Analyse

Mes (très brève) notes du cours d'analyse

Bloc 1

Les cours d'analyse du bloc 1

Introduction

L'*analyse* est le procédé de raisonement qui va de la connaissance desparties à celle du tout. C'est l'opposé de la synthèse. Étude et examen sont des synonymes.

Il s'agit de déterminer les contours et le fonctionnement général de quelque chose avant de le construire. Par exemple à l'aide de dessins, plans, maquettes, etc. Cette description permet de communiquer avec les clients et les concepteurs.

En informatique il s'agit d'identifier les besoins d'un organisme ou d'un ensemble d'utilisateurs. De représenter ces besoins avec des modèles spécifiques et de faciliter la communication avec les différents acteurs (clients, analytes et concepteurs)

Les raisons d'échec d'un projet sont souvent liée à une mauvaise définition des objectifs métiers et de la formulation des besoins.

- Périmètre (ce qu'il faut faire)
- Business rules (les règles métier)
- Data modelling (les données nécessaire)
- UX design (l'expérience utilisateur)

Les règles métier sont les règles du fonctionnement de l'entreprise, comment elle fonctionne, son organisation, etc à fin de la rendre plus efficace.

L'analyste va identifier les besoins en rencontrantles clients, utilisateurs, sponsors, etc. Il va consulter les ressources existantes tel que la documentation, il pose des question et met en évidence les incohérences.

Ensuite l'analyste crée des modèles sur différents aspects du système (données, processus, fonctionalités, UI, etc) pour engager la discussion.

L'analyste va ensuite faire valider ses modèles en discuttant avec l'équipe avec l'aide d'autres documents (cahiers des charges, spécifications fonctionnelles, etc) et peut aussi participer aux aspects budgétaires (évaluation de la charge de travail)

Un modèle en informatique a pour objectif de structurer les informations et activités d'un système (données traitements, flux d'informations). Il a pour but de couvrir les points les plus importants. C'est un outil de communication et d'information sur le projet.

L'analyste estla personne qui sert de pont entre les clients et les concepteurs. Il doit pouvoir faire la "traduction" du client au programmeurs. L'analyste se doit d'être précis et rigoureux pour déceler les trous dans le discours du client. Il doit être un bon négociateur et communiquant.

Sa place dans l'organisation depends de l'entreprise, du projet et de la méthodologie utilisée.

La méthode de l'analyse est de définir les besoins, déterminer la faisabilité et la cohérence de la demande et de structurer les données. Le cours va être plus basé sur la partie identifications des besoins et modélisation de données et interface.

Le but du cours est de décortiquer et interpreter le discour du client, de maitriser les concepts, outils et vocabulaire et de pouvoir modéliser des systèmes complexes en suivant une méthode.

Bloc 1

Diagramme de Use Case

- Wikipedia - Unified Modelling Language
- Wikipedia - Use case diagram

Le but d'un diagramme Use Case est de décrire visuellement les besoins d'un organisme (intéraction des aceturs avec le système et ses fonctionalités). Il vient de l'UML (Unified Modelling Language)

On va ainsi définir quelles sont les fonctionalités (dans le cadre de l'analyse, donc seulement les principales) et qui peut y accéder, pour y faire quoi et comment.

Voici les différents éléments du diagramme :

- Le système est représenté par un rectangle délimitant le périmètre.
- Les acteurs externes (éléments externent qui intéragit avec celui ci), sont représenté par leur rôle sous forme de bonhomme baton (stickman). Les acteurs non humains sont définis par un rectangle libéllé
  - Les acteurs primaires qui initient les CUs (cas d'utilisation) sont placés à gauche du système
  - Les acteurs secondaires qui aidentà la réalisation des CUs mais qui n'en sont pasles bénéficiaires à droite.
  - Une flèche peut être tracée d'un acteur à un autre si le premier acteur peut faire tout ce que le deuxième peut. Exempe: Administrateur → éditeur
- Les cas d'utilisations (CU) sont représenté par des ovales libéllé. Le nom contient un verbe d'action à l'infinitif, choisi du point de vue de l'utilisateur, réalise un service du début à la fin. Exemple: « *Lire un message* ». Il sont aussi reliés avec les acteurs avec une ligne

Ce diagramme permet d'avoir une vision globale des fonctionnalités du système, compréhensible par tout le monde, même non initié.

Ensuite il y a la « *Spécification textuelle* » qui décrit avec plus de détails les cas d'utilisations. Le texte doit fournir, pour chaque fonctionnalité:

- Le même nom que dans le diagramme (ex: Lire un message)
- L'identification de l'évènement déclencheur (ex: L'utilisateur veut lire ses nouveau messages ou relire d'anciens messages)
- La description de la fonctionnalité. La description doit être suffisament générale pour ne pas être dépendante sur l'interface.
- L'identification du/des acteurs impliqué(s) (ex: l'utilisateur).

- Les éventuelles pré/post conditions (ex: L'utilisateur doit être authentifié et une fois lu les messages ne doivent plus apparaitre en gras). Les post conditions sont toutes les choses qui doivent avoir été fait pour que l'UC soit considéré comme terminé.

En pratique

Pour faire le diagrame :

1. Trouver quel est le système de la situation. Il s'agit bien du système informatique. Donc il faut éviter de représenter un élément comme "Bibliothèque" ou "nom de la société" et être plus précis comme "Système de gestion biliothécaire" ou "Site marchant BeGood". Créer un rectangle libéllé représentant le système.
2. Trouver les acteurs primaires et secondaires. Les acteurs sont ceux qui interagissent *directement* avec le système. Les acteurs primaires (qui initient les UC) sont à droite du système, les acteurs secondaires (qui aident à la concrétisation du UC mais qui n'en bénéficie pas) sont mis à droite. Les acteurs humains sont représentés par des stickman tandis que les non humains peuvent être représenté par un petit rectangle libellé. Le nom doit être un *rôle*, et pas un nom ou personne en particulier.
3. Pour chaque acteur noter les actions possible en excluant toutes celles qui sont en dehors du système. Essayer de le synthétiser un maximum, mais tout en incluant toutes les actions dans le système. Les UC doivent commencer par un *verbe à l'infinitif*. (Relire les consignes en se mettant dans la peau de chaque acteur pour voir ce qu'iel pourrait faire). Puis ensuite relire les UC pour vérifier leur portée (mot "Gérer" par exemple)
4. Lier les acteurs avec les UC via une ligne (sans flèche), et lier les acteurs entre eux quand un role peut faire tout ce qu'un autre role peut faire (administrateur --> utilisateur par exemple)

Pour faire la spécification textuelle (à faire par UC) :

1. Définir quel est l'élément déclencheur, cela peut être un souhait ou un désir d'un acteur, ou encore lié au résultat d'un autre UC.
2. La description doit être complète mais pas trop longue et ne doit pas dépendre de l'interface utilisateur (qui peut changer dans le temps).
3. La liste des acteurs impliqués.
4. Les préconditions (tel que l'authentification par exemple), pour que le UC prenne place.
5. Les postconditions, qui définissent l'état du système après le UC. C'est à dire ce qui doit avoir changé pour que le UC soit défini comme terminé.

> Note: Le mot "gérer" fait référence à CRUD (créer, lire, mettre à jour et supprimer) en analyse

Bloc 1

Diagramme activité

Le diagramme d'activité sert à donner une vision globale et temporelle d'une partie dynamique d'un système. C'est a dire un enchainement d'activité que l'utilisateur ne voit pas forcément et qui sont opérée par un système.

Cela peut servir à modéliser un algorithme, la dynamique d'un cas d'utilisation, ou un "processus métier".

Fonctionnement du diagramme

Distributeur de billets

Image not found or type unknown

Le diagramme est divisé en 2 colonnes (qui sont appellées swimlanes), la colonne de gauche sert à représenter les intéraction avec le système. Tandis que la colonne de droite est la dynamique du système en elle même. Cela permet de modéliser les actions entre le systèmes et les différents acteurs.

Le début et la fin sont symbolisés par un point noir ou un point noir entouré. Pour symboliser la fin d'un seul flux seulement on utilise un rond avec une croix dedans.

Il y a ensuite différents "noeuds" sur notre diagramme :

Symbole	Nom	Description
Ovale	Noeud d'exécution	Fait une action
Losange (1 input, plusieurs output)	Noeud de décision	En fonction d'une condition, elle va continuer dans un ou l'autre direction. Ses conditions sont représentés sur les lignes
Losange (plusieurs input, 1 output)	Noeud de fusion	A l'inverse d'un noeud de décision qui permet de diviser le processus en plusieurs possibilités, le noeud de fusion va recombiner les différentes possibilités.
Barre (1 input, plusieurs output)	Fork	Cela permet de créer des flux parallèles (des actions qui vont se déclencher simultanément)
Barre (plusieurs input, 1 output)	Join	Fait l'inverse de fork en recombinant les flux parallèles en un seul

Symbole	Nom	Description
Un genre de sablier	Evenement temporel	Modélise un évènement qui se déclenche à un moment prédéfinis (par exemple fin du mois)
Un rectangle avec une flèche et avec une "réception de flèche"	Envoi et réception d'un signal	Symbolise une signal asynchrone. C'est a dire que le programme va envoyer un signal vers un autre thread et attendre qu'une tache soit effectué avant de continuer

Bloc 2

Les cours d'analyse du bloc 2

Bloc 2

Introduction

Buts du cours

- Analyser les besoins d’un·e client·e et modéliser une situation
- Concevoir un logiciel de qualité
  - Correct, réponds au spécifications (soit ne pas faire quelque chose dont lea client·e n’a pas besoin)
  - Robuste, résiste aux situations difficiles
  - Extensible, peut être amélioré et étendu
  - Réutilisable, peut être réutilisable par d’autres logiciels. Soit éviter de réinventer la roue.
- Communiquer de manière claire
- Approfondir notre connaissance du langage UML (type diagrammes vu en bloc 1). L’UML est principalement un moyen de communication pour l’équipe de développement (et pour certain diagrames comme les use cases, avec lea client·e également).

Contenu du cours

- Modélisation des données
- Compléments UML (diagrames use cases, classes, séquences, etc)
- User stories (description d’une fonctionalité à implémenter) et IHM (Interface Human Machine, maquette de l’interface pour communiquer avec lea client·e)
- Méthode d’analyse (agile et scrum)
- Designs patterns (moyens de résoudre des problèmes en informatique quelque soit le langage de programmatoin)

Projet professionel

- Modélisation des besoins d’une vertiable entreprise de notre choix
- Il est très recommandé de ne pas l’avoir en seconde sess car on a un suivi durant l’année mais pas pendant la seconde session

- Travail effectué en groupe de 2
- Remise d’un rapport avec défense orale (pour poser des questions sur le rapport, et pour s’entrainer pour la défense orale du bloc 3), vers le 20 décembre

Examen

- QCM rapide
- Modélisation de situations, approfondissement des diagrammes dont on a déjà parlé en bloc 1 (use cases, séquence, classe, etc)

Types de métiers (débouchés de la formation)

- Administration
  - Administration système (monitoring, backup, restore, installation des logiciels et périphériques, gestion des comptes utilisatuers, droits d’accès, upgrade, etc)
  - Administration réseaux (extension des réseaux, sécurité, politique d’accès)
  - Administration base de données (utilisateurs, droits d’accès, configuration, fine-tuning, validation des scripts, monitoring des performance et volumes)
- Support (première ligne, assistance technique pour utilisateur·ice·s, suivi d’incidents et pannes, procédures, guides utilisateur·ice·s, formation utilisateur·ice·s)
- Informatique décisionnelle/Business Intelligence, prendre des décisions sur base de données en utilisant des programmes (collecte d’informations, reporting, recommendations, etc)
- Formation (concevoir des supports, dispositifs pédagogiques, évaluation, etc)
- Développement de logiciels (chefs de projets, analyste, architecte (définition de la structure de l’application, va définir quel langage, quel frameworks vont être utilisés, etc), dévelopeur, testeur)
  - Lea **chef de projet** est responsable du projet, toujours présent et suis tout le processus, calculer le budget, supervise le développement, organise les équipes de développement, contact avec lea client·e. Iel doit avoir des aptitudes techniques, une bonne communication, un bon leadership et de la rigueur.
  - L’**analyste** doit comprendre les besoins du client, traduire les besoins en modèles, est le lien entre l’équipe de développement et lea client·e. Iel doit avoir une bonne communication, des aptitudes techniques, de la rigueur et une bonne compréhension du domaine du problème ou une capcité à aquérir cette compréhension rapidement (l’informatique est rarement une fin en soit, on fait des programmes pour résoudre des problèmes d’autres personnes. Iel doit donc

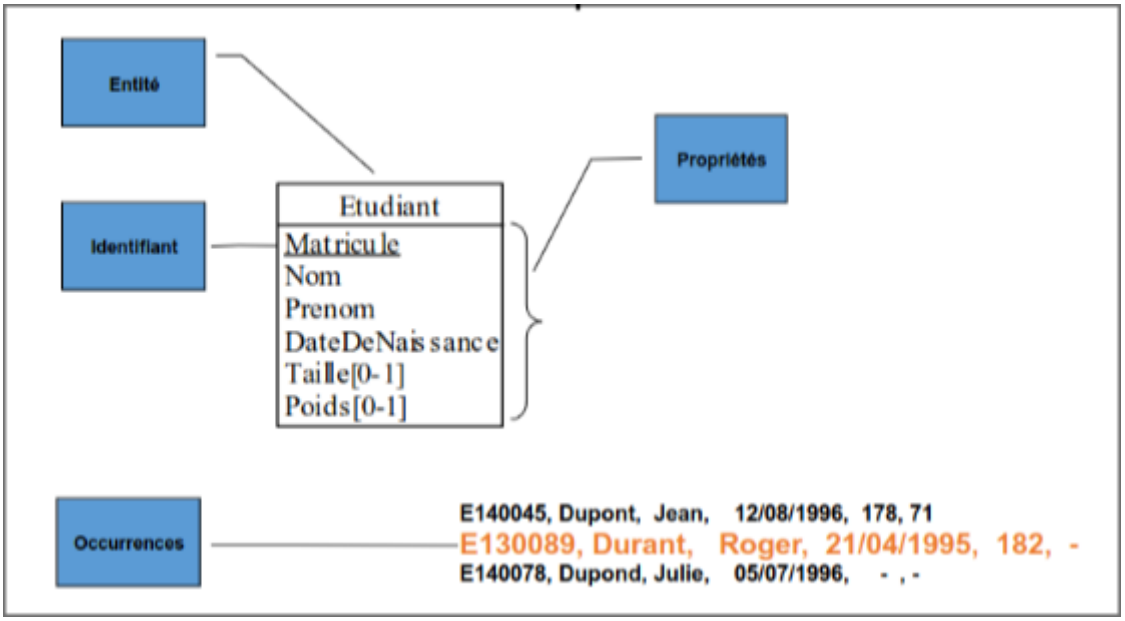
s’imerger dans le domaine du client pour comprendre comment résoudre le problème. Le but est de faire gagner du temps au utilisateur·ice·s).
- L’**architecte** va gérer les aspects techniques du projet (structure du logiciel, réutilisation des composants, langages, serveurs, frameworks, etc) en fonction des exigences établies par les analystes (performance, contraintes économiques, *montée en charge*). Iel sera le point de référence technique pour l’équipe de développement et a une collaboration étroite avec le chef de projet (pour le budget et le planning). Iel doit avoir de bonnes compétences techniques, bonne connaissance du paysage IT en place, bonne communication, leadership technique et rigueur. *Voir tiobe*
- Lea **développeur·euse** va écrire le code sur base du travail de l’analyste et de l’architecte, tester le code, automatisation des tests (intégratoin continue), et collaborer avec l’analyste et l’architecte pour détecter des lacunes dans l’analyse, valider le respect de l’architecture.
- Lea **testeur·euse** doit faire des tests fonctionels (fonctionalités demandées), tests de performance (utilisateurs concurrents, bande-passante, pannes, etc), de robustesse (memory leaks, gros volume de données, etc), de vulnérabilité (resistant à differentes attaques, injection SQL, *fuzzing*). Iel doit communiquer les résultats avec lea chef de projet et l’équipe de developpement. Iel doit être rigoueu·euse, méthodique, avoir un bon esprit critique et une bonne communciation et diplomacie.

Mais il y a également plein d’autres nouveaux métiers en lien avec l’intégration continue, les réseaux sociaux, le cloud, le développement mobile, l’intelligence artificielle, etc.

Bloc 2

MCD (Modélisation conceptuel des données)

Il est très important de savoir bien modéliser les données



Outils

- DB-main, est un logiciel de l'université de Namur permettant de faire des MCD
- Looping, est le logiciel utilisé en B1 pour faire les MCD (fonctionne aussi sur Linux et macOS avec Wine)
- Mermaid, qui fonctionne avec du texte/code (attention un MCD s'appelle un ER dans la doc), qui est open source, gratuit et basé sur le web. Cependant la fonctionalité de l'héritage n'a pas encore été implémentée.

Modélisation des données

Tout système d’information manipule des données, les données doivent être et organisée, structurée de manière à faire apparaitre des dépendances. La modélisation des données propose une représentation abstraite des relations entre les différentes informations (entités). Ici on va plus précisément voir le modèle conceptuel de données qui permet de représenter toutes les données d’un systèmes et leurs relations de manière à plus tard le traduire dans un schéma de base de donnée.

Définition des éléments d’un MCD

Entité

Une **entité** représente un groupement d’informations communes à une classe d’objets (exemple, une entité User peut reprendre les informations “addresse email”, “numéro de téléphone” et “nom”). Chaque entité a un **nom** (exemple “User”) et des **propriétés** (exemple “nom”). Le concept d’entité et d’*association* sont suffisant pour modéliser n’importe quoi.

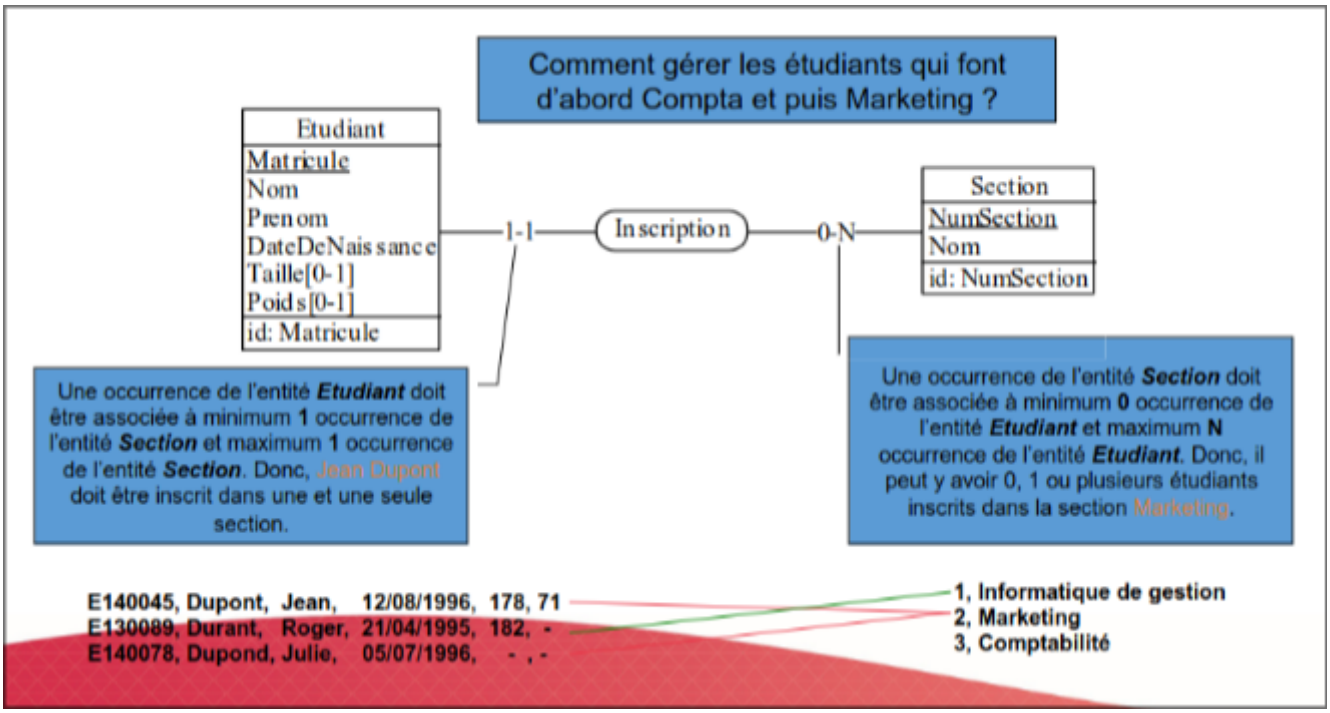
Propriétés

Les propriétés sont des caractéristiques qui dérivent les entités et qui doivent être pertinente pour le système d’information que l’on modélise (on ne vas pas mettre la taille des cheveux pour un forum). Les valeurs de ces propriétés dépendent donc de chacune des occurences de l’entité.

Identifiant

L’identifiant d’une entité est un ensemble minimal de propriétés (minimum 1 mais il faut en avoir le moins possible) permettant de distinguer une occurence parmis les autres occurences (existantes ou potentielles) de la même entité (exemple : propriété “id” par exemple).

Association



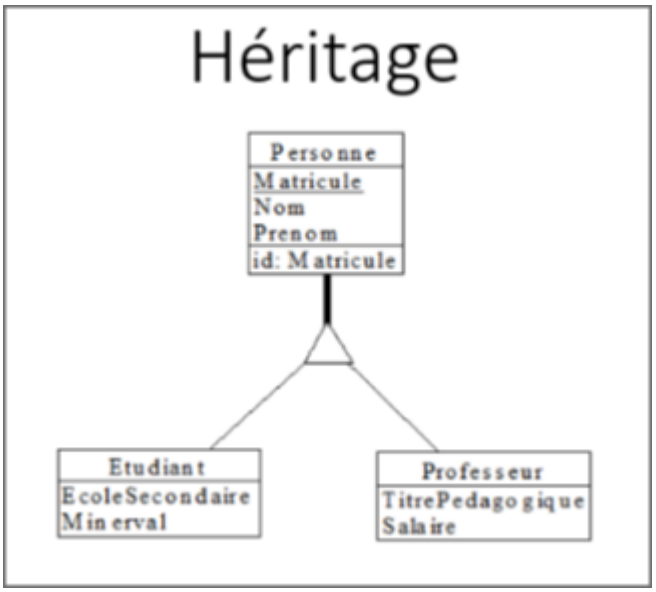
Une association est un lien logique entre 2 entités ou plus. L'association permet ainsi d'établir des liens entre certaines occurrences d'une entité. L'association elle même peut être porteurse de propriétés. L'identifiant d'une association est composé des identifiants des entités qu'elle relie (éventuellement augmenté des propriétés de l'association). Le nom d'une association est généralement un verbe liant les deux entités (par exemple "Utilisateur·ice *possède* Role")

Cardinalité

Les cardinalités d'une association indiquent le nombre minimum et maximum d'occurencenes de l'entité d'arrivées qui peuvent être associés à une occurences de l'entité de départ. Les cardinalités peuvent être 0, 1, ou n (n signifiant sans limite). Par exemple un le lien entre une entité "Utilisateur·ice" et une entité "Rôle"

- Est de 0,n dans le sens Utilisateur·ice → Role car un·e utilisateur·ice peut avoir 0 ou plusieurs rôles.
- Est de 0,n dans le sens Role → Utilisateur·ice car un rôle peut être relié à 0 ou plus utilisateur·ice·s

Héritage



Le MCD permet aussi de représenter l'héritage d'entités. L'avantage de l'héritage c'est que l'on hérite aussi des propriétés. Par exemple dans l'exemple ici, les étudiants et les professeurs hérites des attributs "matricules", noms et prénoms.

La notion d'héritage est surtout intéressante pour représenter des personnes par exemple.

> L'héritage n'est pas représentable dans Mermaid diagrams.

Créer un MCD depuis un énoncé

- Souligner les phrases avec les éléments inmportatnes (entités et informations pertinentes)
- Lister les questions auquelles le modèle pourra répondre (Exemple, *Peut-il déterminer la section d'un·e étudiant·e afin de pouvoir lui afficher ses cours ?*)
- Etablir la liste des entités et des liens qui les relient
- Créer le MCD sur bases des éléments listés précédemment

MCD to MLD

MCD vs MLD

Attention que le MCD (Modèle Conceptuel de Données) n’est pas la même chose qu’un MLD (Modèle Logiques de Données, aussi appellé Modèle Relationnel). Le MLD est exactement ce qui sera présent dans la base de données tandis que le MCD est purement conceptuel. Le MCD peut donc être converti en MLD, qui lui même peut être converti en code SQL.

Traduction

Une entité devient une table, une propriété devient un attribut (ou une colonne), un identifiant devient une clé primaire et une association deviennent des relations ou des clé étrangère.

Associations - Clés étrangères

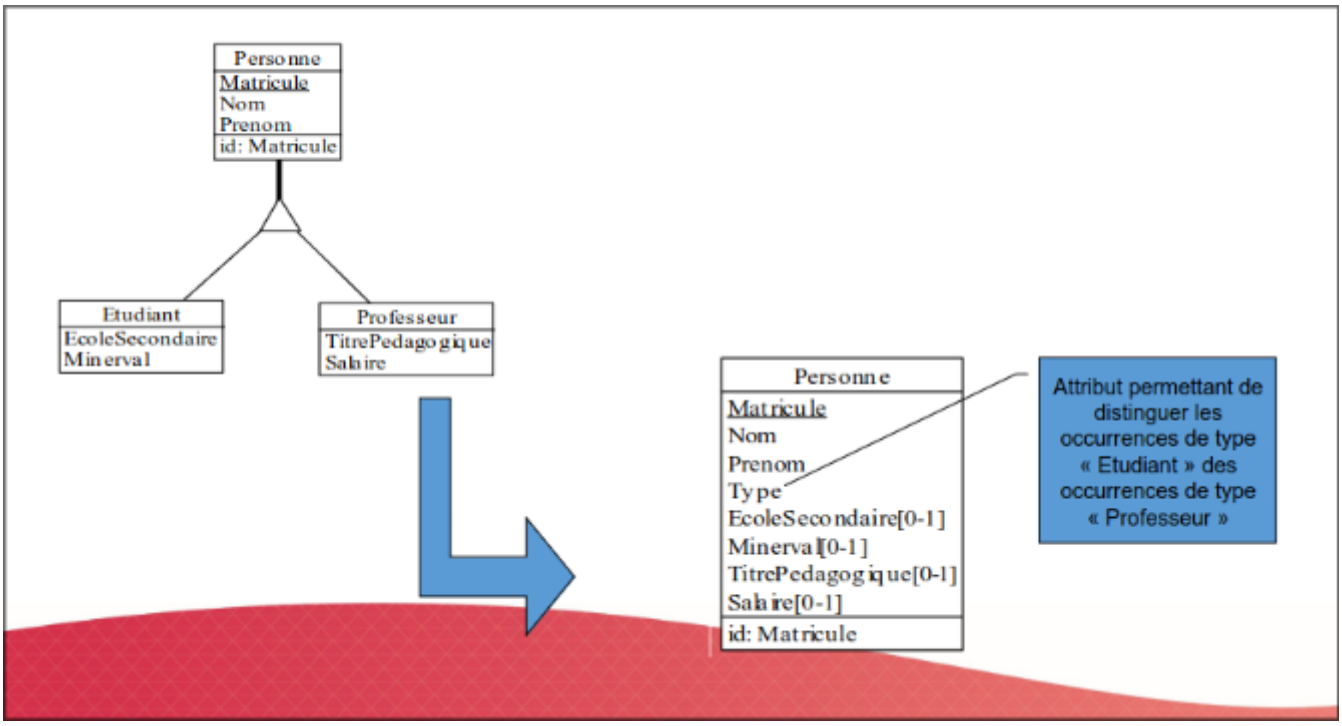
- Les cardinalités `1,1`, `0,1` ou `1,n` deviennent des clés étrangères
- Les cardinalités `0,n`, `1,n` ou `n,n` génèrent une table permettant de faire la liaison, la table a donc comme clé primaire les ids des deux autres tables.

Par exemple, pour chercher la relation entre un·e utilisateur·ice et un role, on va créer une table de liaison. On pourra donc aller chercher dans la table les ids de tous les utilisateur·ice associés à un role et inversément.

Héritage

Il existe plusieurs techniques pour traiter le cas de l’héritage.

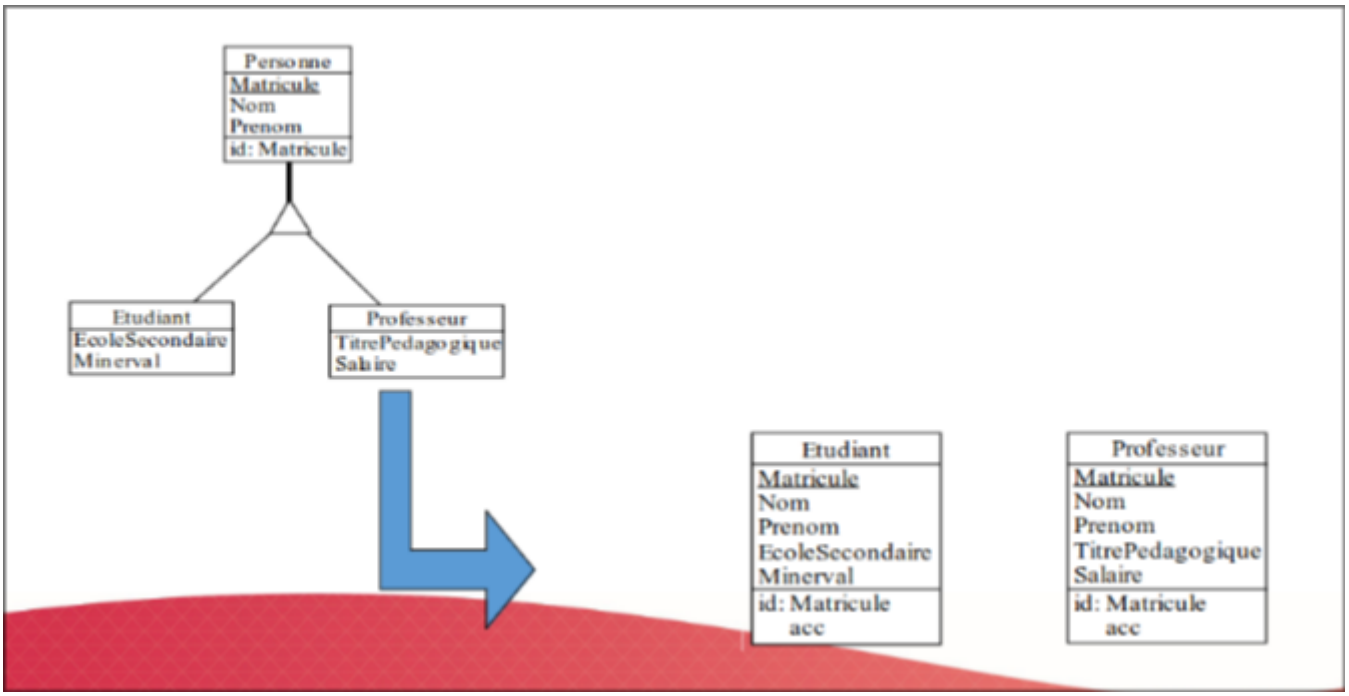
Héritage ascendant



Les entités générales et spécialisées fusionnent en une seule table qui aura tous les attributs et un attribut supplémentaire “type” permettant de définir quel spécialité c’est. Le problème c’est que beaucoup des attributs seront null.

Il faut généralement utiliser celle ci lorsque qu’il y a peu de propriétés.

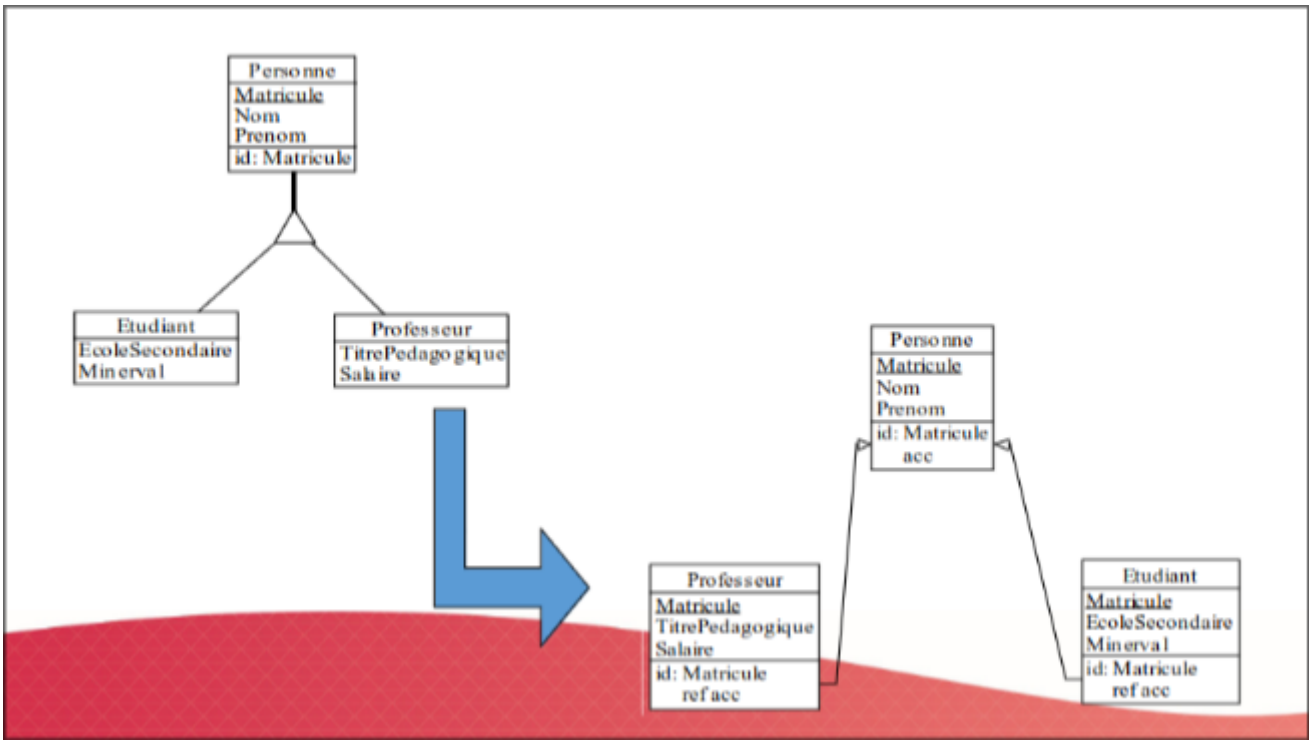
Héritage descendant



On crée des tables pour chaque entité spécialisée en y ajoutant tous les attributs de l’entité générale. Le seul “inconvénient” c’est que les attributs se retrouvent dupliqués.

Il faut mieux utiliser cette méthode lorsqu’il y a beaucoup d’attributs.

Héritage avec clé étrangère



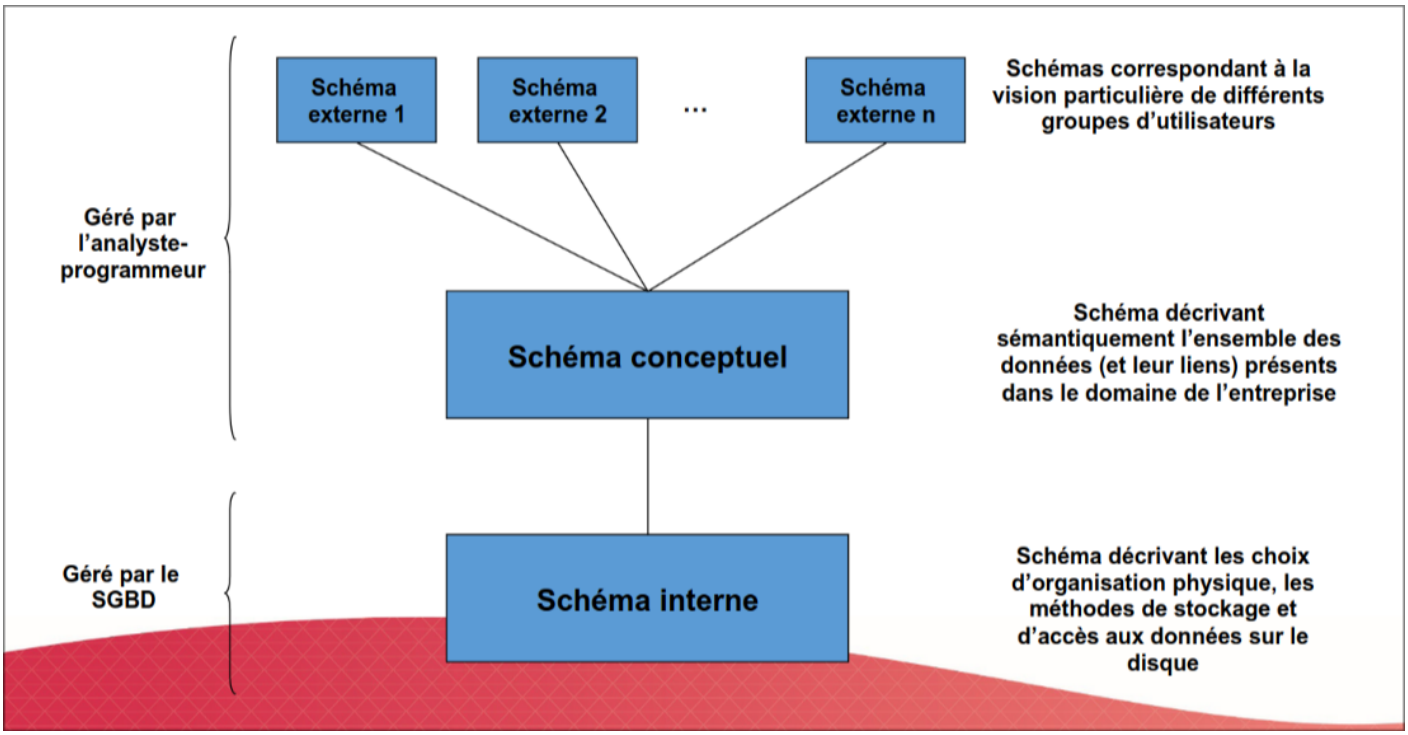
On crée une table générale et des tables sépécialisées. Chaque table spécialisée ayant un attribut lié à la table générale. L'inconvéninent est que cela nécessite de toujours utiliser des jointures pour avoir toutes les informations sur une occurence. Et que cela augmente la complexité des relations.

Il faut mieux utiliser celle ci si on veux accèder aux informations via l'entité générale. Par exemple dans le screenshot, accéder à la liste des personnes est plus simple car il suffit de lister la table Personnes, alors que dans l'héritage descendant il aurait fallu lire les tables Etudiant et Professeur.

Base de donnée

Une base de donnée est un système qui permet de mémoriser de façon durable les données sur un support physique (mémoire secondaire) Les données modélisent des objets du monde réel qui pourront être organisé de manière à faire apparaître les relations existant entre les objets. Les données doivent pouvoir être **CRUD** (lues, ajoutées, modifiées et supprimée - create, read, update, delete).

Trois niveaux de schémas



On crée plusieurs schémas pour plusieurs groupes d'utilisateur·ice·s et qui permettent aussi de parler avec le client. Enfin, on crée un schéma conceptuel reprenant l'ensemble des données et des liens. Enfin il est converti en MLD pour devenir le schéma interne de la base de donnée.

Bloc 2

Notes pour chaque diagramme + mini tuto pour EA

Procédure réponse examen analyse

1. Souligner les phrases et éléments importants (entités potentielles, fonctionalités, formulaires, informations, données, etc)
2. Lister les questions auquel le diagramme pourra réponse (exemple *Peut-on déterminer la section d'un·e étudiant·e afin de pouvoir lui afficher ses cours ?*)
3. Etablir la liste des éléments (entités, classes, dialog, controleurs, ou autres) et leur reponsabilités éventuelles (retenir XYZ)
4. Etablir les relations entre les différents éléments (entités, classes)
5. Etablir la liste des propriétés de chaque entité
6. Tester le diagramme selon les questions, vérifier qu'il respecte les règles d'UML et les flèches/associations.
7. Justifier toutes les décisions sur le diagramme ou sur le côté

MCD

- Attention associations
- Attention MCD != MLD (pas de tables d'association par exemple), pas de FK
- Pas oublier les clés primaires

Use Cases

- Utiliser de l'infinitif partout
- Pas de flèches entre les acteur·ice·s et les UC
- UC humains = stickman, UC non-humains = boites
- include = flèche vers un autre UC indispensable, vs extend = flèche vers un autre UC facultatif
- Description textuelle : Titre, résumé, acteurs, pré/post-conditions, déclencheur, scénario nominal, erreur et alternatif
- Scénario en tableau (colonne gauche = acteur, colonne droite = action du système)

Utilisation EA

1. Model > Add a model using wizard > basic use case diagram
2. Create element : Use Cases > Add element > toolset > usecase.
3. Add arrows and lines : Select the element then drag the arrow button to the other element then release and select the type of relation.
4. Switch between human actor and system actor : click on the element and click on the magnifying glass

Class

- Dépendence (A ..> B, A a besoin de B pour fonctionner)
- Association (Personne 1–utilise–0..* Voiture, Une personne a 0 ou plus de voiture, mais une voiture n'a qu'une personne), lien structurel durable entre des instances de classes. Flèche optionelle. Attention inversée par rapport au MCD.
- Aggrégation (Personne –<> Groupe, Personne fait partie de Groupe mais Personne et Groupe sont indépendant car si le groupe disparait, les personnes restent)
- Composition (User –<x> App, User fait partie intégrante de App et ne peut pas exister sans elle)
- Généralisation (Voiture –|> Véhicule, une voiture est une sorte de véhicule)
- Réalisation (Voiture ..|> Véhicule, une voiture a les opérations d'un véhicule)
- Penser aux patterns OOP (exemple, Repository, Strategy, etc)

Utilisation EA

1. Model > Add a model using wizard > starter class diagram
2. Configure > Settings > Code Engineering Datatypes… > Java > New > *String* > New > Close
3. Add new class : Starter Class Diagram > Add element > toolset > class
4. Add arrows and lines : select the element then drag the arrow button to the other element then release and select the type of relation

5. Add attributes or methods : Right click on class > Features & Properties > Attributes/Operations
6. Change multiplicity : for each side of the association, right click and click on multiplicity.
7. Create generic interface : Create a new interface, right click on it, select properties, go the templates tab and add a new one named "T". Then on the realization line, double click on it and write something like "T = String"

Séquence

- Boundary est la vue, control est le contrôleur et entity est une classe métier. Boundary et entity n'intéragissent jamais directement.
- Les valeurs de retours sont des flèches avec lignes pointilliées
- Les diagrammes de séquence systèmes représentes seulement les intéractions entre les acteurs et le système tandis que les diagrammes de séquence normaux (complets) représentent les intéraction entre toutes les classes métiers, les contrôleurs, les formulaires et les acteurs pour chaque fonctionalité.

Utilisation EA

1. Model > Add a model usig wizard > starter sequence diagram
2. Double click in void of the diagram > Features > Supress brackets for Operations without Parameters
3. To add actor, boundary, control or entity : Right click on "starter sequence diagram" > Add Element... > Toolset > interaction, then drag it to the diagram as a link
4. To add an interaction, select the origin, drag the arrow to the end point. Then double click on the arrow and complete message (add ~() ~ if you need to), parameters and return value. Eventually tick the "is Return" box to have dotted arrow.

Objet

- Diagramme d'objet représente différentes configurations d'instances d'objets du diagramme de classes métier.
- Chaque objet peut être nommé avec un nom d'objet et/ou une classe et/ou un état. Par exemple `nom d'instance :Classe [état]` ou `nom d'instance :Classe` ou `:Classe` ou encore `nom d'instance`
- Les valeurs des attributs peuvent être définie comme `attribut: type = valeur`, `attribut = valeur` ou encore `valeur`

Utilisation EA

1. Model > Add a model using wizard > started objet diagram
2. To add object : Right click on starter object diagram > Add Element… > Toolset > Object
3. To link objects : select object, drag arrow to other object and choose "association"
4. To add states : CTRL+MAJ+R on an object or right click on object > Features & Properties > Set Run State…