté par tous les langages de programmation. Il pose un problème au niveau des propriétés de classes qui peuvent avoir le même nom et des fonctions différentes.

I.3- Autres caractéristiques de l'approche objet

- **Encapsulation :** principe consistant à cacher les données et les algorithmes des objets en ne laissant visible que les interfaces.

- **Réutilisabilité** : étant donné qu'un objet est défini par son comportement, il est facile de l'inclure dans une bibliothèque et l'utiliser pour construire soit des objets de même type, soit des objets spécifiques par spécialisation et composition des objets existants.

- **Modularité** : principe consistant à regrouper plusieurs classes en un ensemble cohérent de classes.

- **Polymorphisme :** concept selon lequel un nom peut référencer des objets instances de plusieurs classes regroupées dans une hiérarchie de généralisation.

- **Surcharge :** redéfinition d'une méthode héritée avec un code différent.

II- METHODES D'ANALYSE ET DE CONCEPTION OBJET

Le première partie de cet article décrit les concepts et notations impliqués dans la modélisation orientée objet. Les méthodes permettent de construire des modèles à partir de ces éléments de modélisation qui constituent des concepts fondamentaux pour la représentation de systèmes ou phénomènes. Dans cette partie, nous traiterons, en particulier, les méthodes d'analyse et de conception orientées objets tout en passant par les méthodes traditionnelles et leurs limites.

II.1- Définition d'une méthode

Une méthode définit une démarche reproductible pour obtenir des résultats fiables. Tous les domaines de la connaissance utilisent des méthodes plus ou moins sophistiquées et plus ou moins formalisées. Une méthode informatique est une démarche organisationnelle et conceptuelle permettant la mise en œuvre d'une solution informatique. De manière générale, les méthodes permettent de construire des modèles à partir d'éléments de modélisation et constituent des concepts fondamentaux pour la représentation de systèmes ou de phénomènes.

Les méthodes définissent également une représentation –souvent graphique- qui permet d'une part de manipuler les modèles, et d'autre part de communiquer entre les différents intervenants.

En plus des éléments de modélisation et de leur représentation graphique, une méthode définit les règles de mise en œuvre qui décrivent l'articulation

des différents points de vue, l'enchaînement, l'ordonnancement des tâches et la répartition des responsabilités.
Une méthode de développement comprend :

- Des éléments de modélisation qui sont les briques conceptuelles de base,
- Une notation dont l'objectif est d'assurer le rendu visuel des éléments de modélisation,
- Un processus qui décrit les étapes à suivre lors du développement du système,
- Du savoir-faire, plus ou moins formalisé.

II.2- Des méthodes fonctionnelles aux méthodes objet

Les méthodes structurées et fonctionnelles se sont imposées les premières car elles s'inspirent directement de l'architecture de l'ordinateur, domaine éprouvé et bien connu des informaticiens.
La séparation entre les données et le code qui existe dans le matériel, a été transposée vers les méthodes ; c'est ainsi que les informaticiens ont pris l'habitude de raisonner en terme de fonctions du système. Cette démarche est devenue anachronique à cause de son manque d'abstraction.
Bien que les racines de l'objet soient solidement ancrées dans les années 60, ce n'est que vers les années 80 que les méthodes objet ont commencé à émerger.
Le cheminement des méthodes fonctionnelles et des méthodes objet est similaire.

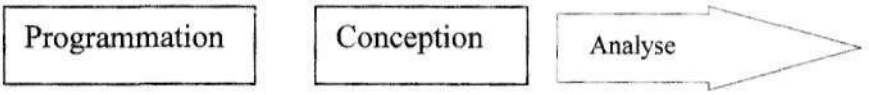


Figure4 : *Evolution des méthodes: de la programmation vers l'analyse*

II.3- Motivation pour les méthodes d'analyse et de conception orientées objet

Deux types de motivation orientent actuellement les concepteurs dans le choix des méthodes de conception : la possibilité de prendre en compte des applications de plus en plus complexes, et le souci de diminuer les coûts de développement et de maintenance.

Les applications de gestion par exemple, couvrent un spectre d'activités plus large que celui d'une simple tache de stockage et de restitution de données structurées. En effet, depuis le milieu des années 80, les applications de gestion ont évolué vers des traitements plus élaborés, tels que l'élaboration des tableaux de bord pour le contrôle de gestion, le raisonnement sur des

règles et des heuristiques pour l'aide à la décision, etc. les frontières entre les domaines aussi variés que la gestion, la Bureautique, les systèmes d'aide à la décision et les systèmes experts se sont estompées pour laisser place à des activités groupées dans un objectif global de représentation et de gestion des SI de l'entreprise. On peut rencontrer aujourd'hui un système de facturation qui édite en même temps que les factures, les lettres d'accompagnement de ces factures ou des relances clients. De même, de nombreuses applications de gestion de stock font appel à des supports multimédias pour stocker et gérer la documentation technique (schémas électriques, graphiques, photos, manuels d'utilisation) du parc de matériel concerné [BOU 97a].
L'approche objet répond le mieux à ces nouveaux besoins.

II-4 Limites des approches traditionnelles

Le bon sens réclame la meilleure adéquation possible entre le modèle informatique et la réalité. Par modèle informatique, on pense aux programmes mais aussi aux descriptions produites pendant l'analyse-conception.
Les méthodes traditionnelles imposent leur ordre propre, artificiel. Elles réduisent la modélisation à une projection selon le principe de la séparation donnée – traitement.
Nous pouvons admettre que cette séparation donnée – traitement était fondé en ce qui concerne les phases basses du développement (analyse organique ; programmation et test) puisqu'elle traduit justement un état de la technologie, celui des langages de deuxième et troisième génération.
Mais ce principe a indûment contaminé les phases hautes qui se doivent pourtant de rester au plus prés du discours de l'utilisateur.
Ainsi, si on met l'accent sur les données, on préserve une bonne part d'expression disponible dans le discours de l'utilisateur, mais on s'oblige à traiter des opérations de façon globale et détachée des données. Si on met l'accent sur le traitement, on obtient une solution fonctionnelle claire, mais c'est au prix de l'abstraction des données.
On n'évite pas la faille sémantique entre le discours initial (représentation implicite de l'utilisateur) et le modèle final (le modèle informatique). Cette distorsion est un vide natif : elle exige une transformation mentale permanente. Elle est une source de difficultés non seulement lors des développements, mais aussi au-delà, au cours de la maintenance et des évolutions du système.

II.5- Evaluation des méthodes objet

L'approche orientée objet est devenue incontournable pour le développement de systèmes, qu'ils soient industriels ou de gestion, de simulation ou de contrôle de processus. Devant cet engouement pour les techniques orientées objet, il est naturel de s'attendre à une augmentation du nombre de méthodes.

Pour pouvoir choisir une méthode plutôt qu'une autre, nous allons tenter de comparer cinq méthodes objet parmi les plus diffusées à l'heure actuelle : OOA, OOD, OOA/OOD, OMT et OOSE.

II.6- Quelques méthodes d'analyse et de conception orientées objet

La communauté objet et plus particulièrement celle des utilisateurs des méthodes d'analyse et de conception orientée objet a été confronté à l'évolution rapide de l'offre. En 1989, il y avait moins de 10 méthodes, en 1994, plus d'une cinquantaine. Entre ces méthodes, il n'y avait pas d'incompatibilité réelle et leur force et faiblesse étaient complémentaires [PER 97].

- **OOD (Object Oriented Design)** : elle est basée sur le modèle objet et fait parti des méthodes dédiées à ADA [COA 91a] [COA 91b].
- **HOOD (Hierarchical Object Oriented Design)** : c'est une méthode de conception hiérarchique des systèmes scientifiques et techniques [DEL 93].
- **OOA (Object Oriented Analysis)** : une méthode d'analyse pure [COA 91a] [COA 91b].
- **OOM (Object Oriented Merise)** : extension de la méthode Merise vers l'approche objet [BOU 94].
- **OOSE (Object Oriented Software Engineering)** : accorde une place importante aux objets d'interface [JAC 94].
- **OMT (Object Modeling Technique)** : Méthode qui couvre tout le cycle de vie d'un logiciel, elle comprend les phases d'analyse, de conception (système et objet) et d'implémentation [RUM 96] [MOH 99].

II.7- Les critères de comparaison

Cette comparaison a été développée autour de plusieurs axes, représentant les préoccupations des différents niveaux et acteurs d'une organisation.

La liste des critères est la suivante :

- Domaine d'application ;
- Cycle de développement ;
- Démarche adoptée ;
- Application des concepts objet ;
- Exploitabilité ;
- Adéquation au développement à grande échelle.

II.8- Comparaison par critère

- **Domaine d'application**

L'adéquation des méthodes aux divers domaines d'application est résumé dans le tableau suivant [JAC 94], [BOU 97a].

Domaine	OOA	OOA/OOD	OOD	OMT	OOSE
Gestion	•	••	•	•	•
Scientifique	•	•	•	•	•
Contrôle de processus	•	•	••	•	•
Contrôle de processus temps réel	••	•	••	•	•
Intelligence artificielle					

Adéquation : •bonne, ••aptitude particulière

Tableau I : **les méthodes et les domaines d'application**

Nous pouvons préciser que :

- OOA est une méthode fortement influencée par les applications temps réel [BOU 97a] ; mais applicable à tout type d'application ;
- OOA/OOD à pour origine les méthodes " d'organisation humaine " [BOU 97a] et donc prévue au départ pour le développement de systèmes intégrés de gestion.

- **Cycle de développement : (la couverture du cycle de vie)**

Le positionnement des méthodes par rapport au cycle de développement est résumé dans le tableau suivant [BOU 97a]. Celui-ci synthétise l'avis de la majorité des auteurs consultés

Etape du cycle	OOA	OOA/OOD	OOD	OMT	OOSE
Planification	O				O
Analyse des besoins	O	O			•
Spécification formelle	•	•	O	•	•
Spécification technique	•	•	•	•	•
Implémentation	O	O	•	•	•
Integration et test		O	O		•
Validation du système					
Maintenance et évolution					•

Couverture : · existence d'un formalisme et de technique, O portée

Tableau 2 : les méthodes et le cycle de vie de référence

- **Le processus de développement propre**

Mis à part les méthodes d'analyse OOA et OOA/OOD, les méthodes basent leur processus sur un cycle de vie propre. Nous pouvons faire le parallèle avec les étapes du cycle de développement de référence :

OOD se base sur un processus de développement incrémental et itératif, organisé autour de prototype opérationnel (phase d'évolution). Dans le cas d'une modification, une deuxième itération est enclenchée et le produit est considéré comme le prototype de la première itération.

Etapes du cycle	OOD	OMT	OOSE
Planification			
Analyse des besoins			Analyse des besoins et de robustesse
Spécification formelle	Analyse	Analyse	
Spécification technique	Conception	Conception système et objet	Construction
Implémentation	Evolution (prototypage)	Implémentation	
Integration et tests			Test
Validation du système			
Maintenance et évolution	Modification		

Tableau 3 : *Comparaison du processus de développement*

- OMT : semble s'orienter vers la conception totale avant l'implémentation [BOU 97a]. Pourtant, J. Rumbaugh dans [RUM 96] affirme que la méthode est tout autant applicable à une démarche incrémentale, basée sur le prototypage. Contrairement à OOD, les itérations sont seulement permises vers l'étape antérieure [BOU 97a].
- OOSE : préconise également un processus itératif. La méthode est conçue pour construire un système extensible qui peut être développé en plusieurs versions.

A propos de transition...

- OOA/OOD : facilite le passage de la phase d'analyse à la phase de conception par l'adoption d'une démarche et de notations communes.
- OOD : permet une transition douce entre l'analyse et la conception. Grâce à une méthode commune et à un dictionnaire objet unique. Par contre, la transition entre le niveau logique et le niveau physique n'est pas

clairement formalisée en fonction de l'environnement cible [BOU 97a].

- OMT : favorise la transition entre l'analyse et la conception par l'utilisation des mêmes modèles, réorganisés et enrichis de détails lies à l'implémentation.
- OOSE : la transition entre la phase d'analyse et la phase de conception puis l'implémentation se fait aisément grâce à l'utilisation commune du modèle objet [BOU 97a], [JAC 94].

En ce qui concerne l'organisation du projet...

- Les méthodes OOD, OMT et OOSE peuvent intégrer une ou plusieurs étapes de prototype. Ce principe de développement incrémental rend particulièrement possible des tâches. Cependant, un partitionnement adéquat du domaine en sous-systèmes est un autre moyen de parallèlisation, accessible à toute méthode.
- Il est entendu que les méthodologies objets entraînent un report conséquent des charges du codage vers les phases d'analyse et de conception.
- De manière générale, ces méthodes objets n'apportent aucune aide directe pour la phase de planification, si ce n'est OOSE. Le livre de I. Jacobson [JAC 94] consacre tout un chapitre au génie logiciel et donne des éléments intéressants sur la conduite de projet et les métriques du logiciel. En outre, le livre de M. Bouzeghoub [BOO 94] donne quelques indications sur la gestion de projet.

- **Démarche adoptée**

Toute démarche méthodologique est constituée d'étapes basées sur le cycle de développement, permettant d'élaborer un ou plusieurs modèles.

- **Les étapes**

Le tableau suivant décrit les étapes principales de la démarche adoptée par chacune des méthodes, fortement liées au processus de développement de chacune d'elles :

OOA	• Modèle d'information (identification des objets et des relations) • Modèle d'états (cycle de vie des objets) • Modèle de processus (interactions entre objets)
OOA/OOD	• Trouver les classes et les objets • Identifier les structures (relations d'agrégation et de classification) • Identifier les sujets • Définir les attributs • Définir les services
OOD	Processus récursif : • Identifier les classes et les objets à un niveau d'abstraction donné • Identifier la sémantique des classes et des objets (comportement) • Identifier les relations entre objets • Compléter ces classes et ces objets (et construire un prototype) • Vérifier la cohérence du système • Raffiner les classes, les objets, leur sémantique et leur structure
OMT	Affinage successif tout au long du cycle de vie : • Modèle objet (objet, classe et relations) • Modèle dynamique (états et transitions des objets) • Modèle fonctionnel (procédés de transformation par les opérations)
OOSE	• Modèle des besoins (expression par cas d'utilisation) • Modèle d'analyse (objets et relations) • Modèle de conception (blocs d'objets et stimuli) • Modèle d'implémentation • Modèle de test

Tableau 4 : les méthodes et leur démarche

- **Les modèles**

Le tableau suivant [BOU 97a] précise les modèles conceptuels utilisés dans chacune des trois dimensions statique, dynamique et fonctionnelle :

Dimensions	OOA	OOA/OOD	OOD	OMT	OOSE
Statique (objet)	Relationnel + héritage	E-A binaire + agreg.,gen.	Modèle spécifique	E-A n-aire + agreg., gen.	Modèle spécifique
Dynamique (événements/états)	Diagramme d'état		Diagramme d'état, de temps	Diag. Etat événements transitions	Objets de contrôle
Fonctionnelle (flux/processus)	Diagramme de flux de données			Diagramme de flux de données	Objets interface

Tableau 5 : les méthodes et les modèles conceptuels selon les trois dimensions

- OOA et OMT basent leur démarche sur les trois modèles, correspondant aux trois dimensions.

- OOA met l'accent sur le cycle de vie des objets. Les trois modèles interagissent en mettant en commun soit des objets, soit des événements, soit des opérations.

- OMT définit les diagrammes d'état au niveau des classes. Le lien entre les modèles se situe au niveau de l'opération. Ils sont enrichis au fur et à mesure des phases d'analyse et de conception. Leur cohérence est assurée par amélioration du modèle global. La démarche comprend l'élaboration d'un dictionnaire d'objets.

- OOA/OOD est basée sur un modèle de représentation unique qui décrit les objets et leur comportement. Cependant, l'aspect dynamique peut être élaboré par un diagramme d'états [BOU 97a]. La méthode utilise la même démarche aux niveaux conceptuel et logique, pourtant elle semble moins bien formalisée pour la conception.

- OOD se focalise sur le modèle statique et aborde seulement le modèle dynamique. G. Booch préconise l'utilisation de SADT (analyse structurée) pour la modélisation fonctionnelle. Toutefois, les diagrammes de temps donnent une idée de l'ordonnancement des opérations des divers objets. Les étapes définies dans le premier tableau amènent progressivement du niveau logique (voire conceptuel) au niveau physique. Les activités de l'étape d'évolution ne sont cependant pas bien formalisées [BOU 97a].

- OOSE commence par décrire le système par différentes vues, appelées cas d'utilisation (ou scénarios fonctionnels). Chaque vue est ensuite décrite sous trois facettes particulières, correspondant aux trois dimensions ci-dessus, et pour lesquelles il propose trois types d'objets : information (objets entités), comportement (objets de contrôle) et présentation (objets d'interface). des diagrammes d'états sont utilisés au niveau logique pour décrire l'évolution d'un objet. L'élaboration des modèles conduit de l'analyse des besoins à la vérification interne du système. La cohérence de tous ces modèles est assurée par le concept de cas d'utilisation. Les règles de construction du niveau physique ne sont pas claires [BOU 97a].

Dans toutes ces méthodes, le processus de modélisation est itératif, ce qui permet une consolidation des modèles.

- **Application des concepts objets**

Le tableau 6 tente de mettre en correspondance les principaux concepts objets utilisés par les méthodes [JAC 94] :

- OOA n'inclut pas vraiment les concepts de l'objet : pas de notion de classe, d'objet complexe, d'identité, d'opération, de message, support d'encapsulation faible et non respecté lors de l'accès aux objets (méthodes non encapsulées).

- OOA/OOD propose la notion de classe et objet. Chaque classe doit avoir des services pour ajouter, modifier, détruire les instances (constructeurs) et d'autres spécifiques au comportement (accesseurs et transformateurs). OOA/OOD intègre la notion de filtrage d'héritage.

- OOD présente des lacunes au niveau de la composition d'objets : il est parfois difficile de choisir entre une relation "utilise" et une relation "contient".

- OMT propose les concepts de classe abstraite, de classe et association dérivées. Elle est la seule à permettre la représentation des associations n-aires. Par contre, il n'existe pas de notion d'identité au sens objet.

OOA	OOA/OOD	OOD	OMT	OOSE
Objet	Classe	Classe	Classe	Objet entité
Instance spécifiée	Objet	Objet	Objet	Instance
		Meta classe	Meta classe	
Attribut atomique	Attribut	Champ	Attribut	Attribut
	Service	Opération	Opération	Opération
Association binaire	Structure tout-partie	Relation contient	Agrégation	Constitué de
Référence / objet associatif	Connexion d'instances	Relation utilise	Association / lien	Accointance (association)
Généralisation / spécialisation	Généralisation / spécialisation	Relation hérite de	Généralisation / spécialisation	Héritage
Flot de données	Connexion message	Relation utilise / instancie	Flot de données	Communication
Evénement	Message	Message	Evénement	Stimulus

Tableau 6 : Les méthodes et les concepts objets.

- OOSE n'utilise pas clairement la notion de classe : " seul le contexte permet de déterminer s'il s'agit d'une classe ou d'un objet " [JAC 94]. Par contre, trois types d'objets sont proposés : les objets entités, les objets de contrôle et les objets d'interface. l'encapsulation est réalisée au niveau logique du bloc (ensemble d'objets) et non de l'objet.
- Dans OOSE et OOD, les structures permettant de décrire les objets ne sont pas spécifiées

A propos des attributs...

- Si aucune indication n'est donnée quant à la nature et la complexité des attributs dans les méthodes OOA/OOD, OOD et OMT, OOA spécifie clairement qu'ils sont atomiques et ne peuvent donc être ni multivalués, ni composés, au contraire d'OOSE qui les caractérise par un type simple ou structuré.

A propos du polymorphisme...

- OOD, OMT et OOSE prennent en compte le polymorphisme.

· **Exploitabilité :**

- OOA peut être suivie d'une programmation orientée objet ou classique. Elle est plus adaptée à des systèmes temps réel ou interactifs qu'à des applications batch [BOU 97a].
- OOA/OOD : la conception logique intègre les interfaces homme machine, la gestion des tâches et la gestion des données. Cette dernière peut mettre en œuvre des bases de données objets ou relationnelles, des fichiers... .

- OOD : l'implémentation physique est proposée principalement dans les environnements ADA et C++, mais est également possible avec Objet Pascal, SmallTalk ou CLOS. Les systèmes de base de données ne sont pas pris en considération.

- OMT : la phase de conception système permet de définir l'architecture finale (systèmes temps réel, fenêtrage, base de données, simulation dynamique...). Nombreuses solutions pour des environnements objet, relationnel ou fichier sont présentées dans [RUM 96].

- OOSE : décrit des implémentations possibles pour des systèmes temps réel, des bases de données objets et relationnelles [JAC 94]. La méthode s'applique aussi bien à des traitements par lot (batch) , qu'à des traitements transactionnels ou temps réel [BOU 97a].

Toutes ces méthodes garantissent l'indépendance vis-à-vis du langage cible [BOO 94].

· **Adéquation au développement à grande échelle**

- OOA semble difficilement applicable à de gros systèmes si l'on considère les diagrammes d'états réalisés au niveau des objets, solution viable si leur nombre est restreint [BOU 97a].

- OOA/OOD est plutôt conçue pour de petits systèmes, même si aujourd'hui on l'utilise quand même pour de grands systèmes.

- OOD s'attaque au problème de complexité lié au développement à grande échelle, notamment par sa démarche incrémentale, à différents niveaux d'abstraction. Toutefois, les diagrammes (de classes, d'objets, d'états et de temps) ne sont pas adaptés à de grands systèmes faisant intervenir une multitude d'objets [GRA 94] [BOU 97a].

- OMT n'offre pas un édifice satisfaisant pour la maîtrise des projets importants selon [BOO 96], malgré le choix de réaliser les diagrammes d'états et les diagrammes de flot de données (DFD) au niveau des classes et non des objets [BOU 97a]. Toutefois, elle a prouvé son efficacité sur des projets de tailles diverses dans le domaine des télécommunications notamment.

- OOSE propose une démarche pour l'analyse et la conception des systèmes de taille importante. D'ailleurs, "la méthode a été conçue à l'origine pour de grands projets et a été édulcorée afin de l'appliquer aux petits " [JAC 94].

Toutes ces méthodes fournissent un concept permanent de partitionner le système afin de gérer sa complexité et d'organiser le travail d'équipe. Ce procédé est généralement plus efficace lorsqu'il est soutenu par des outils aux fonctionnalités appropriées (gestion de configuration et de versions, dictionnaire objet unique...).

II.9- Synthèse sur les méthodes

Les méthodes étudiées se répartissent en deux catégories [BOU 97a] :

1. Les méthodes plutôt orientées développement (spécification technique et implémentation) : OOD et OMT.
2. Les méthodes plutôt orientées conception (analyse des besoins et spécification formelle) : OOA, OOA/OOD.

- OOSE quant à elle offre une démarche plutôt globale (de l'analyse des besoins à l'implémentation).
- La méthode OOA est d'avantage une méthode relationnelle qu'objet [BOU 97a], ce qui lui vaut un certain succès. Elle ne fait qu'adapter au monde de l'objet des concepts classiques.
- La méthode OOA/OOD est essentiellement une approche orientée objet. Une attention particulière est portée sur la compréhension des domaines

d'application. Elle intègre pleinement les concepts objets et offre une représentation graphique pauvre, mais claire et conviviale. L'aspect dynamique est faiblement développé. Sa démarche d'analyse et de conception basée sur un modèle unique en fait une méthode cohérente et facile d'accès. Elle semble plus adaptée aux petits systèmes.

- La méthode OOD est l'une des plus anciennes méthodes objets. C'est une référence dans le domaine malgré des lacunes dans la structuration des objets [BOU 97a]. La représentation graphique n'est pas des plus conviviales. C'est une méthode ouverte qui peut être précédée d'une analyse structurée (SADT) ou objet (OOA de COAD & YOURDON).
- La méthode OMT apparaît comme la méthodologie la plus développée de toutes [JAC 94]. Elle couvre une bonne partie du cycle de développement. Elle est riche en concepts et propose une représentation graphique très conviviale. L'accent est mis sur les associations. Elle possède une démarche adaptée aux applications complexes. C'est une bonne combinaison des approches objets et fonctionnelles.
- La méthode OOSE est l'une des méthodes dont le cycle de développement est le plus complet. Sa démarche de conception par cas d'utilisation est originale mais semble difficile d'accès. Elle intègre mieux l'aspect modulaire et incrémental du système d'information, ainsi que la conception des interfaces homme – machine (IHM). Elle est conçue pour le développement à grande échelle. Sa philosophie proche des types abstraits, la rapproche plus de l'objet que certaines autres méthodes [BOU 97a].

Le point faible de toutes ces méthodes reste l'aide à la planification.

III- PRÉSENTATION D'UML

La plupart des méthodes objets sont mal positionnées par rapport à leur cycle de développement (analyse, conception, développement). Elles ambitionnent de tout couvrir alors que dans l'effet elles ne proposent des techniques que pour une ou deux phases, de là est venue l'idée de standardiser ces méthodes [BOU 97b].

Pour la standardisation, deux approches ont été proposées par les concepteurs de méthodes : une fusion des méthodes ou une intégration par complémentarité des méthodes. Les deux approches n'ont conduit qu'à la proposition de deux méthodes supplémentaires qui s'ajoutent à celles déjà connues. Une nouvelle tentative se restreint à l'uniformatisation d'une notation graphique (UML : Unified Modeling Language) dont l'espoir est d'obtenir un consensus minimum sur les interfaces graphiques laissant à chacun le soins de définir ou d'utiliser le processus méthodologique de son choix [BOU 97b].

III.1- La genèse d'UML

La notation UML (Unified Modeling Language), a été développée en réponse à l'appel à la proposition lancée par l'OMG (Object Management Group), dans le but de définir la notation standard pour la modélisation des applications construites à l'aide d'objets.

La notation UML représente l'état de l'art des langages de modélisation objet. Elle se place comme le successeur naturel des notations des méthodes de OOD (Object Oriented Design), OMT (Object Modeling Technique) et OOSE (Object Oriented Software Engineering) et, de ce fait, UML s'est très rapidement imposée, à la fois auprès des utilisateurs et sur le terrain de la normalisation.

UML est un langage objet graphique, autrement dit un formalisme orienté objet ; ce n'est pas une méthode, il ne propose donc pas une démarche [MUL 98] [DES 97] [CHE 99].

Avant de présenter les différents modèles d'UML, nous tenons à définir les méthodes objet les plus connues qui lui ont donné naissance à savoir OOD, OMT et OOSE en se basant sur l'origine de la méthode, ses particularités, ses modèles et sa démarche.

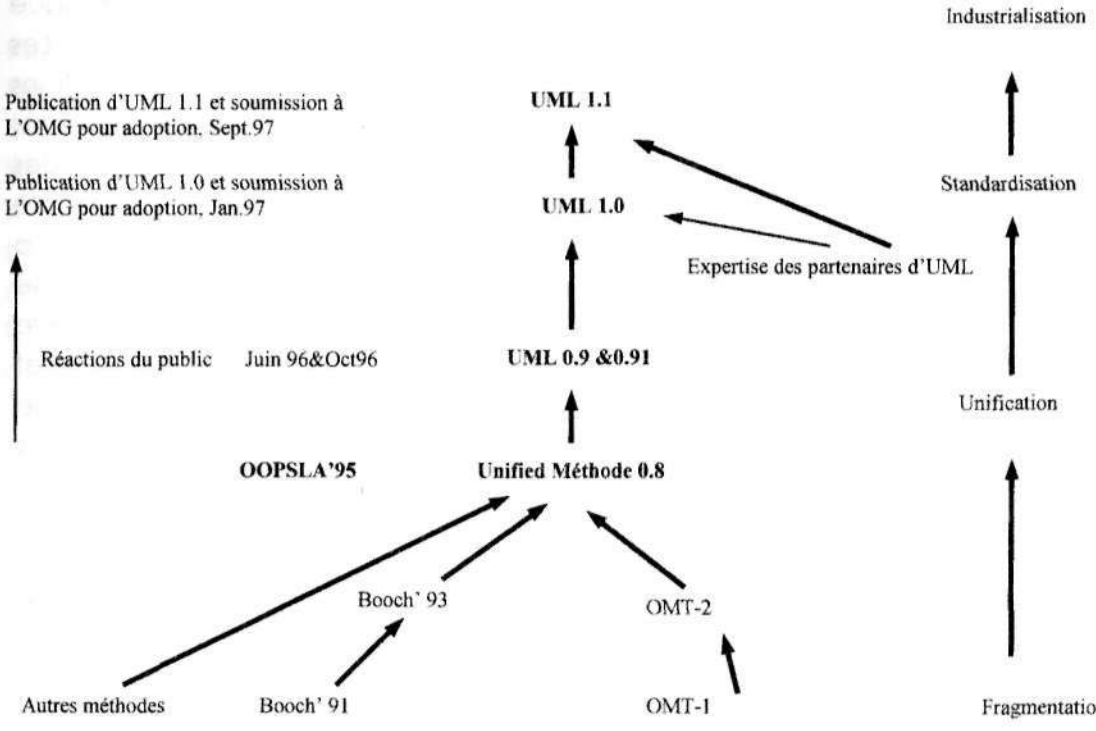


Figure 5 : *Historique et perspectives d'UML*

METHODE OOD (Object Oriented Design) – G. BOOCH

- *Origine* : Créer au début des années 80 afin de nationaliser les développements d'application en ADA.
- *Date diffusion réelle* : 1985.
- *Particularité* : méthode de développement (spécification technique et implémentation).
- *Modèles* : modèle statique (objet, association), modèle dynamique (état-transition).
- *Démarche* : cycle de développement itératif et incrémental. L'analyse n'est couverte que depuis 1993.
- *Formalisme* : objet représenté par des nuages, association représentée par des arcs reliant les nuages.

METHODE OMT (Object Modeling Technique) – J. RAMBAUGH

- *Origine* : créer en 1987/1989 par le centre de recherche et de développement de Général Electric.
- *Date de diffusion réelle* : 1991.
- *Particularité* : suit OOA, mais plus riche en concepts et sa démarche permet de travailler sur des cas complexes.
- *Modèles* : modèle statique, dynamique et fonctionnel. Le modèle statique est dérivé du modèle Entité-Association (concepts et graphique). Les associations peuvent être n-aires et porteuses de données. Les notions d'agrégation et de généralisation sont gérées. Les cardinalités sont utilisées pour préciser les relations ; il est possible de spécifier des contraintes supplémentaires comme la disjonction et l'insertion. Le modèle dynamique est présenté sous la forme de diagramme état/transition. La transition est caractérisée par l'événement déclencheur, OMT propose l'emploi de scénarios qui retracent une succession particulière d'événements entre les acteurs et les classes d'objets (notion de sujet dans OOA/OOD ou de cas d'utilisation dans OOSE). Le modèle fonctionnel utilise les diagrammes des flux qui relient les acteurs et les classes d'objets pour décrire les processus de l'application (fonction). Le processus est une autre formalisation du scénario.
- *Démarche* : le cycle en cascade classique.
- *Formalisme* : le modèle statique est dérivé du modèle Entité-Association. Les modèles état/transition, de scénarios et fonctionnel sont basés sur les diagrammes de flux.

METHODE OOSE (Object Oriented Software Engineering) – I. JACOBSON

- *Origine* : créer en 1980 à partir d'une méthode suédoise.
- *Date de diffusion réelle* : 1992.
- *Particularité* : OOSE accorde une place importante aux objets d'interface qui constitue une première étape de modélisation.
- *Modèles* : au nombre de cinq ; il s'agit du modèle des besoins (délimitation des frontières du système), du modèle d'analyse (niveau conceptuel), du modèle de conception (niveau logique), du modèle d'implémentation (niveau physique), du modèle de test (vérification de la cohérence).
  Les modèles d'analyse prennent en charge des cas d'utilisation ; l'ensemble intégré grâce à des liens inter-vues donne une vue globale du système. Chaque cas d'utilisation peut comporter des entités (aspect statique), des objets d'interface (aspect fonctionnel), des objets de contrôle (aspect dynamique).
- *Démarche* : OOSE comporte trois phases formant un cycle complet ; la phase d'analyse fournit un modèle des besoins et un modèle d'analyse, la phase de construction d'un modèle de spécifications (logique) et un modèle d'implémentation (physique), une phase de test.
- *Formalisme* : cas d'utilisation, trois types d'objet (entité, interface, contrôle).

III.2- Les diagrammes d'UML

Un diagramme donne à l'utilisateur un moyen de visualisation et de manipulation des éléments de modélisation.

UML définit neuf diagrammes pour présenter les différents points de vue de modélisation.

L'ordre de présentation de ces différents diagrammes ne reflète pas un ordre de mise en œuvre dans un projet réel. De plus, UML n'est pas une notation fermée ; en cas de besoin particulier, des précisions peuvent être apportées au moyen de mécanismes d'extension et de commentaires textuels.

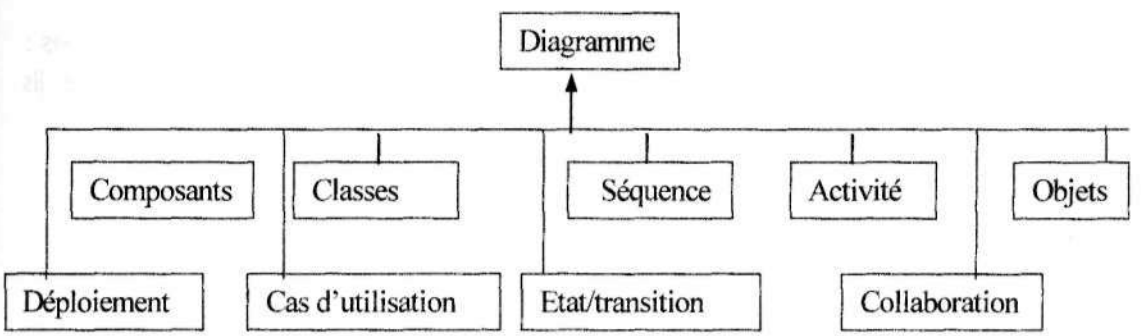


Figure 6 : *Différents types de diagrammes définis par UML*

III.2.1- Les diagrammes de cas d'utilisations (use cases)

Les uses cases (cas d'utilisations) constituent le concept principal de la méthode OOSE de I. JACOBSON, un des pères du langage UML. Dans le processus de standardisation des méthodes, le concept de use cases a été repris dans le but d'effectuer une bonne délimitation du système et également d'améliorer la compréhension de son fonctionnement.
Les use cases représentent le moyen de décrire le caractère fonctionnel des objets. Ce sont des suites d'événements, qui correspondent à une utilisation particulière du système.

III.2.1.a- Place des use cases dans la modélisation

Les use cases représentent le premier modèle du système à concevoir. Ce modèle s'appuie sur le seul document existant : spécifications. Il pourra également s'appuyer sur la connaissance du domaine d'application que l'on peut avoir au préalable. Les use cases permettent de modéliser les attentes des utilisateurs.

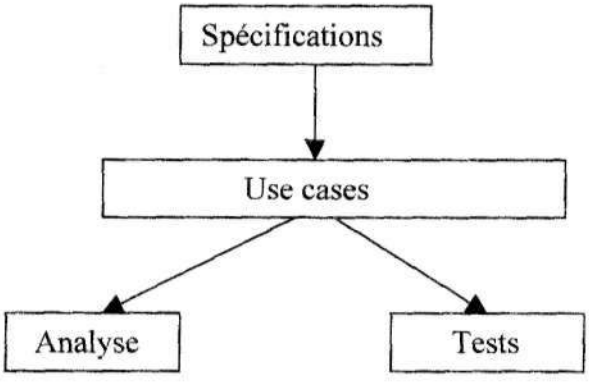


Figure 7 : *La place des use cases dans le processus de développement*

III.2.1.b- Acteurs et use cases pour décrire le système

Il existe deux concepts fondamentaux dans la modélisation par les use cases :

- Les acteurs qui utilisent le système : les utilisateurs extérieurs au système, ils peuvent être de trois types : humains, logiciels ou matériels.
- Les use cases qui représentent l'utilisation du système par les acteurs.

III.2.1.c- Représentation UML des cas d'utilisation :

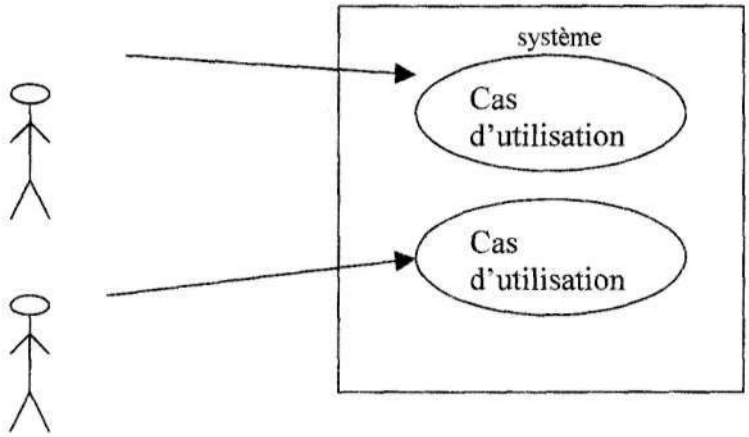


Figure 8 : *Modèle des cas d'utilisation*

III.2.1.d- Les différents types de use cases

Nous pouvons trouver dans les use cases les cas d'utilisations suivants :

- Les principales tâches de chaque acteur.
- Les fonctionnalités mal décrites dans la spécification.
- Les modifications (lecture, écriture) des données du système.
- Les cas d'anomalies.

Exemple :

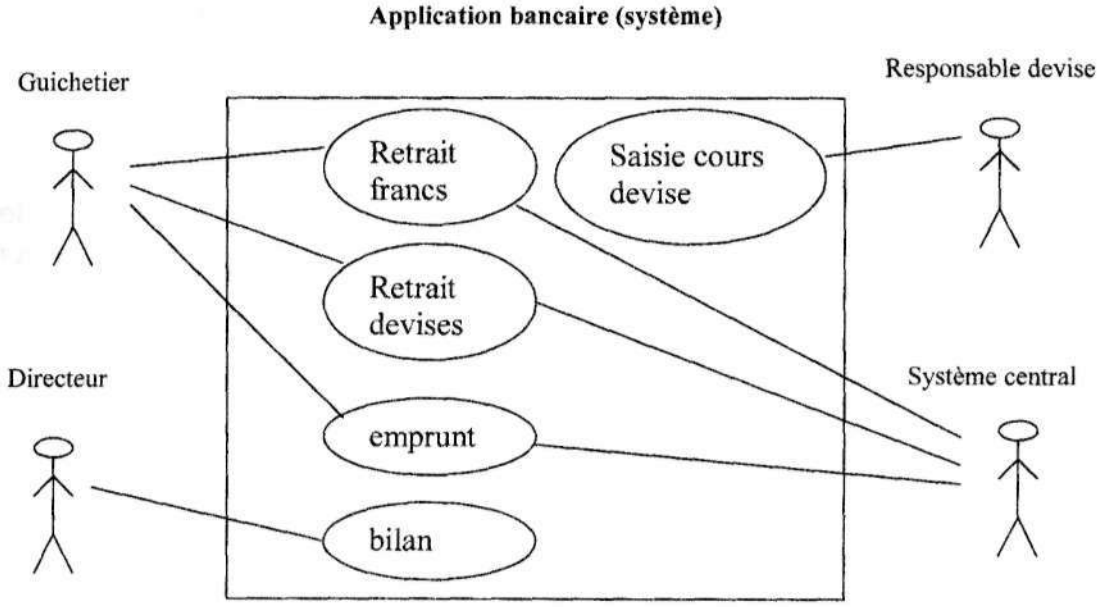


Figure 9 : *Quelques use cases d'une application guichet bancaire*

III.2.2- Le diagramme d'objet

Le modèle objet (appelé également modèle statique) permet de décrire les objets et les relations qui interviennent dans le cadre de la problématique. Cette description est double :

- Une description de la structure des objets et de leurs caractéristiques.
- Une description des associations qui existent entre les différents objets.

➢ **Place du modèle objet dans le processus de développement**

La vue fonctionnelle (use cases, spécifications) ne permet pas de dégager les entités intrinsèquement liées au domaine d'application mais exprime simplement un besoin à un moment donné.

Cette phase d'analyse (modèle objet) a donc pour but de déterminer les objets utiles pour les phases d'analyse (objet puis dynamique) ainsi que pour les phases de conception et d'implémentation.

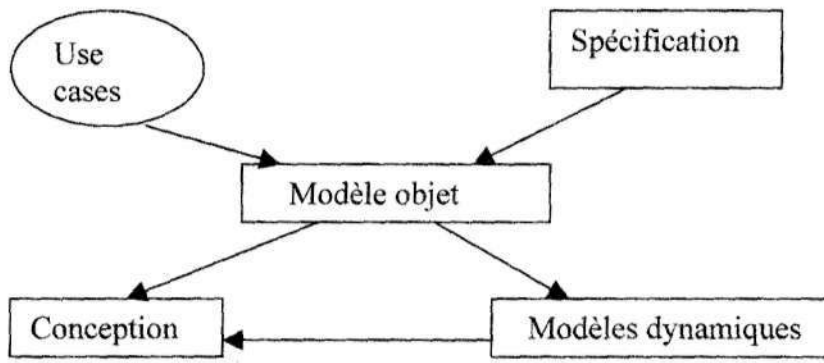


Figure 10 : *Place du modèle objet dans le processus de développement.*

Exemple :

Le diagramme d'objet suivant montre une partie de la structure générale des voitures. Chaque voiture possède un moteur et quatre roues (roue de secours exclue).

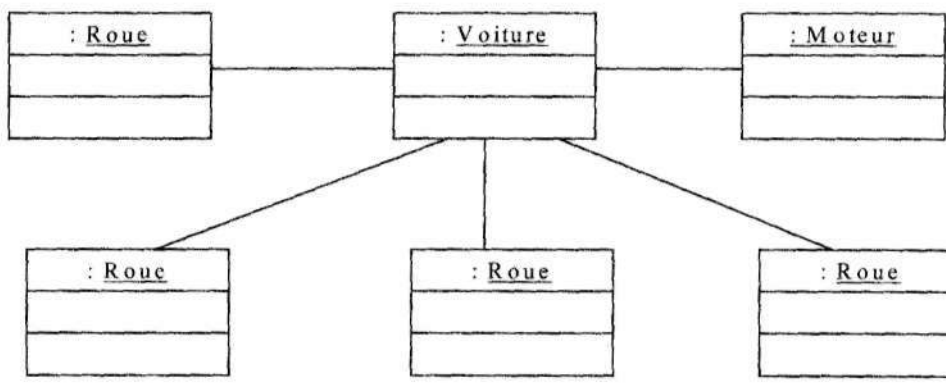


Figure 11 : *Exemple d'un modèle objet*

III.2.3- Le diagramme de classes

Le diagramme de classes exprime de manière générale la structure d'un système, en terme de classes et de relations entre ces classes. De même qu'une classe décrit un ensemble d'objets, une association décrit un ensemble de liens. Les objets sont instances des classes et les liens sont instances de relations [MUL 98].

Un diagramme de classes n'exprime rien de particulier sur les liens d'un objet donné, mais décrit de manière abstraite les liens potentiels d'un objet vers d'autres objets.

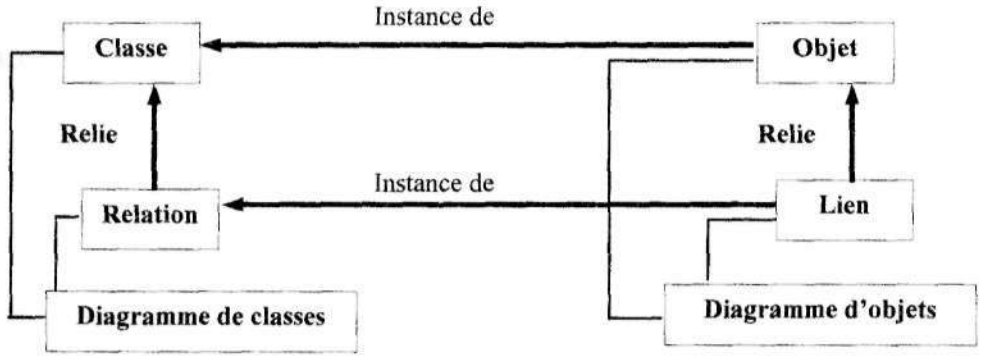


Figure 12 : Méta modèle de modèle de classes

Exemple :

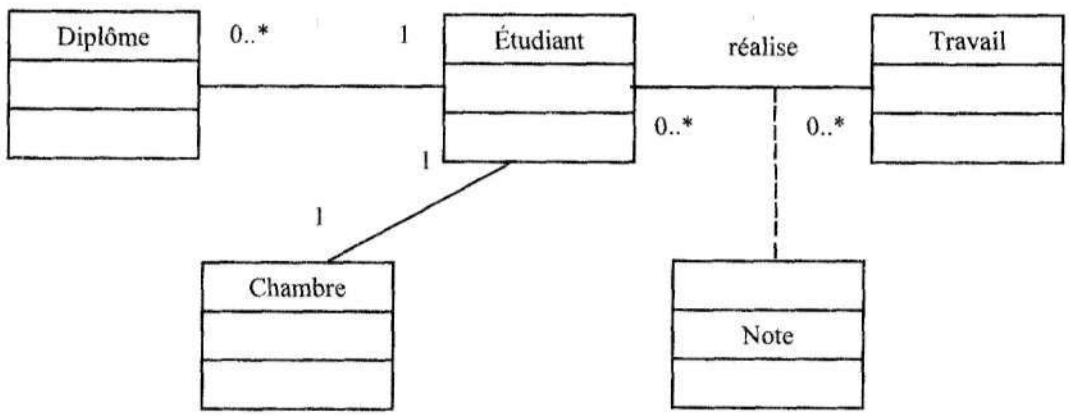


Figure 13: Exemple de modèlede classes

III.2.4- Les diagrammes de collaboration

Un diagramme de collaboration entre objets vise à représenter du point de vue statique et dynamique les objets impliqués dans la mise en place d'une fonction applicative. L'objectif est de construire un modèle expliquant la coopération entre les objets utilisés pour la réalisation d'une fonctionnalité.

Deux notions fondamentales sont utilisées dans la mise en place d'un diagramme de collaboration : le contexte et les interactions.

- Le contexte est une vue statique partielle des objets qui collaborent pour réaliser une fonction.
- Les interactions entre objets sont des séquences de messages échangés par les objets dans le cadre de la réalisation d'une opération.

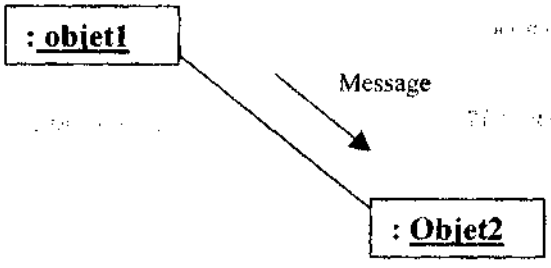


Figure 14 : *Représentation graphique d'échange de messages entre les objets.*

Exemple :

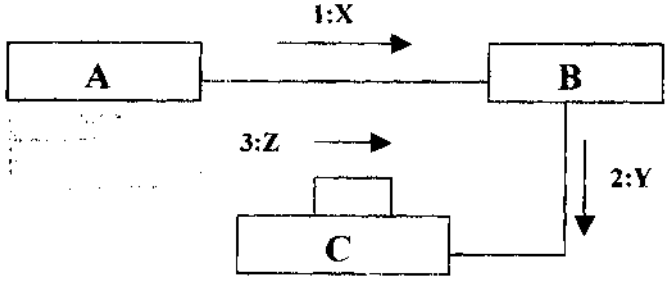


Figure 15 : *Exemple de diagramme de collaboration*

III.2.5- Les diagrammes de séquence

Les diagrammes de séquence montrent des interactions entre objets selon un point de vue temporel. La représentation des interactions est la suivante :

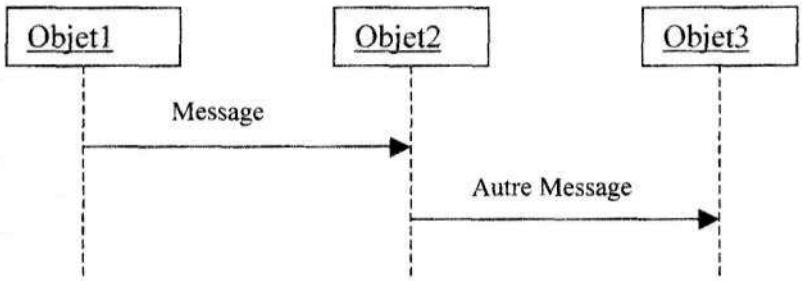


Figure 16 : *Présentation des diagrammes de séquences.*

En modélisation objet, les diagrammes de séquences s'utilisent de deux manières :

1 Dans la documentation des cas d'utilisation, la modélisation se concentre sur la description de l'interaction entre l'utilisateur et le système.
2 Pour la représentation précise des interactions entre les objets du système.

Exemple :

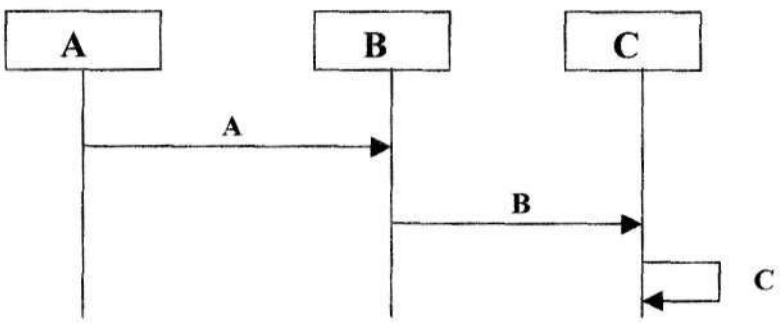


Figure 17 : *Exemple d'un diagramme de séquence*

III.2.6- Les diagrammes d'état et de transition

III.2.6.a- La notion d'état

Chaque objet est, à un moment donné, dans un état particulier. L'état d'un objet est défini à la fois par la valeur de ses variables d'instance et par les valeurs de ses liens avec d'autres objets.

Les états se caractérisent par la notion de durée et de stabilité. Un objet est toujours dans un état donné pour un certain temps, un objet ne peut pas être dans un état inconnu ou non défini.

Exemple : Les états d'un objet personne

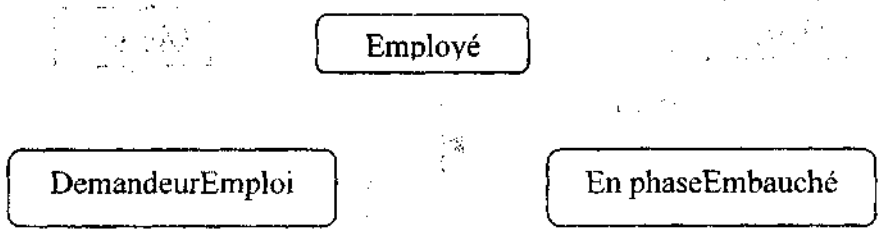


Figure 18 : *Les états d'un objet Personne*

III.2.6.b- La notion de transition

Une transition est le changement d'état d'un objet causé par un événement qui survient dans le domaine du problème.

Un événement peut être émis par un objet du système ou par un objet externe au système, cela correspond à l'appel d'une méthode.

L'événement est effectué par l'intermédiaire des actions et des activités :

- **Action** : correspond à une des opération de la classe de l'objet destinataire de l'événement; les actions correspondent à des opérations dont le temps d'exécution est négligeable.

- **Activité** : l'activité correspond à une opération dont le temps d'exécution est non négligeable, elle est exécutée pendant que l'objet est dans un état donné.
  Contrairement aux actions, les activités peuvent être interrompues à tout moment, dès qu'une transition de sortie de l'état est déclenchée.

Deux types d'éléments permettent de préciser une transition :

- **Gardiens (conditions)**: ce sont des fonctions booléennes. Une transition portant un gardien ne peut être effectuée que si le gardien est vérifié, elle peut éventuellement porter plusieurs gardiens.

- **Attributs** : correspondent à des informations ou des paramètres portés par des événements. Une transition peut porter une liste d'attributs.

Exemple :

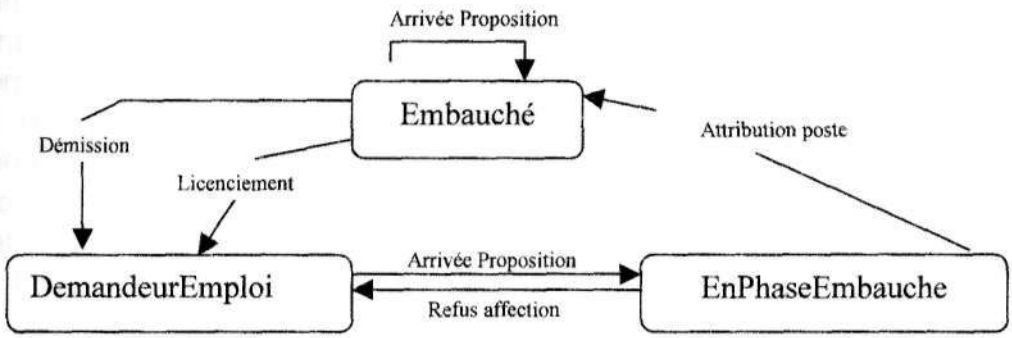


Figure 19 : *Les transitions pour un objet Personne*

Plusieurs sous états peuvent être associés à un état.

Exemple :

PersonneEmbauche

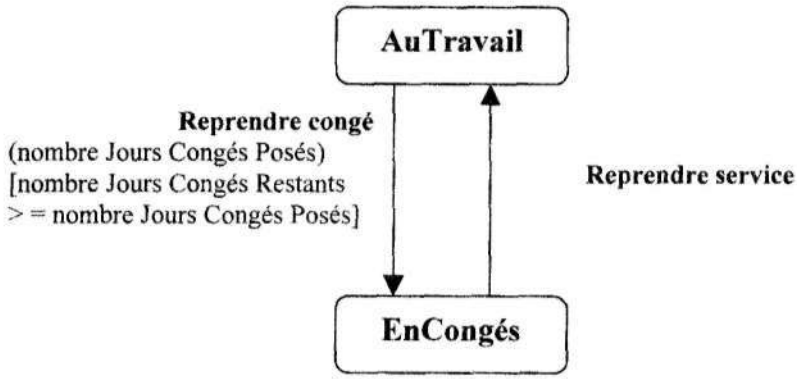


Figure 20 : *Le sous-diagramme de l'état Embauché*

III.2.7- Les diagrammes d'activités

Un diagramme d'activités est une variante des diagrammes d'états-transitions organisés par rapport aux actions et principalement destiné à représenter le comportement interne d'une méthode (la réalisation d'une opération) ou d'un cas d'utilisation [MUL 98].

Le diagramme d'état-transition et le diagramme d'activité représentent tous les deux le comportement des objets, le premier sous l'angle de l'enchaînement des états et le second sous l'angle des activités.

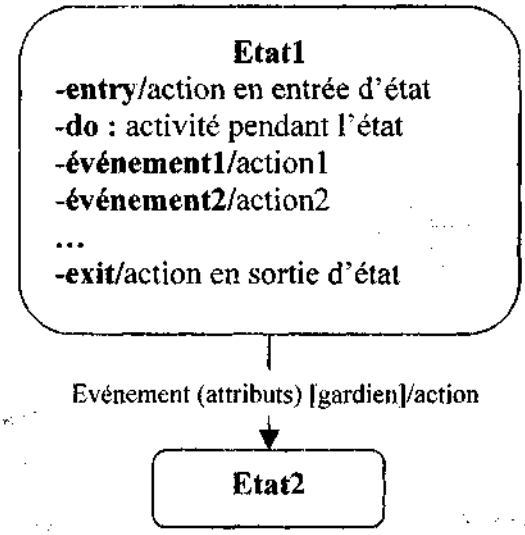


Figure 21: *Représentation des opérations.*

III.2.8- Le diagramme de composants

Les diagrammes de composants décrivent les éléments physiques et leurs relations dans l'environnement de réalisation. Les diagrammes de composants montrent le choix de réalisation.

Les composants sont de types : sous-système, module, programme et sous programme, processus et tache.

- **Les sous-systèmes :** facilitent la réalisation des applications, ils organisent la vue de réalisation d'un système ; chaque sous-système peut contenir des composants et d'autres sous-systèmes, les sous-systèmes peuvent être décomposés en modules.
- **Les programmes principaux :** le nom de programme principal est souvent utilisé par l'éditeur de liens pour nommer le programme exécutable correspondant à l'application. Ceci permet, entre autre, de relier le modèle de composants et le modèle de processus (ou de déploiement).
- **Les sous programmes** : regroupent les procédures et les fonctions qui n'appartiennent pas à des classes.
- **Tâches** : correspondent à des composants qui possèdent leur propre flot de contrôle, les tâches peuvent être contenues par d'autre composants.
- **Les modules** : représentent toutes les sortes d'éléments physiques qui entrent dans la fabrication des applications informatiques (fichier, bibliothèque, sous-ensemble de logiciels...). Les modules peuvent être décomposés en processus et tâches.

III.2.9- Le diagramme de déploiement

Le diagramme de déploiement représente l'architecture physique des composants matériels (les nœuds) qui supportent l'exécution du système. Les liens entre ces composants, permettent de décrire les supports physiques de communication.

Le diagramme de déploiement permet de représenter les classes de composants matériels ainsi que les instances particulières de ces classes (objet).

Deux types de composants matériels sont distingués dans UML.

- Un processus (de données ou de traitements par exemple).
- Un dispositif périphérique par rapport au processus.

Exemple :

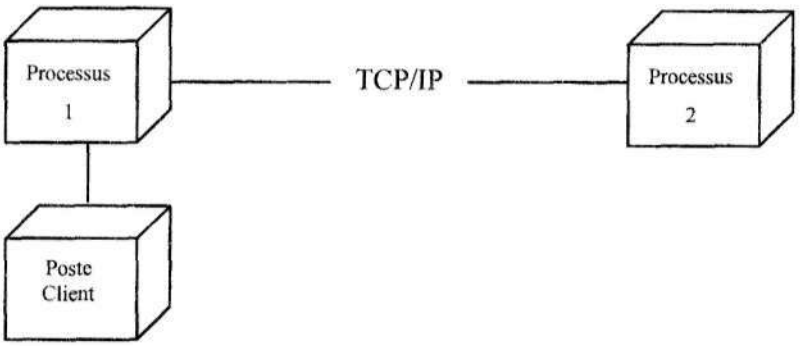


Figure 22 : *Exemple de type de diagramme de déploiement à trois composants*

Exemple :

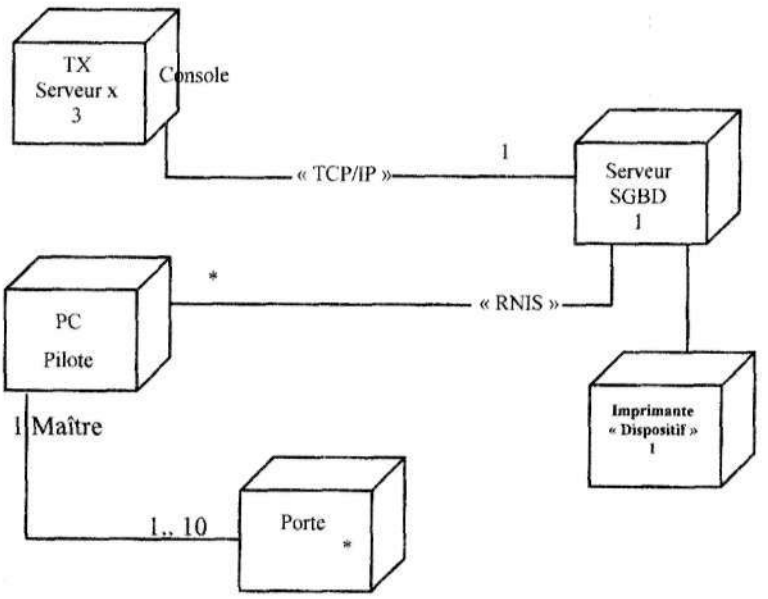


Figure 23 : *Exemple de classe dans un diagramme de déploiement*

Le diagramme montre que le système est constitué de serveurs ; autour duquel gravitent des PC pilotant l'ouverture et la fermeture de portes, trois terminaux jouent le rôle de console pour accéder au système, une imprimante est reliée au serveur.
Le diagramme décrit aussi la nature des liens de communications entre les différents nœuds.

III.3- Les mécanismes d'extensibilité

En plus de ces diagrammes, la notation UML comporte également d'autres concepts et d'autres représentations graphiques relevant du métalangage : les éléments, les stéréotypes, les contraintes, les notes et les étiquettes. Enfin, un dernier concept, paquetages, permet de gérer la complexité des schémas.

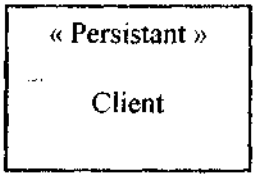
III.3.1- Les stéréotypes

Le concept de stéréotype peut être associé à tout élément du modèle, à savoir les classes, les associations, les opérations, les attributs, les paquetages etc.
Les stéréotypes ont deux fonctions :

- Ils classifient des éléments du modèle : ils permettent donc de créer des familles d'éléments associés à un même stéréotype ;
- Ils modifient la sémantique des éléments associés et permettent la création de nouveaux concepts propres à une application.

Les stéréotypes sont représentés par un nom entouré de guillemets.

Exemple :



Le stéréotype "persistant" indique que la classe Client est capable de se connecter à une base de données.

III.3.2- Les étiquettes

Une étiquette est une paire (nom, valeur) qui décrit une propriété d'un élément de modélisation. Les propriétés permettent l'extension des attributs des éléments du méta modèle. Une étiquette modifie la sémantique de l'élément qu'elle qualifie.

III.3.3- Les contraintes

Une contrainte est une relation sémantique quelconque entre éléments de modélisation. UML ne spécifie pas de syntaxe particulière pour les contraintes, qui peuvent ainsi être exprimées en langage naturel, en pseudo-code, par des expressions de navigation ou par des expressions mathématiques.

III.3.4- Les notes

Une note est un commentaire attaché à un ou plusieurs éléments de modélisation. Par défaut, les notes ne véhiculent pas de contenu sémantique. Toutefois, une note peut être transformée en contrainte au moyen d'un stéréotype, et dans ce cas, elle modifie la sémantique des éléments de modélisation.

III.3.5- Les paquetages

Les paquetages offrent un mécanisme général pour la partition des modèles et le regroupement des éléments de modélisation. Chaque paquetage est représenté graphiquement par un dossier.

Les paquetages divisent et organisent les modèles de la même manière que les répertoires organisent les systèmes de fichier. Chaque paquetage correspond à un sous-ensemble de modèles et contient, selon le modèle, des classes, des objets, des relations, des composants ou des nœuds, ainsi que les diagrammes associés.

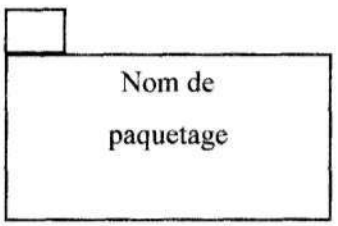


Figure 24 : *Représentation des paquetages*

III.4- Les cycles de vie objet [LOP 00]

Le cycle de vie d'un logiciel a pour but d'expliquer le séquencement des différentes étapes (expression des besoins, spécification, analyse, conception, implémentation, test et vérification, validation, maintenance et évolution) ainsi que la manière dont elles coopèrent. Nous exposerons le cycle de vie objet préconisé par les concepteurs d'UML et utilisé dans la plupart des projets orientés objet.

Le cycle de vie d'un *projet objet* possède trois caractéristiques fondamentales :

- Une bonne traçabilité entre les étapes ;
- Un caractère itératif ;
- Un caractère incrémental.

- **Une bonne traçabilité :**

La traçabilité est la possibilité d'effectuer aisément une correspondance entre les éléments définis dans deux phases successives. Elle est un bon indicateur du niveau de rupture entre deux étapes du cycle de vie.

Dans un projet traditionnel utilisant une méthode d'analyse/conception systémique et un langage structuré ; par exemple, les concepts utilisés dans les différentes phases sont totalement distincts. Cela engendre une difficulté d'effectuer une correspondance entre les éléments dans les phases successives : la traçabilité des éléments est faible.

Dans un projet objet il existe une bonne traçabilité car les concepts qui sont véhiculés peuvent être utilisés à tous les stades de cycle de vie, favorisant ainsi la communication entre les équipes mais aussi le passage d'une phase à une autre. Par exemple une classe définie dans la classe d'analyse devient généralement une classe dans la phase de conception et dans la phase de développement.

- **Un cycle de vie itératif**

Le cycle de vie objet est itératif, en ce sens qu'il propose de répéter toutes les étapes depuis la phase d'expression des besoins jusqu'à la validation, tant que la validation n'est pas satisfaisante : ceci améliore la cohérence de la gestion des projets.

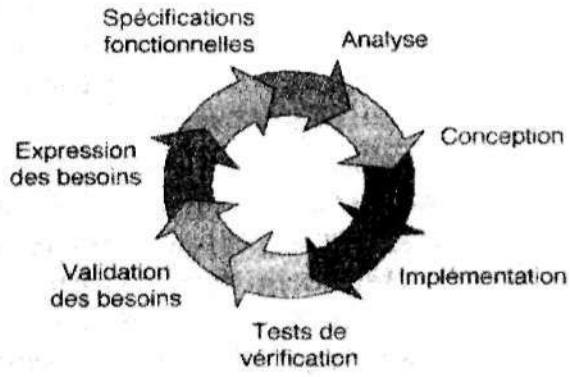


Figure 25 : *Le cycle de vie itératif* [LOP 00].

Ce type de construction itérative s'avère très productif car il permet très tôt des retours sur :

- les modifications des spécifications ;
- l'adéquation entre l'analyse, la conception et l'implémentation ;
- les domaines potentiels à risques (performance des accès en base de données par exemple).

- **Un cycle de vie incrémental**

Une construction incrémentale est basée sur la réalisation d'une série de **prototypes** permettant d'évaluer sur un nombre réduit de fonctionnalités, la faisabilité du projet (performance du langage, les accès aux bases de données, du matériel, etc.), ainsi que le processus de développement et l'organisation générale du projet. Chaque prototype, par des évolutions incrémentales successives, permettra d'aboutir à la réalisation finale du système. Le passage d'une génération d'un prototype à un autre permet de détecter rapidement des défauts et les dérivés de la construction logicielle et donc de faire à chaque fois une réévaluation des risques du projet et un réaménagement des objectifs.

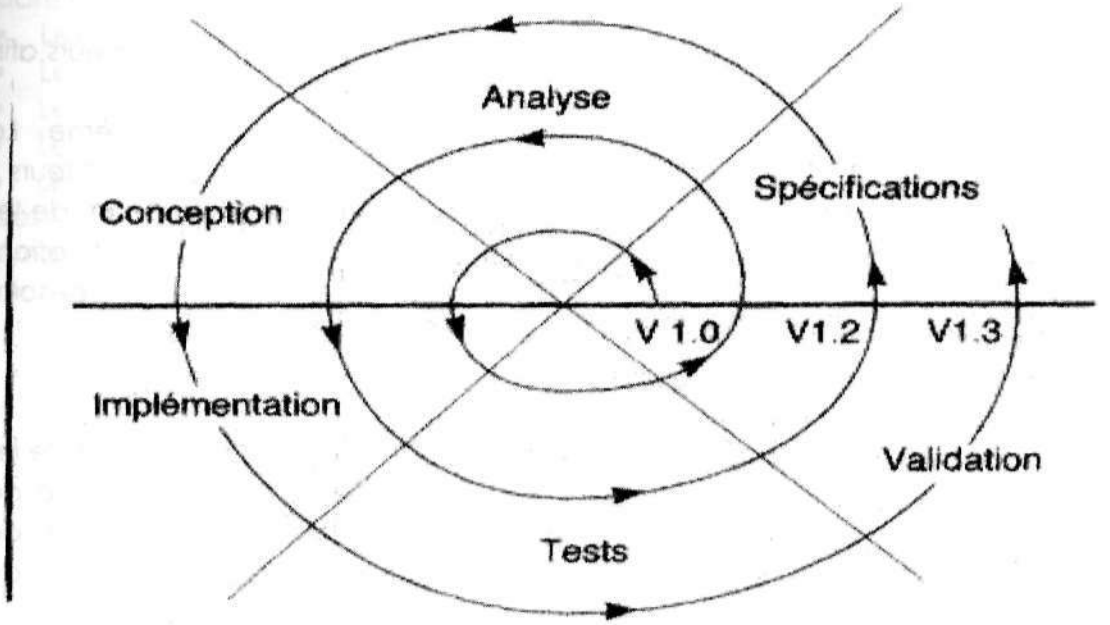


Figure 26 : *Cycle de vie incrémental [LOP 00]*

III.4.1- Les étapes de cycle de vie d'une application [LOP 00]

III.4.1.a- L'expression des besoins

La phase d'expression des besoins est la première étape dans le cycle de développement d'un logiciel.
L'expression des besoins doit traduire ce que le futur système est susceptible d'apporter aux utilisateurs. Elle définit les fonctionnalités du système et surtout la façon de l'utiliser.

L'expression des besoins doit être élaborée au court d'un dialogue entre les individus qui souhaitent mettre en place un nouveau système ou faire évoluer un système existant et des individus qui sont aptes à juger si ces aspirations sont techniquement réalisables. L'expression des besoins se matérialise par un document dans lequel la terminologie doit être compréhensible par les deux parties.

III.4.1.b- Les spécifications

L'expression des besoins est insuffisante et trop imprécise pour servir de base à la réalisation d'un système. Pour l'affiner il est nécessaire de mettre en place une phase de spécification. L'objectif des spécifications est de délimiter précisément le système et de décrire les différentes manières de l'utiliser du point de vue des divers utilisateurs.
Les spécifications se constituent à base de discussion avec les utilisateurs afin de tenir compte de leurs attentes et préférences.
Les use cases est une bonne approche pour spécifier le système. La confrontation des deux points de vue des utilisateurs et des " spécificateurs " permet de compléter les spécifications déjà établies, de les améliorer, de les valider. Il n'est pas possible de définir en une seule fois des spécifications exhaustives, en raison de modifications des données externes, des besoins supplémentaires qui sont mis en évidence par des réflexions.

III.4.1.c- L'analyse

L'analyse du système s'appuie sur les spécifications et vise à comprendre *les problèmes métiers* qui sont posés. Cette phase doit être indépendante de toute considération technique et informatique et de tout aspect de réalisation : langages de programmation, systèmes de gestion des bases de données, configuration matérielle etc.
L'objectif est de déterminer les éléments intervenant dans le système, leur structure ainsi que leurs relations.

L'analyse doit faire apparaître des caractéristiques métiers des objets suivant trois axes :

- Un axe fonctionnel décrivant le savoir-faire de l'objet.
- Un axe statique représentant la description structurelle de l'objet.
- Un axe dynamique décrivant le cycle de vie de l'objet au cours de l'application (les états par lesquels passe l'objet ainsi que les événements qui lui sont envoyés).

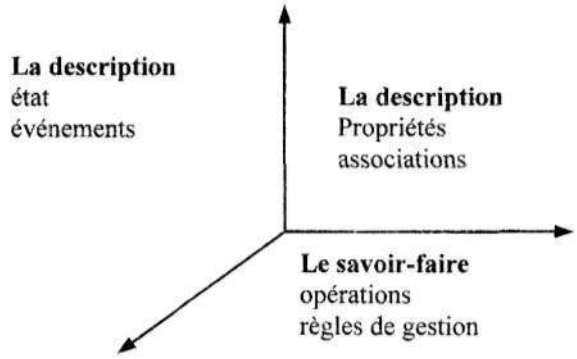


Figure 27 : *La représentation triaxiale d'un objet*

Parmi les nombreux diagrammes définis par UML ceux qui peuvent être utilisés dans l'analyse sont :

- Les spécifications des classes et des objets.
- Les diagrammes de collaboration.
- Les diagrammes de séquence.
- Les diagrammes de classes.
- Les diagrammes d'état-transition.

Cette étape d'analyse a donc pour but de déterminer les objets et leur dynamique.

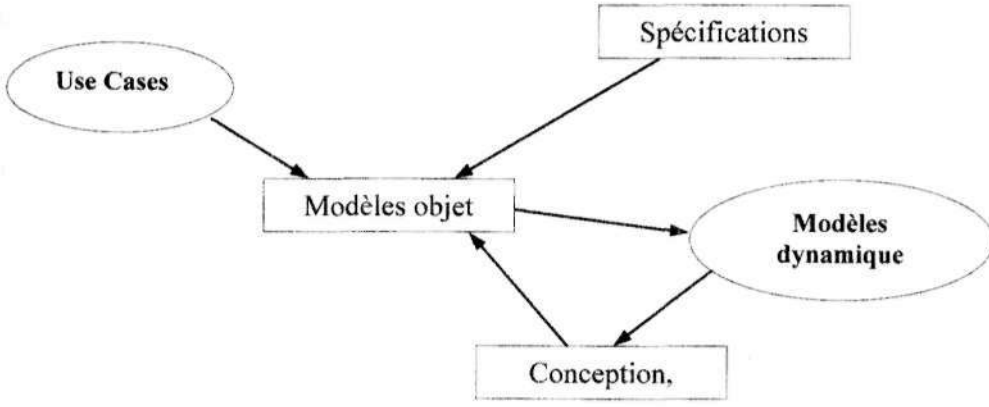


Figure 28 : *La place des modèles d'analyse dans le processus de développement*

III.4.1.d- La conception

La phase de conception s'appuie sur les documents et les modèles élaborés au cours de l'analyse et les affine en tenant compte de l'environnement technique dans lequel évolue le système. Son objectif est de déterminer la manière de résoudre le problème étudié par l'analyse et donc de proposer des solutions de réalisation. Pour cela, il s'agit d'effectuer de nombreux choix :

- Les choix d'une *architecture technique* sont stratégiques et doivent être déterminés en parallèle des phases de spécification et d'analyse ;
- Les décisions de performance et d'optimisation ;
- Les stratégies de programmation et de persistance des données, qui ne peuvent être choisis qu'après une phase d'analyse.

C'est plus tard dans le cycle de développement, au cours de la phase de test, que la validation de la conception sera effectuée.

III.4.1.e- L'implémentation

L'implémentation est la phase au cours de laquelle les structures et les algorithmes définis pendant la conception sont traduits dans un langage de programmation et/ou une base de données.

L'utilisation d'outils de génération de code à partir de modèles de conception détaillés permet de diminuer de façon significative l'étape d'implémentation, car ils permettent de créer rapidement un grand nombre d'instructions de programmation, comme ils peuvent intervenir dans la production du support de persistance des données (tables relationnelles,bases de données orientées objet, etc.). Cela peut réduire significativement les "risques de dérapage " du projet si mal maîtrisés aujourd'hui.

III.4.1.f- Les tests de vérification

Ils ont pour objectif de contrôler le travail effectué par les développeurs et de relever tous les défauts de conception et d'implémentation, afin de garantir la robustesse et la cohérence du système.

Cette étape de vérification est souvent reléguée en fin de projet, ce qui engendre trois conséquences néfastes :

- La durée de test est souvent sous estimée dans la planification du projet ;
- Les erreurs décelées sont mal perçues par les concepteurs et les développeurs ;
- Les erreurs ont des répercussions importantes car elles sont décelées trop tard (leur correction en est plus coûteuse).

Pour contourner ces problèmes, il s'agit de prévoir des tests de vérification très tôt dans le cycle de développement.

III.4.1.g- La validation

La validation peut être considérée à plusieurs niveaux dans le cycle de développement du logiciel ; en effet le client peut valider les spécifications, l'analyse et parfois, suivant ces compétences techniques, la conception et le développement.
La validation doit être effectuée une fois que la qualité technique du système (tests et vérification) a été démontrée.
Pour la validation on peut s'appuyer sur les use cases qui sont décrits car ils constituent des scénarios d'utilisation du système.

III.4.1.h- La maintenance et l'évolution

Les phases de maintenance et d'évolution interviennent à partir du moment ou le logiciel entre en exploitation.
Deux types de modifications peuvent être mises en œuvre, celles qui sont du ressort de la maintenance et celles qui relèvent de l'évolution :

- La maintenance consiste à effectuer des corrections "mineurs " mais nécessaires ; visant à augmenter la robustesse et les performances du produit et à augmenter sa convivialité ;
- L'évolution consiste d'une part à adapter le logiciel pour répondre aux modifications de l'environnement tel que l'ajout de fonctionnalités et d'écrans au fur et à mesure que de nouveaux besoins apparaissent, et d'autre part à effectuer des modifications de l'architecture interne de l'application destinées à corriger des erreurs de conception.

III.5- Usage d'UML et compléments nécessaires [POT 97]

Dans une perspective " système d'information " plusieurs lacunes apparaissent :

- Le cycle d'abstraction utilisé est incomplet. En UML, pour ce qui concerne les traitements, existe un niveau logique, voir organisationnel (use cases). En revanche, il manque un niveau conceptuel, *indépendant de toute technique et de toute organisation*, indispensable pour la définition des processus métier.

- Il manque la notion de *vue*, vision partielle d'un modèle, que ce soit pour une famille d'utilisateurs (modèle externe utilisateurs) ou pour une couche de composants d'architecture (vue locale d'usage). Son absence amène à multiplier des sous-ensembles de définitions de classes.

La nécessité d'étudier une entreprise comme un système global qui interagit avec son environnement, implique l'utilisation d'une approche systémique, existant depuis longtemps dans des méthodes européennes comme Merise. Cette approche met l'accent, comme base de construction des SI, sur ce qui est invariant, c'est à dire indépendant de toute organisation et de toute informatisation existante. Les deux éléments invariants sont les *processus* de réaction de l'entreprise à son environnement et la *sémantique* des informations manipulées. Ne se préoccuper (sérieusement) que des systèmes informatisés et de leurs interactions avec des " utilisateurs " est une vision réductrice pour concevoir un nouveau système.

Le rôle des cas d'utilisation (use cases) d'UML est de décrire l'expression des besoins fonctionnels de chaque catégorie d'utilisateurs. Or, les processus doivent décrire le comportement global du système. Il est bien entendu possible de modifier ce processus à l'aide des cas d'utilisation en leur associant des diagrammes d'activité de contenu volontairement conceptuel. Cependant, cela " détourne " ces modèles de leur vocation réelle au prix, qui plus est, d'une perte de sémantique.

Les diagrammes de classe d'UML expriment aussi bien la sémantique des éléments du SI que la structuration du SI en classes d'objet. Si l'on veut *commencer* par modéliser cette sémantique *avant de choisir une solution de structuration* en classes, il est possible de " détourner " la modélisation des classes en se limitant aux aspects sémantiques.

En ce qui concerne les préoccupations organisationnelles, il est possible de les modéliser en UML, en sachant, cependant, qu'il est difficile d'exprimer des vues de cas d'utilisation (les diagrammes de collaboration obligent à penser déjà en termes d'objets) et impossible de représenter de modèles externes des classes (vues de la sémantique). De même, les diagrammes et les couloirs d'activité ont une granularité trop fine pour décrire la contribution globale d'un même acteur à un processus (métier) donné.

CONCLUSION

La modélisation et la conception orientée objet proposent une nouvelle façon de penser aux problèmes en utilisant des modèles organisés autour du monde réel. Le développement orienté objet possède comme principaux avantages par rapport au développement conventionnel la vocation à permettre une futur réutilisabilité et une réduction des erreurs en aval ainsi que l'effort de maintenance.

Le formalisme de type UML est une tentative d'exploiter les avancées récentes introduites par une modélisation objet conséquente (héritage, composition), il vise à standardiser les concepts graphiques.

Notre conclusion ne peut être que provisoire car nous sommes en présence d'un domaine en pleine évolution. Néanmoins, nous sommes convaincu que l'avènement d'un modèle objet standardisé, assurera la promotion du développement de logiciels guidés par la modélisation, et qu'ainsi l'automatisation de la conception technique et du codage sera une tendance majeure dans la décennie à venir.

Bibliographie

Ouvrages :

[BOO 94] G. Booch,
"Object Oriented Analysis and Design with Application", 2eme édition, Addision-Wesley, 1994

[BOO 96] G. Booch,
"Object Solutions", Addision-Wesley, 1996

[BOU 94] Mokrane Bouzeghoub & George Gardarin & Patrick Val duriez, "Objets: du C++ à Merise Objet", Editions Eyrolles, 1994

[BOU 97a] Mokrane Bouzeghoub & George Gardarin & Patrick Val duriez," Les Objets", ISBN 2-212-08957-0, Edition Eyrolles, 1997

[COA 91a] P. Coad, E. Yourdon,
"Object Oriented Analysis", Second Edition, Yourdon Press, Prentice Hall, Inc., 1991

[COA 91b] P. Coad, E. Yourdon,
"Object Oriented Design", Second Edition, Yourdon Press, Prentice Hall, Inc., 1991

[DEL 93] B. Delatte, M. Heirtz, J.F. Muller,
"HOOD Reference Manuel 3.1", Editions Masson, 1993

[GRA 94] I. Graham,
"Object Oriented Methods", 2eme édition, Addision. We sley, 1994

[JAC 94] I. Jacoboson, M.Christeron, P.Jonsson , G.Overgaard
"Le génie logiciel orientée objet. Une approche fondée sur les cas d'utilisation ", Addision. Wesley, 1994

[LOP 00] Nathalie Lopez & Jorge Migues & Emmannuel Pichon

" Intégrer UML dans vos projets ", EYROLLES 2000

[MUL 98] Pierre Alain Muller,
" Modélisation objet avec UML ", EYROLLES 1998

[RUM 96] James Rumbaugh et al.
" OMT, Modélisation et conception orientées objet ", MAS SON 1996

Mémoires et thèses :

[MOH 99] Abdlkrim Mohammedi,
" Conception et réalisation d'une base documentaire : Modélisation par la méthode OMT "
Université des Sciences et de la Technologie Houari-Boumedien, 1999

[CHE 99] Chelroum Nadjim & Ziadi Toufik
" Conception et réalisation d'un SI pour le traitement des redevances aéronautiques "
Institue Nationale d'Informatique, INI, 1999

Articles :

[BOU 97b] Mokrane Bouzeghoub,
Unification des méthodes objets : fusion ou confusion
" Ingénierie des systèmes d 'information " .volume5- n°5/1997

[POT 97] François Potier- Arnold Rochfeld – Yves Tabourier,
UML et systèmes d'information
" Ingénierie des systèmes d 'information " .volume5- n°5/1997

[DES 97] Philippe Desfray,
UML, méta-modélisation design patterns et hypergénéricité
" Ingénierie des systèmes d 'information " .volume5- n°5/1997

[PER 97] Philippe Perrin
La guerre des standards des méthodes orientées –objet n'aura pas lieu !
" Ingénierie des systèmes d 'information " .volume5- n°5/1997

Méthodes d'analyse et de conception orientées objet : présentation d'UML (Unified Modeling Language)

Bessai F.Z.
CEntre de Recherche sur l'Information Scientifique et Technique
Laboratoire Base de Données et Système d'Information
***E-Mail**: ZBessai@mail.cerist.dz*

INTRODUCTION

L'approche objet a pour but une modélisation des propriétés statiques et dynamiques de l'environnement dans lequel sont définis les besoins. Il est appelé le domaine de problème. Elle formalise notre perception du monde et des phénomènes qui s'y déroulent et met en correspondance l'espace de problème et l'espace de la solution, en préservant la structure et le comportement du système analysé.

L'approche objet est loin d'être une idée récente. Simula, premier langage de programmation a implémenté le concept de type abstrait à l'aide de classes, date de 1967. En 1976, Smalltalk implémente les concepts fondateurs de l'approche objet: encapsulation, agrégation et héritage.

L'approche objet est donc devenue une réalité. Utiliser l'approche objet n'est plus une mode, mais un réflexe quasi-automatique dès lors qu'on cherche à concevoir des logiciels complexes qui doivent résister à des évolutions incessantes.

Bien que les racines de l'objet soient solidement ancrées dans les années 60, ce n'est que vers les années 80 que les méthodes objet ont commencé à émerger. Par la suite, l'approche objet est devenue incontournable pour le développement de systèmes, qu'ils soient industriels ou de gestion, de simulation ou de contrôle de processus. Devant cet engouement pour les techniques orientées objet, il est naturel de s'attendre à une augmentation du nombre de méthodes.