Introduction à UML
Unified Modeling Language

Cours UML

Hammal Youcef

1 / Historique

- **Les premières méthodes d'analyse (années 70)** Découpe cartésienne (fonctionnelle et hiérarchique) d'un système.
- **L'approche systémique (années 80)** Modélisation des données + modélisation des traitements (Merise, Axial, IE...).
- **L'émergence des méthodes objet (1990-1995)**
  - Prise de conscience de l'importance d'une méthode spécifiquement objet :Comment structurer un système sans centrer l'analyse uniquement sur les données ou uniquement sur les traitements (mais sur les deux) ?
  - Plus de 50 méthodes objet sont apparues durant cette période (Booch, Classe-Relation, Fusion, HOOD, OMT, OOA, OOD, OOM, OOSE...) !
  - Aucun méthode ne s'est réellement imposée.

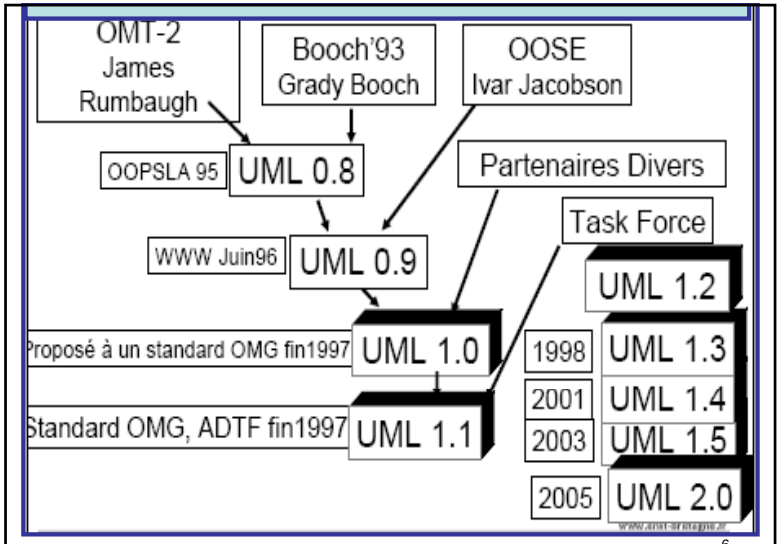
Les premiers consensus (1995)

- **OMT** (James Rumbaugh) : vues statiques, dynamiques et fonctionnelles d'un système
  - Issue du centre de R&D de General Electric.
  - Notation graphique riche et lisible.
- **OOD** (Grady Booch) : vues logiques et physiques du système
  - Définie pour le DOD, afin de rationaliser de développement d'applications ADA, puis C++.
  - Ne couvre pas la phase d'analyse dans ses 1ères versions (préconise SADT).
  - Introduit le concept de package (élément d'organisation des modèles).
- **OOSE** (Ivar Jacobson) : couvre tout le cycle de développement
  - Issue d'un centre de développement d'Ericsson, en Suède.
  - La méthodologie repose sur l'analyse des besoins des utilisateurs.

L'unification et la normalisation des méthodes (1995-1997)

- UML (Unified Modeling Langage), la fusion et synthèse des méthodes dominantes :
  - OMT
  - OOD
  - OOSE
- Langage normalisé par l'OMG et supporté par les grandes compagnies industrielles et informatiques.

5



6

UML : un standard incontournable

- UML est le résultat d'un large consensus (industriels, méthodologistes...).
- UML est le fruit d'un travail d'experts reconnus.
- UML est issu du terrain.
- UML est riche (il couvre toutes les phases d'un cycle de développement).
- UML est ouvert (il est indépendant du domaine d'application et des langages d'implémentation).
- Les outils qui supportent UML se multiplient (GDPro, ObjectTeam, Objecteering, OpenTool, Rational Rose, Rhapsody, STP, Visio, Visual Modeler, WithClass...).
- XMI (format d'échange standard de modèles UML).

7

UML évolue mais reste stable !

- L'OMG RTF (nombreux acteurs industriels) centralise et normalise les évolutions d'UML au niveau international.
- Les groupes d'utilisateurs UML favorisent le partage des expériences.
- De version en version, UML gagne en maturité et précision, tout en restant stable.
- UML inclut des mécanismes standards d'auto - extension.
- La description du **métamodèle** d'UML est standardisée (OMG-MOF).

8

2 / A quoi sert UML ?

9

UML n'est pas une méthode ou un processus !

- UML est un langage et non pas une méthode
- Une méthode propose aussi un processus, qui régit notamment l'enchaînement des activités de production d'une entreprise.
- UML a été pensé pour permettre de modéliser les activités de l'entreprise, pas pour les régir.
- Un processus de développement logiciel universel est une utopie (Impossible de prendre en compte toutes les organisations et cultures d'entreprises).

10

UML est un langage pseudo-formel

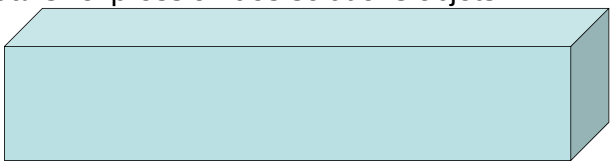
- UML est fondé sur un métamodèle, qui définit :
  - les éléments de modélisation (les concepts manipulés par le langage),
  - la sémantique de ces éléments (leur définition et le sens de leur utilisation).
- Un métamodèle est une description très formelle de tous les concepts d'un langage. Il limite les ambiguïtés et encourage la construction d'outils.
- Le métamodèle UML est lui-même décrit par un méta-métamodèle (OMG-MOF).
- UML propose aussi une notation, qui permet de représenter graphiquement les éléments de modélisation du métamodèle.

11

UML cadre l'analyse objet

Il Offre :

- différentes vues complémentaires d'un système, qui guident l'utilisation des concept objets (Aspects statiques, dynamiques…)
- plusieurs niveaux d'abstraction, qui permettent de mieux contrôler la complexité dans l'expression des solutions objets.



12

UML est un support de communication

- Sa notation graphique permet d'exprimer visuellement une solution objet.
- L'aspect formel de sa notation limite les ambiguïtés et les incompréhensions.
- Son aspect visuel facilite la comparaison et l'évaluation de solutions.
- Son indépendance (par rapport aux langages d'implémentation, domaine d'application, processus...) en font un langage universel.

13

3/ Qu'est-ce qu'un modèle ?

14

Un modèle est une abstraction de la réalité

- Un modèle est une **description abstraite** d'un système, destinée à en préciser certaines caractéristiques.
- L'abstraction est un des piliers de l'approche objet.
  - Il s'agit d'un processus qui consiste à identifier les caractéristiques intéressantes d'une entité, en vue d'une utilisation précise.
  - L'abstraction désigne aussi le résultat de ce processus, c'est-à-dire l'ensemble des caractéristiques essentielles d'une entité, retenues par un observateur.

15

Un modèle est une vue subjective mais pertinente de la réalité

- Un modèle définit une frontière entre la réalité et la perspective de l'observateur. Ce n'est pas "la réalité", mais une vue très subjective de la réalité.
- Bien qu'un modèle ne représente pas une réalité absolue, un modèle reflète des aspects importants de la réalité, il en donne donc une vue juste et pertinente.

16

Quelques exemples de modèles

- Modèle météorologique :
  - à partir de données d'observation (satellite ...), permet de prévoir les conditions climatiques pour les jours à venir.
- Modèle économique :
  - peut par exemple permettre de simuler l'évolution de cours boursiers en fonction d'hypothèses macro-économiques (évolution du chômage, taux de croissance...).
- Modèle démographique :
  - définit la composition d'un échantillon de population et son comportement, dans le but de fiabiliser des études statistiques, d'augmenter l'impact de démarches commerciales, etc...

17

Caractéristiques fondamentales des modèles

- Le caractère abstrait d'un modèle doit notamment permettre :
  - de faciliter la compréhension du système étudié
    > Un modèle réduit la complexité du système étudié.
  - de simuler le système étudié
    > Un modèle représente le système étudié et reproduit ses comportements.
- Un modèle réduit (décompose) la réalité, dans le but de disposer d'éléments de travail exploitables par des moyens mathématiques ou informatiques.

18

4/ Modélisation en Génie Logiciel

19

Rôles d'un modèle en Génie Logiciel

- communication entre les différentes phases du développement.
- documentation de la structure interne du logiciel.
- En pratique, un modèle est représenté par un ensemble de documents (diagrammes, textes, …).
- Il existe un standard industriel pour l'écriture de différents types de modèles : UML (Unified Modeling Language).

20

Pourquoi modéliser en G-L

- Un modèle est une simplification de la réalité qui permet de mieux comprendre le système à développer.
- Il permet :
  - De visualiser le système comme il est ou comme il devrait l'être.
  - De valider le modèle vis à vis des clients
  - De spécifier les structures de données et le comportement du système.
  - De fournir un guide pour la construction du système.
  - De documenter le système et les décisions prises.

21

Les principes de la modélisation

1. Le modèle doit être connecté au monde réel
2. Un modèle peut être exprimé avec différents niveaux de précision
3. Un simple modèle n'est pas suffisant, il y a plusieurs façons de voir un système.
   - plan de masse
   - vue de face, de coté, …
   - plan des niveaux
   - plan électrique
   - plan de plomberie
   - plan des calculs de construction

22

Qu'apporte la modélisation objet

- Plus grande indépendance du modèle par rapport aux fonctionnalités demandées.
- Des fonctionnalités peuvent être rajoutées ou modifiées, le modèle objet ne change pas.
- Plus proche du monde réel.

23

Les objectifs d'UML

- Représenter des systèmes entiers
- Établir un couplage explicite entre les concepts et les **artefacts** exécutables
- Prendre en compte les facteurs d'échelle
- Créer un langage de modélisation utilisable à la fois par les humains et les machines.
- Recherche d'un langage commun :
  - Utilisable par toutes les méthodes
  - Adapté à toutes les phases du développement
  - Compatible avec toutes les techniques de réalisation

24

UML un langage pour

- visualiser; chaque symbole graphique a une sémantique
- spécifier de manière précise et complète, sans ambiguïté,
- Construire; les classes, les relations SQL peuvent être générées automatiquement
- Documenter; les différents diagrammes, notes, contraintes, exigences seront présentés dans un document.

25

UML et les domaines d'utilisation

- Systèmes d'information des entreprises
- Les Banques et les services financiers
- Télécommunications
- Transport
- Défense et aérospatiale
- Scientifique
- Applications distribuées par le WEB.

26

5/ Éléments de Base en UML

27

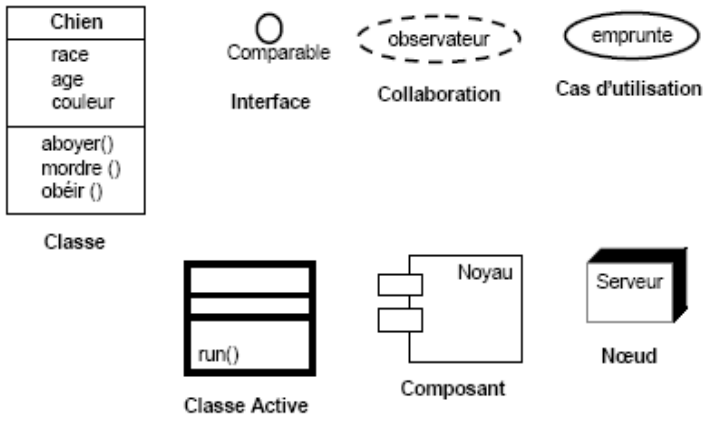
Les trois éléments de base en UML

1. les blocs de base pour construire
   - les entités utilisées
   - la notion de relation
   - les diagrammes
2. les règles à observer pour utiliser ces blocs de base
   - règles sémantiques
   - règles de présentation
3. les mécanismes communs
   - spécification
   - présentation
   - extension des modèles

1. Entités structurelles
2. Entités de comportement
3. Entités de regroupement
4. Entité d'annotation

28

Les entités structurelles



29

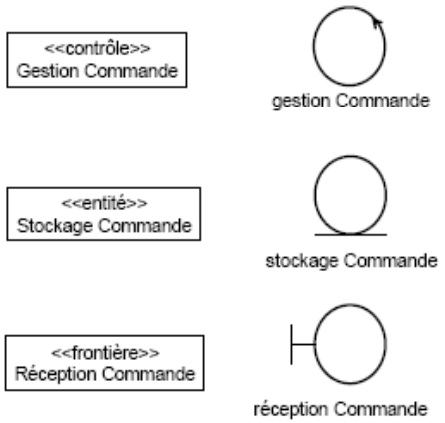
Les entités de comportement

Les entités de groupement

Les entités de notation

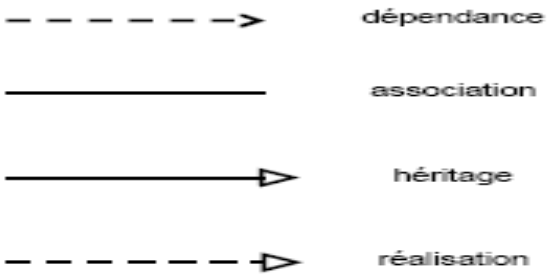
30

Stéréotypes et icônes associées



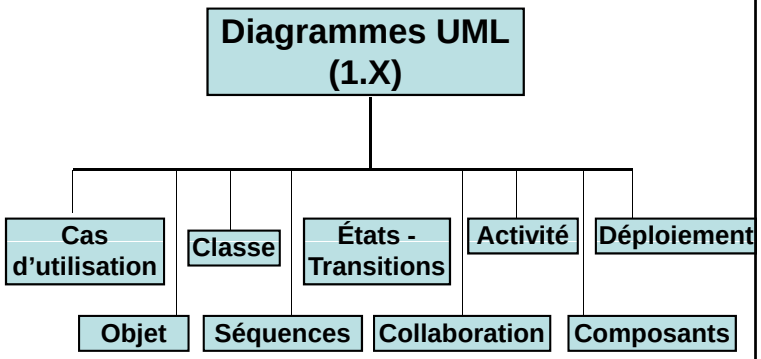
31

Les relations



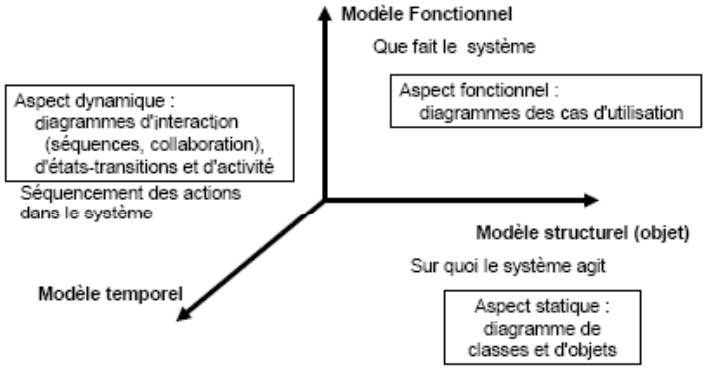
32

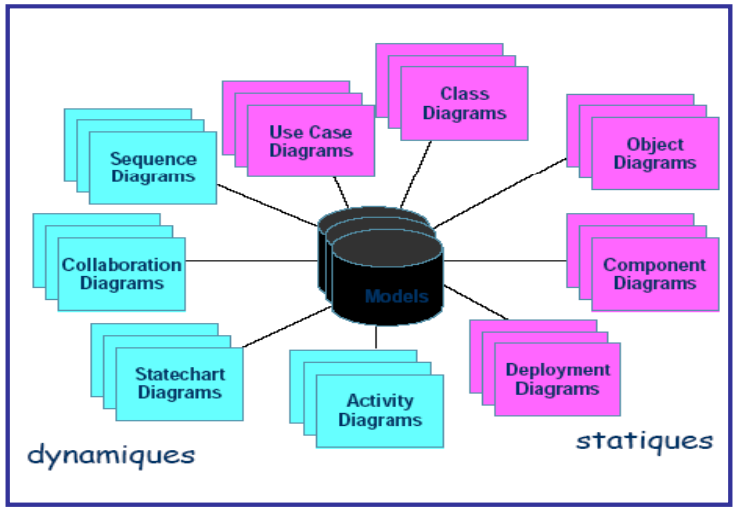
Les Diagrammes d'UML



33

Les trois composantes d'une modélisation





35

- Diagramme de cas d'utilisation : Décompose le cahier des charges en parties significatives.
- Diagramme de structure statique : Définit l'organisation des concepts (représentations mentales d'éléments du problème),
  - Classes, Objets.
- Diagramme de Composants : Met en évidence les dépendances entre groupes de concepts ou d'éléments logiciels.
- Diagramme d'activité : Ordonne les tâches devant être effectuées par le logiciel.

36

- Diagramme d'état : Détaille le comportement dynamique d'un composant.
- Diagramme d'interaction : Précise les interactions (collaborations) entre différents composants accomplissant une tâche.
  - Deux variantes : diagramme de séquence, diagramme de collaboration.
- Diagramme de déploiement : Définit la situation physique des composants du logiciel par rapport aux éléments matériels.

37

6/ Comment modéliser avec UML ?

38

- **UML est un langage qui permet de représenter des modèles, mais il ne définit pas le processus d'élaboration des modèles !**
- Cependant, dans le cadre de la modélisation d'une application informatique, les auteurs d'UML préconisent d'utiliser une démarche :
  - itérative et incrémentale,
  - guidée par les besoins des utilisateurs du système,
  - centrée sur l'architecture logicielle.
- D'après les auteurs d'UML, un processus de développement qui possède ces qualités devrait favoriser la réussite d'un projet.

39

Une démarche itérative et incrémentale ?

- L'idée est simple : pour modéliser (comprendre et représenter) un système complexe, il vaut mieux s'y prendre en plusieurs fois, en affinant son analyse par étapes.
- Cette démarche devrait aussi s'appliquer au cycle de développement dans son ensemble, en favorisant le prototypage.
- Le but est de mieux maîtriser la part d'inconnu et d'incertitudes qui caractérisent les systèmes complexes.

40

Une démarche pilotée par les besoins des utilisateurs ?

- **Avec UML, ce sont les utilisateurs qui guident la définition des modèles :**
  - Le périmètre du système à modéliser est défini par les besoins des utilisateurs (les utilisateurs définissent ce que doit être le système).
  - Le but du système à modéliser est de répondre aux besoins de ses utilisateurs (les utilisateurs sont les clients du système).
- **Les besoins des utilisateurs servent aussi de fil rouge, tout au long du cycle de développement (itératif et incrémental) :**
  - A chaque itération de la phase d'analyse, on clarifie, affine et valide les besoins des utilisateurs.
  - A chaque itération de la phase de conception et de réalisation, on veille à la prise en compte des besoins des utilisateurs.
  - A chaque itération de la phase de test, on vérifie que les besoins des utilisateurs sont satisfaits.

41

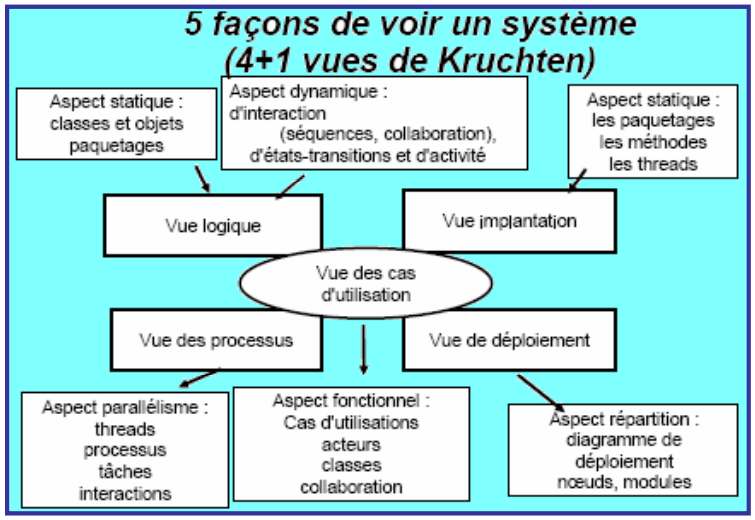
Une démarche centrée sur l'architecture ?

- **Une architecture adaptée est la clé de voûte du succès d'un développement.**

  Elle décrit des choix stratégiques qui déterminent en grande partie les qualités du logiciel (adaptabilité, performances, fiabilité...).
- **Ph. Kruchten propose différentes perspectives, indépendantes et complémentaires, qui permettent de définir un modèle d'architecture (publication IEEE, 1995).**

  Cette vue ("4+1") a fortement inspiré UML :

42



43

Point de vue des cas d'utilisation

- Cette vue (**La vue des besoins des utilisateurs**), guide toutes les autres.
  - Dessiner le plan (l'architecture) d'un système informatique n'est pas suffisant, il faut le justifier !
  - Cette vue définit les besoins des clients du système et centre la définition de l'architecture du système sur la satisfaction (la réalisation) de ces besoins (Réponses aux questions QUOI et QUI).
  - A l'aide de scénarios et de cas d'utilisation, cette vue conduit à la définition d'un modèle d'architecture pertinent et cohérent.
  - Cette vue est la "colle" qui unifie les quatre autres vues de l'architecture.
- Elle motive les choix, permet d'identifier les interfaces critiques et force à se concentrer sur les problèmes importants.

44

Point de vue logique

- Définition du système vue de l'intérieur.
- Expliquer **Comment** peuvent être satisfaits les besoins des acteurs.
- Décomposition orientée objet
  - Décomposition en objets et classes
  - Regroupement en paquetages. Connexions par héritage, association, etc.
  - Accent sur l'abstraction, l'encapsulation, l'uniformité.
  - Réalisation des scénarios des cas d'utilisation.

45

Point de vue implantation (1/2)

- Cette vue de bas niveau (aussi appelée «vue de réalisation» , «vue de composants »), montre :
  - L'allocation des éléments de modélisation dans des modules (fichiers sources, bibliothèques dynamiques, bases de données, exécutables, etc...).
  - En d'autres termes, cette vue identifie les modules qui réalisent (physiquement) les classes de la vue logique.
  - L'organisation des composants, c'est-à-dire la distribution du code en gestion de configuration, les dépendances entre les composants...
  - Les contraintes de développement (bibliothèques externes...).

46

Point de vue implantation (2/2)

- La vue des composants montre aussi l'organisation des modules en ***"sous-systèmes"***, les interfaces des sous-systèmes et leurs dépendances (avec d'autres sous-systèmes ou modules).
  - Réduire le couplage et la visibilité (dépendances entre modules)
  - Augmenter la robustesse
- Information sur les caractéristiques suivantes :
  - Facilite de développement
  - Potentiel de réutilisation
  - Gestion de configuration

47

Point de vue déploiement

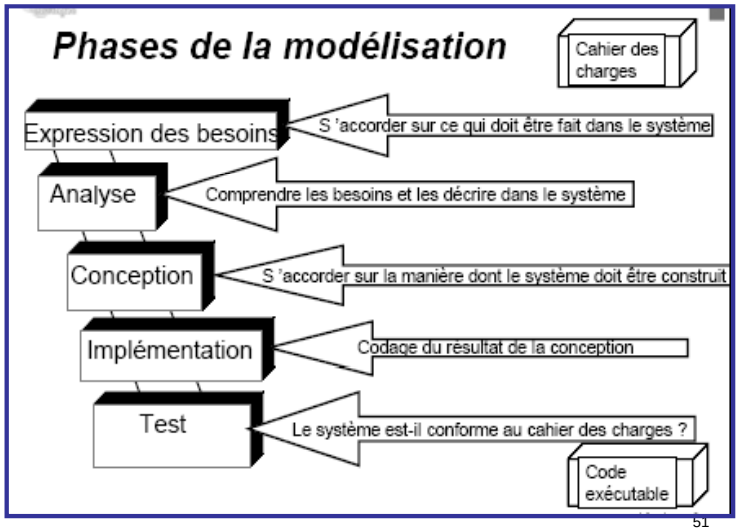
- Cette vue très importante dans les environnements distribués, décrit les ressources matérielles et la répartition du logiciel dans ces ressources (Où?) :
  - La disposition et nature physique des matériels, ainsi que leurs performances.
  - L'implantation des modules principaux sur les noeuds du réseau (Décomposition en noeuds d'exécution).
  - Inter - connectivité, topologie
- Les exigences en terme de performances (temps de réponse, tolérance aux fautes et pannes...).

48

Point de vue processus

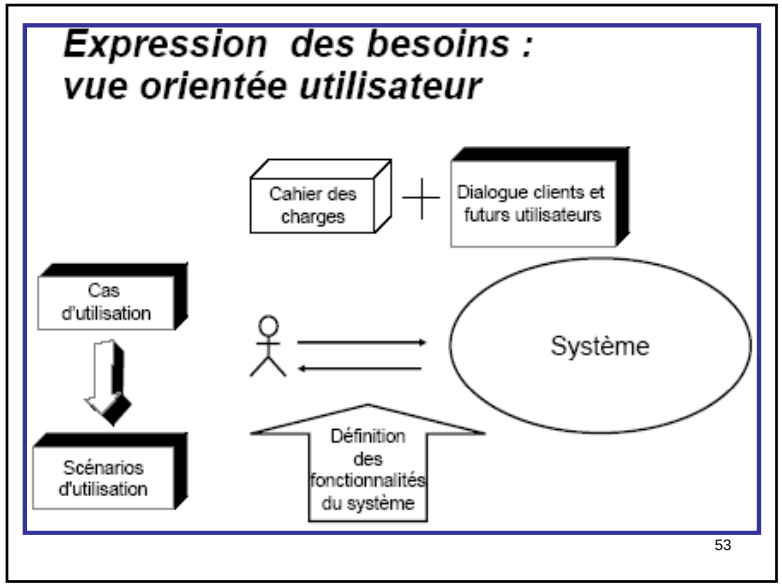
- Cette vue est très importante dans les environnements multitâches ; elle montre :
  - La décomposition du système en terme de processus et tâches.
  - Les interactions entre les processus (leur communication).
- La synchronisation et la communication des activités parallèles (threads).
- Information sur les caractéristiques suivantes :
  - Disponibilité, fiabilité
  - Intégrité, performance
  - Contrôle

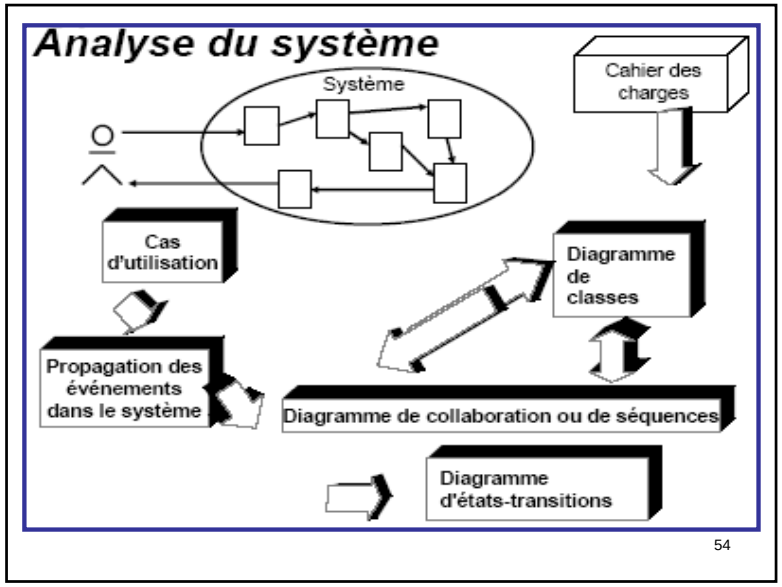
7/ Phases de modélisation



Expression des besoins

- Comprendre e contexte du système en définissant un modèle du domaine et du métier
- Recenser les besoins fonctionnels et les définir par des cas d'utilisations
- Noter les contraintes, exigences non fonctionnelles (temps de réponse, performance...)
- Le modèle du domaine regroupe les objets qui se situent dans le contexte du système :
  - entités existantes ou événements qui s'y produisent.





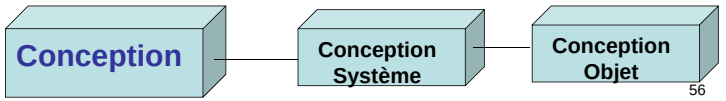
L'analyse

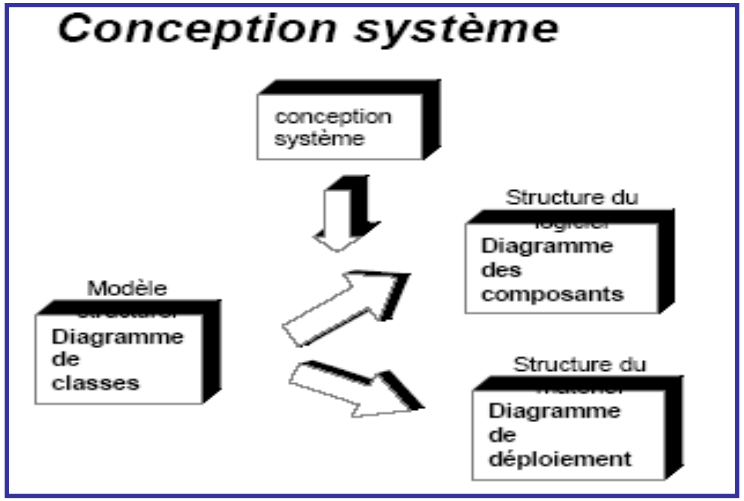
- Le but de l'analyse est de traduire dans un langage proche de celui des informaticiens les modèles exprimes dans l'expression des besoins.
- Cependant pour rester compréhensible par les clients ou utilisateurs, il n'intervient que des entités du domaine (métier) considéré.
- Elle sert d'interface, avec l'expression des besoins, aux dialogues et aux contrats avec les clients et les utilisateurs.
- L'analyse doit servir de support pour la conception, l'implémentation et la maintenance.

55

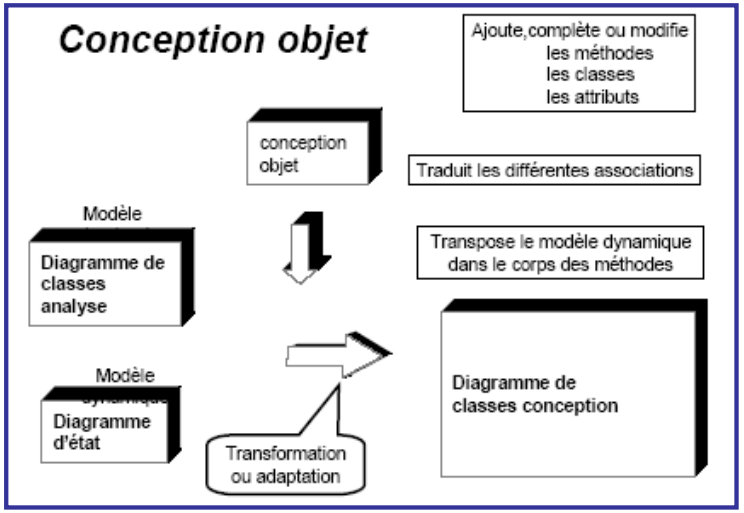
Conception

- La conception a pour objectif de rechercher comment le système va être réalisé, contrairement à l'analyse qui recherche ce qui doit être fait (quoi).
- Elle élabore les différentes parties du système et leurs interactions d'abord à un niveau général puis à un niveau de plus en plus détaillé.
- Elle tient compte des contraintes matérielles et logicielles : langages, bases de données, processeurs, périphériques, etc..
- C'est une étape où ne peuvent intervenir que des informaticiens spécialisés dans les différentes technologies utilisées.





57



58

Remarque Importante

- Certains diagrammes s'appliquent à plus d'une phase de développement.

	Besoins	Analyse	conception
cas d'utilisation	X		
struct. statique		X	X
Activité		X	X
État		X	X
Interaction	X (Scénarios)	X	X
Composant		X	X
Déploiement			X

59

Diagrammes de UML 2.0

60

Diagrammes Statiques d'UML 2.0

- **Diagramme de classes :** représente les classes intervenant dans le système.
- **Diagramme d'objets :** sert à représenter les instances de classes utilisées dans le système.
- **Diagramme de Composants :** permet de montrer les composants du système d'un point de vue physique, tels qu'ils sont mis en œuvre (fichiers, bibliothèques, bases de données …).
- **Diagramme de déploiement :** sert à représenter les éléments matériels (ordinateurs, périphériques, réseaux, systèmes de stockage …) et la manière dont les composants du système sont répartis sur ces éléments matériels et interagissent avec eux.
- **Diagramme de paquetages :** un paquetage étant un conteneur logique permettant de regrouper et d'organiser les éléments dans le modèle UML, le diagramme de paquetage sert à représenter les dépendances entre paquetages, c'est-à-dire les dépendances entre ensembles de définitions.
- **Diagramme de structures composites :** permet de décrire sous forme de boîte blanche les relations entre composants d'une classe.

61

Diagrammes Dynamiques d'UML 2.0

- **Diagramme de cas d'utilisation :** permet d'identifier les possibilités d'interaction (i.e., services fournis) entre le système et les acteurs.
- **Diagramme d'activités :** permet de décrire sous forme de flux ou d'enchaînement d'activités le comportement du système ou de ses composants.
- **Diagramme de séquences :** représentations séquentielle du déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs.
- **Diagramme d'états-transitions :** permet de décrire sous forme de machine à états finis le comportement du système ou de ses composants.
- **Diagramme de communication :** représentation simplifiée d'un diagramme de séquence, se concentrant sur les échanges de messages entre les objets.
- **Diagramme global d'interaction :** permet de décrire les enchaînements possibles entre les scénarios préalablement identifiés sous forme de diagrammes de séquences.
- **Diagramme de temps (Timing diagram) :** permet de décrire les variations d'une donnée au cours du temps.

62