SOMMAIRE

Généralités

A

A Généralités

I Élaboration d'un projet TITANIC

- Fixer la date de fin des travaux
- Baptiser le projet
- Trouver un dindon (le chef de projet)
- Trouver un analyste
- Ordonnancement rapide des travaux
- Première maquette d'écran
- Embaucher des programmeurs
- Bâtir des programmes de contrôle
- Concevoir les bases de données
- Réaliser un prototype
- Embaucher des programmeurs supplémentaires
- Corriger les plannings
- Réaliser un nouveau prototype
- Lâcher prise
- Paniquer
- Chercher un coupable
- Punir un innocent
- Revenir en 1

Comme son nom l'indique, ce type de projet est voué à l'échec Cependant, on peut retenir trois points forts de ce planning :

- L'ordonnancement des travaux
- Les points de contrô1e
- La réalisation de prototypage

II Notion de méthode

La méthode est faite donc d'une part pour pallier la démarche intuitive, et d'autre part, pour maîtriser la complexité des problèmes à résoudre De plus, il faut réussir à sortir de l'empirisme individuel pour la gestion de projet Pour cela on dispose des moyens suivants :

- **Faciliter la communication** entre les différents acteurs du projet par l'intermédiaire d'un langage commun
- Construire un SI **pertinent, fiable et homogène**
- Être à même d'**évaluer** le SI à tout moment

II.1 Les composants d'une méthode

Une méthode doit être composée des éléments suivants :

- Un cadre général de réflexion (principes fondamentaux)
- Une démarche
  - Étape de mise en oeuvre
- Raisonnement
  - Langages
  - Des modèles ou artefacts
- Moyens de mise en œuvre
  - Organisation

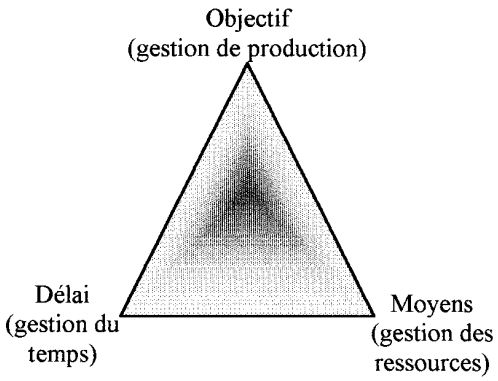
Document créé par picquart.ludwig@free.fr dans le cadre des cours de méthodologie du CNAM pour l'année universitaire 2001-2002.

- Partenaires et rôles définitifs
- Outils logiciels
- Une maîtrise du projet

III Notion de projet

Le terme de projet représente d'abord une intention souvent floue dont la réalisation peut être lointaine (Ex : projet de vacance). La deuxième définition décrit un projet comme une étude préparatoire, parfois exhaustive, qui va être soumise à décision (Ex. projet de loi, d'urbanisation...).

On parle de projet informatique lorsque celui-ci correspond à la situation a laquelle on se trouve, quand on doit atteindre un objectif avec des moyens et un délai donné. On parle du TRIANGLE DE PROJET.

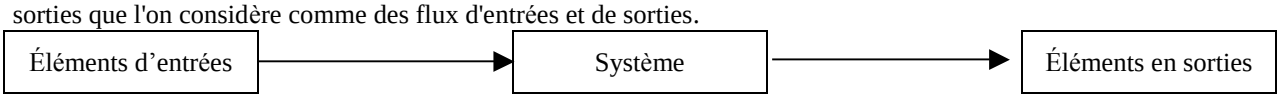


Un chef de projet est donc un gestionnaire (production, temps, ressources), le plus complexe étant celui de la gestion des ressources. En effet, c'est de cette partie où la variable financière est omniprésente que les décisions seront les plus difficiles à faire admettre ou à argumenter auprès des instances dirigeantes.

IV Qu'entend-on par S.I ?

IV.l Définition (1) :

Un système est un ensemble d'éléments (matériels ou non) transformant des éléments d'entrées en éléments de sorties que l'on considère comme des flux d'entrées et de sorties.



Ex.
Une entreprise commercialise une liste de produits:

- En entrée. Les produits achetés, commandes, paiements (clients)
- En sortie. Les produits vendus, factures, paiements (fournisseurs)

Ou en terme de flux

- Flux physiques : produits acheter, produits vendus
- Flux d'information : paiements clients/fournisseurs

On peut donc conclure qu'un flux déclenche un **PROCESSUS**.

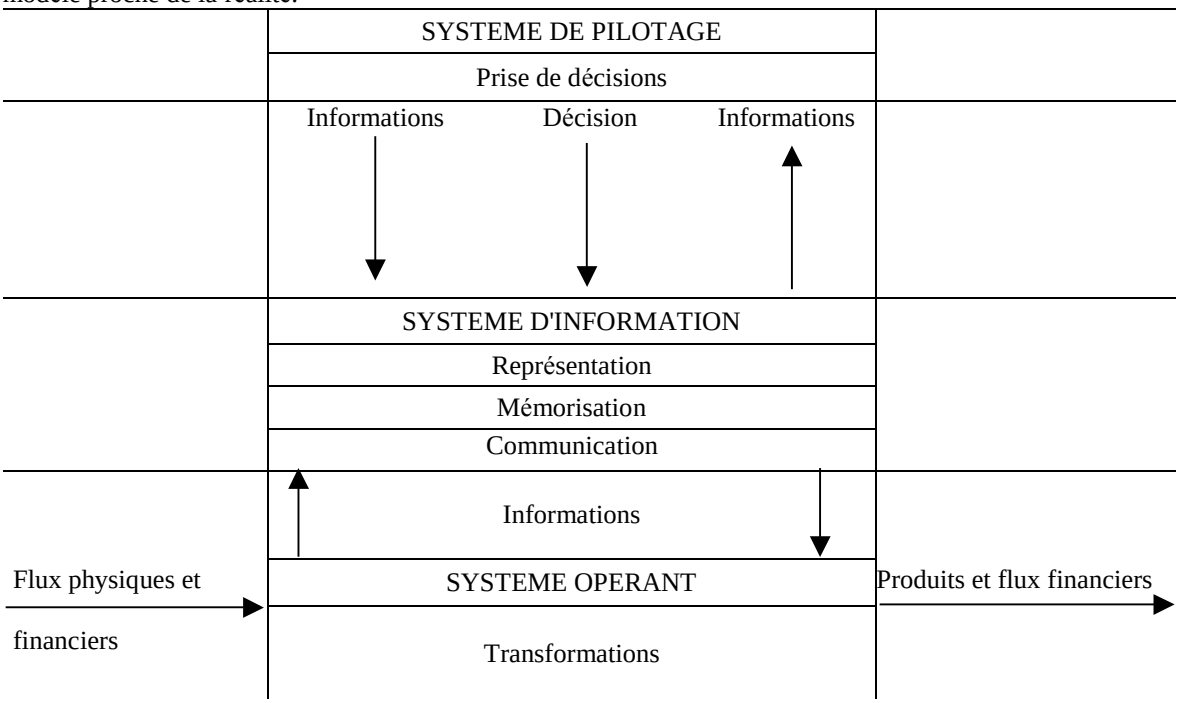
IV.2 Définition (2) :

On peut distinguer 3 sous-systèmes :

- Le système de pilotage : direction, régulation, contrôle, décisions, définition des objectifs
- Le système opérant : Réalisation des tâches
- Le système d'information : Interface entre les deux systèmes précédents. Le SI est la mémoire de l'organisation.

V Le S.I

L'approche systémique (relatif à un système dans son ensemble), appliqué au système de l'entreprise, offre un modèle proche de la réalité.



- Le système de pilotage : Contrôle et dirige l'ensemble
  - Finalité : Élaborer des règles de gestions et produire des décisions
- Le système opérant : transforme les flux d'entrées et flux de sorties
  - Finalité La transformation des flux
- Le SI transmet les ordres au système de production (ou système opérant) et renvoie au système de pilotage les informations en aval
  - Finalité :
    - § Assure une représentation d'un certain nombre d'éléments de l'entreprise
    - § Gère la mémoire collective de l'entreprise
    - § Offre des services d'accès rapide à l'information

Ex.
Une voiture
Syst. de pilotage : le chauffeur
Syst. Opérant : le moteur (flux d'entrée -- carburant, flux de sortie -- énergie)
Syst. d'Information : le tableau de bord
Le système d'information comprend les informations relatives :

- Aux flux (produits en stock, produits commander, .)
- A l'univers extérieur (clients, fournisseurs, ...)
- A l'organisation de l'entreprise (que se passe t-il entre l'enregistrement d'une commande et sa livraison ?)
- Aux contraintes légales (lois, règlement, paramètres financiers)

V.l Typologie des SI

<table>
<tr><td>Système d'information de pilotage et stratégique (politique)</td><td colspan="3">Pilotage de l'évolution (accroissement de l'entreprise Pilotage de mutation (fusions,. . .)</td></tr>
<tr><td>SI de pilotage de gestion</td><td>Système d'alerte sur
Données agrégées
Tableaux de bords
Faits graves
Contrôle de gestion (rentabilité)</td><td>Recherche d'informations :
Interrogation de bases de données</td><td>Préparation des décisions :
Prévisions
Simulations
Analyse multicritères</td></tr>
<tr><td>SI opérant</td><td colspan="2">Mémoire</td><td>Processus de traitement</td></tr>
<tr><td>SI physique</td><td colspan="3">Architecture technique : support du SI
Ressources
Processus de production du SI ; Méthode de conception du SI, outils de conception et de développement (AGL)</td></tr>
</table>

V.2 Le tout informatisé ?

- Données formalisables .les renseignements sur les clients, les fournisseurs, ...
- Données non formalisables les choix
- Traitements formalisables .édition des factures, ..
- Traitements non formalisables .les choix

On parle de système d'information automatisé (SIA). Il communique avec son environnement extérieur par des saisies et des accès.

VI Pourquoi une analyse informatique ?

Lors des premières informatisations, de nombreux problèmes furent rencontrés, entre autre.

- Logiciel ne réalisant pas ce pourquoi il était prévu
- Logiciel ne pouvant pas évoluer en fonction de .
  - Nouveaux matériels
  - Changements dans l'entreprise
  - La taille de l'entreprise
- Études correctes sur le papier mais impossibles à implémenter
- Informatisation bouleversant l'organisation humaine de l'entreprise

Alors, deux questions se posent :

- Comment réaliser un logiciel conforme au cahier des charges?
  - Avec les Ateliers de Génie Logiciel ou AGL
- Comment réaliser un cahier des charges qui décrive exactement et précisément ce que l'on attend?
  - L'analyse informatique

VII Le cycle de vie d'un S.I

Il existe plusieurs modèle de cycle de vie:

VII.1 Le modèle « code and fix »

Ce modèle date des premières informatisations. Donc il repose sur l'idée que la détermination détaillée des besoins se fait à travers de la mise au point des programmes Le schéma donnait.

COMPREHENSION DU PROBLEME		
ò		
PROGRAMMATION		
ñ	ð	ò
ñ		MISE AU POINT
ñ	ï	ò

Ce modèle utilise donc trois étapes :

- Une brève étape de compréhension du futur système
- Une étape de programmation
- Une étape de mise au point en collaboration avec l'utilisateur final

Mais ce modèle présente plusieurs problèmes

- Source importante de difficultés dans l'étape de mise au point (Répétition)
- Économie des étapes de modélisation et de conception

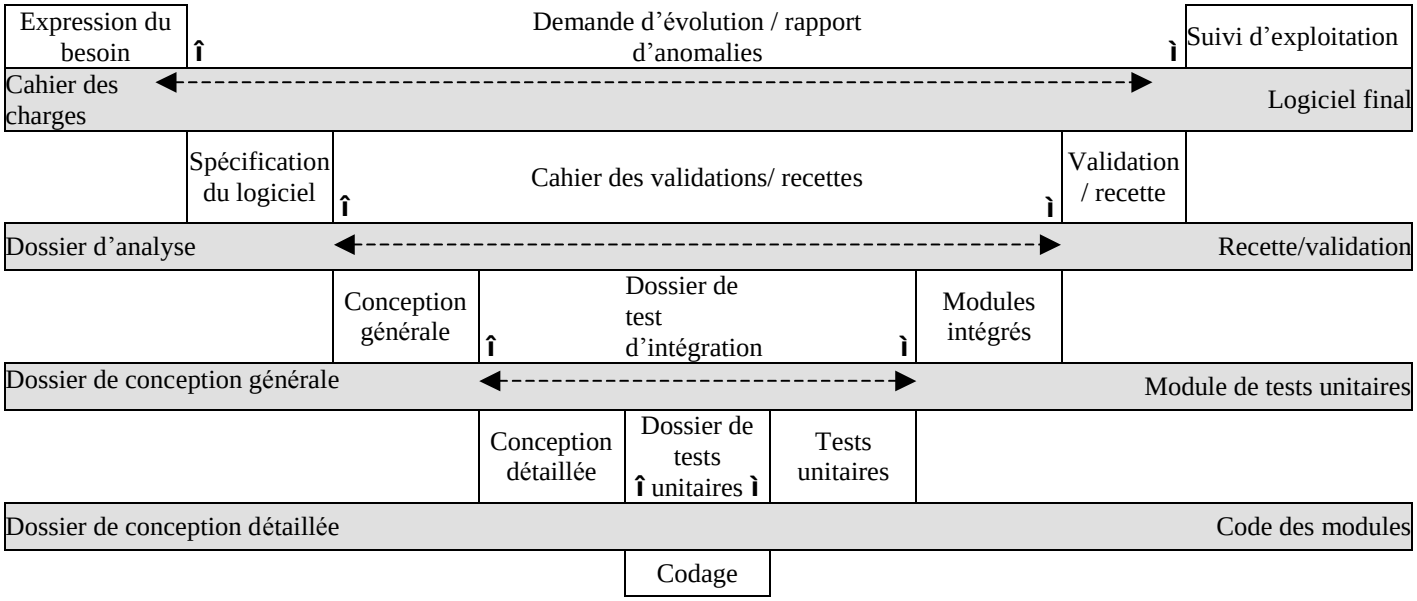
VII.2 le modèle en « cascade »

C'est une succession d'étapes uniques fixées à l'avance On ne peut passer à l'étape suivante que si le résultat du contrôle est satisfaisant C'est à dire que dans ce système, on ne remet pas en cause les étapes antérieures si une erreur est détectée, à l'exception de l'étape suivante Le modèle donne le schéma suivant

Étude de faisabilité	î					
Validation						
	Expression des besoins					
Validation						
		Définition et spécification des besoins (analyse des flux)				
			î			
Validation						
			Conception générale			
				î		
Vérification						
				Conception détaillée		
					î	
Vérification						
					Codage	
						î
Tests unitaires						
						Intégration
						î
Tests d'intégration						
						Implémentation
Recette / réception						
						Source d'exploitation

VII.3 le modèle en « V»

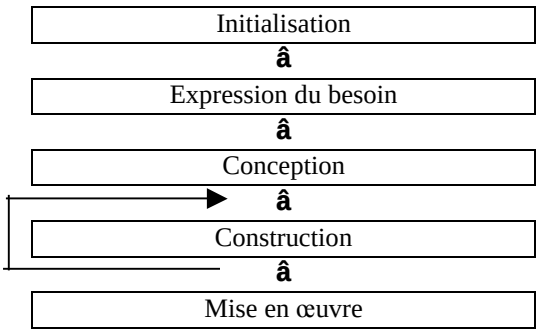
Le modèle prend en charge le plus rapidement possible une erreur et l'ensemble de ses conséquences Il part d'une conception générale et décompose systématiquement le système en sous-ensemble (découpage structurel) qui seront développés et testés séparément Son schéma se présente de la manière suivante



VII.4 Le modèle du cycle « RAD » (Rapid application development)

Ce modèle comprend un découpage en cinq étapes

- Initialisation
- Expression des besoins
- Conception
- Construction
- Mise en oeuvre



L'étape de construction se compose d'un nombre variable de cycles de prototypage La participation des utilisateurs joue un rôle fondamental ainsi le modèle comprend une structuration plus fine. Chaque phase présente une structure à trois étapes. qui sont.

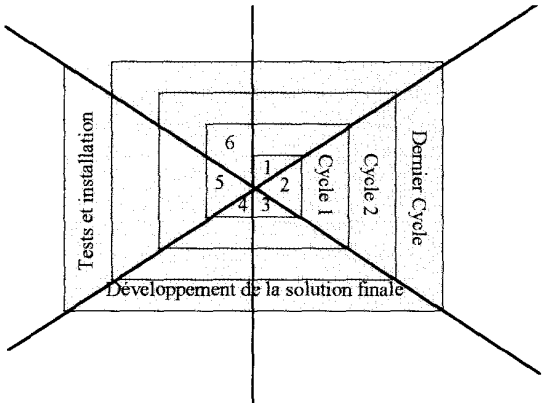
Travaux préparatoires	Session participative qui se fait avec le groupe de travail, les utilisateurs	Travaux de conclusions

VII.5 Le modèle en « spirale »

Le développement se fait à travers une succession de cycles. Chaque cycle peut être considéré comme une étape qui comporte 6 phases.

- **Analyse** du risque
- **Développement** d'un prototype

- **Simulations** et essais du prototype
- **Détermination** des besoins (à des mailles différentes) à partir des essais
- **Validation** des essais par un comité de pilotage
- **Planification** du cycle suivant



Le dernier cycle permet de développer la version finale et d'implémenter le logiciel. Sa structure est.

Les principaux facteurs de risques.

- La taille du projet
- La difficulté technique
- Le degré d'intégration
- La configuration organisationnelle
- Le changement
- L'instabilité de l'équipe de projet

Merise

B

B Merise

I Présentation générale

I.1 Principes généraux

Nous avons vu qu'avant la méthode d'analyse consistait alors à rédiger les résultats attendus (les sorties), à en déduire les traitements informatiques à appliquer aux informations nécessaires (les entrées) pour obtenir ces résultats Mais les années 70 sont marquées par trois évènements :

- Le développement informatique
- L'apparition de base de données
- La crise économique avec les différents chocs pétroliers

Ils eurent pour conséquences.

- De développer l'automatisation et donc l'informatisation des entreprises afin de diminuer les coûts en réponse à la crise économique
- De rendre plus cohérent l'utilisation des informations exploitées dans les différentes applications
- Le besoin d'informatisations capables de s'adapter sans remettre en cause l'existant

Sur ces conséquences est née la méthode Merise. Les concepts développés seront novateurs, il s'agit.

- De considérer l'entreprise et les informations nécessaires à son bon fonctionnement comme un SYSTEME
- D'abandonner définitivement l'approche par les traitements. Les données sont modélisées indépendamment des traitements ce qui amènera à l'utilisation de bases de données

Les données étant relativement stables, leur modélisation permettra de ne pas remettre en cause le S.I existant. La vocation de Merise est double ; en effet, c'est d'une part une méthode de conception du S.I mené en parallèle sur les données et traitements, et d'autre part une démarche méthodologique de développement du S.I

En tant que méthode de conception, on aura

- Une approche globale du SI (dichotomie entre les données et les traitements)
- Une description du S.I par niveau. Niveau conceptuel, Niveau logique, Niveau organisationnel et le Niveau physique
- Une description du S.I en utilisant un formalisme normalisé sur le plan international par l'ISO selon le modèle *ENTITE -RELATION*

De plus, on aura un découpage du processus de développement en 4 étapes:

- Une étude préalable
- Une étude détaillée' La description détaillée de la structure de travail sera composée.
  - D'un comité directeur (le COD!)
  - D'un groupe de projet
  - D'un comité d'utilisateurs
- La réalisation
- Et la mise en oeuvre

I.2 Description du S.I : les Quatre niveaux du cycle d'abstraction de Merise

2.1 Le niveau conceptuel

Il correspond aux finalités de l'entreprise et reflète les choix de gestion. Il s'agit de décrire le « QUOI faire» et avec « QUELLES données» ? ceci sans tenir compte ni du matériel, ni de la manière dont sera organisé le travail. Il est donc essentiellement de la compétence des gestionnaires. Les modèles utilisés sont :

- Pour le MCD qui est élaboré à l'aide de trois concepts du formalisme individuel.
  - Objet (ENTITE)

- Relation
- Propriétés

- Pour le MCT est réalisé à partir des concepts suivant.
  - Les processus
  - Les opérations (évènements et synchronisation)

2.2 Le niveau organisationnel

L'évolution permanenta de la méthode Merise, fait que le modèle organisationnel et le modèle logique des traitements, initialement confondus, tendent maintenant à être réalisés séparément. Le modèle organisationnel est en effet de la compétence des utilisateurs alors que le modèle logique, plus technique, sera de la compétence des informaticiens.

A ce niveau, les choix d'organisation (moyens, ressources) sont pris en compte:

- Le mode de fonctionnement
- La répartition géographique des données et des traitements
- La répartition homme/machine

Les modèles sont :

- Le MOD : Il permettra d'apporter des précisions sur le MCD validé. Ces précisions porteront sur la mémorisation, la quantification et la sécurité des données utilisées.
- Le MOT. Le modèle organisationnel des traitements consistera à déterminer puis à réaliser les PROCEDURES fonctionnelles. C'est procédures devront ensuite être détaillées

Cela revient à se demander QUI fait QUOI et *OU* ?

2.3 Le niveau Logique

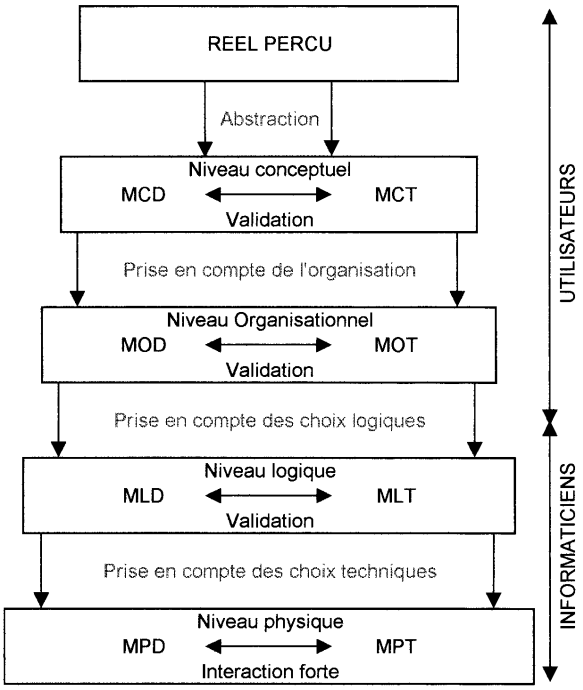
Au niveau logique, on trouvera.

- Le MLT. Le modèle logique des traitements consistera à réaliser les unités logiques des traitements et les algorithmes correspondants
- Le MLD .Ici se posera le problème de l'implémentation. C'est à dire le choix d'un logiciel s'il s'agit d'une première informatisation ou l'utilisation du logiciel existant dans le cas contraire. Le MLD devra être construit en fonction de l'outil logiciel. Il sera donc différent suivant l'implémentation adoptée.

2.4 Le niveau physique

On trouve à ce niveau :

- Le MPD .le modèle physique des données consiste à traduire le MLD dans le langage retenu. C'est donc un problème de programmation qui ne relève plus de la méthode Merise.
- Le MPT : Le MPT correspond à l'ensemble des programmes qui permettront d'assurer les traitements informatisés du S.I. Comme pour le MPD, la diversité des langages et des machines ne permet pas à la méthode Merise de définir un seul modèle. Cependant, il est important d'utiliser une méthode de programmation structurée.



I.3 Découpage du processus de développement: le cycle de vie

Merise propose un découpage en 4 étapes qui sont elles-mêmes découpé en phases

- L'étude préalable (EP) : Elle regroupe trois phases
  - Observation ou phase Recueil
  - Phase de conception ou d'organisation, c'est à dire la

conception globale ou générale de la solution
    - Une phase appréciation qui reste une des phases la plus difficile (seul l'expérience permet une maîtrise de plus en plus significative de cette phase).
- L'étude détaillée: Partie faite uniquement si l'EP n'a pas été suffisamment détaillé (par exemple sur un gros projet)
- La réalisation : Elle comporte deux phases :
    - La phase étude technique
    - La phase production du logiciel
- La mise en oeuvre comporte également deux phases:
    - La mise en place
    - La réception et le lancement qui se divisera en deux parties distinctes

Chaque étude puis phase comporte des objectifs et des résultats qui sont définis.

3.1 L'étude préalable (EP)

Objectifs à atteindre:

Faire des choix structurant pour la future application C'est à dire évaluer l'adéquation de la solution et de l'objectif.

- Prévoir, si possible, plusieurs solutions tout en présentant un attrait particulier pour tel ou tel projet avec de l'argumentation devant le CODI
- Évaluer l'investissement. C'est à dire prévoir le budget, le temps nécessaire à la réalisation du projet, les moyens,...
- Et enfin, ajuster la solution à l'enveloppe budgétaire.

Tous ces objectifs ont le point commun de fournir une base de référence pour la suite du développement.
Les résultats attendus sont:

- Une synthèse avec les grandes options retenues et les estimations
- Une description précise de la solution sur un sous-ensemble représentatif. S'il est facile de trouver le sous-ensemble, la condition représentative est plus difficile à trouver.

Entre les objectifs et les résultats de l'EP, Merise donne une découpe en trois phases afin de répondre méthodiquement à chaque partie. Nous allons donc détailler chacune de ces trois phases.

a. Phase d'observation ou de recueil d'informations (EP-PR)

La phase d'observation a pour objectif de donner une photographie pertinente du domaine étudié afin d'établir un diagnostic et de mettre en évidence les besoins. Cette phase ne doit pas être une phase d'interprétation, c'est à dire que celle-ci n'est rien d'autre qu'un glanage d'informations en laissant de coter les problèmes rencontrer par les utilisateurs.
Elle aura pour résultat:

- Le recueil de l'ensemble des documents du domaine. Les documents sont de tous types (papier, état, ...)
- D'établir une structuration du domaine en processus par une étude des flux d'information (on parle de diagramme des flux)
- De faire le choix du SER (sous-ensemble représentatif) dans le cas où le domaine étudié serait important
- De faire une description modélisée des données par le biais du MCD et pour les traitements par le biais du MOT.
- Et donc d'établir un diagnostic

b. Phase de conception (EP-PC)

L'objectif de cette phase et de composer une ou plusieurs solutions, au niveau conceptuel, organisationnel, et technique. Cela aura pour résultat :

- De définir les nouvelles orientation de gestion, d'organisation et techniques.
- D'obtenir une description des données du MCD consolidées et enrichies des l'existant (prise en compte des problèmes)
- Faire l'évaluation du volume des données à gérer
- Faire l'ébauche du MLD
- Faire la description des traitements (MCT et MOT)
- Valider les différents modèles (MCD, MCT, MOT, MOD)
- Définir la solution technique. C'est à dire de définir l'architecture technique.
- Faire l'étude des variantes possible de la solution

c. Phase appréciation (évaluation)

Les différents objectifs de cette phases auront pour but de dresser un bilan des avantages attendus et des coûts prévisibles (Étude de rentabilité) et d'élaborer un plan pour la poursuite du projet. C'est à dire identifier les sous-projets qu'il faudra ordonnancer.

Ces différents objectifs auront pour résultat :

- D'établir un bilan qualitatif de la nouvelle solution,
- D'évaluer les matériels physiques nécessaires (capacité de stockage, nombre de postes de travail, configuration technique du matériel et des postes,. ..)
- D'évaluer le logiciel à développer ou bien de faire un choix sur un progiciel(le plus souvent le cas)
- D'établir un bilan économique de la solution retenue.
- De découper et ordonnancer le projet en sous-projets qui se fait selon :
  - une éventuelle primauté stratégique de certains processus
  - De définir la périodicité
  - Et les contraintes d'approvisionnement (capacité des fournisseurs)
- D'établir le plan de développement. C'est à dire d'affecter les différentes ressources humaines pour les différentes étapes donc d'établir le planning général.
- Et enfin de rédiger le dossier d'étude préalable

3.2 L'étude détaillée

Son objectif sera de concevoir et de décrire, de façon exhaustive, la solution sur tout le champ d'étude. Elle aura pour résultat :

- De relever les interfaces homme/machine
- De décrire les traitements en une maille suffisamment fine donc l'ambiguïté fonctionnelle sera enlevée De compléter la description des données et des traitements
- D'étudier la mise en oeuvre dans son cadre général. C'est à dire d'établir les plan de formations du personnel, d'établir la documentation détaillée, de décrire le plan de démarrage, le plan d'initialisation des données, ...
- D'étudier les solution dégradées. C'est à dire de définir les solutions de secours en cas de disfonctionnement du système(mirroring, RAID, ...). Il faut envisager les cas où le système serait partiellement interrompu ou totalement (il faut prendre en compte les, délais autorisés par les utilisateurs)
- Spécifier les différentes phases de façon détaillées:
  - Description du dialogue homme/machine (écran, états, enchaînement d'écrans, ..)
  - Décrire les algorithmes de calculs et de contrôles
  - Définir les conditions de sécurité en cas d'incident (sauvegarde des données)
  - Décrire les états papiers
  - Définir le plan de développement

3.3 Réalisation

L'étape réalisation se décompose en deux phases

a. Phase technique ou étude technique

Les objectifs de cette phase seront d'optimiser les structures physiques de données, de construire les traitements (dossier du programme donc réalisation du code).

Cela aura pour résultats :

- De vérifier le respect des normes techniques (Base de données, programmation)
- De décrire l'environnement technique (type de matériel retenu, son OS, choix du SGBD, Normes du standard de développement, le plan de test et de réception)
- De décrire l'architecture du logiciel (procédures, découpage des procédures en unités de traitement)
- D'établir la description complète du MPD
- Définitions et procédures de sécurité

b. Production du logiciel

L'objectif de cette phase va être de produire un logiciel testé et surtout conforme aux spécifications du cahier des charges.

Les résultats attendus seront :

- Production du logiciel en utilisant une démarche structurée ayant pour objectif de faciliter la maintenance future

- De faire les tests unitaires à partir d'un jeu d'essais
- De faire les tests d'intégration C'est à dire de vérifier le bon fonctionnement de l'ensemble du programme
- Et d'optimiser le MPD

3.4 La mise en oeuvre

Elle comporte deux phases. La première consiste à mettre en place tous les moyens nécessaires à la réception du matériel. La seconde, subdivisée en deux sous-parties, consistera d'une part à la réception et d'autre part au lancement.

a. La mise en place

La mise en place couvre l'ensemble des préparatifs nécessaires avant la réception du matériel et au lancement du nouveau système.
Cette phase aura donc pour résultats :

- La mise en place des moyens techniques. C'est à dire, de préparer les locaux pour la configuration matériel (Câbles de liaisons, installation mobilière, ...)
- Mettre en place les moyens informatiques (Serveurs, Mainframe, réseaux, ...)
- La mise en place des moyens humains. C'est à dire, de préparer d'organiser et de réaliser les formations du personnel prévus lors de l'étude détaillée
- Et enfin, la mise en place des fichiers et de la documentation. (Reprise de l'existant ou initialisation du système ? Documentation d'utilisation du système à l'attention des utilisateurs,...)

b. Réception et lancement

Cette dernière phase a pour objectifs de réaliser les tests dans l'environnement opérationnel, de tirer un bilan du S.I et de s'y installer suivant différents critères de qualités. En réponse à ces objectifs, les résultats seront.

- Réception ou recette, livraison des logiciels et de la documentation aux utilisateurs.
- Exécution des jeux d'essais utilisateurs. C'est à dire appréciation de la conformité des résultats par rapport aux spécifications du dossier d'étude
- Lancement progressif du nouveau système en parallèle au système existant
- Et enfin arrêter l'ancien système.

I.4 description de la structure de travail: le cycle de décision ou points de contrôles

Ce cycle concerne les différentes décisions et choix qui sont effectués tout au long du cycle de vie. Comme nous l'avons déjà vu, la structure de travail comprend:

- Le CODI : Il fixe les orientations majeures et prend les décisions importantes
- Le groupe ou équipe de projet. Seule structure permanente, il comprend:
    - Le CPI ou Chef de Projet Informatique
    - Les concepteurs et réalisateurs des logiciels
    - Les représentants des utilisateurs

Ce groupe participe à l'élaboration de la solution, réalise et valide les dossiers d'études produits au cours du projet, et planifie les différentes tâches à réaliser.

- Un comité d'utilisateur

Le cycle de décision permet de fixer des objectifs et de prendre des décisions.

- EP
    - Objectif.
        - Plan de développement
        - Dossier de choix par rapport aux solutions proposées
    - Décision.
        - Approbation et mise en oeuvre
        - Choix de la solution
- ED
    - Objectifs
        - Définir les spécifications fonctionnelles détaillées
    - Décision
        - Accord par les utilisateurs des spécifications fonctionnelles

- Réalisation.
  - Objectifs.
    - Spécifications techniques
    - Développer le S.I
  - Décision
    - Accord par les utilisateurs
    - Recette provisoire de conformité par rapport à l'étude préalable
- Mise en oeuvre.
  - Objectif:
    - Le système doit être installé dans l'organisation prévue
    - Maintient du système (maintenance)
  - Décision
    - Recette définitive de la nouvelle application

II Le MCD et le MOD

II.1 Élaboration du MCD

Le but du MCD est de modéliser (formaliser) les données mémorisées du S.I On ne tient pas compte des aspects techniques, économiques, ni du problèmes du stockage de l'information, ni des problèmes d'accès aux informations et ni des conditions d'utilisation. On a deux démarche pour explorer le S.I :

- La démarche déductive (ascendante) .Cette démarche s'appuie sur une liste d'informations.
- La démarche inductive (descendante). Elle met en évidence les entités de gestion en les décrivant

1.1 La démarche déductive

Elle nécessite un recensement exhaustif de toutes les informations brassées par le S.I (on appelle ça la déstructuration). Cette phase fournit un dictionnaire des toutes les informations (dictionnaire des données). C'est le point de départ d'une recomposition du S.I, d'une structuration des ses informations en objets et relations.

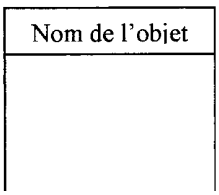
- La démarche
  - Constitution d'une liste de données (recueil des données à travers des entretiens et des documents)
    - § L'information est -elle vraiment nouvelle ?
    - § Est-ce que l'information a déjà été répertoriée sous un nom différent ou sous un même nom mais avec un sens différent ?
  - Constitution d'un dictionnaire des données où chaque données figurera avec sa description (mot clef, longueur, texte libre, règle de calcul, ...)
  - Constitution des objets et des relations entre les objets en agrégeant les propriétés pour décrire des entités de gestion (gérer en entreprise)

II.2 Les concepts de base

2.1 Le concept objet

Définition : Un objet (ou entité, individu) est la représentation d'un objet pourvu d'une existence propre et conforme aux choix de gestion de l'entreprise

Formalisme



Exemple.
Pour une université, on aura.

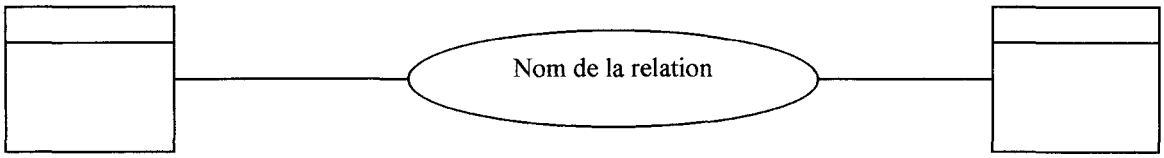
- Les enseignants
- les étudiants
- Les cours
- Les salles

- Les examens
- …

2.2 Le concept de relation

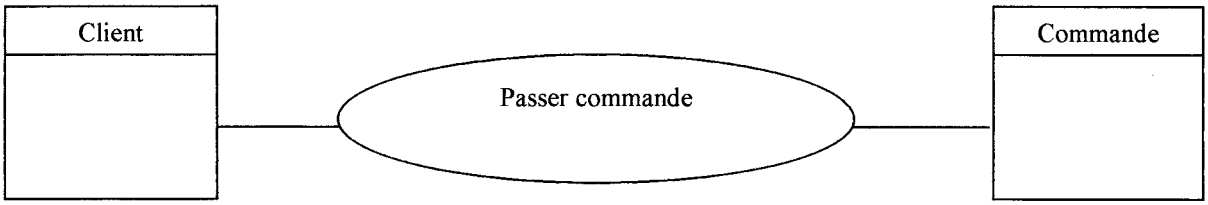
Définition : Une relation entre plusieurs occurrences traduit l'existence de liens entre les entités. C'est une association perçue dans le monde réel entre deux ou plusieurs entités

Formalisme



Exemple

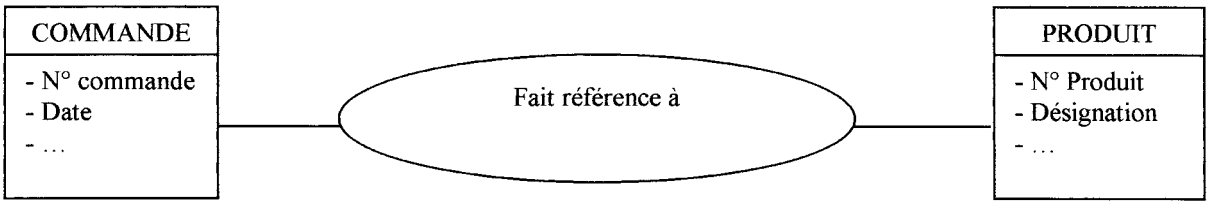
- Mariage est un lien entre deux personnes
- Le travail est un lien entre l'employé et son employeur
- Le client passe une commande



2.3 Le concept de propriété

Définition : les entités ont des propriétés ou attributs. Chaque attribut d'une entité prend une valeur parmi une variété de valeurs possibles. Une propriété est une donnée élémentaire que l'on perçoit sur un objet ou sur une relation

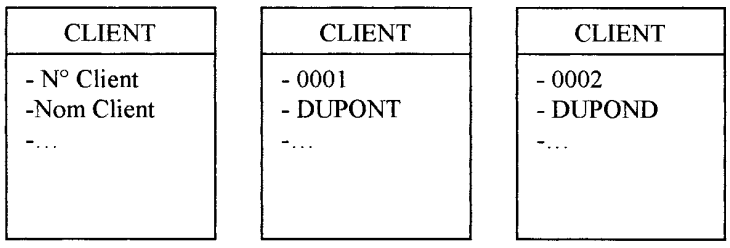
Formalisme : Le nom est inscrit à l'intérieur de l'objet ou de la relation



2.4 Le concept de cardinalité

Occurrence d'un objet : Une occurrence d'un objet est un élément particulier ou individualisé.

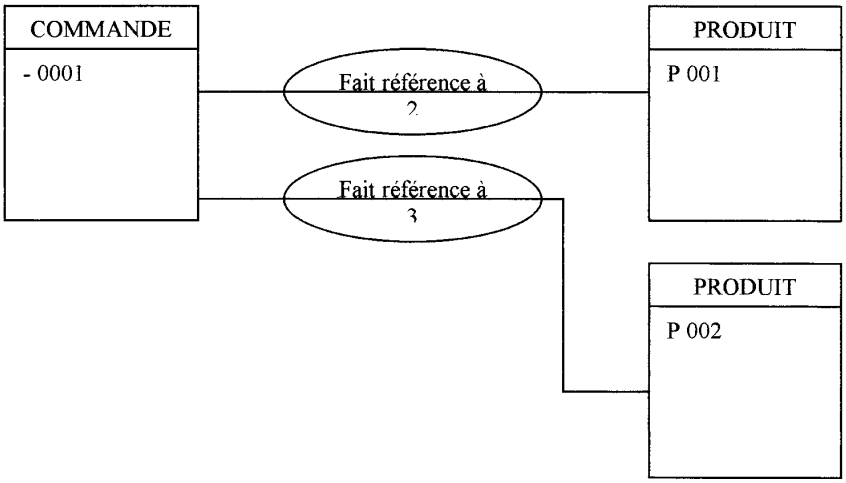
Représentation



Occurrence d'une relation

Représentation :

COMMANDE	N° PRODUIT	QTE
001	P001	2
	P002	3

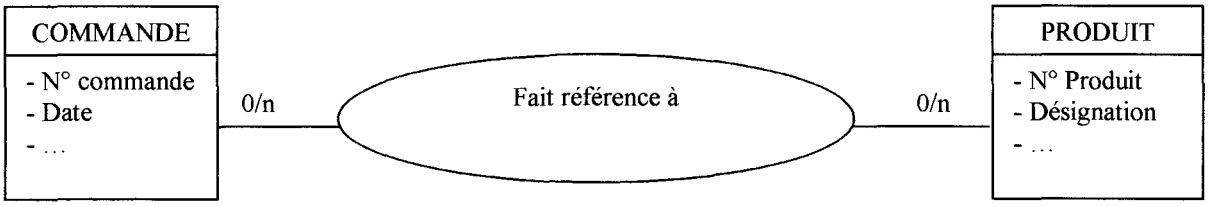


2.5 Les cardinalités

Définition : La cardinalité d'une entité par rapport à une relation s'exprime par deux nombres. On 1es appelle cardinalité minimale et cardinalité maximale.

- La cardinalité minimale est dénombrable, c'est le nombre de fois minimum qu'une occurrence d'un objet participe aux occurrences de la relation.
- La cardinalité maximale est égal à 1. ou tous nombre fixé.

Formalisme



II.3 Les règles de normalisation du MCD

- Existence d'un ID ou clef pour chaque objet
- Un ID est une propriété de l'objet telle qu'à chaque valeurs de la propriété corresponde une et une seule occurrence de 1'objet
- Un ID d'un objet est obtenu par concaténation des ID des objets participants à 1a relation
- Toutes les propriétés doivent être élémentaires, car non décomposables
- Pour chaque occurrence d'un objet, chaque propriété ne peut prendre qu'une seule valeur
- Toutes propriétés doivent dépendre de l'ID
- Une entité ne figure qu'une seu1e fois dans le MCD
- Une propriété ne figure qu'une seule fois dans le MCD

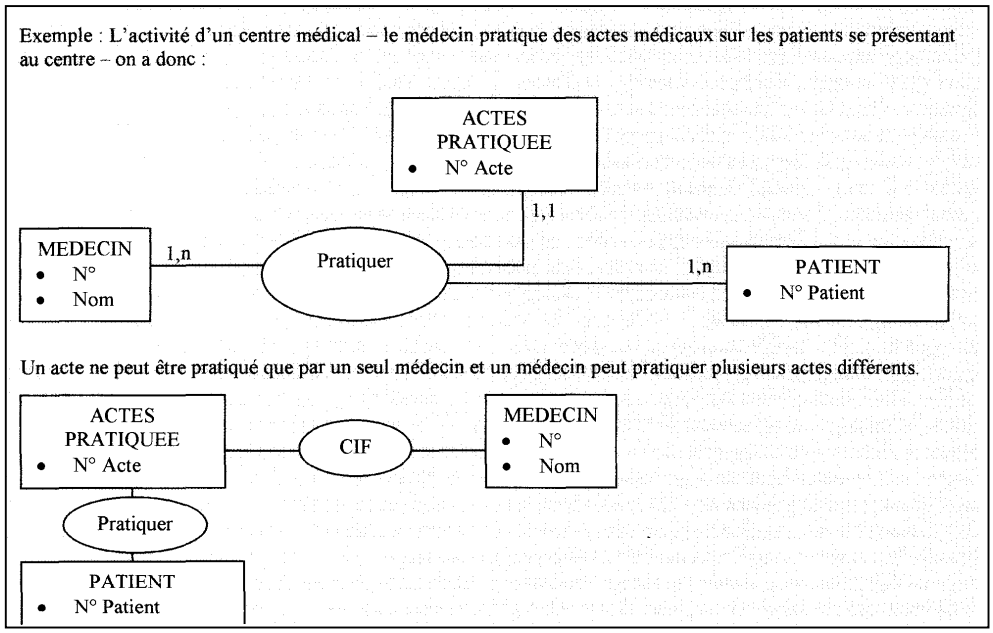
(Voir exercice en annexe)

II.4 Les contraintes d'intégrités fonctionnelles (CIF) et les dépendances fonctionnelles (DF)

Définition :

Une CIF, sur plusieurs entités participant à une même relation, exprime que l'une des entité est totalement déterminée par la connaissance d'une ou plusieurs entités. Elle traduit un lien fort et permanent de la dépendance d'une entité par rapport à une autre entité. Dans le cas ou le lien n'est pas permanent, il s'agira alors d'une DF.

L'intérêt, dans une relation de dimension supérieure à deux, est de diminuer cette même dimension (c'est à dire le nombre d'entité qui participent à 1a relation).



Les DF entre propriétés. on dit que A et B sont reliés par une DF. la connaissance d'une valeur de A une et une seule valeur de B on note a **à** df **à** b

Ex Commande client **à** df **à** Nom client

On plusieurs types de DF .

- **La DF élémentaire:**

On a une Dfe entre A et B si on a d'abord A ~ B et si aucune partie de A détermine B

Ex .code client + Nom client **à** @client

Code client **à** @ client

- **Les DF entre entités**

A **à** B si toutes les occurrences de A déterminent une et une seule occurrence de B

Ex : CLIENT N° Client	O,n —— Passer commande —— 1, l	COMMANDE N° commande

II.5 Les extensions du formalisme Entité-Relation

Les extensions traitent des concepts de généralisation/Spécialisation, d'héritage et de représentation de nouvelles contraintes.

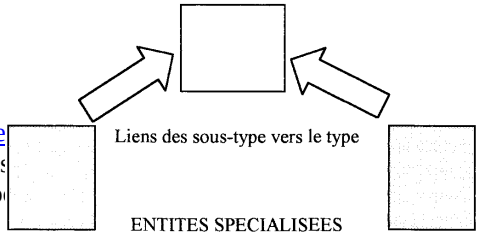
5.1 Généralisation spécialisation

déf. :

La généralisation est un processus de modélisation permettant des rassembler dans une même entité toutes les propriétés communes relatives à cette entité, vis à vis d'autres entités spécialisés (regroupant des propriétés propre à un sous-ensemble d'occurrences de l'unité générique).

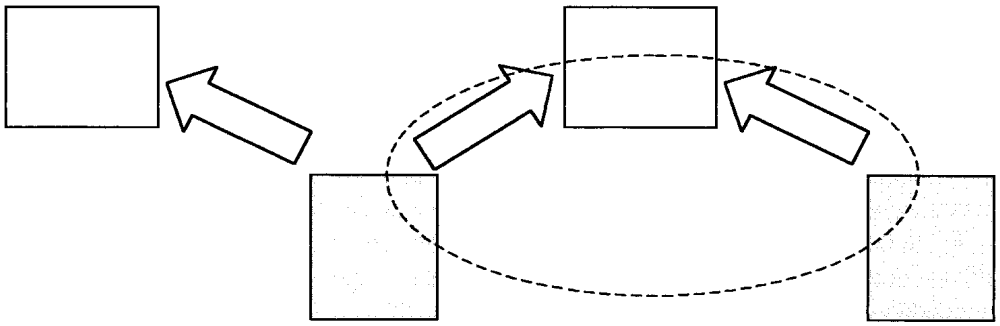
Formalisme:

- **Le formalisme SIMPLE**



Document créé par picquart.ludwig@fre gie du CNAM pour l'année universitaire 2001-2002. Ce document es ent. Aucune rémunération de quelque ordre que ce soit ne doit être asso é à être modifié.

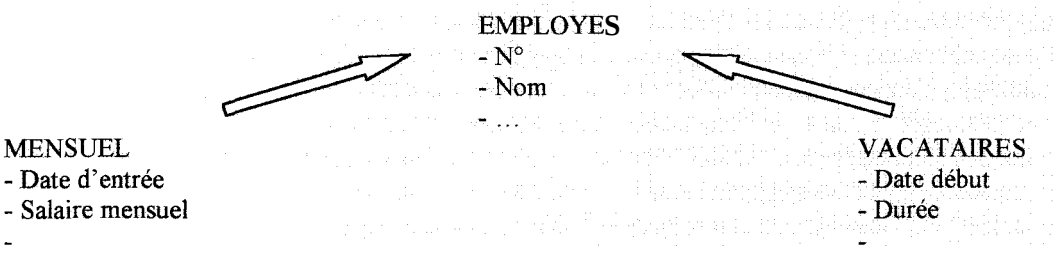
- **La généralisation multiple**



Ex ; Soit des employés d 'une entreprise, où nous prenons en compte les éléments de rémunération concernant les catégories de personnels a savoir .

- Les employés mensuels avec les informations
  - Salaires mensuels
  - Nb de jours
- Les employés vacataires
  - Début de la vacation
  - Durée de la vacation
  - Coût horaire
  - Nb jours cumuls de vacation

On a donc une table Employés puis ensuite1es spécialisations



Cette solution permet d'appliquer le concept d'héritage qui consiste à transmettre les propriétés de l'entité type (générique) vers les sous-type (spécialisés)

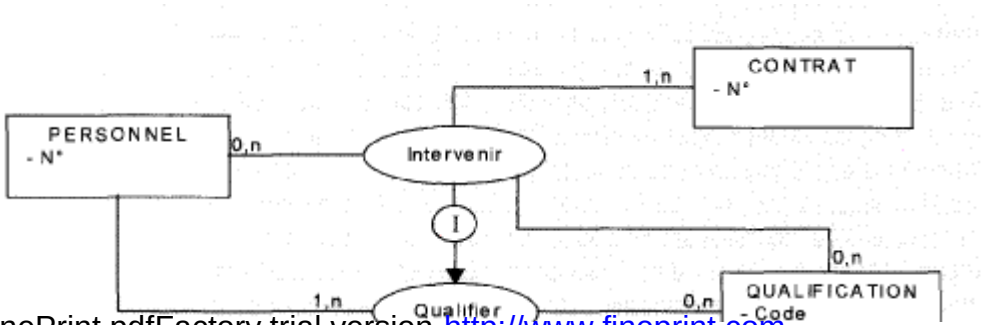
5.2 La contrainte d'inclusion

déf. :

Une contrainte d'inclusion d'une relation 1 vis à vis d'une relation 2 exprime le fait que les occurrences des entités de la relation 1 participe implicitement à la relation2.

Ex :

Soit un employé possédant plusieurs qualifications et cet employé intervient au titre d'un contrat avec une de ses



Document our l'année universitair nération de quelque orc

qualifications. On a :

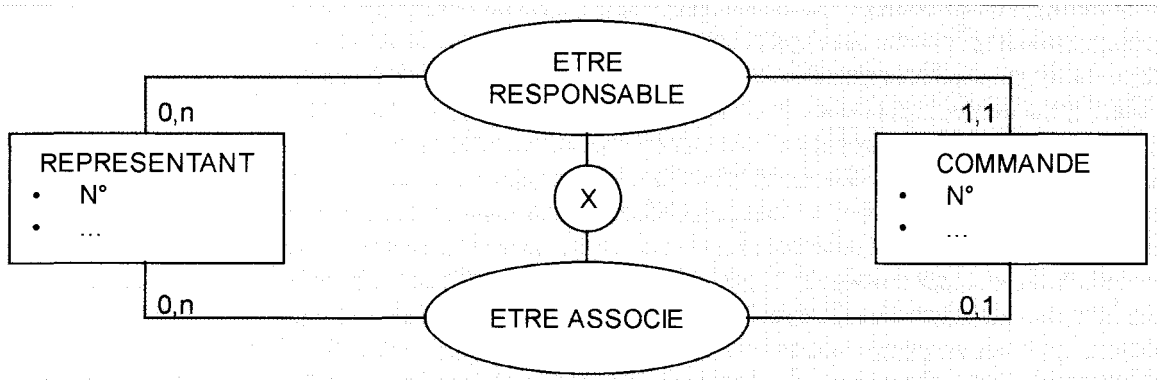
5.3 Contrainte d'exclusion

déf. :

Une relation 1 soumise à une exclusion vis-à-vis d'une relation 2 et d'une ou plusieurs entités SSI toutes les occurrences des entités concernées de la relation 1 ne participent pas à le relation 2.

Ex :

Soit un S.I de gestion commerciale où un représentant peut intervenir soit en tant que responsable soit en tant qu'associé, on a :



II.6 Élaboration du MOD

Le développement du Client/Serveur nécessite une répartition des données et des traitements entre les clients et un ou plusieurs serveurs.. le MOD se caractérise par des préoccupations spécifiques.

- Détermination des données retenues au niveau organisationnel
- Détermination des droits d'accès aux données.
  - Pour chaque donnée ou ensemble de données
  - Les droits d'accès en consultation, MàJ, doivent être définis pour chaque utilisateur
- La visibilité des données par site organisationnel (S.I organisé en plusieurs sites).
- La volumétrie des données actives (MàJ) et passives (Archivage.

Le formalisme du MOD est identique à celui du MCD

6.1 Élaboration du MOD

1. Identification des types de sites et des types d'acteurs :

   a. **Type d'acteur**s : Ensemble d'occurrence d'acteurs travaillant sur un même type d'activité (ex : le service comptable)

   b. **Type de site** : Ensemble de type d'acteurs regroupés sur un critère organisationnel et/ou fonctionnel (ex. Agence régional)

Documents types :

TYPES DE SITES	TYPES D'ACTEURS
SITE N° 1	Acteur N°1 Acteur N° 2
SITE N° 2	Acteur N° 3

2. Détermination des données à retenir au niveau organisationnel
   - Le MOD ne doit contenir que les données utilisées dans les traitements automatisées.
   - Le MOD peut contenir de nouvelles informations ou données de type organisationnel par rapport au MCD. (Ex' données ayant un lien avec le site organisationnel → @ du site, dépôt de rattachement, nombre de connexions…)
3. Détermination des droits d'accès

Les droits d'accès doivent être définis par type d'acteur et/ou par type de site.

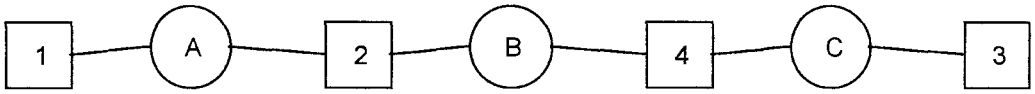
- Droit d'accès à la consultation largement autorisé sauf dans les cas spécifiques de données sensibles (salaires, données stratégiques,. ..)
- Les droit d'accès en MàJ représentent un enjeux fondamental dans la construction du SI S'assurer de l'unicité de la responsabilité de la MàJ
- Définir des droits du MàJ par ensemble homogène et cohérent de données et par type d'acteur responsable.
- Distinguer les droits d'accès à la création initiale et les droits d'accès à la modification.

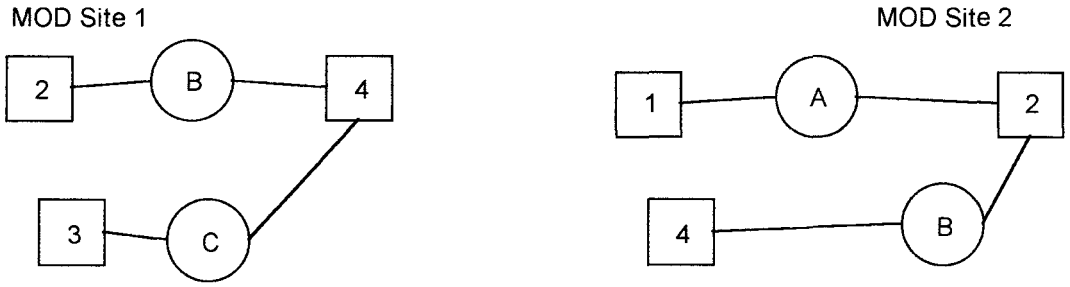
Tableau des droits d'accès

Entité/Relation	Type d'acteur 1		Type d'acteur 2	
	Consultation	MàJ	Consultation	MàJ
Entité 1	X	X	X	
Entité 2	X			X

4. La visibilité des données par site organisationnel

A partir d'un MOD général, on établit un MOD par site. Cette opération s'appelle la fragmentation.





Exercice en annexe 2

III La modélisation des flux

III.1 Le MCF (Modèle Conceptuel des Flux) :

1.1 Domaine d'activité d'une entreprise

Il faut identifier les domaines d'activité de l'entreprise, c'est à dire chaque finalité de l'entreprise. C'est un découpage de l'entreprise en domaine d'activité (Ex Domaine commercial, de la production, des ressources humaines. .)

1.2 Notion de flux

Définition:

Un flux est la représentation de l'échange d'informations entre deux activités. Entre une activité de l'entreprise et un partenaire extérieur à l'entreprise.

Caractéristiques

Un flux est caractérisé par son nom et la liste des données qui le compose (Ex. COMMANDE; N° commande, Date,.). Dans le cadre d'un S.I, seul sont décrit les flux de données.

1.3 Champ d'étude ou domaine d'application

définition.

le champ d'étude représente le domaine d'activité à étudier (Ex. Domaine de gestion commerciale est composé de la gestion des commandes, de la gestion des factures,...).

1.4 Définition du MCF

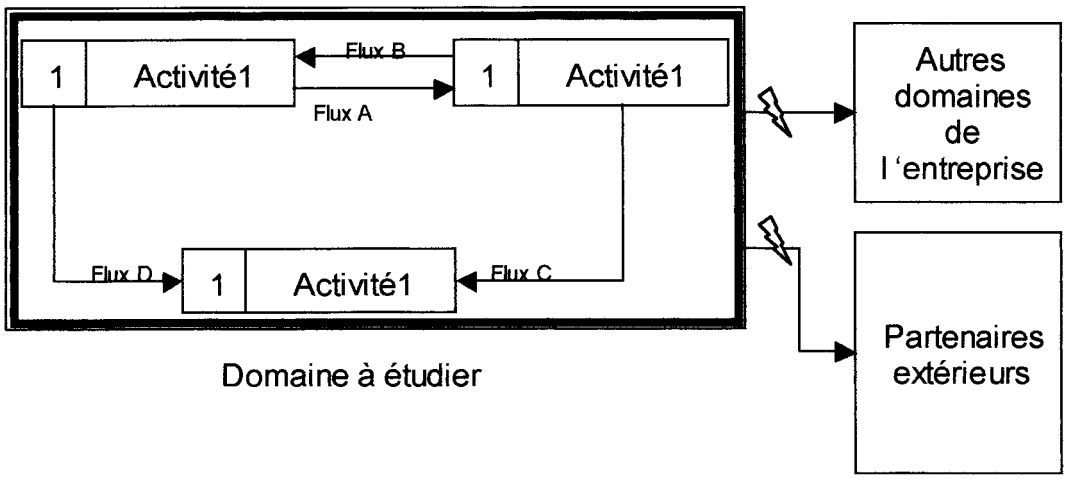
Le MCF permet.

- De représenter des flux échangés
- De délimiter précisément un champ d'étude (Attention aux interactions)

Les flux sont

- Internes au champ d'étude
- Externes au champ d'étude

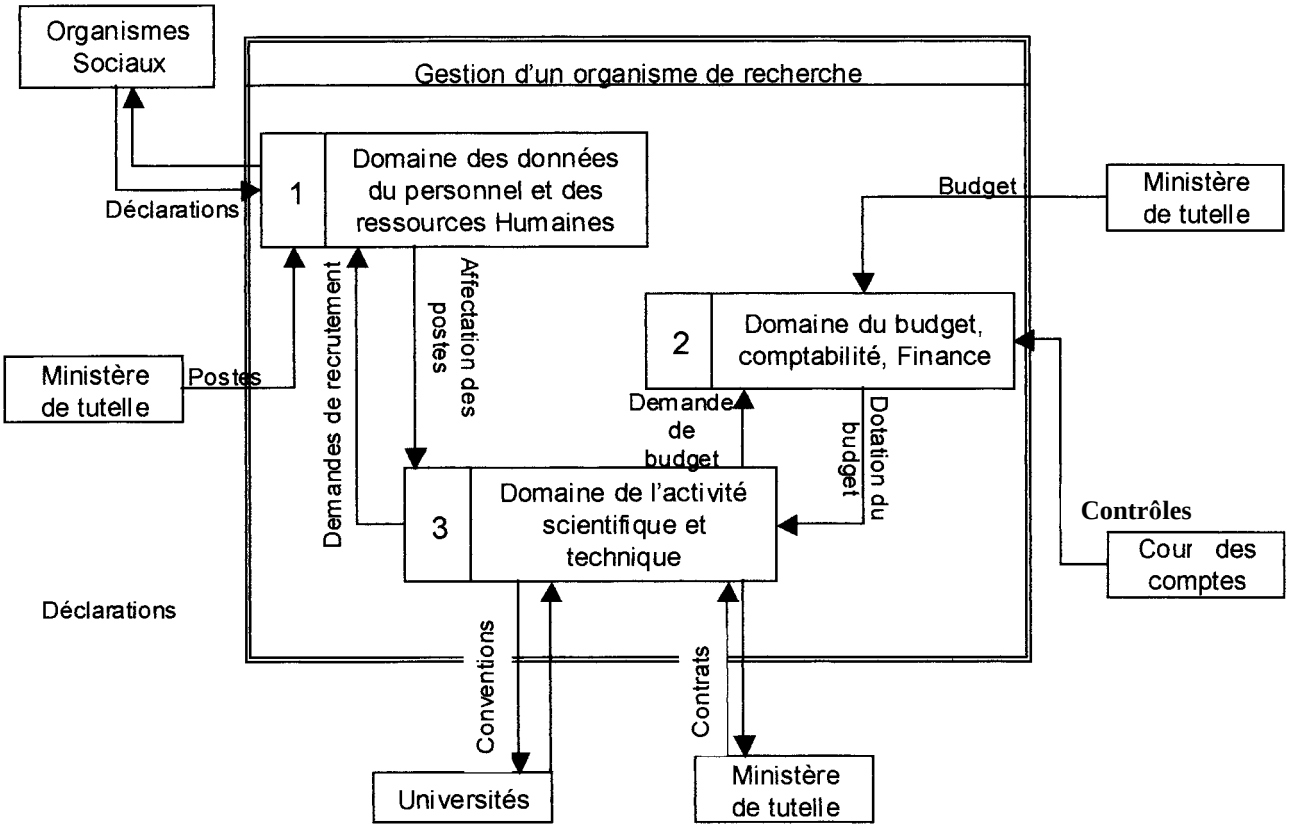
Formalisme:



1.5 Le modèle de contexte

Le modèle de contexte permet de représenter le flux échangé avec l'extérieur.

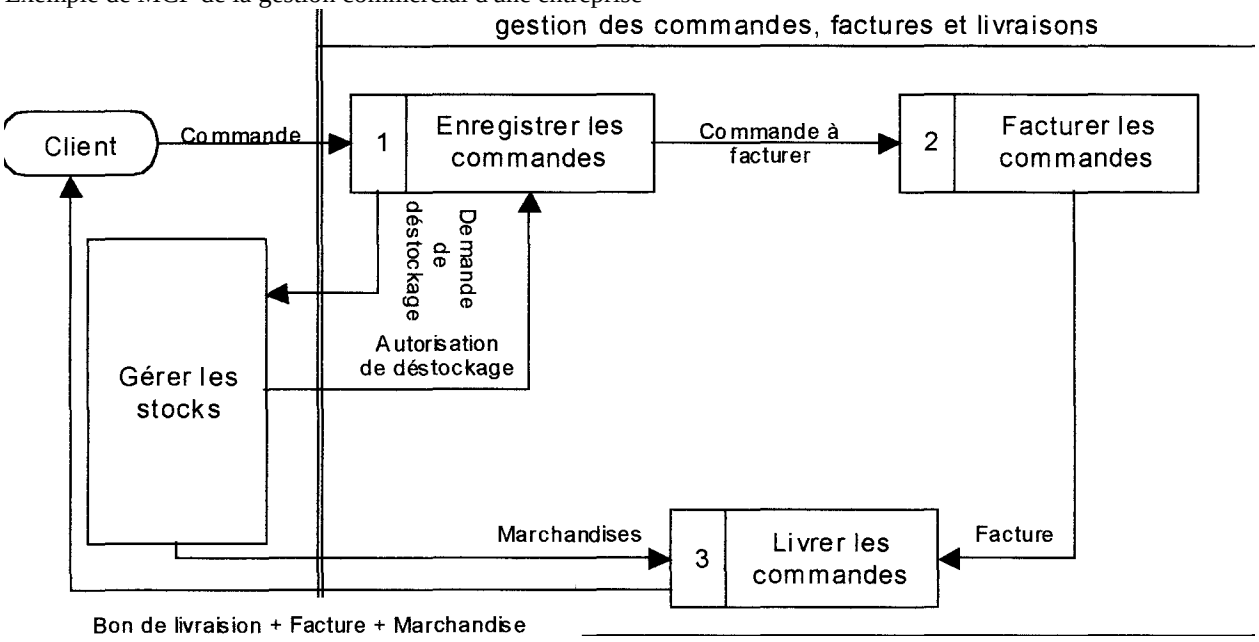
Exemple de contexte du système de gestion d'un organisme de recherche



1.6 Le modèle détaillé

Il faut représenter les flux échangés entre les activités Donc on détaille au niveau le plus fort (une opération est considérer comme une fonction) Le degrés du détail dépend de l'étude

Exemple de MCF de la gestion commercial d'une entreprise



III.2 Le MOF

2.1 Les concepts de base

Acteur:

Un acteur est une entité organisationnelle identifiable pour les missions qu'il remplit dans le cadre du champ d'étude (Internes ou Externes)

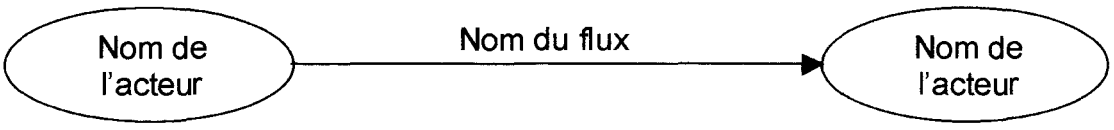
Flux:

Le support du flux peut être précisé par des symboles

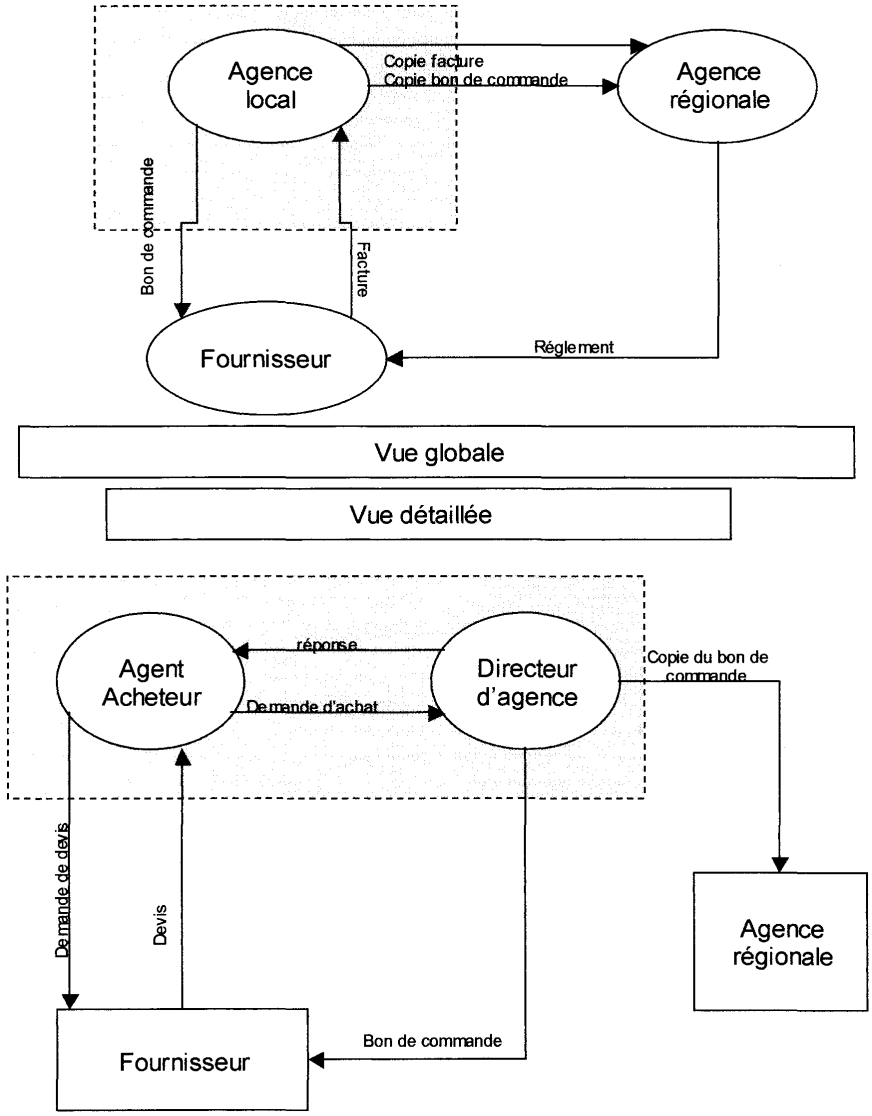
2.2 Définition du MOF ou diagramme des flux ou DFD (pour *Data Flow diagram)*

Le MOF représente les flux échangés entre les acteurs du domaine et les partenaires extérieurs.

Formalisme:



Exemple de gestion d'une agence locale (vue globale) :



2.3 Matrice des flux

la matrice des flux fait apparaître l'ensemble des flux émis par un acteur qui correspond aux lignes de la matrice et l'ensemble des flux reçus par un acteur, qui correspond aux colonnes de la matrice.
Dans notre exemple ci-dessus, on a :

La matrice suivante :

	Acteurs internes		**Acteurs externes**	
	Agent acheteur	*Directeur d'agence*	*Agence régionale*	*Fournisseur*
Agent	-	Demande d'achat	-	Demande de devis
Directeur d'agence	Réponse	-	Copie du bon de commande	-
Fournisseur	Devis	-	-	-

IV Le MCT (pour Modèle Conceptuel des Traitements)

Les traitements constituent la partie dynamique du SI. Les traitements sont la traduction, en action, des règles de gestion qui composent l'activité de l'entreprise.
Vue d'ensemble de la finalité de l'entreprise

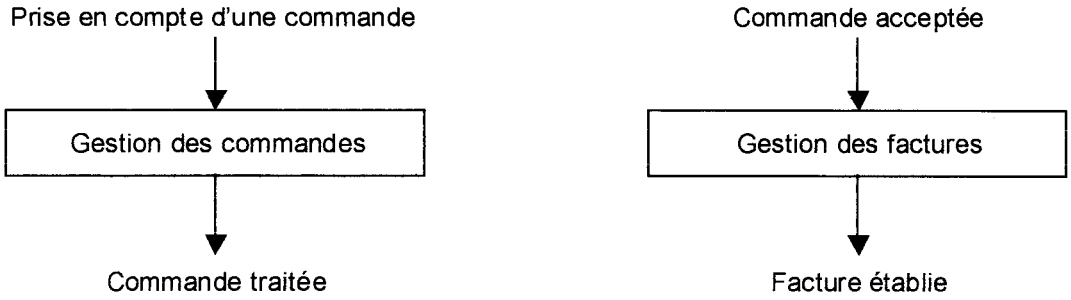
CONCEPT	OBJECTIF	PREOCCUPATION
MCT	Représenter les traitements exécutés par l'entreprise indépendamment de l'organisation (des données).	Il s'agit de répondre QUOI ? sans le QUI, ni le COMMENT

IV.l Le processus et l'opération

Le concept de processus:

Un processus constitue un sous-ensemble de l'activité de l'entreprise dont les points d'entrés et de sortis sont stables, et surtout indépendants des choix organisationnels

Ex .Activité commerciale



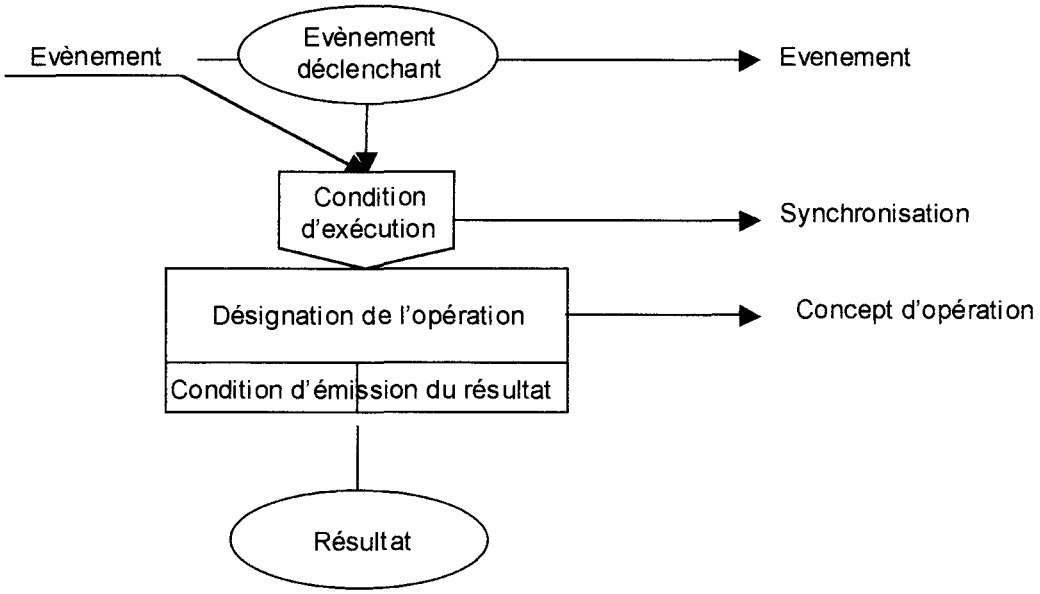
Le concept d'opération:

Une opération est constituée d'un ensemble d'actions accomplies par le S.I qui sont exécutables sans interruption en réaction d'un ou plusieurs éléments.

IV.2 Le formalisme du MCT

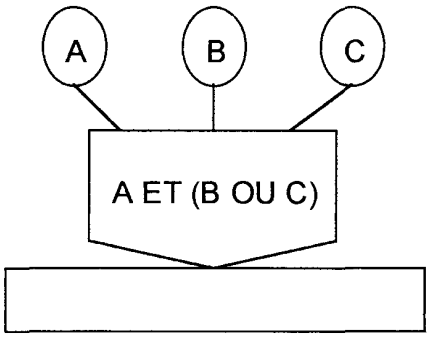
Trois formes représentent le formalisme de base du MCT .

- L'évènement
- La synchronisation
- Le concept d'opération



la synchronisation:

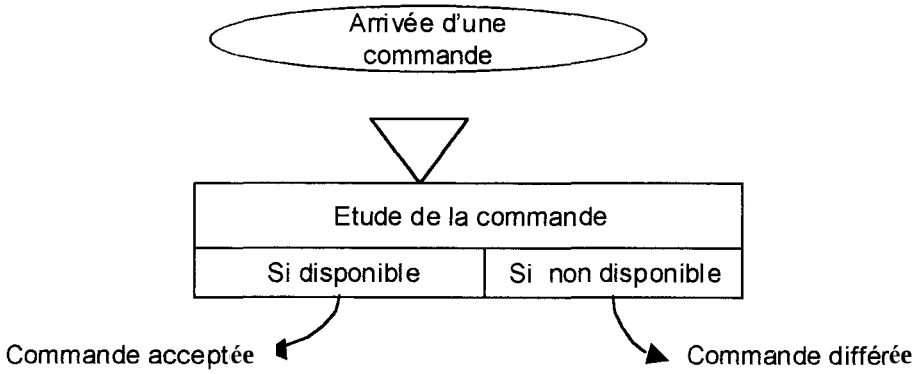
La synchronisation se sert de la conjonction « ET » et « OU », qui définissent les conditions d'exécution de l'opération.



Règles d'émission d'un résultat:

L'évènement résultat d'une opération peut être conditionnée par des règles.

Ex



2.1 Notions complémentaires

- Fréquence d'un évènement: Ceci correspond à la fréquence de survenance des occurrences de cette évènement.
- Capacité d'un évènement: Nombre maximum d'occurrence qu'un évènement peut accepter par défaut.

Quelques recommandations:

L'élaboration du MCF constitue une aide appréciable pour la construction du MCT

- Établir un MCT par processus
- Ne représenter aucun élément d'origine organisationnel ou physique
- Il est possible d'admettre la découpe en « opération fine » (suivant le niveau de détail)
- Tout évènement possède au moins un producteur interne ou externe
- Tout résultat doit avoir au moins un consommateur
- Vérifier que l'expression logique ne donne pas toujours un résultat faux.

V le MOT (Modèle Organisationnel des traitements

V.l Concept et formalisme

1.1 Concept

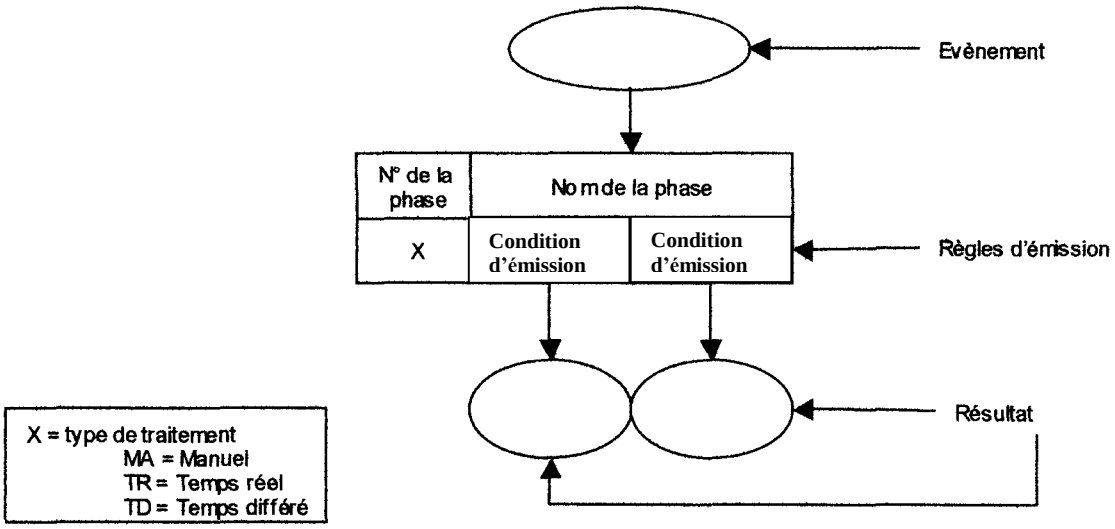
Le MOT intègre les notions de temps, de durée, de ressources, de lieux et la nature du traitement. On répond aux questions: Qui, où , quand et comment? Plusieurs préoccupations sont prises en compte:

- L'affectation des traitements aux postes de travail
- Le niveau et le type d'automatisation des traitements
  - Traitements .
    - Manuels
    - Automatisés

- § Temps réel
- § Temps différé

- Les orientations générales de l'informatisation
  - o Traitements relevant d'un serveur en informatique centrale
  - o Traitements relevant d'un serveur en informatique locale
  - o Traitements relevant d'un serveur en informatique répartie
- Le MOT prend en compte l'organisation de l'entreprise
- Dans le MOT, une procédure est composée d'une ou plusieurs phases. Une phase peut correspondre soit à un traitement, soit à une suite de traitements

1.2 Formalisme



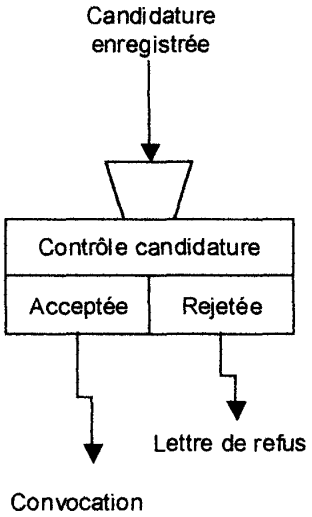
V.2 Élaboration du MOT

Le MOT doit être établi avec soins car il constitue la vision globale et cohérente des taches effectuées par les différents acteurs, de leurs échanges et des contraintes d'organisation. A chaque opération du MCT correspondra soit:

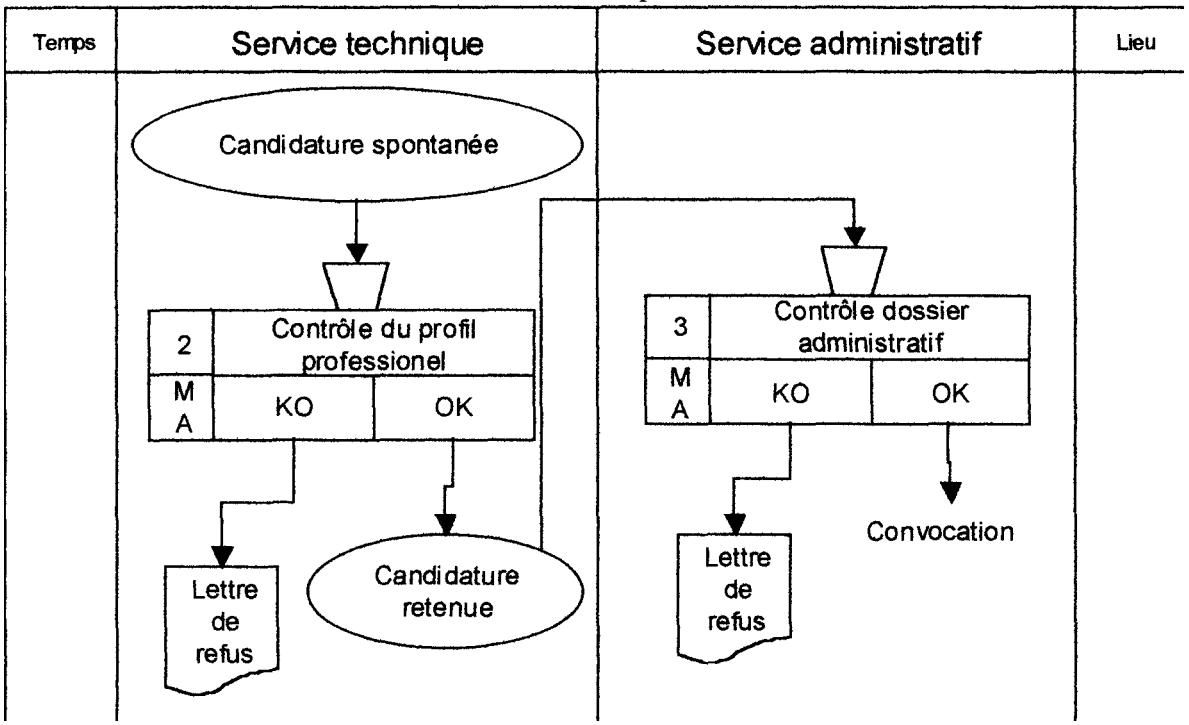
- Une phase unique dans le MOT d'une opération exécutée par un même poste
- Plusieurs phases dans le MOT, opération du MCT exécutée par plusieurs postes de travail.

Exemple: pour une candidature

MCT



MOT: Les processus



VI Synthèse de la méthode merise

VI.1 Classification des projets

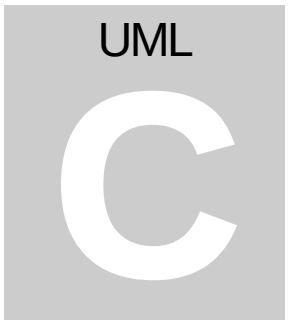
Les question à se poser sont.

- Quel modèle faut-il réaliser à chaque étape de développement?
- Quel est l'enchaînement à suivre (pour l'élaboration des modèles?
- Quelles sont les interactions des modèles aux différents niveaux?
- Quel est le niveau de description à atteindre pour chaque modèle aux différentes étapes?

Comment évaluer la taille d'un projet?

La taille du projet est données par plusieurs paramètres. On trouve.

- La classe du projet, la charge de travail, la charge de développement
  - Type 1 .
    - Très gros projet si la charge est supérieure à 100 mois/homme (m/h)
    - Gros projet si la charge de travail est comprise entre 30et 100 m/h
    - Projet moyen si la charge est comprise entre 12 et 30 m/h
  - Type 2 .
    - Petit projet si la charge de travail est comprise entre 6 et 12 m/h
    - Très petit projet si la charge de travail est inférieure à 6 m/h



C UML

I Introduction

I.1 La genèse d'UML

UML est la forme contractée de « Unified Modeling Language » qui peut se traduire par *Langage Unifié de modélisation* II fournit les fondements pour spécifier, construire, visualiser et décrire les artefacts d'un système logiciel Pour cela, UML se base sur une sémantique précise et sur une notation graphique expressive Il définit des concepts de bases et offre également des mécanismes d'extension de ces concepts

UML est un langage de modélisation objet En tant que tel, il facilite l'expression et la communication de modèles en fournissant un ensemble de symboles (la notation) et de règles qui régissent l'assemblage de ces symboles (la syntaxe et la sémantique)

La réalité des entreprises conduit à construire des S.I plus importants, plus complexes et sujets à un constant changement Pour surmonter ces difficultés, les informaticiens doivent apprendre à faire, à expliquer et à comprendre

1.1 Les méthodes d'analyse et de conception

Une méthode définit une démarche reproductible qui fournit des résultat fiables Une méthode d'élaboration de logiciel décrit comment modéliser et construire des systèmes logiciels de manière fiable et reproductible Elle définit également des règles de mise en œuvre qui décrivent l'articulation des différents points de vue, l'enchaînement des actions, l'ordonnancement des tâches et la répartition des responsabilités Ces règles définissent un processus qui assure l'harmonie au sein d'un ensemble d'éléments coopératifs et qui explique comment il convient de se servir de la méthode

1.2 L'unification des méthodes

UML 1.0 naît en janvier 1997. La version II est remise à l'OMG (Objet Management Group) en septembre 1997.

Aujourd'hui, nous sommes à la version 1.3 (2.0 est en cours) Cette version apporte de nombreux changements qu'il s'agisse de correction ou d'ajout. Des incohérences entre les concepts présentés dans les documents traitant de la sémantique et de la notation ont été levées ; des définitions et des explications ont été clarifiées

1.3 Modèle et métamodèle

Un modèle est une description abstraite d'un système ou d'un processus, une représentation simplifiée qui permet de comprendre et de simuler La modélisation n'est pas un problème à solution unique Bien souvent, le même problème analysé par des personnes différentes conduit à des modèles différents

Un métamodèle d'écrit de manière formelle les éléments de modélisation ainsi que la syntaxe et la sémantique de la notation qui permet de les manipuler Le gain d'abstraction induit par la construction d'un métamodèle facilite l'identification d'éventuelles incohérences et encourage la généricité

Le terme de **modélisation** est souvent employé comme synonyme d'**analyse** C'est à dire de **décomposition en éléments simples**, plus facile à comprendre La modélisation fonctionnelle décompose les tâches en fonctions plus simples à réaliser La modélisation objet décompose des systèmes en objets collaborants. Chaque métamodèle définit des éléments de modélisation et des règles pour la composition des ces éléments de modélisation

Un modèle est l'unité de base du développement: il est fortement cohérent avec lui-même et faiblement couplé avec les autres modèles par les liens de navigation En règle générale, un modèle est relié à une phase précise du développement

Les utilisateurs regardent et manipulent les modèles au moyen de vues graphiques, véritables projections au travers des éléments de modélisation contenus dans un ou plusieurs modèles. De nombreuses vues peuvent être construites à partir des modèles de bases. A chaque vue correspondent un ou plusieurs diagrammes. UML définit neuf types de diagrammes (5 pour les vues statiques et 4 pour des vues dynamiques) :

- **Vues statiques**
  - Les diagrammes de classes (DCL);
  - Les diagrammes d'objets (DOB);
  - Les diagrammes de cas d'utilisation;
  - Les diagrammes de composants (DCP);
  - Les diagrammes de déploiement (DPL) ;
- **Vues dynamiques**
  - Les diagrammes de séquence (DES) ;
  - Les diagrammes de collaboration (DCO) ;
  - Les diagrammes états-transitions (DET) ;
  - Les diagrammes d'activité (DAC).

1.4 Le Processus Unified (PU)

Le processus de développement doit être piloté par les cas d'utilisation. La modélisation est guidée par l'identification des séquences d'évènements qui correspondent à l'utilisation typique d'un système vu comme une boîte noire. Le PU montre que le système à construire se définit d'abord avec les utilisateurs (capter les besoins). Cette approche permet un découpage du développement par cas d'utilisation et la réception du logiciel se fera aussi par cas d'utilisation.

II L'approche objet

D'une manière générale, toute méthode de construction de logiciels doit prendre en compte l'organisation, la mise en relation et l'articulation de structures pour en faire émerger un comportement macroscopique complexe du système à réaliser. Son étude doit donc nécessairement prendre en considération l'agencement collectif des parties, et conduire à une vue globale des éléments qui le composent. Elle doit progresser à différents niveaux d'abstraction et, par effet de zoom, s'intéresser aussi bien aux détails qu'à l'ordonnancement de l'ensemble.

La construction d'un logiciel est une suite d'itérations (répétitions) du genre « division-réunion »; il faut décomposer –division- pour comprendre et il faut composer –diviser- pour construire. Cela conduit à une situation assez paradoxale, il faut diviser pour réunir

Face à ce paradoxe, le processus de décomposition a été dirigé traditionnellement par un critère fonctionnel. Les fonctions du système sont identifiées, puis décomposées en sous fonctions, et cela récursivement jusqu'à l'obtention d'éléments simples, directement représentable dans les langages de programmation (par les fonctions et procédures).

L'approche objet repose à la fois sur le rationalisme d'une démarche cartésienne et sur une démarche systémique qui considère un système comme une totalité organisée, dont les éléments solidaires ne peuvent être définis que les uns par rapport aux autres.

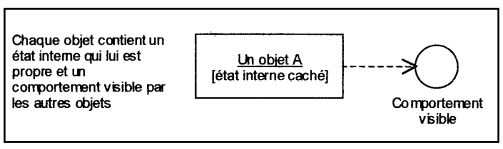
L'approche objet tire sa force de sa capacité à regrouper ce qui a été séparé, à construire le complexe à partir de l'élémentaire, et surtout à intégrer statiquement et dynamiquement les constituants d'un système.

Une environnement de développement ou langage de programmation est dit « objet » si :

- Il fournit un support pour l'encapsulation
- Il permet l'utilisation efficace des concepts d'héritage
- Il supporte le polymorphisme

- Les langages modernes offrent :
  - Un support pour les interfaces;
  - L'agrégation et paquetage (Regroupement de plusieurs objets) ;
  - Le support pour les attributs.

II.1 Les objets

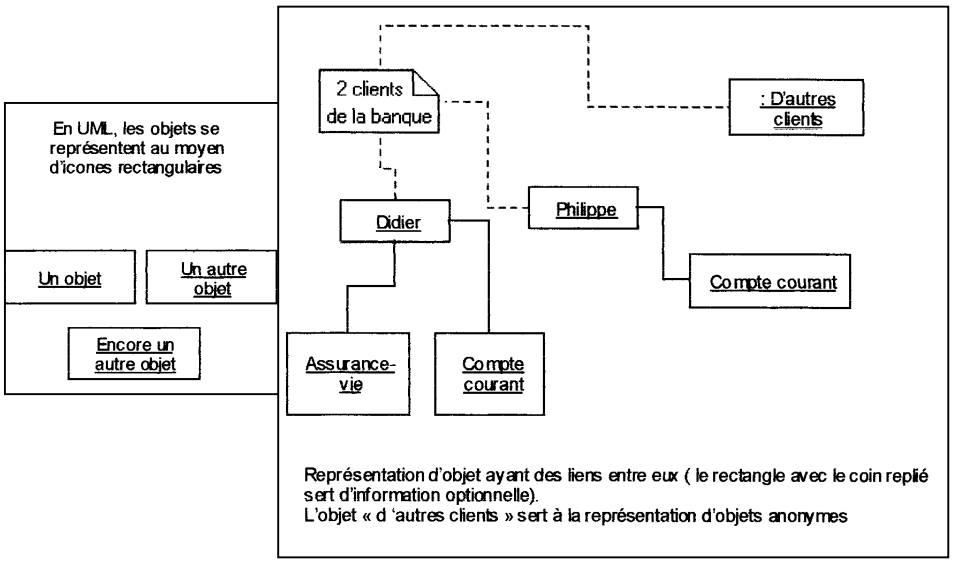


L'objet est une unité atomique formée de l'union d'un état et d'un comportement. Il fournit une relation d'encapsulation qui assure à la fois une cohésion interne très forte et un faible couplage entre le dit objet et l'extérieur. L'objet révèle son vrai rôle et sa

vraie responsabilité lorsque, par l'intermédiaire de l'envoi de message, il s'insère dans un scénario de communication. Le monde dans lequel nous vivons est constitué d'objets matériels de toutes sortes. Par extension, il est possible de définir d'autres objets, sans masse, comme les comptes en banque, les polices d'assurance... Ces objets correspondent plutôt à des concepts plutôt qu'à des entités physiques. Toujours par extension, les objets peuvent appartenir à des mondes virtuels, par exemple, des communautés de personnes qui ne sont pas situées au même point géographique et qui sont reliées par Internet.

Les objets informatiques définissent une représentation abstraite des entités d'un monde réel ou virtuel, dans le but de les piloter ou de les simuler Comme les êtres vivants, les objets du monde réel qui nous entourent naissent, vivent et meurent; la modélisation objet permet de représenter le cycle de vie des objets au travers de leurs interactions.

Formalisme



1.1 Caractéristiques fondamentales d'objet

Tout objet présente les trois caractéristiques suivantes: un état, un comportement et une identité.

OBJET = ETAT + COMPORTEMENT + IDENTITE

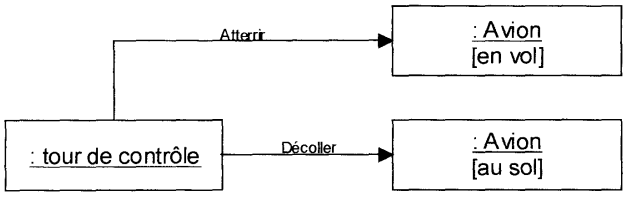
L'état;

L'état regroupe les valeurs instantanées de tous les attributs d'un objets (c'est à dire ses propriétés) sachant qu'un attribut est une information qualifiant l'objet qui le contient. Chaque attribut peut prendre une valeur dans un domaine de définition donné. L'état d'un objet, à un instant donné, correspond à une sélection de valeurs, parmi toutes les valeurs possibles des différents attributs.

Cet état évolue au cours du temps; ainsi, lorsqu'une voiture roule, la quantité de carburant diminue, les roues s'usent. Certaines composantes de l'état peuvent être constantes, c'est le cas de la marque,…Toutefois, en règle générale, l'état d'un objet est variable et peut être vu comme la conséquence de ses comportements passés.

Le comportement

Le comportement regroupe toutes les compétences d'un objet et décrit las actions et les réactions de cette objet. Chaque atome de comportement et appelé opération (ou méthode). Les opérations sont déclenchées suite à une simulation externe, représenté sous la forme d'un message envoyé par un autre objet



L'état et le comportement sont liés; en effet, le comportement à un instant donné dépend de l'état courant, et l'état peut être modifié par le comportement Il est possible de faire atterrir un avion à la condition qu'il soit en train de voler; en d'autre termes, le comportement atterrir n'est valide que si l'information en vol est valide. Après l'atterrissage, l'information en vol devient invalide et l'opération atterrir n'a plus de sens

L'identité

En plus de son état, un objet comporte une identité qui caractérise son existence propre L'identité permet de distinguer tout objet de façon non ambiguë, et cela indépendamment de son état. Ainsi, il est possible de distinguer deux objets dont toutes les valeurs d'attributs sont identiques L'identité est un concept ; elle ne se présente pas de manière spécifique en modélisation. Chaque objet possède une identité attribuée de manière implicite à la création de l'objet et jamais modifiée

En phase de réalisation, l'identité est souvent construite à partir d'un identifiant issu naturellement du domaine du problème. Ces identifiant uniques sont appelés clef naturelle et peuvent être rajoutés dans l'état des objets afin de les distinguer.

1.2 Contraintes de réalisation

Les notions d'état, de comportement et d'identité s'appliquent aux objets de manière très générale, quel que soit l'environnement de réalisation. Toutefois, les objets peuvent également posséder des caractéristiques plus informatiques, liées aux contingences de réalisation, comme la distribution de programmes, les bases de données ou les développements multi langages.

La persistance des objets

La persistance désigne la capacité d'un objet à transcender le temps ou l'espace. Un objet persistant sauvegarde son état dans un système de stockage permanent, de sorte qu'il est possible d'arrêter le processus qui l'a créé sans perdre l'information représentée par l'objet (passivation de l'objet). Par la suite, l'objet peut-être reconstruit (activation de l'objet) par un processus et se comportera exactement comme dans le processus initial, Les objets non persistants sont dits transitoires ou éphémères. Par défaut, les objets ne sont pas considérés comme persistants.

II.2 Les classes

Le monde réel est constitué de très nombreux objets en interaction. Ces objets sont des amalgames souvent trop complexes pour être compris du premier coup dans leur intégralité. Pour réduire cette complexité, ou du moins pour la maîtriser, et comprendre ainsi le monde qui l'entoure, l'être humain a appris à regrouper les éléments qui se ressemblent et à distinguer des structures de plus haut niveau d'abstraction, débarrassées de détails inutiles

2.1 La démarche d'abstraction

L'abstraction est une faculté des êtres humains qui consiste à concentrer la réflexion et l'attention sur un élément d'une représentation ou d'une notion en négligeant tous les autres. La démarche d'abstraction procède de l'identification des caractéristiques communes à un ensemble d'éléments, puis de la description condensée des ces caractéristiques dans ce qu'il est convenu d'appeler une classe.

La classe décrit le domaine de définition d'un ensemble d'objets. Chaque objet appartient à une classe. Les généralités sont contenues dans la classe et les particularité sont contenues dans les objets. Les objets informatiques sont construits à partir de la classe, par un processus appelé instanciation. De ce fait, tout objet est une instance de classe.

Une classe est l'abstraction d'un ensemble d'objet qui possède une structure identique (Liste des attributs) et un même comportement (liste des fonctions).

2.2 Les attributs et les opérations

Les attributs d'une classe correspondent aux propriétés de la classe. Ils sont définis par un nom, un type et éventuellement une valeur initiale. Chaque objet, instance d'une classe, donne des valeurs particulières à tous les attributs définis dans sa classe et fixe par-là même son état.

La spécification du comportement d'un objet est définie par les opération décrites dans sa classe. La réalisation du comportement est exprimée dans les méthodes. UML distingue clairement ces deux termes. **Une opération est un service offert par les instances de la classe alors qu'une méthode est une implémentation d'une opération**.

Il existe cinq principales catégories d'opérations :

- Les constructeurs qui créent les objets;
- Les destructeurs qui détruisent les objets;
- Les sélecteurs (ou opération de consultation) qui renvoient tout ou partie de l'état d'un objet;
- Les modificateurs qui changent tout ou 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.

2.3 Description des classes

La description des classes est séparée en deux parties :

- **La spécification** d'une classe qui décrit le domaine de définition et les propriétés des instances de cette classe, qui correspond à la notion de **type.**
- **La réalisation** qui décrit comment la spécification est réalisée et qui contient le corps des opérations et les données nécessaires à leur fonctionnement.

La séparation entre la spécification et la réalisation des classes participe à l'élévation du niveau d'abstraction. Les traits remarquables sont décrits dans les spécifications alors que les détails sont confinés dans les réalisations. Le regroupement de la description de la structure des attributs (propriétés) et de la description des fonctions (opérations) dans une même classe porte le nom **d'encapsulation**. Les détails de l'implémentation d'un objet sont masqués aux autres objets du système.

Par défaut, les valeurs d'attributs d'un objet sont encapsulés dans l'objet et ne peuvent pas être manipulées directement par les autres objets. Toutes les interactions entre objets sont effectuées suite au déclenchement de diverses opérations déclarées dans la spécification de la classe et accessibles depuis les autres objets. Les règles de visibilité viennent compléter ou préciser la notion d'encapsulation. Il existe trois niveaux distincts d'encapsulation couramment rencontrés.

- **Public (+)** .C'est le niveau le plus faible, qui est obtenu en plaçant les attributs et/ou les opérations dans la partie publique de la classe. Cela revient à se passer de la notion d'encapsulation et à rendre visibles les attributs pour toutes les classes.
- **Protégé (#)** .Il est possible de relâcher légèrement le niveau d'encapsulation, en plaçant certains attributs dans la partie protégée de la classe; Ces attributs sont alors visibles à la fois pour les amis et les classes dérivées de la classe fournisseur; Pour toutes les autres classes, les attributs restent invisibles.
- **Privé (-)** .C'est le niveau le plus fort; la partie privée de la classe est alors totalement opaque et seuls les amis peuvent accéder aux attributs placés dans la partie privée.

L'ensemble des éléments (attributs et opérations) d'une classe rendus visibles aux autres porte le nom **d'interface.** L'interface de la classe correspond à l'ensemble des méthodes publiques de la classe.

II.3 Les relations entre classes

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

3.1 L'association

L'association exprime une connexion sémantique bidirectionnelle entre classes. Elle représente une relation entre plusieurs classes. Une association est une abstraction des liens qui existent entre les objets instance des classes associées.

3.2 L'agrégation

Une relation exprime une forme de couplage entre abstractions. La force de ce couplage dépend de la nature de la relation dans le domaine du problème. Par défaut, l'association exprime une relation à couplage faible, les classes associées restant relativement indépendantes l'une de l'autre. L'agrégation est une forme particulière d'associations qui exprime un couplage fort entre classes. Une des classes joue un rôle plus important que l'autre dans la relation. L'agrégation permet d'exprimer des relations de type Composant/Composé et structure d'organigramme

II.4 Les hiérarchies de classes

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

4.1 Généralisation et spécialisation

La généralisation et la spécialisation sont des points de vus portés sur les hiérarchies de classes. La généralisation consiste à factoriser les éléments communs (attributs, opérations, et contraintes) d'un ensemble de classes dans une classe plus générale appelé **Superclasse ou classe de haut niveau.**

Les classes sont ordonnées selon une hiérarchie ; une superclasse est une abstraction de ses sous-classes. La généralisation est une démarche assez difficile car elle demande une bonne capacité d'abstraction. La mise au point d'une hiérarchie optimale est délicate et itérative.

La spécialisation permet de capturer les particularités d'une ensemble d'objets par les classes déjà identifiées. Les

nouvelles caractéristiques sont représentées par une nouvelle classe et devient une sous-classe d'une des classes existantes. La spécialisation est une technique efficace pour l'extension cohérente d'un ensemble de classes.

La généralisation et la spécialisation sont deux points de vue antagonistes du concept de classification; elles expriment dans quel sens une hiérarchie de classes est exploitée; Dans toute application réelle, les deux points de vue sont mis en œuvre simultanément.

La généralisation est plutôt employée une fois que les éléments du domaine ont été identifiés, afin de dégager une description détachée des solutions. La spécialisation est à la base de la programmation par extension et de la réutilisation.

4.2 L'héritage

Il existe de nombreuses manières de réaliser la classification. En programmation objet, la technique la plus utilisée repose sur l'héritage entre classes.

L 'héritage est une technique offerte par les langages de programmation pour construire une classe à partir d'une ou plusieurs autres classes, en partageant des attributs, des opérations, et parfois, des contraintes au sein d'une hiérarchie de classes. Une classe est déclarée comme héritant d'une autre classe si elle peut activer toutes les méthodes de cette dernière et si elle contient tous ses attributs.

4.3 Le polymorphisme

Le terme polymorphisme décrit la caractéristique d'un élément qui peut prendre plusieurs formes, comme l'eau qui se trouve à l'état liquide, solide ou gazeux. En informatique, le polymorphisme désigne un concept de la théorie des types selon lequel un objet peut désigner des instances de classes différentes issues d'une même arborescence.

Le polymorphisme est la capacité donnée à une fonction de s'exécuter différemment suivant le contexte de la classe où elle se trouve. Une opération définie dans une superclasse peut s'exécuter d'une manière différente selon la sous-classe

II.5 L'Unified Processus (PU) (P .263)

5.1 Les principes

Le processus de développement doit être piloté par les cas d'utilisation. La modélisation est guidée par l'identification des séquences d'événements qui correspondent à l'utilisation typique d'un système vu comme une boîte noire. Le PU montre que le système à construire se définit d'abord avec les utilisateurs (capter les besoins). Cette approche permet un découpage du développement par cas d'utilisation ; la réception du logiciel se fera aussi par cas d'utilisation.

5.2 Les processus itératifs et incrémentais

On développe les systèmes par incrémentation (implémentation d'un sous-système de cas d'utilisation initiaux pour les versions de base) Le développement progressif et incrémental est recommandé en s'appuyant sur la décomposition du système en cas d'utilisation.

Le processus itératif repose sur une l'idée simple. Lorsqu'un système est trop complexe pour être compris, conçu ou réalisé du premier coup, voire les trois à la fois, il vaut mieux procéder en plusieurs fois par évolution.

5.3 Processus centré sur l'architecture

Le développement d'un logiciel peut être vu comme la traversée d'un océan en bateau. Le jour de départ est connu, le jour d'arrivée l'est moins; en cours de route, il faudra affronter des tempête et réparer des avaries.

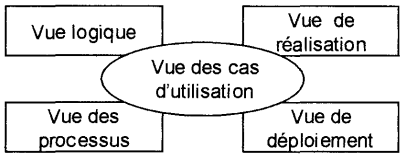
La conception de l'architecture logicielle a pour objet de donner à un effort de développement les moyens de parvenir à bon port L'architecture logicielle se préoccupe d'intégrité, d'uniformité, de simplicité et d'esthétisme. Ca mise en place est la clé de voûte du succès d'un développement. Il importe donc de la stabiliser le plus tôt possible.

Le PU met en avant la préoccupation de l'architecture technique du système dès le début des travaux d'analyse. Il est important de décrire une architecture type qui sera retenue pour le développement, l'implémentation et pour le déploiement du système. Les bonnes architectures se caractérisent par:

- Leur simplicité;
- Leur élégance;
- Leur intelligibilité
- Des niveaux d'abstraction bien définis;
- Une séparation claire entre l'interface et l'implémentation de chaque niveau.

a) La vision de l'architecte

Il n'existe pas une seule manière de considérer un système. De multiples points de vue sont requis Par ailleurs, pour satisfaire les multiples intervenants, chaque type de diagramme ne donne qu'une image partielle d'un système. Cette vision peut s'exprimer à travers plusieurs vues complémentaires, inspirées du modèle dit des« 4 + 1 vues ».



Ce modèle d'architecture est bien adapté à la présentation des contraintes variées que l'on doit prendre en considération.

La vue logique

La vue logique décrit les aspects statiques et dynamiques d'un système en termes de classes et d'objets et se concentre sur l'abstraction, l'encapsulation et l'uniformité. Elle met en jeu les éléments de modélisation suivants :

- Les objets;
- Les classes;
- Les collaborations;
- Les interactions.

La vue de réalisation

La vue de réalisation se préoccupe de l'organisation des modules statiques (les exécutables, code source...) dans l'environnement de développement. Elle montre l'allocation des classes dans les modules et l'allocation des modules dans les sous-systèmes. Elle s'intéresse également à la gestion de configuration (auteurs, version.)

La vue des processus

La vue des processus représente la décomposition en flots d'exécution (processus, threads, tâches...), la synchronisation des flots et l'allocation des objets et des classes au seins des différents flots. Elle se préoccupe également de la disponibilité du système, de la fiabilité des applications et des performances. La vue des processus prend toute son importance dans les environnements multitâches. La vue des processus manipule les éléments de modélisation suivants :

- Les objets actifs (les threads et les processus) ;
- Les interactions

La vue de déploiement

La vue de déploiement décrit les différentes ressources matérielles et l'implémentation du logiciel dans ces ressources. Elle traite les points suivants :

- Les temps de réponse et les performances du système;
- La bande passante des chemins de communication et les capacités;
- Les contraintes géographiques (disposition physique des processeurs) ;
- Les besoins en puissance de calcul distribuée;
- Les surcharges et l'équilibrage des charges dans un système distribué;
- La tolérance aux fautes et aux pannes

La vue de déploiement prend toute son importance lorsque le système est distribué. Elle se concentre sur les éléments de modélisation suivants :

- Les nœuds;
- Les instances de composants.

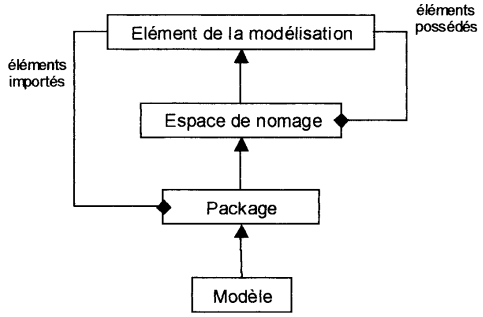
La vue des cas d'utilisation

Les cas d'utilisation forme la colle qui unifie les quatre vues précédentes. Les cas d'utilisation motivent et justifient les choix d'architecture. Ils permettent d'identifier les interfaces critiques ; ils forcent les concepteurs à se concentrer sur les problèmes concrets. Ils démontrent et valident les autres vues d'architecture. La vue des cas d'utilisation rend compte des éléments de modélisation suivants :

- Les acteurs;
- Les cas d'utilisation;
- Les classes;
- Les collaborations

Organisation des modèles

Les cinq vues décrites plus haut, associées au concept de *package,* permettent la structuration hiérarchique d'un modèle. Un modèle décrit par UML prend la forme d'arborescence représentée dans le diagramme de classe suivant :



Les paquetages sont à la fois des éléments de modélisation et des éléments de structuration des modèles. L'arborescence des paquetages organise les modèles, comme les répertoires organisent les systèmes de fichiers.

Documentation de l'architecture

L'architecture est décrite dans un document d'architecture qui comprend :

- Une description textuelle des traits de l'architecture (les vues) et les besoins clés du système;
- Les compromis retenus et les alternatives explorées;
- Une représentation de haut niveau de la vue logique;
- Des scénarios propre s à l'architecture;
- La description des mécanismes clefs.

5.4 Processus dirigé par la réduction des risques

L'analyse des risques est étudié à toutes les étapes de l'analyse. L'analyse des risques aide à la prise de décisions. Chaque phase du développement apporte sa contribution à la réduction des risques, avec un éclairage particulier:

- ***L'étude d'opportunité*** (appelée ***inception*** **ou Phase de conception**) essaie de délimiter les risques du projet, en construisant un prototype pour valider les concepts.
- ***L'élaboration*** **(ou phase d'élaboration**) commence par réduire les risques d'incompréhensions entre les utilisateurs et les développeurs : elle développe une vision partagée de la portée du projet et explore des scénarios avec les utilisateurs et les experts du domaine. Dans un deuxième temps, l'élaboration valide les choix d'architecture. Au besoin, l'architecture est affinée au cours des itérations suivantes
- **La *construction* (phase de construction**) est le siège de la fabrication incrémentale de l'explication. L'intégration progressive des divers composants supprime les risques de discordance rencontrés dans les phases dédiées spécifiquement à l'intégration, lors d'un développement en cascade.
- Lors d'un déploiement graduel de l'application en **phase transition**, les risques de prise en main du logiciel sont réduits, d'autant plus que les utilisateurs ont été impliqués dès les phases précédentes.
- **En phase de *post-déploiement*,** la maintenance, bien souvent plus évolutive que corrective, est effectuée par le même cadre de développement que la construction, par itération et fabrication de nouveaux prototypes La maintenance est une évolution normale du produit, comparable à la construction. Les risques de dégradation de l'architecture et de la conception s'en trouvent fortement réduits.

Risque	Impact	Résolution
Intégration trop complexe Absence de convergence	Qualité Coût Délais	Développement centré sur l'architecture Middleware Développement itératif
Démotivation	Qualité Coût Délais	Développement itératif Objectifs proches Planification des besoins
Environnement de développement inadapté	Coûts Délais	Investissement dans des outils intégrés

Utilisateurs défavorables	Délais Coûts	Implication précoce des utilisateurs Démonstration des prototypes
Technologie trop complexe	Délais Coûts	Développement centré sur l'architecture Achat de compétences externes
Activités manuelles trop lourdes	Coût	Automatisation par des outils partagés
Inadéquation des composants réutilisables achetés	Délais	Prototypes
Performances insuffisantes	Qualité	Indépendance par rapport au matériel acheté Tests précoces de l'architecture
Excès de bureaucratie	Délais	Développement centré sur les prototypes, pas sur la documentation
Fonctions inadaptées	Qualité	Prototypes

a) La phase de démarrage

Cette phase correspond à l'étude de faisabilité et d'opportunité du système à construire. Elle n'est généralement pas menée par les informaticiens, mais par des spécialistes de l'étude des marchés et de l'analyse de la concurrence. La première opération consiste à se doter d'une vision générale du produit afin de mieux le caractériser et de dégager des éléments d'appréciation. La formule suivante peut résumer cette vision :

VISION = QUOI + POUR QUI + COMBIEN

Les différents composants se définissent ainsi :

- **QUOI** exprime les grandes lignes du produit;
- **POUR QUI** détermine la population cible;
- **COMBIEN** estime le prix que les acheteurs du produit seront disposés à payer.

On peut donc dire qu'il y a l'identification du but premier du système, l'esquisse et l'analyse d'architecture provisoire et la préparation d'un plan de développement et d'estimation des coûts.

b) La phase d'élaboration

La phase d'élaboration commence par l'analyse des besoins et la modélisation du domaine. Elle a pour objectif de définir, réaliser et valider les choix d'architecture, d'explorer et de réduire les risques du projet, et finalement de définir un plan de développement complet qui quantifie les moyens à mettre en oeuvre pour mener à bien le développement.

Cette phase peut se faire de manière itérative. Plusieurs itérations sont souvent effectuées pour obtenir l'architecture finale.

- Élargissement de l'appréciation de faisabilité du système futur sur la quasi totalité des cas d'utilisation;
- Spécification détaillée des cas d'utilisation;
- Conception de l'architecture technique;
- Conception et réalisation des cas d'utilisation critiques (évaluation des risques) ;
- Le système de base permet d'estimer les ressources nécessaires;
- L'étude de rentabilité du projet est précisée

c) La phase de construction

Cette phase a pour objectif de développer un produit logiciel prêt à être transféré dans la communauté des utilisateurs.

- Conception et réalisation d'une 1ère version du produit (conception, implémentation, tests) ;
- Correction de la plupart des défauts;
- Livraison du système bêta.

d) La phase transition (livraison du produit)

La phase de transition consiste à transférer le logiciel dans la communauté des utilisateurs. Cette phase est de complexité variable selon le type d'applications ; elle comprend la fabrication, l'expédition, l'installation, la formation, le support technique et la maintenance.

En phase de transition, le développement initial est presque terminé; tous les artefacts ont atteint suffisamment de maturité pour être largement distribués vers deux catégories de destinataires : les utilisateurs et les responsables de

projet.

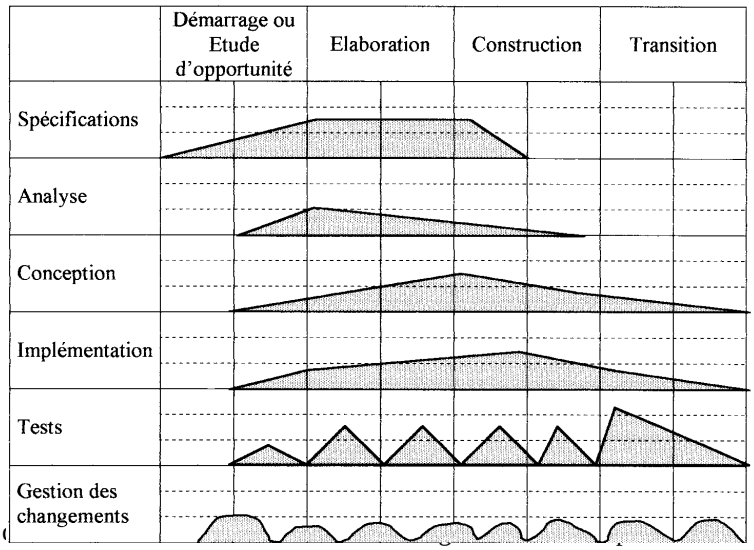
- Les principaux produits à destination des utilisateurs :
  - Les programmes exécutables, les versions bêta puis définitives;
  - Les manuels utilisateurs;
  - La documentation de déploiement et d'installation (Mise en place des procédures liées au déploiement et de l'équipe de maintenance).
- Les principaux produits à destination des responsables de projet :
  - Les modèles révisés;
  - Les critères d'évaluation de chaque itération;
  - La description des livraisons;
  - Les résultats de l'assurance qualité;
  - Les résultats de l'analyse post-mortem des performances du projet.

Chaque phase est dirigée par des activités (ou étapes) qui sont cycliques. C'est à dire qu'elles seront amenées à servir de manière itérative à chaque phase de développement ; on a :

- **L'expression des besoins** (Spécifications, c'est à dire cahier des charges)
  - Elle s'établit par une modélisation des procédures du système existant afin d'élaborer les cas d'utilisation. On trouve à ce niveau, une approche client avec la terminologie du client)
- **L'analyse**
  - Elle permet une formalisation du futur système en réponse à l'expression des besoins exprimés par le client;
  - Élaboration de tous les diagrammes donnant une représentation du système statique (DCL) et dynamique (cas d'utilisation).
- **La conception**
  - Prise en compte des choix d'architecture technique retenus pour le développement et l'exploitation du système
- **L'implémentation**
  - Concerne l'implémentation du logiciel sous forme de composants, de bibliothèques, de fichiers..
  - Il s'agit de l'étape la plus lourde en charge de travail. Elle représente au moins 40% du travail global.
- **Les tests**
  - C'est le niveau des tests.
    - Unitaires
    - D'intégration
    - De réception (validation)
    - De performance
    - De robustesse (Montée en charge, évolution.)

e) Résumé

- Le PU est itératif
- Chaque phase a des jalons pour les éléments fournis
- Artefacts des spécifications du modèle objet et du modèle comportemental
- Système exécutable partiellement implémenté
- Spécification des tests
  - Les artefacts utilisés dans le PU sont :
    - Expression du besoin : Recueil des besoins via les cas d'utilisation et des autres spécifications;
    - L'analyse : Diagramme de classes et validation des *use case* ;
    - Conception : Expansion

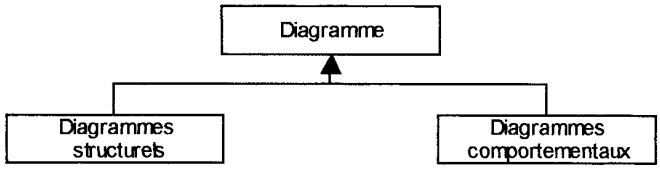


du DCL et des use case ;
- Implémentation. code source.

III Diagramme de classe (DCL)

III.1 Généralités et règles générales

Un diagramme donne à l'utilisateur un moyen de visualiser et de manipuler des éléments de modélisation. UML définit des diagrammes structurels et comportementaux pour représenter respectivement des vues statiques et dynamiques d'un système



1.1 Les stéréotypes

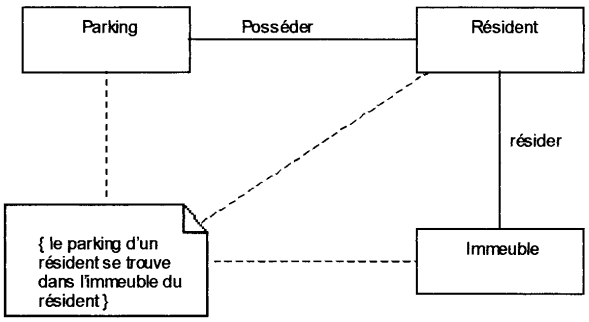
Un stéréotype permet la (méta) classification d'un élément UML. Il permet également de définir le but d'un élément«< *exprime* »). Par exemple, pour un *package.* «système », «sub-système», « souche »... Pour une classe' « interface », « type », « acteur »..

1.2 Les notes

Elles correspondent à un commentaire. Elle prend la représentation d'un rectangle avec le coin droit replié. Elle est reliée aux classes ou objets pour lesquelles le commentaire s'applique par un trait en pointillé.
Ex.

a) Les contraintes

Une contrainte est une note ayant une valeur particulière, elle est indiquée entre accolades.
Ex

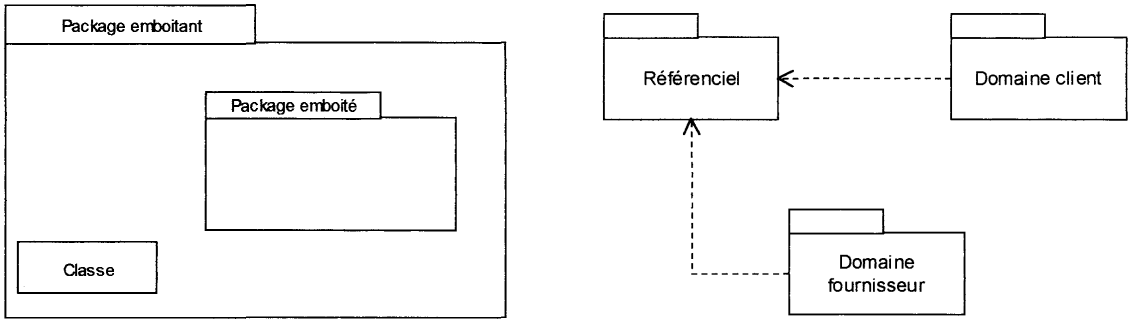


1.3 Les paquetages

Les systèmes complexes doivent être décomposés en sous-systèmes ou paquetages. Ils permettent de répartir le travail entre les différentes équipes de développement et permettent d'avoir une approche modulaire pour l'implémentation. Mais comment faire le 1er découpage d'un sous-système? pour réaliser une découpage, on utilisera les cas d'utilisation établis de manière macroscopique lors de la phase de démarrage. Par exemple, on trouvera des cas d'utilisation suivant différents départements de l'entreprise On peut rapprocher cette conception à Merise lors de l'étude des domaines avec la modélisation des flux (MCF) qui permet de définir les différentes activités (ou domaines) du domaine à étudier. Les paquetages portent sur les cas d'utilisation et sur le DCL

a) Dépendance entre paquetages:

Une relation de dépendance entre les paquetages représente un lien de dépendance entre deux éléments. Son formalisme est :



La relation de dépendance entre un paquetage source et un autre cible (unidirectionnel) .

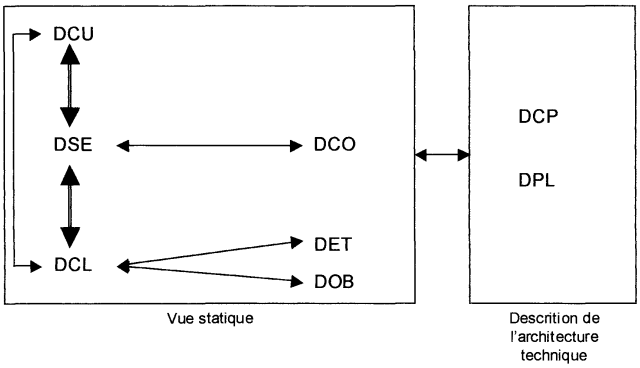
b) Stéréotypes standards associés aux paquetages

- « Importe », ajoute les éléments du paquetage cible (USE) à l'espace du paquetage source (WITH)
- « Accède », permet d'utiliser les éléments du paquetage cible (USE)

1.4 Les diagrammes d'UML

Nom du diagramme	Statique ou dynamique	Fonctions
Cas d'utilisation ou use case (DCU)	Dynamique	Reprend les fonctions du système du point de vue de l'utilisateur
De collaboration (DCO)	Dynamique	Représentation spatiale des objets, des lieux et des interactions en mettant l'accent sur les objets et les messages échangés
D'activités (DAC)	Dynamique	Représente le comportement d'une méthode ou use case
D'états transitions (DET)	Dynamique	Représente les états des objets en relation aux évènement
De séquences (DES)	Dynamique	Représente les scénarios de chaque cas d'utilisation en mettant l'accent sur l'aspect temporel (c'est à dire chronologique). Ce diagramme est en interaction avec celui des objets.
De classes (DCL)	Statique	Représente la structure statique du système en termes de classes et de relations entre classe. Il s'agit du diagramme pivot de la modélisation
D'objets (DOB)	Statique	Représente les objets et leur liens
De composants (DCP)	Statique	Représente les différents constituants du logiciel
De déploiement (DPL)	Statique	Représente le déploiement des composants sur les dispositifs matériels.

a) Liens entre les diagrammes



III.2 Le DCL

2.1 les classes

les classes sont représentées.

Nom classe
Description des attributs
Description des méthodes
Exception et objectifs (responsabilité)

2.2 les attributs

Description complète des attributs dans l'ordre :

- Obligatoire (O) : La visibilité (+ , -, #) ;
- O .Le nom (Nom unique dans la classe)
- Facultatif (F). multiplicité (nombre de valeur pour un attribut) ;
- O .type de l'attribut (prédéfini ou classe) ;
- F/O .valeur initiale;
- F .propriétés entre accolades.

Ex.

(-) prénom [2], text = (TOTO, TITI) {xxxx,yyyy}

2.3 Les méthodes

Description complète du comportement d'une classe :

- O .Visibilité;
- O .Nom de la méthode (polymorphisme) ;
- O .liste des arguments. Chaque paramètre doit être défini par
- O .Son nom;
- O: Son type;
- F. Sa direction (IN, OUT, IN/OUT).
- O. Le type retourné en résultat
- F .Propriétés {}

2.4 La visibilité

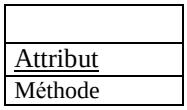
On a trois types de visibilités:

- Visibilité Publique : Elle se représente par le signe (+). Sa finalité est de permettre à tous les clients de la classe de rendre visibles tous les attributs et/ou les méthodes
- Visibilité Protégée : Elle se représente par le signe (#). Ne donne une visibilité qu'à l'intérieur de la classe, de toutes les sous-classes et de ses attributs et/ou méthodes.
- Visibilité privée: Elle se représente par le signe (-) et a pour conséquence de rendre visibles ses attributs et/ou

méthodes uniquement à l'intérieur de la classe.

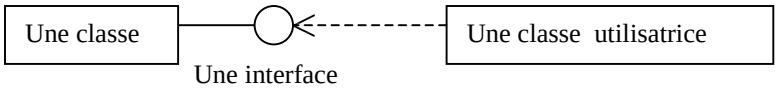
2.5 Attributs ou méthodes de classe

Un attribut ou une méthode peut être défini au niveau de la classe (ex' constante pour toutes les instances ou méthode d'une classe). Sa représentation est:

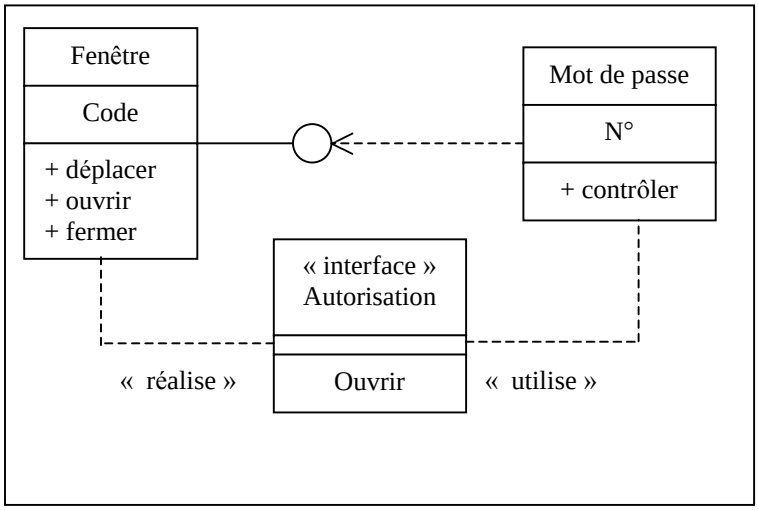


2.6 Les interfaces

Une classe d'interface décrit le comportement visible d'une classe, d'un composant, d'un sous-système, d'un paquetage, ou d'un autre classificateur et est constituée des méthodes ayant une visibilité «public ». UML représente les interfaces au moyen de petits cercles reliés par un trait à l'élément qui fournit les services décrits par l'interface.

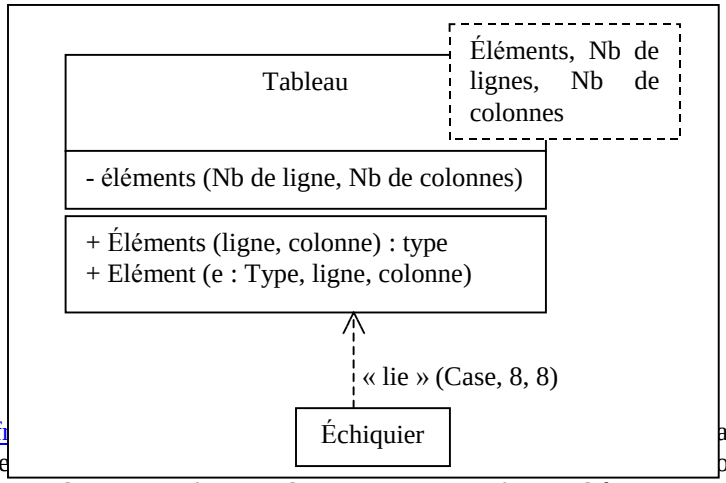


Elles peuvent également se représenter au moyen de classes avec le mot clef «interface ». Cette notation permet de faire figurer dans le compartiment des opérations. La classe qui réalise cette interface est alors indiquée par une relation de réalisation (représentée par une flèche en traits pointillés et à extrémité triangulaire). Une même classe peut réaliser plusieurs interfaces et une interface peut être réalisée par plusieurs classes.



2.7 Les classes paramétrables

Les classes paramétrables, aussi appelées **classes templates**, sont des modèles de classes. Elles correspondent aux classes génériques d'Eiffel et aux templates C++. Une classe paramétrable ne peut être utilisée telle quelle. Il convient d'abord de **lier** les paramètres formels à des paramètres effectifs (le terme *instancier* est également utilisé) afin d'obtenir une classe paramétrée qui pourra à son tour être instanciée afin de donner des objets. Lors de l'instanciation, les paramètres effectifs personnalisent la classe réelle obtenue à partir de la classe template.



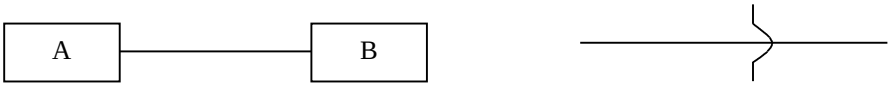
Ce type de classe n'apparaît généralement pas en analyse, sauf dans le cas particulier de la modélisation d'un environnement de développement. Les classes paramétrables sont surtout utilisées en conception détaillée, pour incorporer par exemple des composants réutilisables.

2.8 Les classes utilitaires

Il est parfois utile de regrouper des éléments (les fonctions d'une bibliothèque mathématique, par exemple) au sein d'un module, sans pour autant construire une classe complète. La classe utilitaire permet de représenter de tels modules et de les manipuler graphiquement comme les classes conventionnelles.
Les attributs et les procédures contenus dans une classe utilitaire deviennent des variables et des procédures globales ; la portée n'a pas besoin d'être spécifiée puisque les attributs et opérations sont automatiquement visibles dans la portée lexicale de classes. Le stéréotype « **utilitaire** » spécialise les classes en classes utilitaires (en C++ une classe utilitaire correspond à une classe qui ne contient que des membres statiques (fonctions et données)).

2.9 Les associations

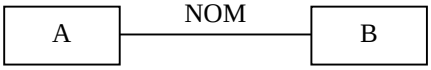
Les associations représentent des relations ou liens (un lien est une instance d'une association) structurels entre classes d'objets. Une association symbolise une information dont la durée de vie n'est pas négligeable par rapport à la dynamique générale des objets instances des classes associées. Une association compte au moins deux extrémités d'association reliées à des classificateurs. La plupart des associations sont binaires, c'est à dire qu'elles connectent deux classes. Las associations se représentent en traçant une ligne entre les classes associées.



Elles peuvent être représentées par des traits rectilignes ou obliques selon les préférences de l'utilisateur. En règle générale, on évitera, tant que possible, que les liens se croisent, si cela n'est pas possible, alors on représentera le croisement à l'aide d'un demi cercle.

a) Nommage des associations

Les associations peuvent être nommées ; le nom de l'association figure alors au milieu de la ligne qui symbolise l'association.

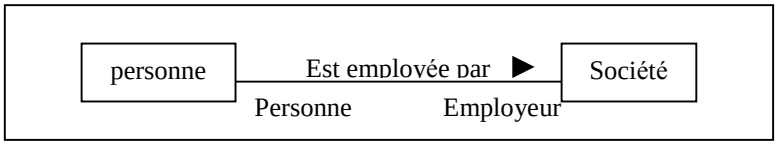


Sans en faire une règle, il est recommandé d'utiliser une forme verbale active ou passive en guise de nom d'association.

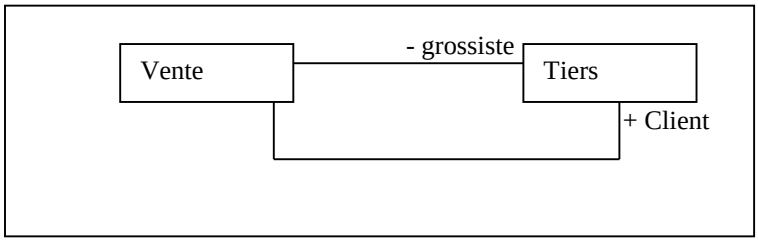
Dans les deux cas, le sens de lecture peut être précisé au moyen d'un petit triangle. Par défaut, le sens de lecture est de gauche à droite. En général, la précision est donnée uniquement lorsque le sens de lecture est ambiguë. Le triangle peut également être remplacé par les signe < et > disponible dans toutes les fontes.
Les associations entre classes expriment principalement la structure statique ; de ce fait, le nommage par des formes verbales qui évoquent plutôt le comportement s'écarte un peu de l'esprit général des diagrammes de classes. Comme le nommage des associations , le nommage des extrémités des relations permet de clarifier les diagrammes, mais d'une manière plus passive, plus en phase avec la tonalité statique des diagramme de classes.

b) Rôles des extrémités des associations

L'extrémité d'une association possède un nom. Ce nom, aussi appelé rôle, décrit comment une classe source voit une classe destination au travers de l'association. Il nomme également le passage d'une instance de la classe source à une ou plusieurs instances de la classe destination. Le rôle représente un *pseudo-attribut* de la classe source et doit avoir un nom unique dans l'ensemble des noms d'attributs et pseudo-attributs de la classe source. Dans l'exemple suivant, **Employeur** est un pseudo-attribut de **Personne**.



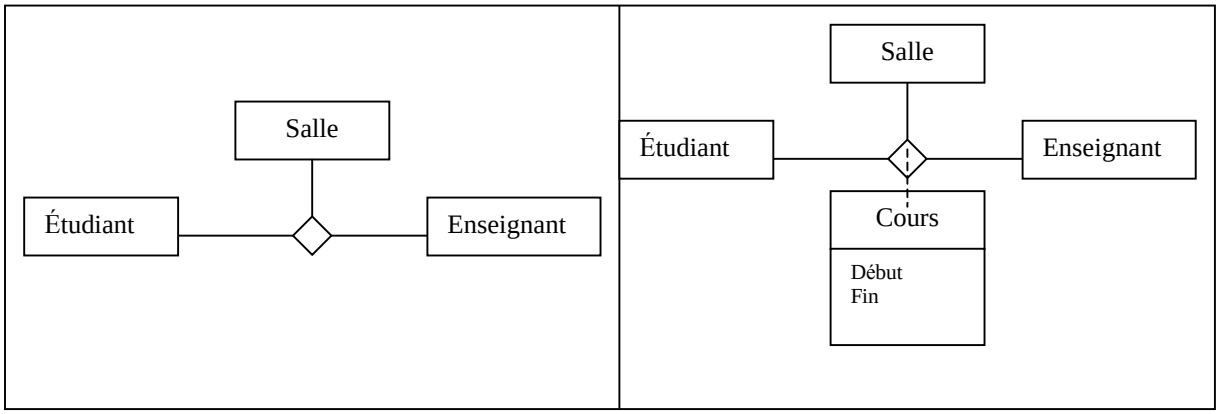
Une indication de visibilité peut être précisée. Elle est placée avant le rôle pour préciser la visibilité du rôle à l'extérieur de l'association ; par défaut, si rien n'est précisé, le rôle est considéré comme **Public**.



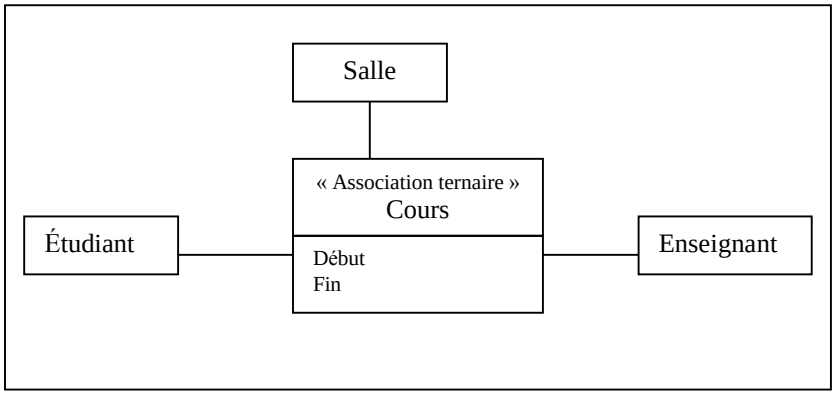
Dans cet exemple, lorsque les instances de **Tiers** jouent le rôle de **grossiste**, elles sont **privées** ; elles sont accessibles uniquement à l'instance de vente à laquelle est sont associées. En observant un objet **Vente**, il est possible de voir le client mais pas le grossiste.

c) Arité des associations

La plupart des associations sont dites binaires car elles relient deux classes. Des arités supérieures peuvent cependant exister et se représentent alors au moyen d'un losange sur lequel arrivent différentes composantes de l'association.



Généralement, les associations n-aires peuvent se représenter en promouvant l'association au rang de classe et en rajoutant une contrainte qui exprime que les branches de l'association s'instancient toutes simultanément, en un même lien. Dans l'exemple suivant, la contrainte est exprimée au moyen d'un stéréotype (non standard) qui précise que la classe cours réalise une association ternaire. Comme pour les associations binaires, il est possible de nommer les extrémités d'une association n-aire pour décrire le rôle des classes dans l'association. La difficulté de trouver un nom différent pour chaque extrémité d'une association est souvent le signe d'une association d'arité inférieure.

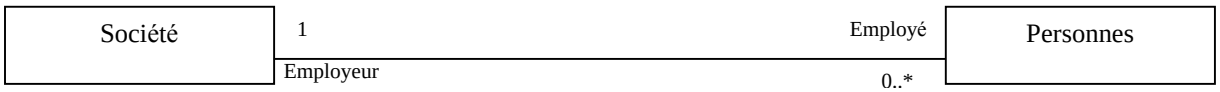


d) Contrainte de multiplicité

Chaque extrémité d'une association peut porter une indication de multiplicité qui montre combien d'objets de la classe considérée peuvent être liés à un objet de l'autre classe. La multiplicité est une information portée par l'extrémité d'association, sous la forme d'une expression entière. Les valeurs conventionnelles sont :

1	Un et un seul
0..1	Zéro ou un
N	Entier
M..N	De M à N (entiers naturels)
*****	De zéro à plusieurs
0..*	De zéro à plusieurs
1..*	D'un à plusieurs

L'exemple suivant rend compte du fait que plusieurs personnes travaillent dans la même société.

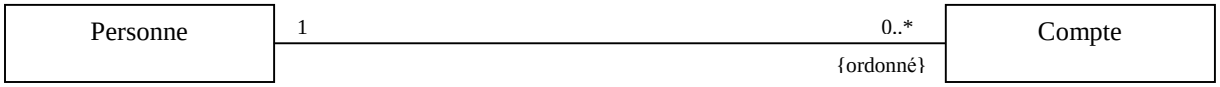


Ce modèle supprime le chômage puisque la multiplicité de valeur **1** du côté de la classe **Société** n'est pas très réaliste car elle signifie que toutes les personnes ont un emploi.
Les valeurs de multiplicité expriment des contraintes liées au domaine de l'application, valable durant toute la vie de l'objet. Le respect des valeurs de multiplicité peut être assuré statiquement ou dynamiquement.

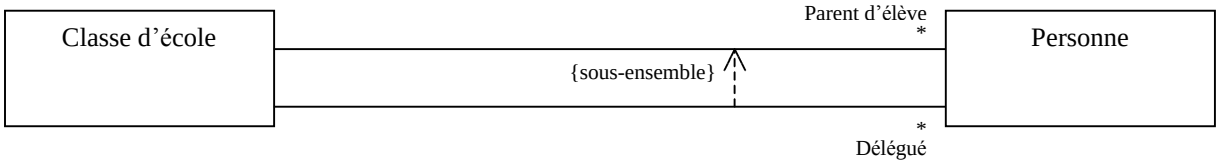
e) Contraintes sur les associations

Toutes sortes de contraintes peuvent être définies sur une relation ou sur un groupe de relations. La multiplicité vu ci-dessus est une contrainte sur le nombre de liens qui existent éventuellement entre deux objets. Les contraintes se représentent dans les diagrammes par des expressions placées entre accolades.

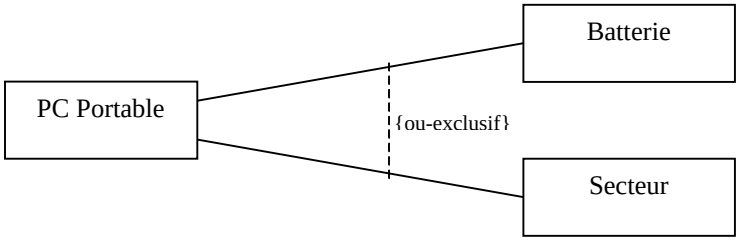
- La contrainte **{ordonné}** peut être placée sur le rôle pour spécifier qu'une relation d'ordre décrit les objets de la collection ; dans ce cas, le modèle ne spécifie pas comment les éléments sont ordonnés, mais seulement que l'ordre doit être maintenu durant l'ajout et/ou la suppression des objets.



- La contrainte **{sous-ensemble}** indique qu'une collection est incluse dans une autre collection. La contrainte est placée à proximité d'une relation de dépendance entre deux associations (La flèche de la relation de dépendance indique le sens de la contrainte). L'exemple suivant montre que les délégués de parents d'élèves sont des parents d'élèves.

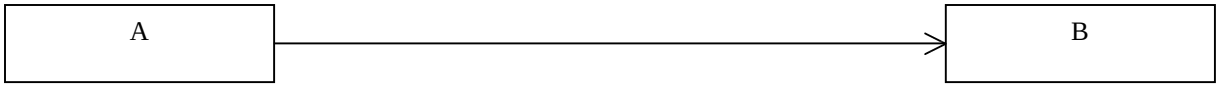


- La contrainte **{ou-exclsusif}** indique que, pour un objet donné, une seule association parmi un groupe d'associations est valide. Cette contrainte évite l'introduction de sous-classes artificielles pour matérialiser l'exclusivité. Dans le diagramme des classes, cette contrainte est placée à proximité d'un trait en pointillés qui relie les associations (au minimum deux) dont les instances sont mutuellement exclusives.



f) Navigabilité

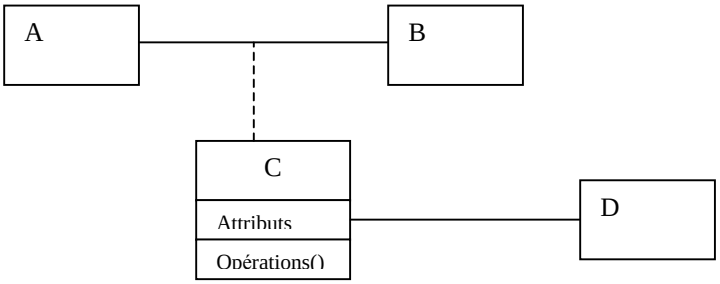
Les associations décrivent le réseau de relations structurelles qui existent entre les classe et qui donnent naissance aux lien entre les objets instances de ces classes. Les liens peuvent être vus comme des canaux de navigation entre les objets. Ces canaux permettent de se déplacer dans le modèle et de réaliser les formes de collaboration qui correspondent aux différents scénarios.
Par défaut, ces canaux sont navigables dans les deux directions. Dans certains cas, une seule direction de navigation est utile ; l'extrémité d'association vers laquelle la navigation est possible porte une flèche. L'absence de flèche signifie que l'association est navigable dans les deux sens.



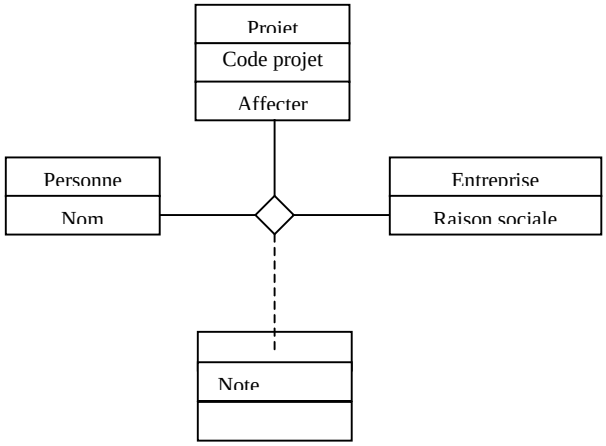
L'association entre les classe A et B est uniquement navigable dans le sens de A vers B.

g) Les classes-associations

Il est possible de représenter une association par une classe pour ajouter, par exemple, des attributs et des opérations dans l'association. Une classe de ce type, appelée **classe associative** ou **classe-association**, possède à la fois les caractéristiques d'une classe et d'une association, et peut à ce titre participer à d'autre relations dans le modèle. La notation utilise une ligne pointillée pour attacher une classe à une association. Dans l'exemple suivant, l'association entre les classe **A** et **B** est représentée par la classe **C**, elle même associée à la classe **C**.

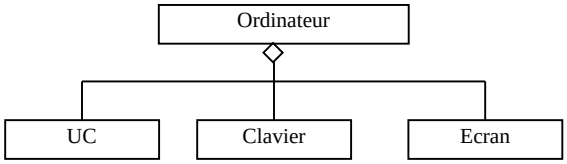


Une association qui contient des attributs sans participer à des relations avec d'autres classes est appelée **association attribuée**. Dans ce cas, la classe attachée à l'association ne porte pas de nom spécifique.
Les classes-associations permettent donc de décrire les attributs et les méthodes propres à l'association.



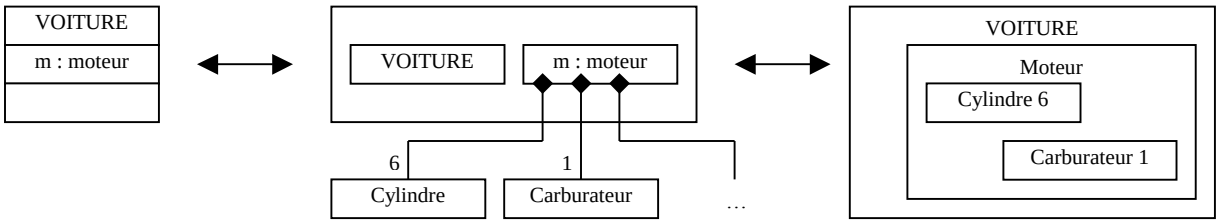
h) Les agrégations

Une agrégation représente une association non symétrique dans laquelle une des extrémités joue un rôle prédominant par rapport à l'autre extrémité. Quelle que soit l'arité, l'agrégation ne peut concerner qu'un seul rôle d'une association.
L'agrégation se représente en ajoutant un petit losange du coté de l'agrégat.



i) La composition

La composition est un cas particulier d'agrégation avec un couplage plus important. La classe ayant le rôle prédominant dans une composition est appelée **classe composite** ou **classe conteneur**.
La composition implique, en plus des propriétés d'agrégation, une coïncidence des durées de vie des composants et du composite. : la destruction du composite implique automatiquement la destruction de tous ses composants. La création, la modification et la destruction des divers composants sont de la responsabilité du composite.



j) Généralisation, spécialisation et héritage

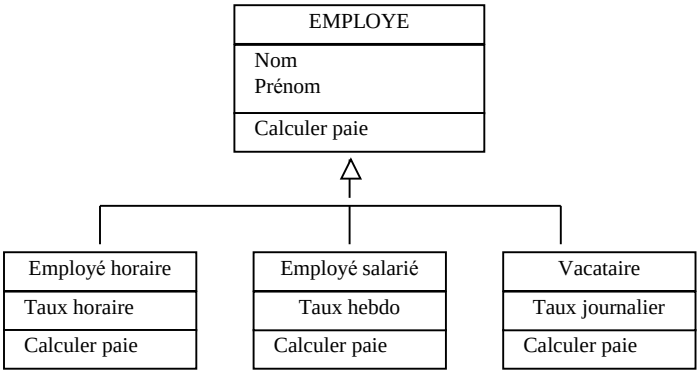
La généralisation est la relation entre une classe et une ou plusieurs autres classes partageant un sous-ensemble commun d'attributs et/ou de méthodes. UML emploie le terme de **généralisation** pour désigner la relation de classification entre un élément plus général et un éléments plus spécifique. En fait, le terme généralisation désigne un point de vue porté sur un arbre de classification.

La classe de plus haut niveau s'appelle **la superclasse** , celle de bas niveau **la sous-classe** ou **classe dérivée**. La généralisation signifie « est une sorte de ». La généralisation s'applique aux classes, aux cas d'utilisation, et aux paquetages ainsi qu'à tous les éléments de modélisation définis dans le méta modèle comme sous-classe d'**élément généralisable**.

La **spécialisation** consiste à créer des sous-classes à partir d'une classe. Les attributs et les méthodes d'une superclasse sont transmis aux sous-classes par héritage.

k) Classes abstraites

Une classe abstraite est une classe qui n'a pas d'instance (mais dont les sous-classes en ont). Elles ne sont pas instanciables directement ;elles ne donnent pas naissance à des objets, mais servent de spécification plus générale pour manipuler les objets instances d'une (ou plusieurs) de leur sous-classes.

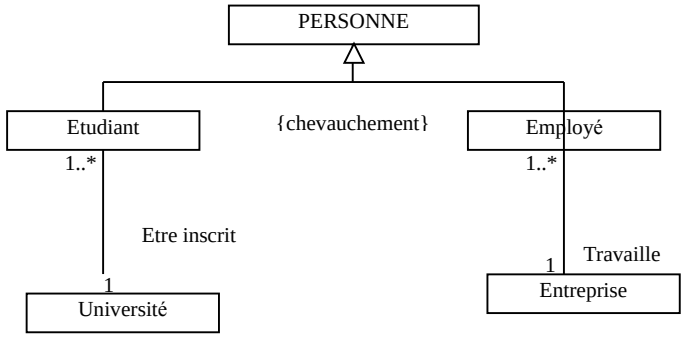


l) héritage et recouvrement (les contraintes et les propriétés de généralisation)

Différentes contraintes peuvent être appliquées aux relations de généralisation, pour distinguer par exemple, les formes exclusives de généralisation des formes inclusives.

Les contraintes sur les relations de généralisation se représentent au moyen d'expression entre accolades, directement attachées aux généralisations agrégées ou associées à une ligne en pointillés qui relie les relations de généralisation concernées. Par défaut, les sous-classes ont des instances disjointes les unes par rapport aux autres. Toutefois, on distingue :

- **{chevauchement}** : Signifie que deux sous-classes peuvent avoir des instances identiques.
- **{disjointe}** : Les instances d'une sous-classe ne peuvent être incluses dans une autre sous-classe de la même classe
- **{complète}** : La généralisation ne peut être étendue (à utiliser avec précaution)
- **{incomplète}** : La généralisation peut être étendue



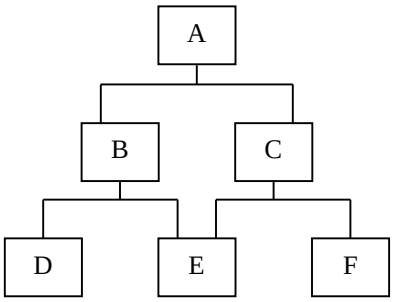
Une personne peut être à la fois **étudiant** et **employé**.

m) Extension et restriction de classe

- L'ajout de propriétés dans une sous-classe correspond à une extension de classe
- Le masquage de propriétés dans une sous-classe correspond à une restriction

n) Héritage multiple

Faire hériter une classe de deux classes « parent » s'appelle **héritage multiple**.



IV Les diagrammes des cas d'utilisation (use case)

IV.1 Généralité

1.1 Définition

Les diagrammes de cas d'utilisation représentent les cas d'utilisation, les acteurs et les relations entre les cas d'utilisation et les acteurs.

Les cas d'utilisation recensent les expressions du besoin sur les utilisateurs. En partant du principe qu'un système est avant tout construit pour les utilisateurs, les cas d'utilisation permettent de structurer et d'articuler les besoins en fonctionnalités et de définir la manière dont les utilisateurs voudraient interagir avec le système.

Ils décrivent, sous la forme d'action et de réactions, le comportement d'un système d'un point de vue utilisateur. Ils permettent de définir les limites du système et les relations entre le système et l'environnement.

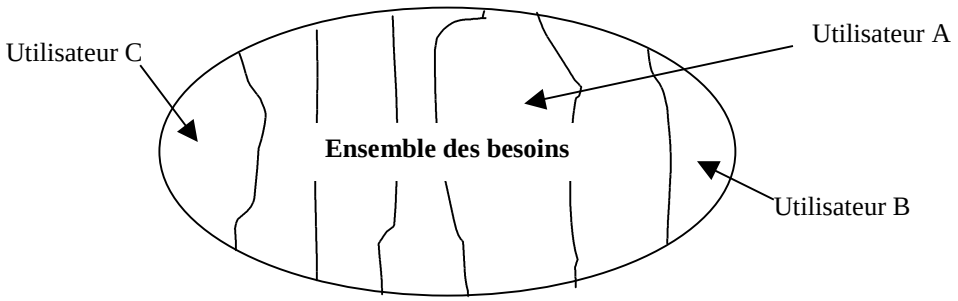
1.2 Intérêt des cas d'utilisation

La détermination et la compréhension des besoins sont souvent difficiles car les intervenants sont noyés sous de grandes quantités d'informations. Les besoins sont souvent exprimés de manière non structurée, sans forte cohérence, de sorte que le cahier des charges sont de longues litanies de paragraphes.

Fréquemment, des besoins sont contradictoires, des oublis sont commis, des imprécisions subsistent, et l'analyse du système part sur de mauvaises bases. Le cahier des charges initial est flou et en constante évolution. Lorsque les besoins se précisent ou évoluent (ce qui est toujours le cas), il devient très difficile d'apprécier l'impact et le coût d'une modification.

Les cas d'utilisation recentrent l'expression des besoins sur les utilisateurs. La structuration de la démarche s'effectue par rapport aux interactions d'une seule catégorie d'utilisateurs à la fois ; cette partition de l'ensemble des besoins réduit la complexité de leur détermination.

Les cas d'utilisation partitionnent l'ensemble des besoins d'un système



Le formalisme des cas d'utilisation, basé sur le langage naturel, est accessible aux utilisateur sans formation particulière ; ils peuvent ainsi exprimer leurs attentes et leurs besoins en communiquant facilement avec les experts du domaine et les informaticiens.
Les use case permettent aux utilisateurs de structurer et d'articuler leurs désirs ; ils les obligent à définir la manière dont ils voudraient interagir avec le système, à préciser quelles informations ils entendent échanger et à décrire ce qui doit être fait pour obtenir le résultat escompté. Les cas d'utilisation concrétisent le futur système en une formalisation proche de l'utilisateur ; ils favorisent la reformulation du cahier des charges afin de refléter réellement les besoins, même en l'absence d'un système à critiquer.

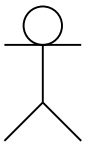
IV.2 Les acteurs

2.1 Définition :

Un acteur représente un rôle joué par une personne ou un élément qui interagit avec le système. La même personne physique peut jouer le rôle de plusieurs acteurs (ex : vendeur, client). Plusieurs personnes peuvent jouer le même rôle (ex : une fonction ne donne qu'un seul acteur (secrétariat)).
Les acteurs se détermine en observant les utilisateurs directs du système, ceux qui sont responsables de son exploitation ou de sa maintenance, ainsi que les autres systèmes qui interagissent avec le système en question.
Les candidats acteurs se recrutent parmi les utilisateurs, les clients, les partenaires, les vendeurs, les autres systèmes ; en résumé, il s'agit des personnes et des choses extérieures à un système qui interagissent avec lui en échangeant de l'information. La détermination des acteurs permet de préciser les limites du système de manière progressive.

2.2 Représentation

Les acteurs se représentent sous la forme de petits personnages ou via un symbole de classe avec le mot clé « acteur ».



Un acteur

2.3 Les différentes catégories d'acteurs

Il existe quatre grandes catégories différentes :

- **Les acteurs principaux** : Cette catégorie regroupe les personnes qui utilisent les fonctions principales du système. Dans le cas d'un distributeur de billets, il s'agit des clients.
- **Les acteurs secondaires** : Cette catégorie regroupe les personnes qui effectuent des tâches administrative ou de maintenance. Dans le cas du distributeur de billet, il s'agit de la personne qui recharge la caisse contenue dans le distributeur.
- **Le matériel externe** : Cette catégorie regroupe les dispositifs matériels incontournables qui font partie du domaine de l'application et qui doivent être utilisés ; Il ne s'agit donc pas de l'ordinateur sur lequel s'exécute

l'application, mais des autres dispositifs matériels périphériques, en l'occurrence, l'imprimante dans le cas d'un DAB.

- **Les autres systèmes** : Cette catégorie regroupe les systèmes avec lesquels le système doit interagir. Dans le cas d'un DAB, le système du groupement bancaire qui gère le parc de DAB est un acteur .

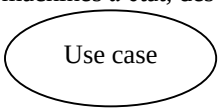
IV.3 Les cas d'utilisation

3.1 Définition

Un cas d'utilisation est un classificateur qui modélise une fonctionnalité d'un système ou d'une classe. L'instanciation d'un cas d'utilisation se traduit par l'échange de message entre le système et ses acteurs.

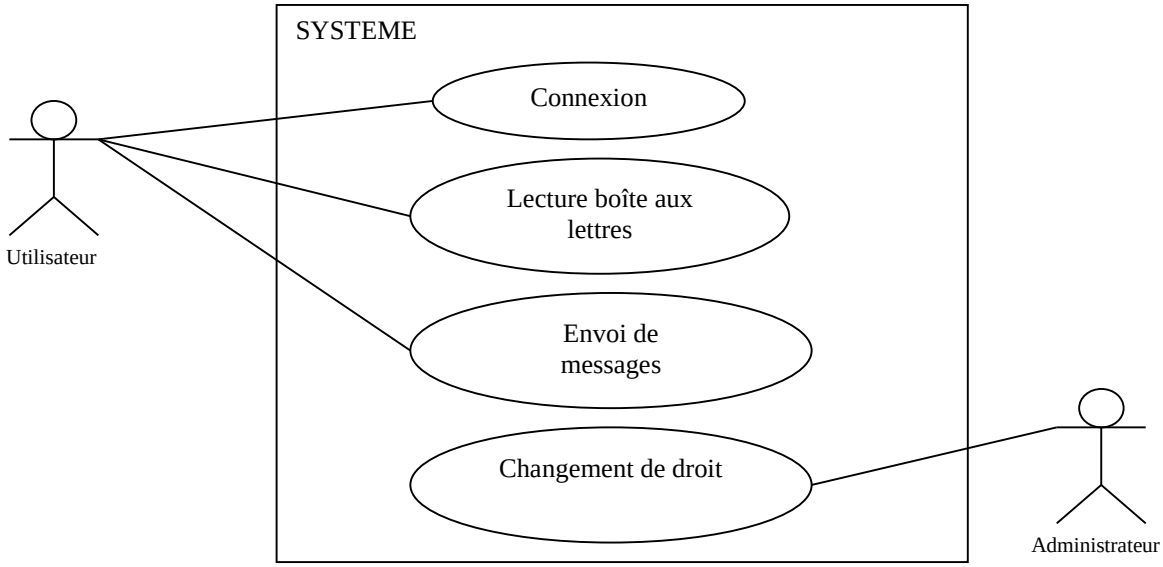
3.2 Formalisme

Les cas d'utilisation sont représentés par des ellipses à l'intérieur desquelles figure le nom du cas d'utilisation (il peut également être placé au-dessus de l'ellipse). Un nom est éventuellement précédé d'un stéréotype et suivi de propriétés.
Les cas d'utilisations peuvent être contenus dans un rectangle qui représente les limites du système. Les acteurs sont alors forcement à l'extérieur du rectangle puisqu'ils ne font pas partie du système ;
Il est possible de préciser les comportement d'un cas d'utilisation via du texte, des diagrammes d'activités, des machines à état, des méthodes…



3.3 Interaction entre l'acteur et le cas d'utilisation

C'est une association entre un acteur et le use case et peut comporter des multiplicités.
Ex : un système de messagerie comportant 4 cas d'utilisation.

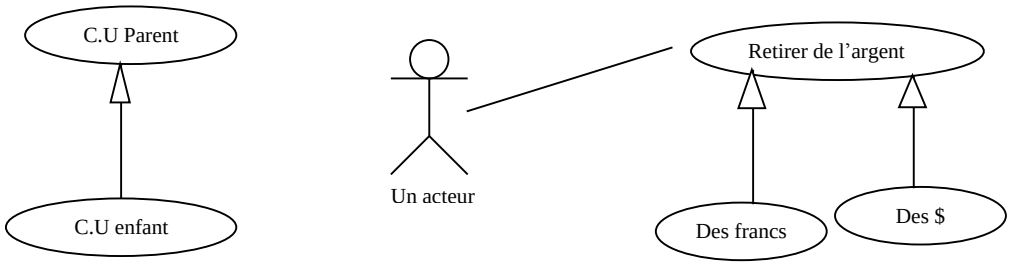


Les cas d'utilisation se déterminent en observant et en précisant, acteur par acteur, les séquences – les scénarios – du point de vue de l'utilisateur. Ils se décrivent en terme d'informations échangées et d'étapes dans la manière d'utiliser le système.
Les cas d'utilisation doivent être vus comme des classes dont les instances sont des scénarios. Chaque fois qu'un acteur interagit avec le système, le Use case instancie un scénario ; ce scénario correspond au flot de messages échangés par les objets durant l'interaction particulière qui correspond au scénario.

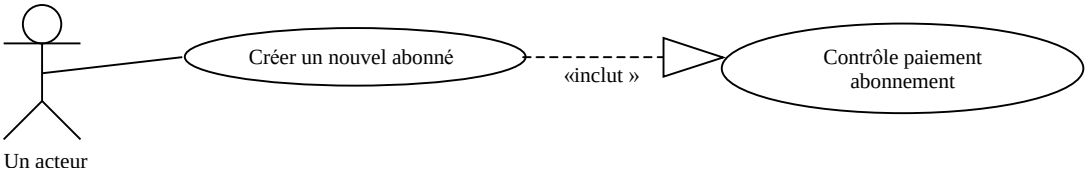
3.4 Les relations entre cas d'utilisation

UML définit trois types de relations entre C.U :

- **La relation de généralisation** : Dans une relation de généralisation entre deux C.U, le cas d'utilisation **enfant** est une spécialisation du C.U **parent**. Le C.U parent peut être abstrait. Une flèche à extrémité triangulaire représente une telle relation.

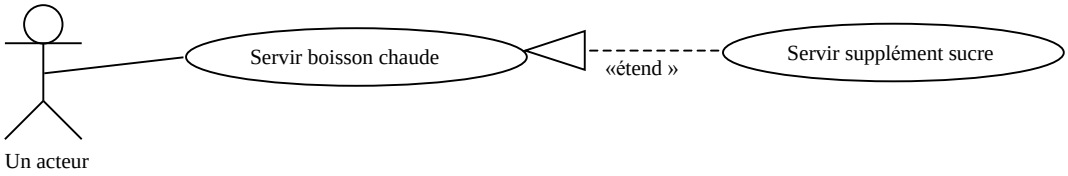


- **La relation d'inclusion** : dans une relation d'inclusion entre C.U, une instance du C.U source comprend également le comportement décrit par le C.U destination.



- **La relation d'extension**

Une relation d'extension, d'un C.U A par un use case B, signifie qu'une instance de A peut être « étendu » par le comportement décrit dans B.



V Les autres diagrammes

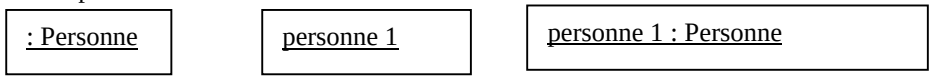
V.1 Le diagramme d'objet

1.1 Formalisme

Le nom de l'objet peut être désigné sous plusieurs formes :

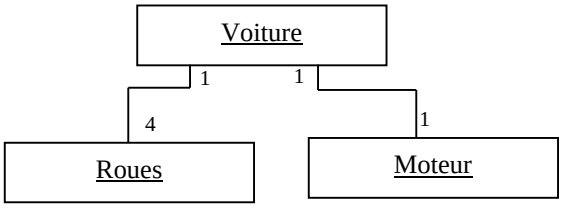
- nom de l'objet
- nom de l'objet : Nom de la classe
- : Nom de la classe

Exemples :

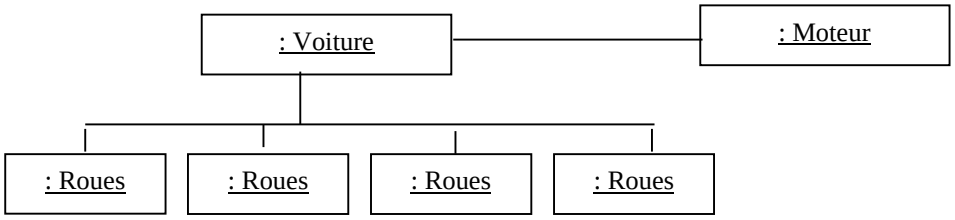


1.2 Représentation des liens

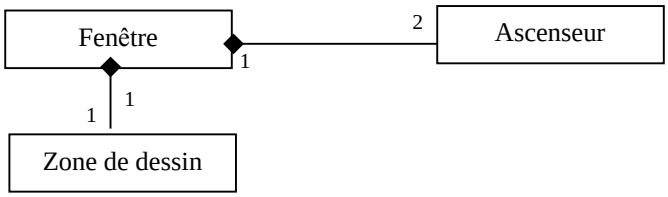
- Soit le diagramme de classe :



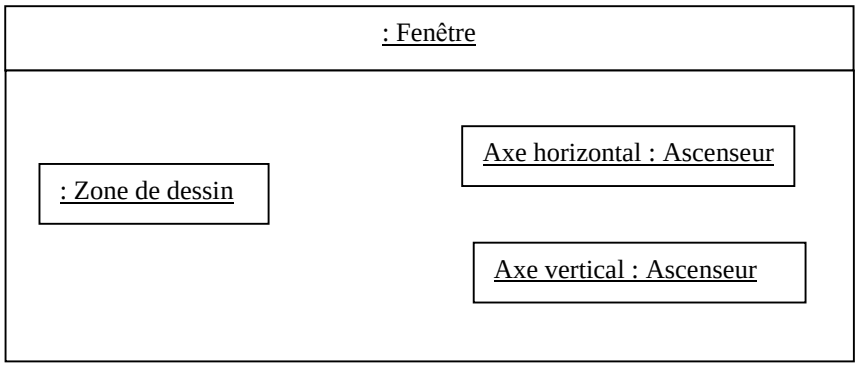
- Une instance de ce diagramme donne :



- Les objets composites
  - DCL



V.2 Le diagramme d'instance

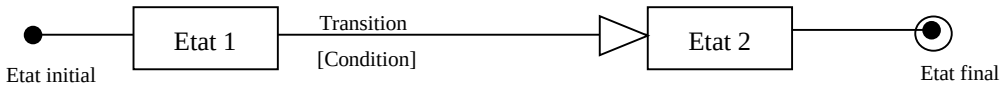


V.3 Le diagramme d'état/transition

a) État, transition, évènement

- L'état d'un objet est défini à un instant T par l'ensemble des valeurs de ses attributs
- Le passage d'un état à un autre s'appelle **la transition** ; Cette transition peut être conditionnée
- Un évènement est un fait survenu qui déclenche une transition (signal, appel, temporel…)

b) Formalisme et exemple



c) Actions et activités

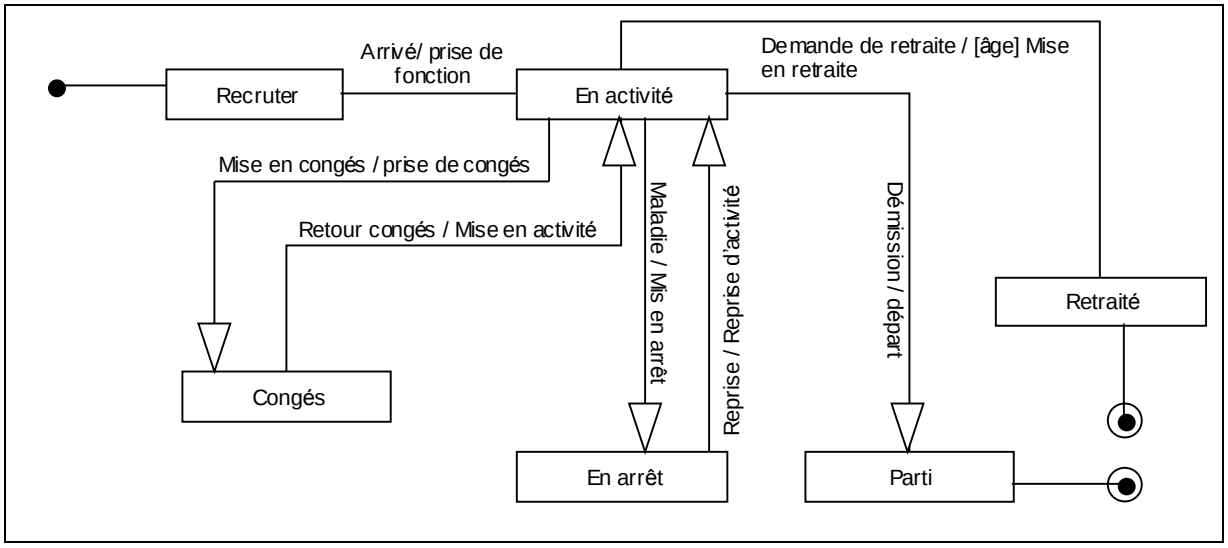
- Une action est une opération instantanée qui ne peut être interrompue ; elle est associée à une transition.
- Une activité est une opération d'une certaine durée qui ne peut être interrompue et elle est associée à un état d'un objet.

d) Exemples d'exercices

Représenter le diagramme d'état/transition d'un objet « personnel » suivant les évènements de gestion depuis le recrutement jusqu'à son départ. Après recrutement, une personne est considérée en activité dès sa prise de fonction. Les évènements à gérer son :

- Le congés maladie
- La prise de congés

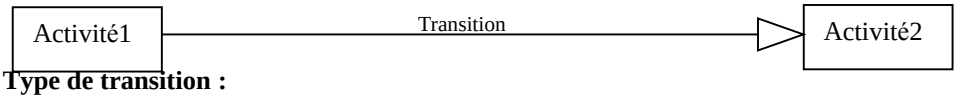
- La fin de carrière
  - Nous retiendrons deux situations :
    - La démission
    - La retraite



V.4 Le diagramme d'activité

Définition : Le diagramme d'activité est très proche du diagramme état/transition, seulement celui-ci se focalise sur les transitions et les activités, et représente le comportement d'une action ou d'un cas d'utilisation en terme d'action.

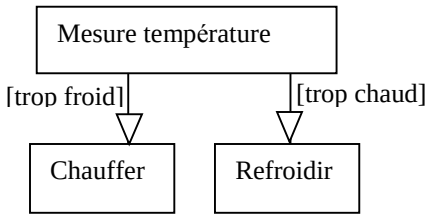
Formalisme :

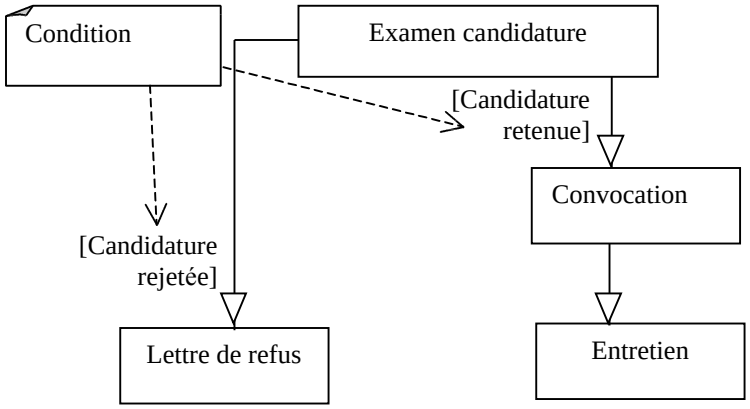


Type de transition :

- Transition automatique ; la fin d'une activité entraîne automatiquement le début de l'activité suivante
- Transition gardée ; Le passage à l'activité suivante est conditionnée, c'est à dire qu'elle s'effectue que si la condition mentionnée est vérifiée

Exemples :

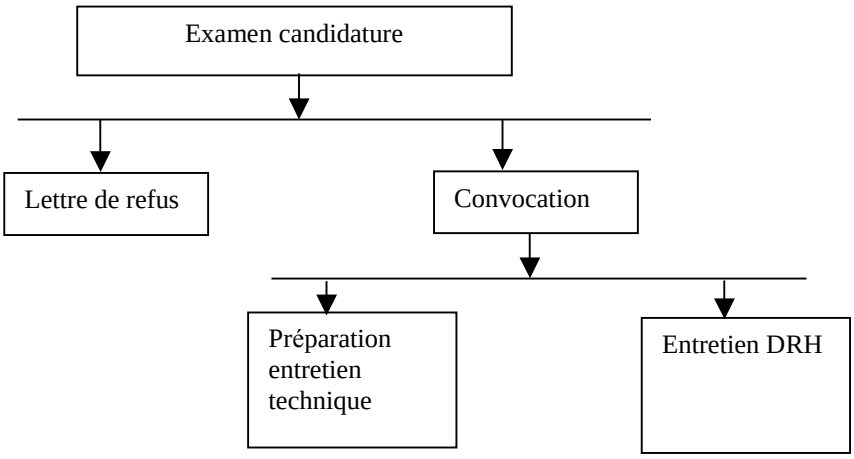




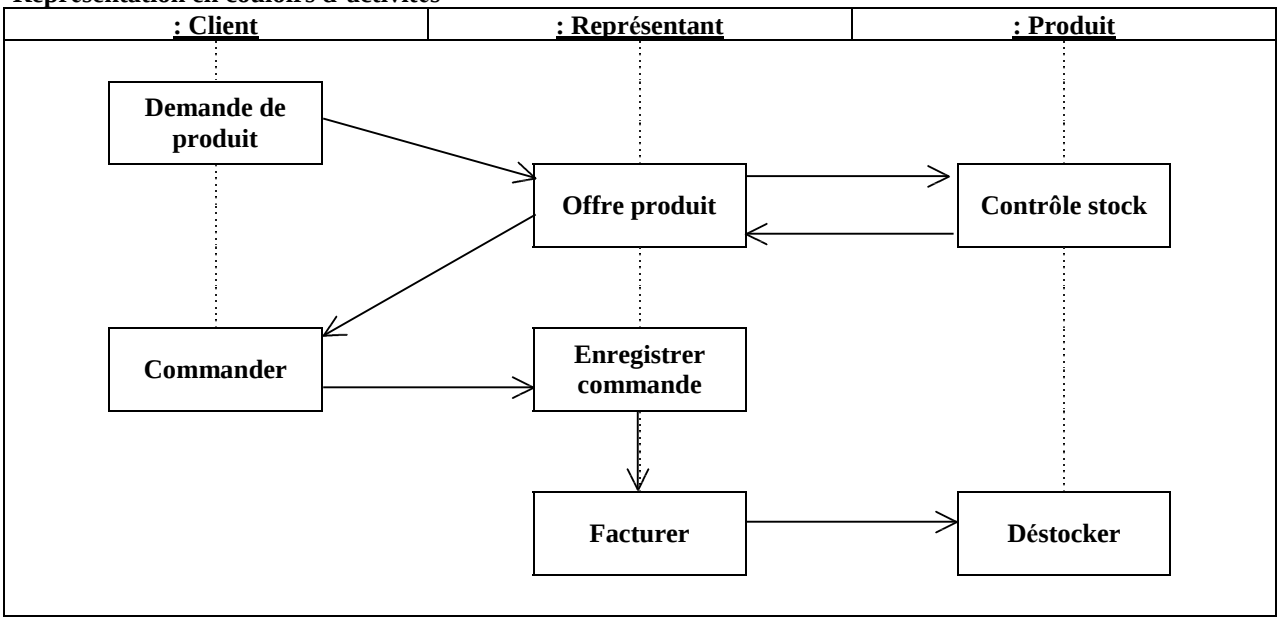
Exécution et synchronisation

- Plusieurs activités peuvent s'exécuter en même temps, de façon parallèle en vue de produire des résultats nécessaires à l'exécution d'une autre activité.

Exemple :



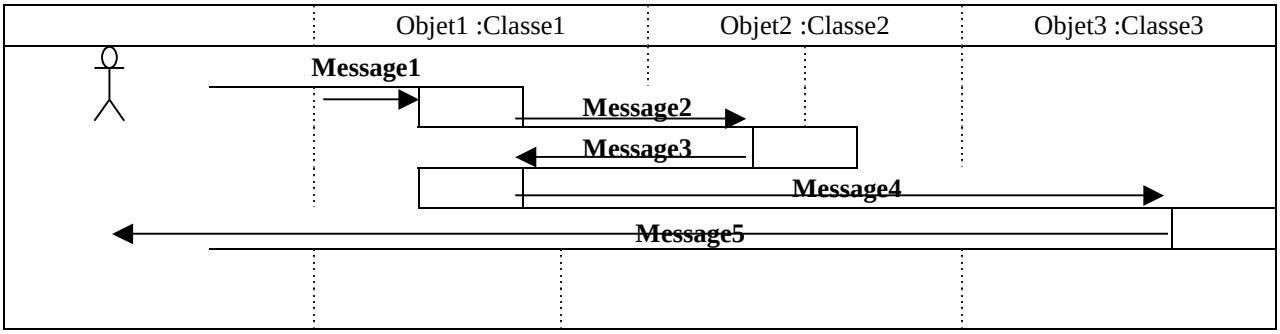
Représentation en couloirs d'activités



V.5 Le diagramme de séquence

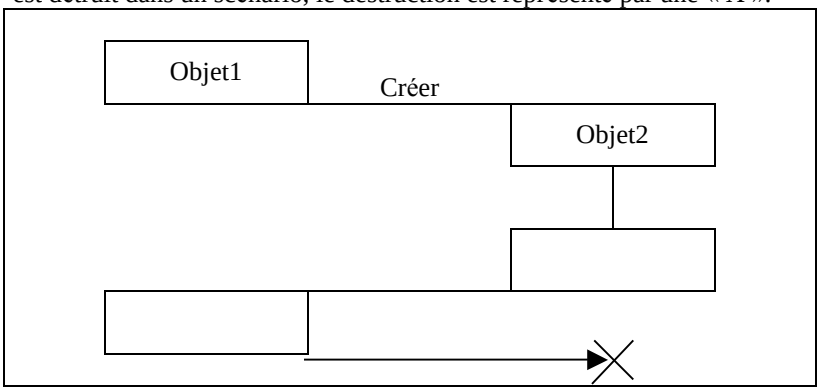
Les diagrammes de séquences montrent la séquence des interactions entre objets selon un point de vue temporel (chronologique). Ce diagramme permet de représenter les scénarios d'un cas d'utilisation. Un scénario est une instance d'un cas d'utilisation. Un message reçu par un objet déclenche l'exécution d'une opération et renvoie un message qui correspond au résultat de l'opération.

5.1 Formalisme :



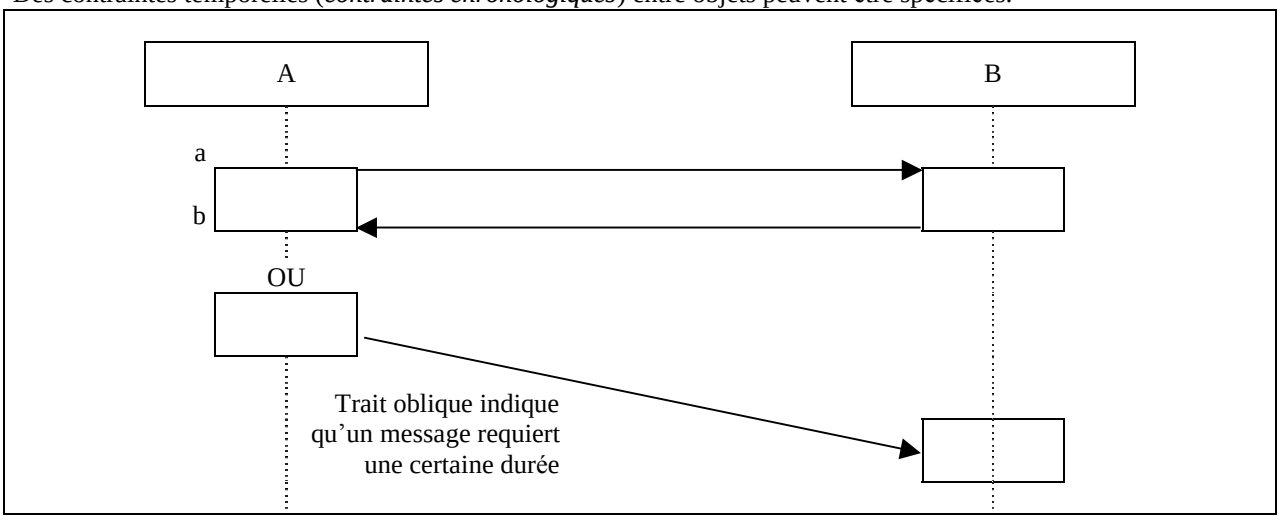
a) Création et destruction d'objets

Si un objet est crée au cours de l'exécution d'un scénario, celui-ci n'apparaît qu'au moment où il est crée. Si l'objet est détruit dans un scénario, le destruction est représenté par une « X ».

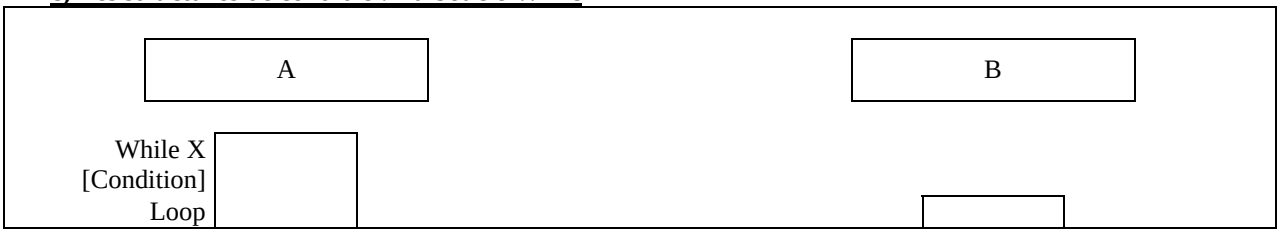


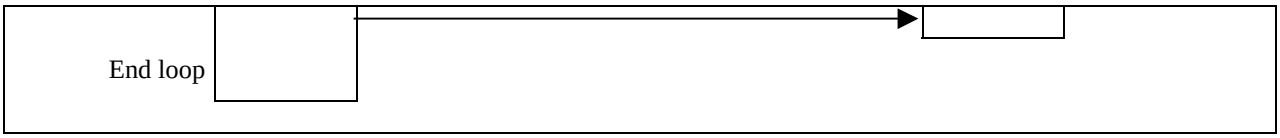
b) Contraintes temporelles

Des contraintes temporelles (*contraintes chronologiques*) entre objets peuvent être spécifiées.



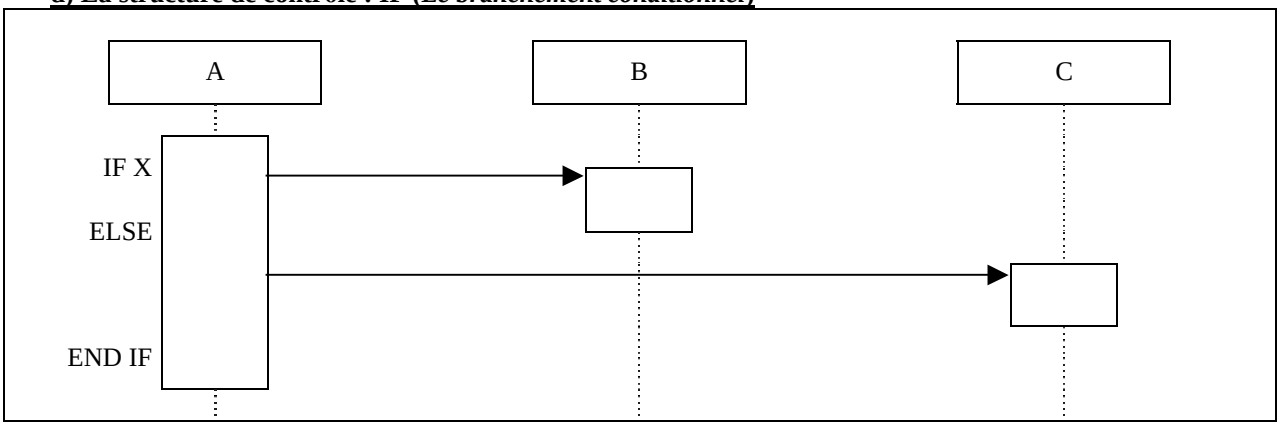
c) Les structures de contrôle : La boucle While





On indique à l'aide de cette structure que tant que la condition X est vérifiée, A envoie un message à B

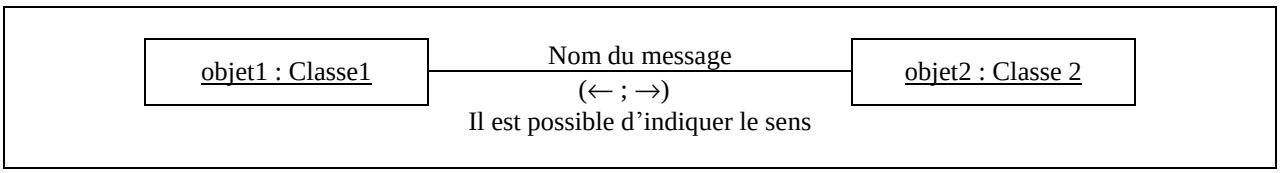
d) La structure de contrôle : IF (*Le branchement conditionnel*)



V.6 Le diagramme de collaboration

Le diagramme de collaboration constitue une autre représentation de l'interaction entre objets, en mettant l'accent sur les relations entre un ensemble d'objets et de messages échangés.

6.1 Formalisme



Les relations entre objets sont représentées par un trait continu entre les objets. Les messages et le sens des échanges sont indiqués par une flèche qui est orientée de l'émetteur vers le récepteur. Il existe deux types de messages :

- Synchrone : Émetteur reste en attente de la réponse à son message. Ce type de message se symbolise par une flèche de type : →
- Asynchrone : dans ce cas, l'émetteur n'attend pas la réponse ; il poursuit l'exécution de ses opérations. Ce type de message se symbolise par une flèche de type :⇀

a) Formalisme associé aux messages

D'autres précisions peuvent être associées à un message :
« Synchronisation / Condition / n° / : type d'envoi / :résultat := / message (paramètres) »

- Synchronisation [pre] indique la liste de messages à envoyer au préalable
- Condition [Cond] définit les conditions d'acheminement des messages
- n° permet de hiérarchiser les messages
- type d'envoi définit le type séquentiel ou parallèle
- Résultat [re] : Valeur à retourner suite à la prise en compte du message, nom du message renvoyé et paramètres associés.

Exemple :
1.5[x>10]1.9. Créer

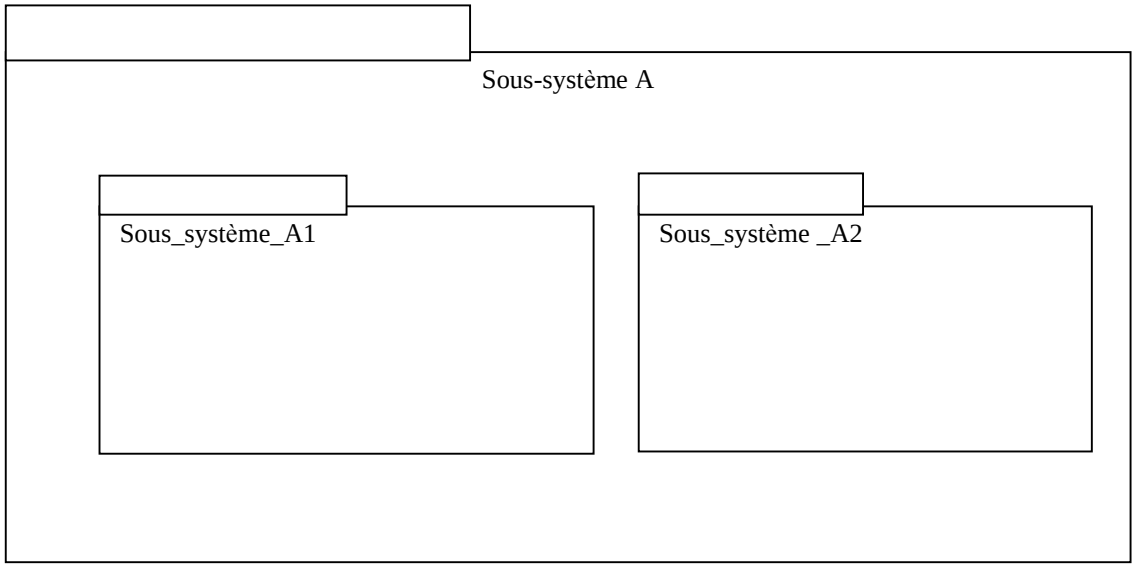
- Envoi du message 1.5
- Le message est envoyé si x>10
- Message numéro 1.9
- Déclencher l'opération « créer », activée sans paramètres

V.7 Le diagramme de composants

Un système informatique comprend des composants propres à l'environnement de réalisation choisi. Ces composants sont :

- Le sous-système
- Le module
- Le programme et sous-programmes
- Le processus
- Et la tâche

7.1 Le sous-système

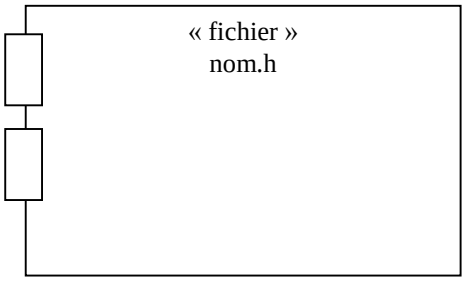


7.2 Le module

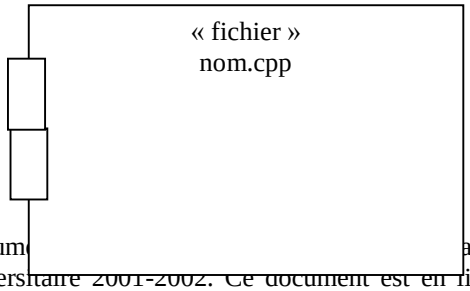
Un sous-ensemble peut être décomposé en modules. Chaque module correspond à un ensemble d'éléments physiques (fichiers, sous-ensemble logiciel…). Chaque classe du modèle logique est réalisé par deux composants : la spécification (ou interface) et le corps (réalisation de la classe).

a) Formalisme

- Spécification :



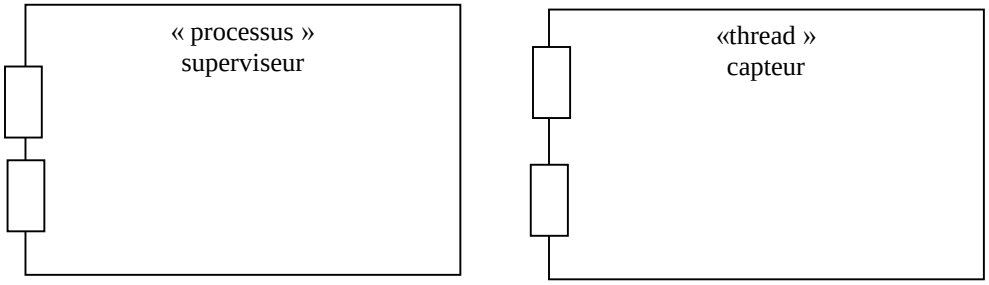
- Corps :



7.3 Processus et tâche

Chaque module peut être décomposé en processus et tâches (« processus », « thread »).

a) Formalisme

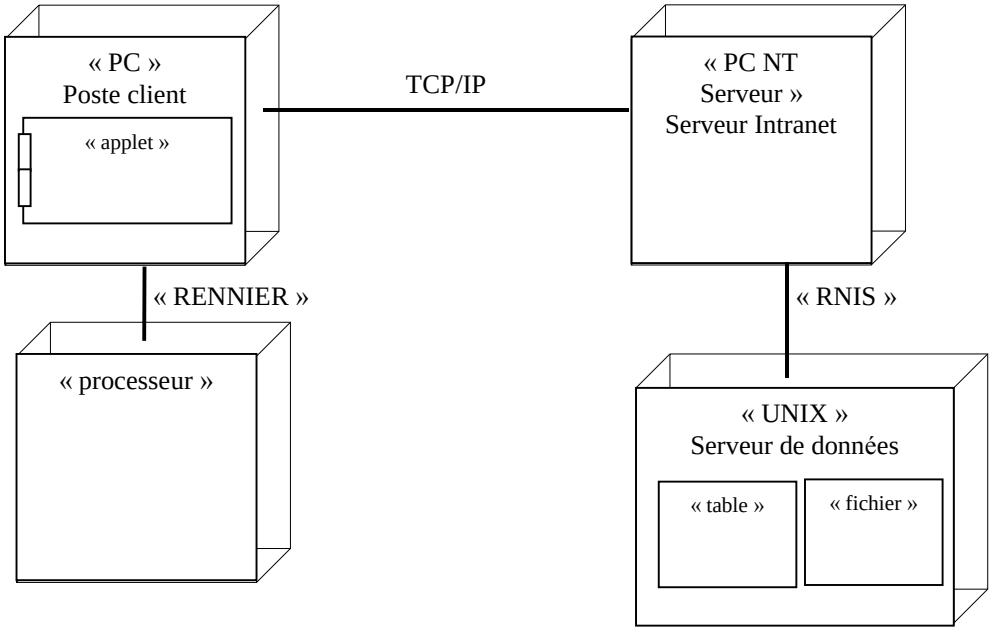


V.8 Diagramme de déploiement

Le diagramme de déploiement représente l'architecture technique des composants matériels (serveur, périphériques, communication…).

- Processeur : « processeur », « pc », « UNIX », « « Linux »…
- Périphériques : « poste client », « dispositif », « mémoire »…
- Communication : « protocole »

a) Formalisme



VI La démarche simplifiée pour l'analyse en sept étapes

1. Élaboration d'un diagramme de contexte du système à étudier (domaine)
2. Identification et représentation des cas d'utilisation (DCU)
3. Description et représentation des scénarios (DES, DCO)
4. Identification des objets et des classes (DOB, DCL)
5. Élaboration du DCL et DOB

6. Élaboration du diagramme État/transition (DET) et d'activité (DAC)
7. Consolidation et vérification des modèles

VII Positionnement de MERISE et UML

MERISE	UML
Points communs	
Entités (objet)	Classes
Propriétés (attributs)	Attributs
Occurrence	Objet, instance
Cardinalités	Multiplicité
Sous-type	Sous-classe
Généralisation /Spécialisation	Généralisation / Spécialisation
Relation	Association
Acteurs	Acteurs
Flux	Messages
Les points divergeant	
Domaine informatique de gestion	Domaine informatique technique (temps réel)
Les concept spécifiques	
MLD	L'agrégation
MCT	Polymorphisme
MOT	Encapsulation
	DCO et DSE
	Diagramme de déploiement
	Typage des langages objets
Sur la démarche	
Découpage du processus de développement en étapes et phases de type « cascades ». L'étape suivante ne peut être commencée que si l'étape précédente est validée	Démarche itérative et incrémentale fondée sur la réalisation de prototypes successifs.

Définitions

A

Abstraction : Faculté des humains de se concentrer sur l'essentiel et d'oublier les détails. Parfois employé comme synonyme de classe.

Abstraite : Se dit d'une classe qui ne peut pas être instanciée directement

Acteur : Classe de personnes ou de systèmes qui interagissent avec un système ou Objet toujours à l'origine d'une interaction.

Action : Opération exécutée lors d'une transition d'un état à un autre état. Une action ne peut pas être interrompue.

Activité : Opération exécutée au sein d'un état. Une activité peut être interrompue.

Agent : Objet qui peut être origine et destination d'une interaction

Agrégation : Forme d'association non symétrique qui exprime un couplage fort et une relation de subordination.

Algorithme : Enchaînement des actions nécessaires à une tâche.

Analyse des besoins : Détermination des besoins des utilisateurs

Analyse du domaine : Partie de l'analyse qui se concentre sur l'environnement de l'application

Analyse informatique : Discipline qui explique comment déterminer un cahier des charges qui décrive précisément et exactement le besoin.

Analyse : Détermination du *QUOI* et du *QUOI FAIRE* d'une application.

Application : Système logiciel élaboré dans un but précis.

Architecture : Art de construire les logiciels, structure d'un logiciel.

Artefact : Élément d'information, produit ou utilisé lors d'une activité de développement logiciel (modèle).

Association dérivée : Association déduite d'autres d'associations.

Association : Relation entre classes qui décrit un ensemble de liens.

Asynchrone : Forme de communication non bloquante et sans accusé de réception

Attribut de lien : Attribut appartenant à un lien entre objets, plutôt qu'aux objets eux-mêmes.

Attribut dérivé : Attribut déduit d'autres attributs.

Attribut : Type d'informations contenues dans les objets d'une classe(propriété).

B

C

Cardinalité : Nombre d'éléments dans une collection.

Cas d'utilisation : Technique d'élaboration des besoins fonctionnels, selon le point de vue d'une catégorie d'utilisateurs.

Classe Abstraite : Classe qui ne s'instancie pas directement et qui sert uniquement de spécification.

Classe active : Classe dont les objets sont actifs.

Classe paramétrable :Classe modèle pour la construction de classes.

Classe racine : Classe au sommet d'une hiérarchie des classes (synonyme de classe de base)

Classe template : Voir classe paramétrable

Classe utilitaire : Classe dégradée, réduite au concept de module.

Classe : Description abstraite d'un ensemble d'objets ; réalisation d'un type.

Classe-association : Association promue au rang de classe.

Classification : Action d'ordonner dans le but de comprendre.

Clé naturelle : Clé primaire issue du domaine de l'application.

Clé primaire : Information désignant un objet de manière bijective.

Clé : Type d'attributs qui réalise la restriction d'une association

Collaboration : Se dit d'une interaction entre objets réalisée dans le but de satisfaire un besoin utilisateur. Ou bien élément structurant d'UML pour la description d'une interaction

Collection : Terme générique désignant tous les regroupements d'objets sans préciser la nature du regroupement.

Composant : Élément physique constitutif d'une application, représenté principalement dans la vue de réalisation.

Composition : Agrégation par valeur

Conception : Détermination du *COMMENT*.

Constructeur : Opération de classe qui construit des objets.

Contrainte : Relation sémantique entre éléments de modélisation qui définit une condition qui doit être vérifiée par les éléments concernés

Cycle de vie : Étapes du développement et ordonnancement des travaux.

Cycle : Passage complet par les quatre phases de la vue de l'encadrement (phase de démarrage, d'élaboration, de construction et de transition).

D

Décomposition : Séparation en éléments plus petits dans le but de réduire la complexité.

Dépendance : Relation d'obsolescence entre deux éléments de modélisation

Domaine : Synonyme de *Champ d'application*.

E

Encapsulation : Occultation des informations contenues par un objet.

Entité : Terme emprunté aux méthodes de modélisation par les données : il est employé comme synonyme d'objet.

État : Situation instantanée dans laquelle se trouve un objet, un système.

Étend : Relation d'extension entre cas d'utilisation.

Étude d'opportunité : Phase de la vue d'encadrement durant laquelle la vision du produit est définie.

F

Factorisation : Identification puis extraction de similitudes entre classes. Permet d'élever le niveau d'abstraction

Flot d'exécution (ou flot de contrôle) : Description de la répartition de l'activité entre objets.

Fonction : Forme de sous-programme qui renvoie à un objet, une donnée. Ou bien, expression du besoin en termes impératifs, sous la forme de tâches à accomplir.

G

Généralisation multiple : Forme de généralisation dans laquelle une classe dérive de plusieurs ancêtres. Souvent synonyme d'héritage multiple.

Généralisation : Point de vue ascendant porté sur une classification ;nom donné par UML à la relation de classification utilisée pour construire des hiérarchies de classes ; souvent synonyme d'héritage.

Génie logiciel : On appelle génie logiciel la discipline qui permet de concevoir un logiciel selon des méthodes (modularité, structuration, programmation descendante) assurant ainsi la conformité d'un logiciel par rapport au cahier des charges.

H

Héritage multiple : Relation entre classes qui permet le partage de propriétés définies dans plusieurs classes.

Héritage : Relation entre classes qui permet le partage de propriétés définies dans une classe.

I

Identité : Caractéristique fondamentale d'un objet qui le distingue de tous les autres objets.

Instance : Une entité créée à partir d'un classificateur

Interaction : Description d'un comportement dans le contexte d'une collaboration.

Interface : partie visible d'une classe, d'un groupe d'objets.

Itération : Action de traverser une collection d'objets. Ou bien, séquence d'activités de la vue technique qui aboutit à la livraison d'un prototype exécutable.

J

K

L

Liaison dynamique : Association, entre un nom d'objet et une classe, réalisée à l'exécution.

Liaison statique : Association, entre un nom d'objet et une classe, réalisée à la compilation.

Lien : Connexion sémantique entre un type d'objet par laquelle un objet peut communiquer avec un autre objet.

M

Message : Élément de communication entre objet qui déclenche une activité dans l'objet destinataire ; la réception ou la livraison d'un message correspond à un événement.

Métaclasse : Classe d'une classe ; elle contient les données et les opérations qui concernent la classe plutôt que les instances de la classe.
Métamodèle : Modèle qui décrit des éléments de modélisation.
Méthode : Ensemble de démarches raisonnées pour parvenir à un but.
Module : Espace lexical dans lequel d'autres constructions peuvent être déclarées.
Monomorphisme : Concept de la théorie des types, selon lequel un nom ne peut référencer que des objets de la même classe.
Multiplicité : Désigne le nombre d'objets qui peuvent participer à une relation.

N

Navigabilité : Qualité d'une relation qui permet le passage d'une classe à l'autre

O

Objet : Entité atomique constituée d'un état, d'un comportement et d'une identité ; un objet est une instance de classe.
Opération : Élément de comportement des objets, défini de manière globale dans la classe.

P

Paquetage : Élément d'organisation des modèles.
Partie privée : Partie de spécification d'une classe qui regroupe des propriétés invisibles de l'extérieur
Partie protégée : Partie de la spécification d'une classe qui regroupe des propriétés invisibles de l'extérieur, sauf pour les sous-classes.
Partie publique :Parie de la spécification d'une classe qui regroupe des propriétés visibles de l'extérieur
Persistance : Qualité d'un objet à transcender le temps ou l'espace.
Phase :Ensemble des activités entre deux points de contrôle d'un processus de développement.
Polymorphisme : Concept de la théorie des types, selon lequel un nom peut référencer des objets, instances de plusieurs classes regroupées dans une hiérarchie de généralisation.
Projection : Relation entre un ensemble et un sous-ensemble.
Propriété : Caractéristique d'un élément de modélisation.
Prototype : Résultat d'un itération ;version partielle d'un système.

Q

R

Responsabilité : Obligation d'une classe ;partie de sa raison d'être.
Revue : Révision formelle d'une documentation, d'un modèle.
Rôle : Extrémité d'un relation ;par extension, manière dont les instances d'un classe voient les instances d'une autre classe au travers de la relation.

S

Sémantique : Règles d'utilisation et de contrôle de cohérence associées aux éléments de modélisation.

Serveur : Objet qui n'est jamais à l'origine d'une interaction.

Sous-classe : Classe spécialisée

Spécialisation : Point de vue descendant porté sur une classification.

Stéréotype : Création d'un nouvel élément de modélisation, basé sur la notion de module et la description des flot de données entre ces modules.

Super-classe : Classe générale.

Synchrone : Forme de communication bloquante, avec accusé de réception implicite.

Synchronisation : Expression de la forme de communication entre deux objets.

T

Template : Traduction de *classe paramétrable*.

Temps réel : Caractéristique d'un logiciel dont le temps de réponse est compatible avec la dynamique du système physique.

Test : Ensemble de mesures et des activités qui visent à garantir le fonctionnement satisfaisant du logiciel.

U

V

Visibilité : Niveau d 'encapsulation des attributs et des opérations dans les classes.

Vue : Manière de regarder des les éléments de modélisation éventuellement issus de modèles différents.

W

X

Y

Z

Index

(Merise de 1 à 29 & UML de 30 à 60)