Université d'Évry-Val d'Essonne

université PARIS-SACLAY

Modélisation Objet

Le langage UML

https://github.com/ELKADIRI/L3_info

hicham.elkadiri@univ-evry.fr

UNIFIED MODELING LANGUAGE

Hicham EL KADIRI 2024-2025

Organisation et plan du cours

CM-01	I - Introduction : l'historique, la modélisation II - Diagrammes de classes et d'objets – concepts de base
CM-02	III - Diagrammes de classes – concepts avancés
CM-03	IV - Diagramme de cas d'utilisation
CM-04	V - Diagramme de séquences
CM-05	VI - Diagramme d'états-transitions VII - Diagramme d'activités
CM-06	VIII - Diagramme de composants IX - Diagramme de déploiement X - Diagramme de paquetages

Université d'Évry-Val d'Essonne

université PARIS-SACLAY

I

Introduction

Pour pratiquer avec argoUML, ouvrir une session linux :
cd /usr/local/vmware/Temp/argouml-0.35.1/
java -jar argouml.jar
Pour StarUML, lancer la commande :
staruml

UML C'est quoi exactement ?

- Comme disait les connaisseurs … un bon dessin vaut mieux qu'un long discours !
- Quel sont les premiers outils que l'on utilise pour construire une maison, un immeuble, une ville :
  - La pelle et le seau ? Le crayon et le papier ?
- Quand un système est complexe, il est nécessaire de pouvoir en faire des représentations simples, axées sur un point de vue particulier.
- Exemples : le plan électrique d'une habitation, une carte routière, …

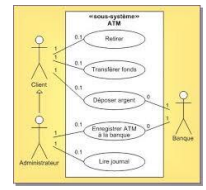
UML C'est quoi exactement ?

- En informatique aussi, les systèmes peuvent être complexes !
- Il devient alors nécessaire, quelque soit la démarche choisie, de faire des représentations – pour un point de vue particulier – du système que l'on construit.
- On parle alors de modèles et de modélisation.
- Les diagrammes **UML** permettent de mieux appréhender un système ou un logiciel complexe grâce à une représentation visuelle de l'**architecture** du code et des relations entre les différents composants.

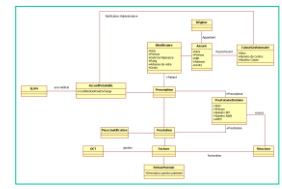
UML C'est quoi exactement ?

- Exemples de modélisations :

Point de vue du client : quelles sont les différentes fonctions de mon système ? Pour qui est-il fait ?



Point de vue de l'informaticien : quels sont les différents composants mis en œuvre ? Comment communiquent ils ? Quelles sont les informations manipulées ?



UML C'est quoi exactement ?

- Pour partager un modèle avec d'autres personnes, il faut s'assurer au préalable :
  - que les personnes comprennent les conventions de représentation et les notations utilisées,
  - qu'il ne pourra pas y avoir d'ambiguïté sur l'interprétation du modèle.
- D'où l'émergence de langages de modélisation de systèmes informatiques.
- Un langage de modélisation doit définir sans ambiguïté les concepts utilisables, leur représentation, les règles et contraintes associées.

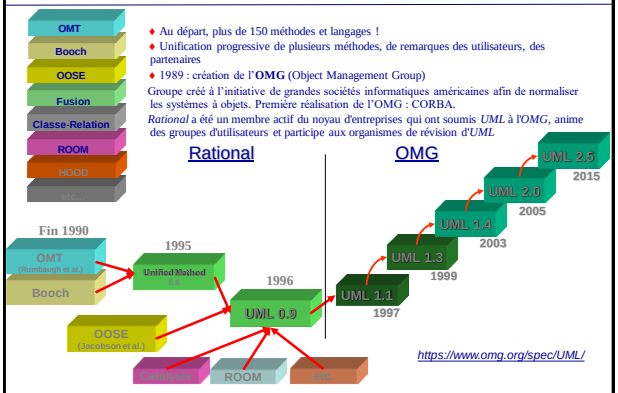
UML C'est quoi exactement ?

UML = Unified Modeling Language
Langage unifié pour la modélisation

Langage de modélisation objet, indépendant de la méthode utilisée.
UML est un langage pour :

- Comprendre et décrire un problème,
- Spécifier un système, de manière précise et complète, sans ambiguïté
- Concevoir et construire des solutions, une partie du code des diagrammes de classes peut être généré automatiquement,
- Documenter un système, les différents diagrammes, notes, contraintes, exigences sont conservés dans un document,
- Communiquer.
- **Unifié** : mise en commun et convergence de bonnes pratiques de langages antérieurs, émergence d'un standard.
- Pourquoi modéliser ? Pourquoi ne pas développer tout de suite

UML C'est quoi exactement ? – La genèse



- Au départ, plus de 150 méthodes et langages !
- Unification progressive de plusieurs méthodes, de remarques des utilisateurs, des partenaires
- 1989 : création de l'**OMG** (Object Management Group)

Groupe créé à l'initiative de grandes sociétés informatiques américaines afin de normaliser les systèmes à objets. Première réalisation de l'OMG : CORBA.

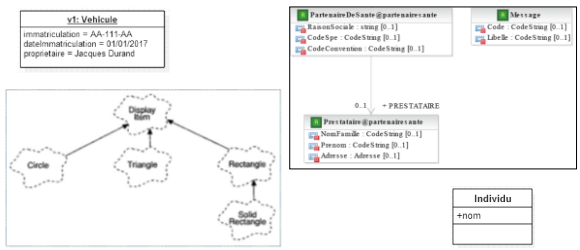
Rational a été un membre actif du noyau d'entreprises qui ont soumis *UML* à l'*OMG*, anime des groupes d'utilisateurs et participe aux organismes de révision d'*UML*

UML c'est quoi exactement ? – La genèse

- UML a été « boosté » par l'essor de la programmation orientée objet.
- Historique de l'approche orientée objet :
  - Simula (1967),
  - Smalltalk (1976),
  - C++ (1985),
  - Java (1995),
  - .net …
- UML permet de modéliser une application selon une vision objet, indépendamment du langage de programmation.

UML c'est quoi exactement ? – La genèse

- Comment représenter une classe ? Un objet ?



- Tout est défini dans la spécification UML !
  - http://www.omg.org/spec/UML/

UML c'est quoi exactement ? – La portée

- UML reste au niveau d'un langage et ne propose pas de processus de développement
  - ni ordonnancement des tâches,
  - ni répartition des responsabilités,
  - ni règles de mise en œuvre.
- Il existe de (très) nombreux outils pour faire de la modélisation UML: StarUML, ArgoUML, Papyrus,…
- Certains ouvrages et AGL basés sur UML proposent un processus en plus de UML.
  - Exemple : le processus unifié UP.

UML c'est quoi exactement ? – Le méta-modèle

- UML est bien plus qu'un outil pour dessiner des représentations mentales !
- La notation graphique n'est que le support du langage
- UML repose sur un **méta-modèle**, qui normalise la sémantique de l'ensemble des concepts.
- UML utile pour mettre les nouveaux développeurs à niveau sur les projets et pour concevoir de nouveaux projets à partir de zéro. La nature visuelle de ces modèles les rend beaucoup plus faciles à comprendre que l'alternative consistant à devoir lire et appréhender potentiellement des centaines de milliers de lignes de code pour comprendre un nouveau système.
- UML Aide à concevoir des projets d'ingénierie complexes

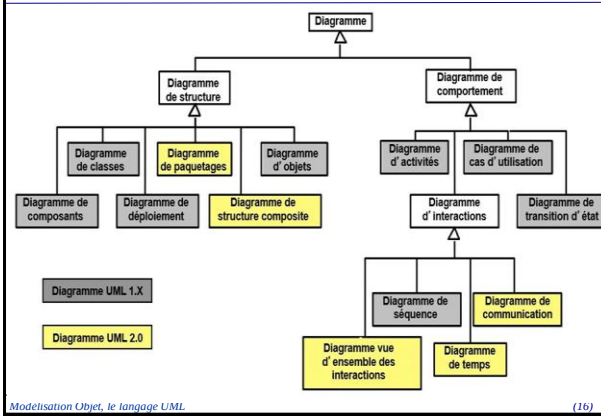
UML c'est quoi exactement ? – En résumé

- UML est un langage de modélisation objet
- UML n'est pas une méthode
- UML convient pour tous les types de systèmes, tous les domaines métiers, et tous les processus de développement
- UML est dans le domaine public
- Les concepts véhiculés dans UML sont définis et non équivoques

Tour d'horizon des diagrammes – 13 diagrammes !

- Historiquement UML proposait 9 types de diagrammes (UML 1.x).
- UML 2 a enrichi les concepts des diagrammes existants et a ajouté 4 nouveaux types de diagrammes.
- Les diagrammes manipulent des concepts parfois communs (ex. classe, objet, …), parfois spécifiques (ex. cas d'utilisation).
- Quelle différence entre un modèle et un diagramme ?

Tour d'horizon des diagrammes – 13 diagrammes !



Tour d'horizon des diagrammes – 13 diagrammes !

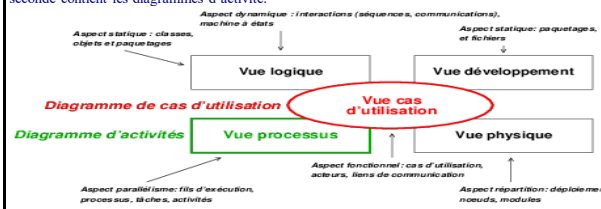
- UML est une grande boîte à outils,
  et comme toute boîte à outils ….
  - On utilise pas tout …
  - Certains outils servent plus souvent que d'autres …
  - Dans chaque situation il faut choisir le bon outil !
- En tant que langage, il faut aussi s'assurer que l'on va être compris
- Modèle pour appréhender la réalisation d'un système informatique
- Diagrammes UML qui permettent d'aider à la réflexion, à la discussion (clients, architectes, développeurs, etc.)
- Pas de solution unique mais un ensemble de solutions plus ou moins acceptables suivant les contraintes du client, et les logiciels et matériels disponibles
- Une solution acceptable est obtenue par itérations successives

Modèle de l'analyse

Vision de l'extérieur du système, le problème :

- Vue cas d'utilisation = ce que fait le système, les fonctionnalités
- Vue processus = ce que fait le système, les règles de gestion

L'élément pivot du modèle du système informatique est la vue cas d'utilisation. Cette vue est la plus proche du client et des utilisateurs finaux. C'est elle que ces derniers comprennent le mieux. Elle est complétée par la vue processus qui décrit comment le système répond aux stimulus externes. Cette vue processus présente les processus métier, par exemple les règles de gestion d'un système d'information. La première vue est composée des diagrammes de cas d'utilisation alors que la seconde contient les diagrammes d'activité.



Diagrammes

- Nous allons nous concentrer sur les 5 diagrammes suivants : Cas d'utilisation, Classes, Séquence, États, Activités.

UML2 apporte 4 nouveaux diagrammes :

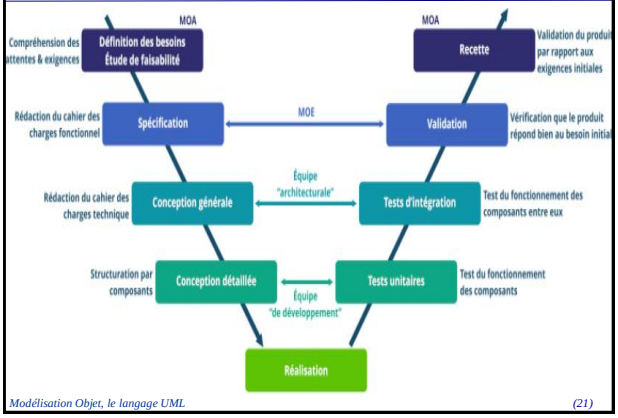
- diagramme de structure composite (composite structure diagram) : permet de décrire la structure interne d'un objet complexe lors de son exécution (au run-time - décrire l'exécution du programme), dont ses points d'interaction avec le reste du système.
- diagramme de paquetages (package diagram) : permet de représenter la hiérarchie des modules du projet, leur organisation et leurs interdépendances. Cela simplifie les diagrammes, et les rend donc plus simple à comprendre.
- diagramme global d'interaction (interaction overview) : permet d'associer les notations du diagramme de séquence à celle du diagramme d'activité, ce qui permet de décrire une méthode complexe. C'est une variante du diagramme d'activité.
- diagramme de chronométrage (timing diagram) : permet de modéliser les contraintes d'interaction entre plusieurs objets, comme le changement d'état en réponse à un évènement extérieur.

Par ailleurs, le diagramme de collaboration est devenu "diagramme de communication", et la plupart des diagrammes ont été revus pour répondre aux nouveaux besoins (abstraction, automatisation...).

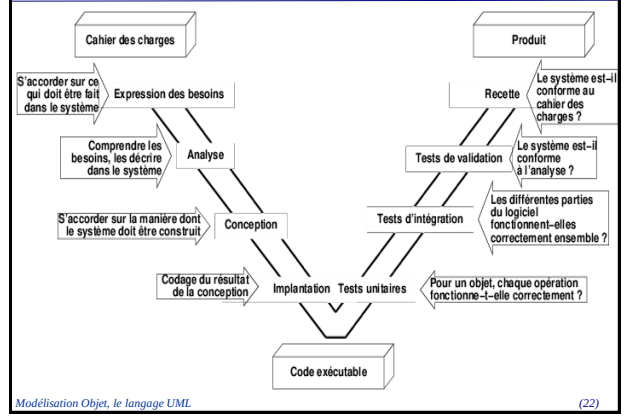
Cycle en V en gestion de projet : conception, réalisation, validation

- Le cycle en V en gestion de projet découle du modèle en cascade théorisé dans les années 1970, qui permet de représenter des processus de développement de manière linéaire et en phases successives.
- Ce mode de gestion de projet a été développé dans les années 1980 et appliqué au champ des projets industriels, puis étendu aux projets informatiques. Il a été remis en cause à partir du début des années 2000, sous l'effet de l'accélération des changements technologiques, favorisant davantage les méthodes dites « agiles ».
- La lettre V fait référence à la vision schématique de ce cycle, qui prend la forme d'un V : une phase descendante suivie d'une phase ascendante. Le cycle en V associe à chaque phase de réalisation une phase de validation.

Schéma de cycle V



Phases du cycle de développement en V



Avantages de cette méthodologie en cycle V

- Le principal avantage du cycle en V est qu'il **évite de revenir en arrière incessamment pour redéfinir les spécifications initiales**.
- Chaque phase de conception demande la rédaction d'une documentation précise et exhaustive, où chaque point doit être validé par le produit final. **Dès lors qu'une étape est validée, on ne revient pas en arrière et on passe à l'étape suivante sur une base solide** ; c'est la principale force du cycle en V.
- De par son aspect à la fois rigoureux et intuitif, le cycle en V demeure un processus facile à mettre en œuvre. Le travail préalable de définition des spécifications en début de projet fait que, une fois lancé, l'ensemble des étapes est connu des collaborateurs, qui peuvent se repérer facilement dans la temporalité du projet et connaître la finalité de leurs tâches. De la même manière, les documentations nécessaires à chaque étape sont réplicables d'un projet sur l'autre dans leur structure (cahiers des charges, cahiers de test…).
- En général, le cycle en V est plus adapté pour des réunions non quotidiennes, mais seulement des réunions de pilotage actant le passage d'une phase à l'autre. Son aspect linéaire autorise donc une organisation géographique éclatée, où le côtoiement des collaborateurs n'est pas clé dans le processus.

Inconvénients cycle en V

- L'inconvénient principal du cycle en V se résume en deux mots : **l'effet tunnel**. Après une phase de définition précise du produit auquel doit l'équipe doit aboutir, le projet est lancé dans un « tunnel » constitué des phases évoquées plus haut. Mais que faire si les spécifications initiales sont dépassées ? Si le besoin du client vient à changer, ou a été mal exprimé ? Le cycle en V supporte donc mal les changements, ce qui est à la fois sa force et sa principale faiblesse.
- Il offre ainsi moins de réactivité par rapport au contexte technologique et économique, aux demandes du client, aux événements inopinés ; la prise de risque s'en trouvera systématiquement limitée. L'effet tunnel est aussi induit par le travail conséquent de production de la documentation en début de projet, qui n'est plus rectifiable par la suite. Enfin, l'image du tunnel illustre le temps (parfois très) long qui sépare l'expression du besoin de la recette du produit final.

Mise en œuvre du cycle en V

Pour quels projets ?

Au regard des éléments exposés précédemment, les éléments suivants favorisent l'utilisation du cycle en V :

- **Des exigences très précises émises par le client**, par exemple dans le cadre d'un appel d'offres.
- **La présence d'un prestataire, qui maîtrise l'ensemble des étapes de réalisation** et requiert ainsi moins de communication entre les différents acteurs.
- **La possibilité de suivre un cahier des charges inchangé du début à la fin**, de par la nature du produit ou du projet.
- **Un projet où l'environnement technologique évolue très peu**, limitant ainsi les risques de décalage inhérents à l'effet tunnel.

Une phase cruciale : la conception

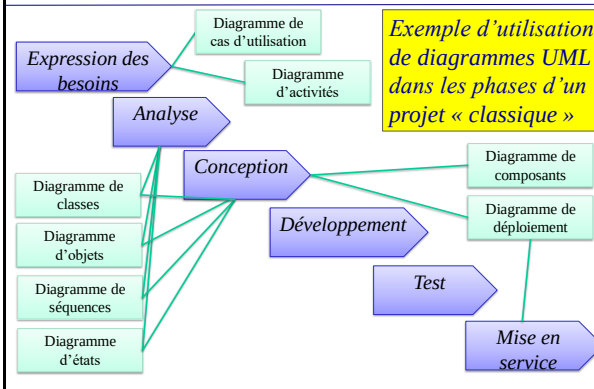
- Les besoins du client doivent être recueillis de manière exhaustive et rigoureuse. **Le dossier spécifications fonctionnelles détaillées (DSFD)** doit faire l'objet d'un processus de validation **suivi et définitif**. Il doit synthétiser les demandes du client tout en couvrant l'ensemble du programme du projet.
- La description des moyens pour parvenir au produit final, quant à elle, ne doit intervenir qu'au stade des **spécifications techniques** (conception générale DAT: dossier d'architecture technique), de façon à éviter que les moyens ne définissent la fin !

Qui valide les étapes de cycle V ?

le cycle en V définit des étapes sans en définir les rôles ou les responsabilités. Il convient donc en début de projet de désigner les personnes ou les entités qui joueront le rôle de (par niveau de détail croissant) :

- Maîtrise d'ouvrage (fonctionnel), ou MOA
- Maîtrise d'œuvre (système), ou MOE
- L'équipe architecturale, ou de conception générale (technique, métier)
- L'équipe de développement (par composant)
- Les rôles sont indiqués par séquence du cycle en V dans le schéma ci-dessus.

Tour d'horizon des diagrammes – 13 diagrammes en cycle V !

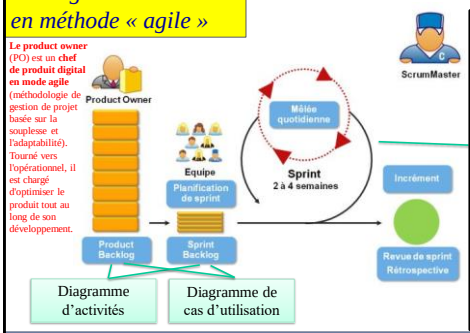


Tour d'horizon des diagrammes – 13 diagrammes en modèle agile !

Exemple d'utilisation de diagrammes UML en méthode « agile »

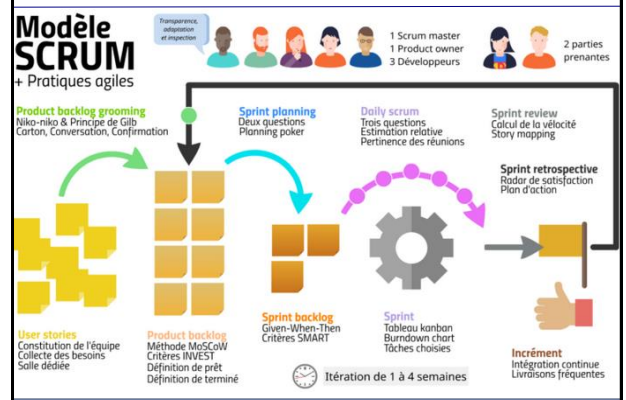
Scrum Master aide à animer la mêlée (Scrum) pour l'équipe dans son ensemble en s'assurant que le framework Scrum est respecté. Il s'engage à respecter les valeurs et les pratiques Scrum, mais doit également rester flexible et ouvert aux possibilités d'amélioration du workflow de l'équipe.

Le product owner (PO) est un **chef de produit digital en mode agile** (méthodologie de gestion de projet basée sur la souplesse et l'adaptabilité). Tourné vers l'opérationnel, il est chargé d'optimiser le produit tout au long de son développement.



- Diagramme de classes
- Diagramme d'objets
- Diagramme de séquences
- Diagramme d'états
- Diagramme de composants
- Diagramme de déploiement
- Diagramme d'activités
- Diagramme de cas d'utilisation

Méthode Scrum du modèle agile



Cycle en V vs. méthodes agiles

- De façon générale, l'on peut affirmer que le **cycle en V se focalise sur le processus**, tandis que **les méthodes agiles privilégient le produit**.
- Dans le cadre des **méthodes agiles** (Scrum, XP, RAD, …), **le projet s'affine par *itérations*** , à travers la répétition d'un cycle d'opérations (le ***sprint*** dans le cadre de **la méthode Scrum**). Comme nous l'avons vu, **le cycle en V définit l'intégralité du produit final dès les premières étapes**, et ne laisse que peu de place à l'adaptation dans la suite du cycle.
- Ensuite, les **méthodes agiles permettent d'élaborer le produit par *incrémentation*** . On produit un peu plus à chaque fois, morceau par morceau, pour aboutir au résultat final. Le cycle en V concentre au contraire la réalisation de l'ensemble dans une seule phase, qui est intégralement conçue en amont et vérifiée en aval.
- Ce manque d'adaptation et de flexibilité du cycle en V a précisément conduit à l'émergence des méthodes agiles, en particulier dans le domaine du logiciel et du marketing, pour répondre aux changements de plus en plus rapides des technologies et des demandes des consommateurs.

Qu'est-ce qu'une méthode agile ?

- Alors que les méthode traditionnelles visent à traiter les différentes phases d'un projet d'une manière séquentielle (que l'on nomme aussi cycle de développement en cascade ou encore cycle en V), le principe des méthodes Agiles est de le découper en sous-parties (ou sous-projets) autonomes (on parle également de développement itératif).
- Les parties (itérations) forment le projet dans sa globalité.

Le Manifeste Agile, les principes fondateurs

Ces méthodes découlent du **Manifeste Agile**, des pratiques édictées par des experts en 2001 pour améliorer le développement de logiciels.

Les 4 valeurs mises en exergues :

- la primauté des personnes et des interactions sur les processus et outils.
- une préférence pour un logiciel fonctionnel plutôt qu'une documentation complète.
- une relation autre avec les clients : une collaboration permanente remplaçant une négociation contractuelle.
- une adaptation continue au changement et non le suivi rigide d'un plan.

Découlant de ces valeurs, le Manifeste agile définit 12 principes

1 - La priorité n°1 est d' **obtenir la satisfaction client au plus tôt** par la livraison rapide et régulière de fonctionnalités attendues.

2 - Accepter **les demandes de changement en cours de projet** . Ce sont des opportunités pour donner plus de valeur au projet et coller aux vrais besoins des clients.

3 - Mettre en œuvre des livraisons rapides reposant sur des cycles courts (quelques semaines). Ces livrables doivent être opérationnels pour permettre des tests de validation des fonctionnalités attendues.

4 - Coopération forte et continue entre les utilisateurs et le développement. A 'inverse des méthode classiques où les rencontres entre les utilisateurs et la maîtrise d'oeuvre interviennent surtout en début et en fin de projet.

5 - Donner de l'autonomie à des personnes impliquées et leur faire confiance.

6 - Privilégier le face à face comme canal de communication entre les parties. Les interactions sont plus efficaces et plus riches. Tout va plus vite.

7 - L'important est d'avoir une application opérationnelle.

8 - Avancer avec un rythme constant compatible avec ce que peut produire l'ensemble des acteurs.

9 - Focus sur la qualité technique et la qualité de conception pour construire une base solide renforçant l'agilité.

10 - Rester simple dans les méthodes de travail : ne faire que ce qui est nécessaire.

11 - Une équipe qui s'organise elle même produit de meilleurs résultats.

12 - En revoyant régulièrement ses pratiques, l'équipe adapte son comportement et ses outils pour être plus efficace.

Quels sont les avantages de la méthode agile ?

Cette approche permet d'obtenir :

- **plus de flexibilité** en travaillant sur des sous-parties autonomes. Elles peuvent être conçues, testée, modifiées de nouveau sans que l'ensemble du projet ne soit impacté. La prise en compte de besoins non identifiés dans la phase d'analyse ou bien l'émergence de nouvelles fonctionnalités au cours du développement peuvent être implémentées. Par expérience, il est difficile de penser à tout dans la phase de définition de besoin pour une approche classique de gestion de projet.
- **Plus de fiabilité et de qualité :** en simplifiant la complexité, en testant en continu, en favorisant les feedbacks, les échanges avec les clients.
- **Des risques réduits :** détection rapide grâce à des cycles courts.
- **Une meilleure maîtrise des coûts :** pas de coûteux retours en arrière - si nécessaire le projet peut être stoppé rapidement.

Mais aussi des limites de la méthode agile

- La flexibilité poussée à l'extrême peut **conduire à un enlisement**
- **du projet** . De nombreuses itérations sans que des directions ou décisions ne soient figées représentent un réel danger. L'une des causes possibles des revirements incessants des clients quant à leurs spécifications.
- Dans ces situations, le chef de projet (quelle que soit sa dénomination dans la méthode choisie) doit être capable d'arbitrer pour le bien du projet, mais également celui... du client.

Les méthodes Agiles

Les principes de l'agilité sont repris d'une manière structurée par plusieurs méthodes. Focus sur l'une des plus populaires : **méthode Scrum**

- Cette méthode propose un cadre très structuré pour appliquer les principes de l'agilité.

Le Sprint, le coeur de Scrum :

- Cette approche repose sur des itérations de 2 à 4 semaines. Ce sont **les fameux "Sprints"** . Il s'agit des sous-parties d'un projet comme le définit le principe Agile. Chaque Sprint a pour objectif de livrer au client une version potentiellement utilisable du produit.

Les Sprints successifs ajoutent des fonctionnalités au produit ou améliorent celles déjà développées. On parle d'incrément de produit.

- Un Sprint démarre lorsque le précédent est terminé. Il s'agit d'un processus incrémental.

Sprint

Le sprint repose sur 3 piliers que sont :

- **la transparence :** élaboration d'un standard commun
- pour permettre une compréhension partagée.
- **l'inspection :** des vérifications sont effectuées régulièrement.
- **l'adaptation :** en cas de dérive constatée lors de l'inspection, des ajustements sont décidés.

Les Sprints se structurent autour de plusieurs outils organisationnels

- **Sprint planning (Planification du sprint) :** réunion pour sélectionner et planifier les priorités de chaque Sprint en terme de liste des fonctionnalités produit (**Sprint Backlog**).
- **Scrum (Mélée quotidienne) :** réunion journalière de coordination entre les membres de l'équipe projet. Elle prend fréquemment la forme de "Stand-up meeting" (réunion de courte durée, 10-15mn, tenue debout).
- **Sprint Review (Revue de Sprint) :** réunion de synthèse à la fin de chaque Sprint afin de valider les fonctionnalités développées.
- **Sprint Retrospective (Rétrospective de Sprint) :** venant immédiatement après la revue de Sprint, il s'agit d'un bilan dont l'objectif est l'amélioration continue des pratiques. L'équipe échange sur les réussites, les difficultés, relève ce qui a fonctionné ou non. Avec toujours des leçons à tirer pour les prochains Sprints.

Comprenant des entrants et sortants du processus, appelés "artéfacts"

- **Product Backlog :** liste des fonctionnalités du produit.
- **Sprint Backlog :** planification des éléments du Product Backlog à mettre en œuvre lors du Sprint pour livrer l'incrément de produit doté des fonctionnalités requises pour cette étape. Le Sprint Backlog n'est pas figé, mais est amené à évoluer durant le Sprint.
- **L'incrément de produit :** déjà évoqué plus haut.

Avec des rôles définis pour chacun :

- **Product Owner - PO (propriétaire du produit) :** l'expert métier, **le maître d'ouvrage**, représente le client et intervient sur le coté fonctionnel.
- **Scrum Master (maître de mêlée) :** le coordinateur du projet et le garant du respect de la méthode Scrum.
- **Team (équipe) :** les autres intervenants sur le projet (notamment les développeurs).

Tour d'horizon des diagrammes – Restrictions & extensions

- **Restrictions** : au sein d'une organisation ou d'une équipe, on utilise un « sous-ensemble » de UML :
  - Sous-ensemble de diagrammes,
  - Sous-ensemble des possibilités offertes par chaque diagramme.
- **Extensions** : UML possède des mécanismes d'extension, qui permettent d'adapter le langage à une organisation, une équipe ou un domaine particulier.
- Un **profil** UML est un ensemble cohérent de concepts UML, d'extensions/restrictions, de contraintes, de règles et notations.
  - Exemples : profil EJB, profil SIG, profil RT, …

Université d'Évry-Val d'Essonne

université PARIS-SACLAY

II
Le diagramme de classes et le diagramme d'objets

Concepts de base

Diagrammes communs à l'analyse et à la conception

Aspects statiques = structure du problème et de la solution
- Diagrammes de classes et d'objets

- Aspects dynamiques = comportement des éléments de la structure
  - Diagrammes de séquence, de communications et d'états

La conception est une phase charnière entre la spécification du problème et l'implantation de la solution. Il est donc important de garder la trace des décisions de conception : telle exigence extraite du cahier des charges et spécifiée dans l'analyse est traduite par tels et tels éléments du modèle de conception. C'est une des raisons qui explique que certains diagrammes sont utilisés dans le modèle de l'analyse et dans celui de la conception, la conception consistant à compléter ou raffiner le modèle de l'analyse avec des éléments techniques, par exemple comment rendre l'application accessible par un service Web sur l'intranet de l'entreprise.

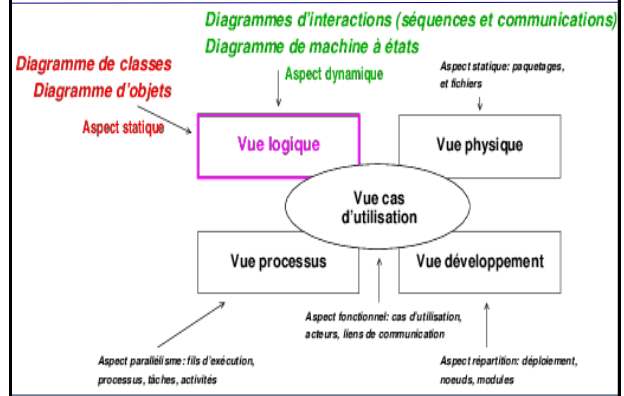
Démarche d'analyse pour conception d'une architecture Objet

- L'analyse va être de plus en plus fine mais il faut une fois de plus éviter de plonger directement dans des solutions techniques connues tout en gardant à l'esprit les contraintes du contexte dans lequel l'application va être développée.
- Chaque cas d'utilisation contient un vocabulaire et des mécanismes que nous devons maintenant projeter à travers des entités informatiques.
- Il faudra durant notre analyse distinguer les classes de représentation des données qui seront dans la base de donnée (qui ne comportent que des attributs) et les classes qui vont constituer la partie logiciel du projet (qui comportent des attributs et des méthodes).

Identification des classes candidates

- Il ne faut pas se tromper, les classes que nous allons identifier ne fournissent qu'une représentation statique du système. Cependant, nous devons garder à l'esprit que chaque classe va se composer d'un ensemble d'attributs et éventuellement d'opérations qui seront utilisés dans la descriptions dynamique du système.
- Chaque classe pourra être affinée par un diagramme d'état et/ou un diagramme d'activité qui va décrire son fonctionnement dynamique. Les diagrammes dynamiques vont utiliser les opérations des classes et vont aussi générer des opérations dans le cas des diagrammes de séquence.
- Enfin, durant la phase d'identification des classes il y a un certain nombre d'associations qui semblent nécessaires et il ne faut pas se priver de les créer.

Diagrammes communs à l'analyse et à la conception



Classes et objets

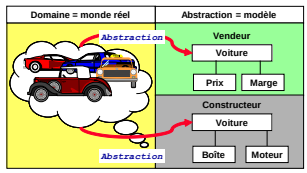
- Le diagramme de classes permet de représenter des classes, leurs propriétés et leurs relations avec d'autres classes.
- Le diagramme d'objets (des instances des classes) permet de représenter des objets, les valeurs de leurs propriétés et leurs relations avec d'autres objets.
- Mais au fait ….
  - Qu'est ce qu'un objet ?
  - Qu'est ce qu'une classe ?
  - Une classe peut-elle être un objet ? Et inversement ?

Les objets

- Un objet est une entité identifiable du monde réel.
- Les objets informatiques définissent une représentation simplifiée des entités du monde réel.
- Un objet informatique est une structure de données valorisées qui répond à un ensemble de messages.
- Un objet peut représenter une entité concrète (personne, guitare, véhicule, …) ou abstraite (gestionnaire de flux, …).

Les objets

- Une **abstraction** est un résumé, un condensé, une mise à l'écart des détails non pertinents dans un contexte donné.
- Mise en avant des caractéristiques essentielles et utiles.
- Dissimulation des détails (complexité).



Les objets

- L'état d'un objet :
  - regroupe les valeurs instantanées de tous les attributs d'un objet
  - évolue au cours du temps
  - est la conséquence des comportements passés
- Exemples :
  - un signal électrique : l'amplitude, la pulsation, la phase, …
  - une voiture : la marque, la puissance, la couleur, le nombre de places assises, …
  - un étudiant : le nom, le prénom, la date de naissance, l'adresse, …

Les objets

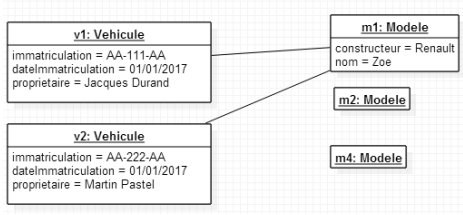
- Le comportement
  - décrit les actions et les réactions d'un objet
  - regroupe toutes les compétences d'un objet
  - se représente sous la forme d'opérations (méthodes)
- Un objet peut faire appel aux compétences d'un autre objet
- L'état et le comportement sont liés
  - Le comportement dépend souvent de l'état
  - L'état est souvent modifié par le comportement

Les objets

- Tout objet possède une identité qui lui est propre et qui le caractérise.
- L'identité permet de distinguer tout objet de façon non ambiguë, indépendamment de son état.

Les objets

- Le diagramme d'objets permet la représentation d'objets du système que l'on modélise.
- Exemple :



Communication entre objets

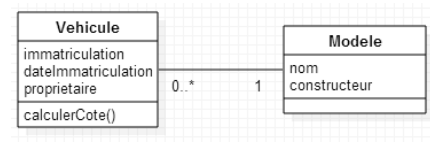
- Application = société d'objets collaborant
- Les objets travaillent en synergie afin de réaliser les fonctions de l'application.
- Le comportement global d'une application repose sur la communication entre les objets qui la composent.
- Les objets
  - ne vivent pas en ermites,
  - interagissent les uns avec les autres,
  - communiquent en échangeant des messages.

Les classes

- La classe :
  - est une description abstraite d'un ensemble d'objets
  - peut être vue comme la factorisation des éléments communs à un ensemble d'objets
  - c'est un modèle d'objets ayant les mêmes types de propriétés et de comportements. Chaque instance d'une classe possède ses propres valeurs pour chaque attribut.
- Il ne faut pas se tromper, les classes que nous allons identifier ne fournissent qu'une représentation statique du système. Cependant, nous devons garder à l'esprit que chaque classe va se composer d'un ensemble d'attributs et éventuellement d'opérations qui seront utilisés dans la descriptions dynamique du système.
- Chaque classe pourra être affinée par un diagramme d'état et/ou un diagramme d'activité qui va décrire son fonctionnement dynamique. Les diagrammes dynamiques vont utiliser les opérations des classes et vont aussi générer des opérations dans le cas des diagrammes de séquence.

Les classes

- Le diagramme de classes permet la représentation des descripteurs d'objets du système que l'on modélise.
- Exemple :

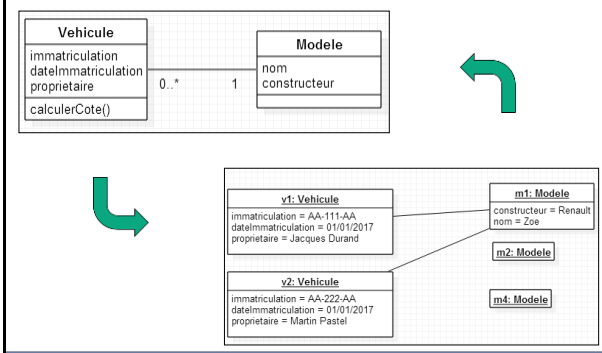


Les classes

- Le diagramme de classes est le diagramme le plus connu et le plus utilisé.
- Structure interne statique d'un système.
- Diagrammes de classes et d'objets sont liés :
  - Le diagramme de classes permet de représenter :
    - des classes (ensembles d'objets),
    - des relations entre classes (ensemble de liens entre objets).
  - Le diagramme d'objets permet de représenter :
    - des instances particulières des classes, des objets,
    - les liens entre ces objets.

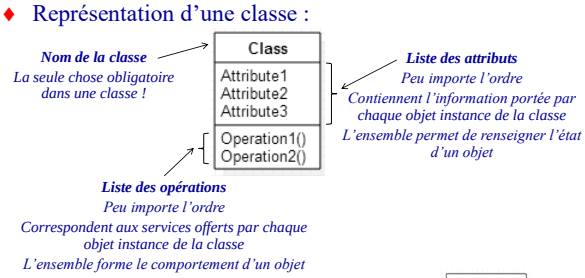
Les classes

- Diagrammes de classes et d'objets sont liés !

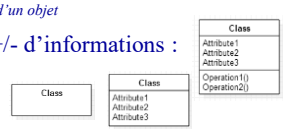


Diagrammes de classes – concepts de base

- Représentation d'une classe :

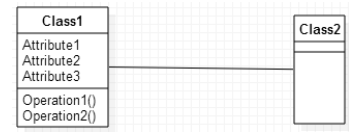


- Possibilité d'afficher +/- d'informations :



Diagrammes de classes – concepts de base

- L'association exprime une connexion structurelle entre classes.
- Notation :



- Exemple : « une personne est employée par une entreprise » pourra se traduire par :

Diagrammes de classes – concepts de base

- Oui mais … l'association entre Personne et Entreprise

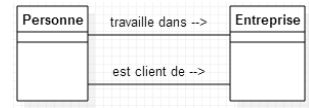
peut être interprétée de plusieurs façons :

- « une personne travaille dans une entreprise »
- « une personne dirige une entreprise »
- « une personne est cliente d'une entreprise »
- ….

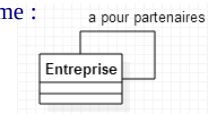
- Il est alors possible d'indiquer la signification de l'association :

Diagrammes de classes – concepts de base

- Sur un diagramme, il est possible de créer plusieurs associations entre les mêmes classes :



- Il est aussi possible de créer une association d'une classe avec elle-même :



- Comment cela se traduit il au niveau des objets ?

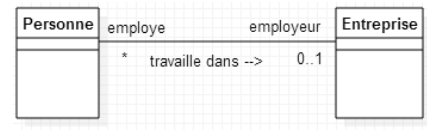
Diagrammes de classes – concepts de base

- Chaque extrémité de l'association peut aussi être nommée :

- Le rôle décrit comment les objets de la classe correspondante sont perçus par les objets de la classe à l'autre extrémité de l'association.
- **Un rôle a la même nature qu'un attribut**
- en conception on nomme les rôles, pour créer au besoin les attributs associés.

Diagrammes de classes – concepts de base

- Chaque rôle porte une indication de multiplicité : il s'agit du nombre d'objets de la classe considérée pouvant être liés à un objet de l'autre classe.

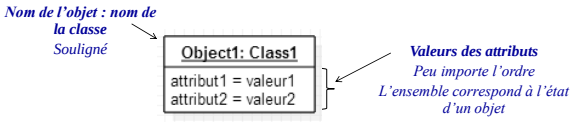


- La multiplicité s'exprime sous la forme d'un intervalle : nombre minimal et nombre maximal d'objets en relation.

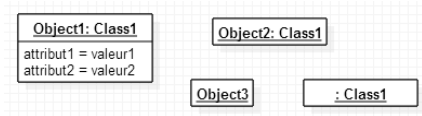
Valeur	Signification
1	un et un seul
0..1	zéro ou un
M .. N	de M à N (entiers naturels)
* ou 0..*	de zéro à plusieurs
1 .. *	un à plusieurs

Diagrammes d'objets – concepts de base

- Représentation d'un objet :

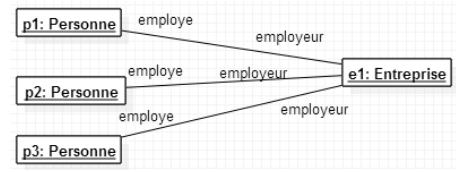


- Possibilité d'afficher +/- d'informations :

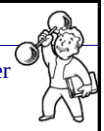


Diagrammes d'objets – concepts de base

- Les associations entre classes permettent de créer des liens entre objets :



Exercice 1



- Réaliser un diagramme de classes pour représenter les concepts suivants issus d'un système de réservation de véhicules :
  - Un client peut effectuer des réservations de véhicules.
  - Chaque client est décrit par son nom, son prénom et son numéro d'inscription.
  - Un véhicule possède un numéro d'immatriculation, une marque, une date de mise en service, une puissance fiscale, une vitesse maximale.
  - Pour un client donné, on souhaite pouvoir calculer la puissance moyenne des véhicules qu'il réserve.
- Illustrer votre diagramme de classes par un diagramme d'objets.

Université d'Évry-Val d'Essonne

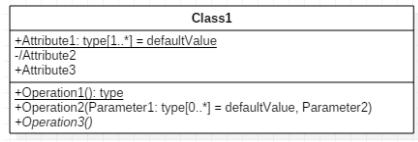
université PARIS-SACLAY

III
Le diagramme de classes et le diagramme d'objets

Concepts avancés

Propriétés complètes des attributs

- La représentation complète d'une classe fait apparaître les attributs avec différentes caractéristiques en plus du nom (type, valeur par défaut, degré de visibilité, …), et les opérations avec leur signature complète :



Propriétés complètes des attributs

- La forme complète de représentation d'un attribut est la suivante :

  <visibilité> **<nomAttribut>** : <type> [borneInf..borneSup] = <valeur par défaut> {propriétés}

- Seul le nom est obligatoire !

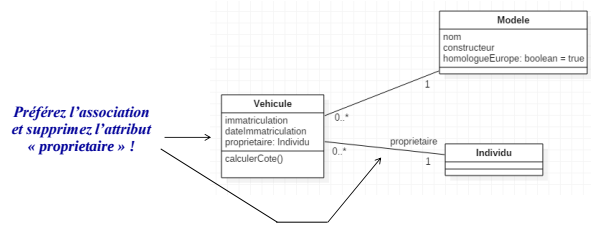
Propriétés complètes des attributs

- Le symbole de visibilité correspond au concept objet d'encapsulation. Il représente le degré de protection de l'attribut :
  - + : publique (accessible à toutes les autres classes)
  - # : protégé (accessibles uniquement aux sous-classes)
  - ~ : paquetage (accessible uniquement aux classes du paquetage)
  - - : privé (inaccessible à tout objet hors de la classe)

- La visibilité peut être précisée sur chaque attribut et sur chaque opération.

Propriétés complètes des attributs

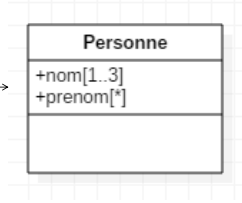
- Le type permet de fixer l'ensemble des valeurs possibles que peut prendre un attribut.
- Il peut s'agir :
  - d'un type standard : integer, string, boolean, real
  - d'une classe : on préfèrera très souvent utiliser une association



Propriétés complètes des attributs

- Un attribut peut prendre plusieurs valeurs.
- Il est alors possible de préciser le nombre minimal et le nombre maximal de valeurs que peut prendre l'attribut.

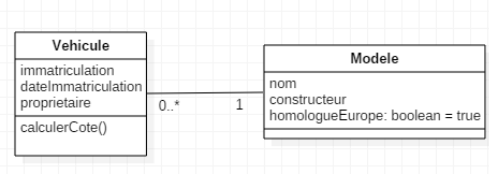
On spécifie ici qu'une personne possède entre 1 et 3 noms et 0 à plusieurs prénoms



- En l'absence d'indication, la valeur est unique ([1..1]).

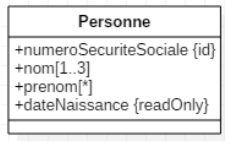
Propriétés complètes des attributs

- Une valeur par défaut peut être précisée sur un attribut. Cette valeur est affectée à l'attribut à la création des instances de la classe.



Propriétés complètes des attributs

- Il est possible d'attribuer des propriétés aux attributs, en les indiquant entre accolades.

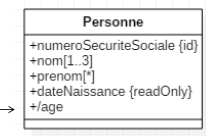


- Parmi les propriétés les plus utilisées :
  - {readonly} indique que la valeur de l'attribut ne peut pas être modifiée,
  - {id} indique que l'attribut fait partie de l'identifiant des instances de la classes.

Propriétés complètes des attributs

- Lorsque la valeur d'un attribut peut être calculée par une fonction basée sur la valeur d'autres attributs, on parle d'attribut calculé ou attribut dérivé.
- Un attribut calculé est précédé du signe « / »

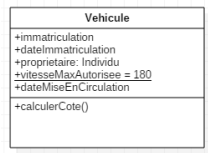
L'âge peut être calculé à partir de la date de naissance !



Propriétés complètes des attributs

- Chaque instance d'une classe contient une valeur spécifique pour chacun de ses attributs. Dans certains cas cependant, il est utile de pouvoir définir des attributs dont la valeur est commune à l'ensemble des instances de la classe.
- On parle alors d'attribut statique ou attribut de classe, et on souligne l'attribut.

La vitesse maximale autorisée est la même pour toutes les instances de la classe ! →



Propriétés complètes des opérations

- La forme complète de représentation d'une opération est la suivante :

  <visibilité> **<nomOpération> (**listeParamètres**)** : <typeRetour> [borneInf..borneSup] {propriétés}

- Comme pour les attributs, seul le nom, suivi des parenthèses ( ), est obligatoire !
- Chaque paramètre est décrit sous la forme :

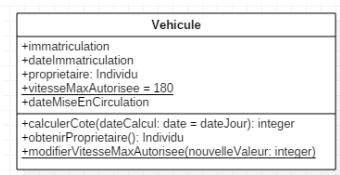
  <direction> <nom> : <type> [borneInf…borneSup] = <valeur par défaut> {propriétés}

Propriétés complètes des opérations

- Certains concepts présentés pour les attributs s'appliquent aux opérations :
  - La visibilité de l'opération,
  - L'intervalle pour indiquer le nombre de valeurs du retour,
  - Sur les paramètres : le type, l'intervalle pour préciser le nombre de valeurs autorisées et la valeur par défaut,
  - Les propriétés.
- Il existe cependant des différences :
  - Lorsque l'opération renvoie un objet, il n'est pas possible de remplacer le typeRetour par une association !
  - Un paramètre peut être préfixé en indiquant sa direction : in, out ou inout.

Propriétés complètes des opérations

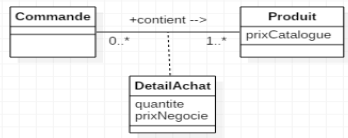
- Il est également possible qu'une opération soit définie comme « statique » : cette opération est appelée sur la classe directement.



- Ces opérations ne peuvent pas manipuler d'attributs qui ne soient pas statiques !

Classe d'association

- La classe d'association est … une classe ! Cette classe permet de faire porter des informations aux liens entre instances de classes.



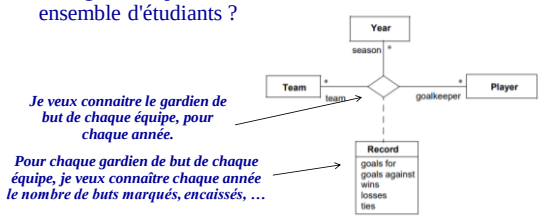
- Une telle classe peut être dotée d'attributs, d'opérations, et de relations avec d'autres classes.
- Attention : une classe d'association ne peut être reliée qu'à une seule association, par contre elle peut être en relation avec d'autres classes.
- Faire exemple avec Achat et Article : où met on l'attribut quantité ?

Classe d'association

- Exemples d'utilisation de la classe d'association :
  - Des personnes empruntent des livres à la bibliothèque. Il est nécessaire de pouvoir retrouver la date de chaque emprunt.
  - Des personnes signent des accords entre eux. Il est nécessaire de conserver la date de ces accords et la ville où a eu lieu la signature.
  - Des personnes travaillent dans des entreprises. Pour chacun de leurs emplois, il est nécessaire de connaître le temps de travail et le type de contrat signé avec l'entreprise.

Associations n-aire

- L'association permet de relier plus de deux classes. On parle d'association ternaire pour trois classes, ou plus généralement d'association n-aire.
- Représentation : un losange blanc ou au moyen d'une classe stéréotypée.
- Comment modéliser avec classes et relations le fait qu'un enseignant dispense un cours dans une salle donnée à un ensemble d'étudiants ?



Associations n-aire

- Attention, il est souvent possible de modéliser un problème avec plusieurs associations binaires plutôt qu'avec une association n-aire.
- Exemple :
  - Un employé emprunte un véhicule de fonction pour se rendre sur un site
  - Un employé possède un véhicule de fonction. L'employé se rend sur des sites avec son véhicule de fonction

Navigabilité d'une association

- Les associations permettent par défaut une navigation bidirectionnelle : il est possible de déterminer les liens de l'association depuis les instances de chaque classe.
- Il est possible de limiter la navigabilité en spécifiant un seul sens de navigation :

- Attention : sans indication particulière sur un diagramme, la navigation peut se faire dans les 2 sens.
- En général c'est en conception que l'on décide du sens de navigation des associations.

Navigabilité d'une association

- Exemple :

Une instance d'utilisateur peut accéder à des instances de Mot de passe, mais pas l'inverse.

- A ne pas confondre avec le sens de lecture de l'association !

Les différentes relations entre classes

Dépendance : un objet d'une classe travaille brièvement avec des objets d'une autre classe.

Association : un objet d'une classe travaille avec des objets d'une autre classe pendant une durée prolongée.

Agrégation : une classe détient et partage une référence à des objets d'une autre classe.

Composition : une classe contient des objets d'une autre classe.

Héritage : une classe est un type d'une autre classe.

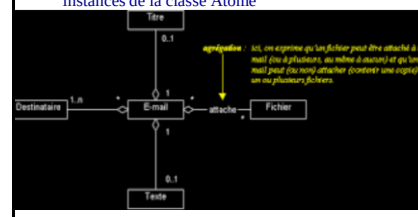
- La composition est une forme particulière d'agrégation. La dépendance est une relation "un peu à part", souvent peu représentée, mais néanmoins importante !

Les différentes relations entre classes

Concernant les relations entre classes, il existe d'autres relations en plus de l'association

- L'agrégation,
- La composition,
- La généralisation/spécialisation,
- La dépendance.

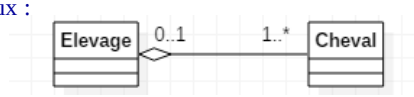
Lorsqu'on représente qu'une molécule est "composée" d'atomes, la destruction d'une instance de la classe "Molécule", implique la destruction de ses composants, instances de la classe Atome



un message électroniqu possède un titre et un destinataire (attributs), est lié un entête et un corps (2 liens de composition), et peut posséder une pièce jointe (1 lien d'agrégation).

Agrégation et composition

- L'agrégation est une forme particulière d'association binaire, mais dissymétrique, où l'une des extrémités est prédominante par rapport à l'autre.
- Représentation des relations de type :
  - tout et parties,
  - composé et composants,
  - maître et esclaves.
- Exemple : un élevage est composé d'un certain nombre de chevaux :



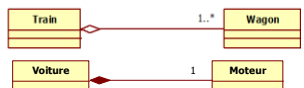
Agrégation et composition

- Deux types d'agrégation :
  - **Agrégation** partagée – notion de co-propriété
    - La création (resp. la destruction) des composants est indépendante de la création (resp. la destruction) du composite
    - Un objet peut faire partie de plusieurs composites à la fois
    - Notation :

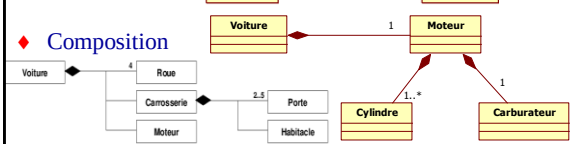
  - **Composition**
    - Cas particulier de l'agrégation : attributs contenus physiquement par l'agrégat
    - La création (resp. la destruction) du composite entraîne la création (resp. la destruction) des composants.
    - Un objet ne fait partie que d'un seul composite à la fois.
    - Notation :

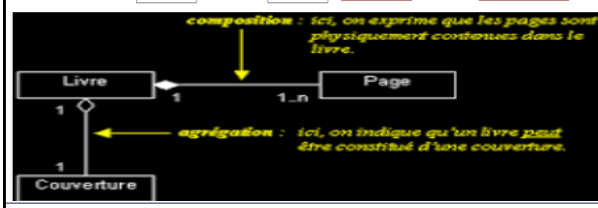
Agrégation et composition

- Agrégation



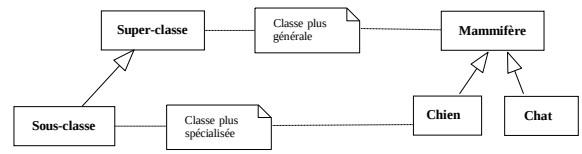
- Composition





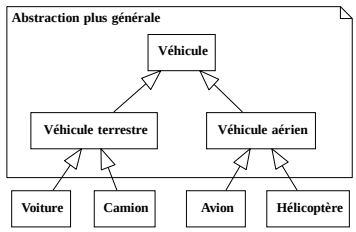
Généralisation/Spécialisation

- La relation de généralisation/spécialisation permet de gérer les relations « est du type » ou « est une sorte de »
- Cette relation correspond au concept d'héritage en objet.



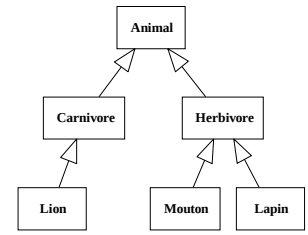
Généralisation/Spécialisation

- Les instances d'une classe fille sont aussi instances des super-classes. Elle « profitent » des attributs, opérations et relations définies dans les super-classes.



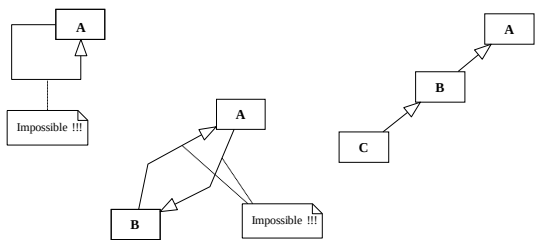
Généralisation/Spécialisation

- La généralisation peut se traduire par « ***est un*** » ou « ***est une sorte de*** »



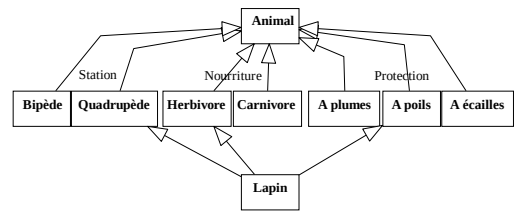
Généralisation/Spécialisation

- Propriétés de la généralisation : non-réflexive, non-symétrique, transitive



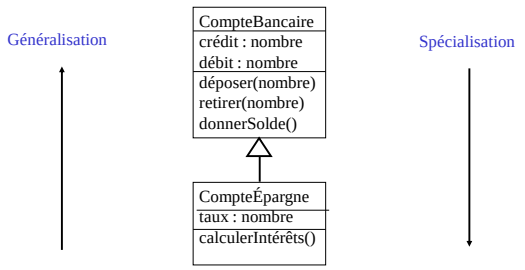
Généralisation/Spécialisation

- La généralisation multiple est modélisable en UML !



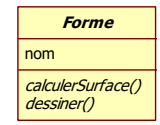
Généralisation/Spécialisation

- Exemple :



Généralisation/Spécialisation

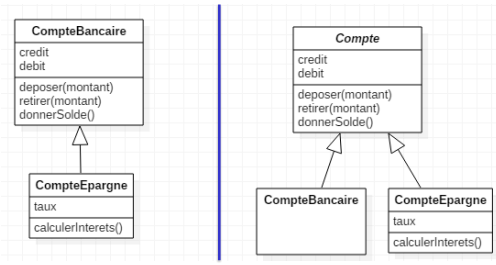
- Classe abstraite : classe qui ne peut avoir aucune instance directe ; on écrit son nom en *italique*
- Opération abstraite : opération incomplète qui a besoin de la classe fille pour fournir une implémentation ; on écrit son nom en italique.



Une classe abstraite n'a pas d'instances directes, mais ses sous-classes peuvent en avoir. Notation UML : nom de classe en italiques et/ou stéréotype de classe <<abstract>>. Une classe concrète est une classe instanciable.

Généralisation/Spécialisation

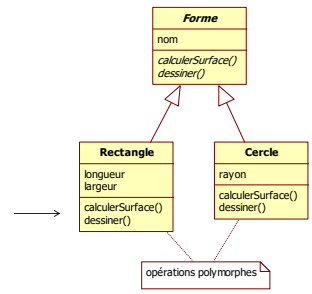
- Ces deux modèles sont ils équivalents ?



Généralisation/Spécialisation - polymorphisme

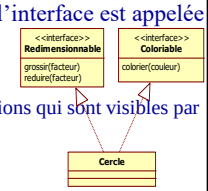
- Le polymorphisme est un concept objet selon lequel un même message peut être interprété de différentes façons, selon le récepteur.

Une méthode peut être définie par le même nom dans différentes classes.



Généralisation/Spécialisation - interfaces

- L'interface est une classe totalement abstraite, sans attribut et dont toutes les opérations sont abstraites.
- Le concept d'interface permet la définition d'un contrat pour toutes les classes qui l'implémentent. La relation entre une interface et une classe qui implémente l'interface est appelée relation de réalisation.
- Notation :



- Une interface est une spécification des opérations qui sont visibles par l'environnement d'une classe. Conséquences :

c'est une classe qui n'a pas d'implémentation (puisqu'il s'agit d'une spécification),

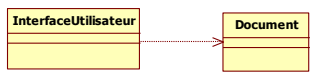
elle ne présente qu'une partie du comportement de la classe correspondante (puisque cette dernière peut implémenter des méthodes "internes"),

Généralisation/Spécialisation - interfaces

- Une interface est une spécification des opérations qui sont visibles par l'environnement d'une classe. Conséquences :
- - c'est une classe qui n'a pas d'implémentation (puisqu'il s'agit d'une spécification),
- - elle ne présente qu'une partie du comportement de la classe correspondante (puisque cette dernière peut implémenter des méthodes "internes"),
- - elle ne possède que des opérations (pas d'attributs, ni d'associations, ni d'états puisqu'elle n'est pas implémentée).
- Une interface est donc équivalente à une classe abstraite qui serait réduite à des opérations abstraites (elle ne possède ni attributs ni méthodes) et qui n'aurait pas de descendants (et donc pas d'instances).

Dépendance

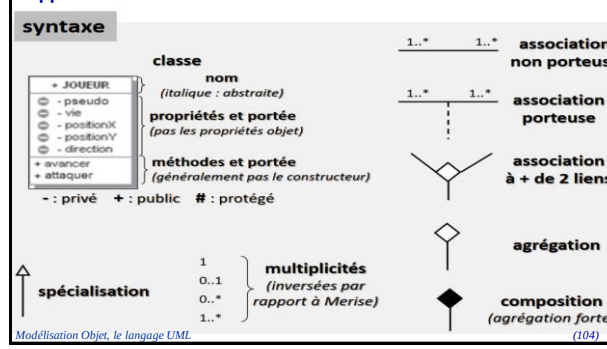
- La relation de dépendance est une relation sémantique entre deux éléments selon laquelle un changement apporté à l'un peut affecter l'autre.
- Implique uniquement que des objets d'une classe peuvent fonctionner ensemble.
- Notation :



Une dépendance entre 2 classes stipule qu'une classe a besoin d'informations sur une autre classe pour pouvoir utiliser les objets de celle-ci.
Indique que la modification de la cible peut avoir une incidence sur la source.

Diagramme de classes

Le diagramme de classes est un schéma :
→présentant l'organisation des données et traitements de l'application



Exercice 2

Agrégation/Composition/Héritage/Dépendance: un pays à une capitale ? / une transaction boursière est un achat ou une vente ? / les fichiers contiennent des enregistrements ? / un polygone est composé d'un ensemble de points ordonné ? / une personne utilise un langage dans un projet ?/ les souris et les claviers sont des périphériques d'entrées sorties ? / des classes d'objets peuvent avoir plusieurs attributs ?

Réaliser un diagramme de classe pour chaque énoncé :

- Une voiture :
  - Quatre roues, des pneus
  - Un moteur,
  - Un coffre,
  - Des passagers….
- Un chien : c'est un mammifère qui appartient à un propriétaire, possède quatre membres, donne naissance éventuellement à des chiots.

Les diagrammes un par un

Université d'Évry-Val d'Essonne

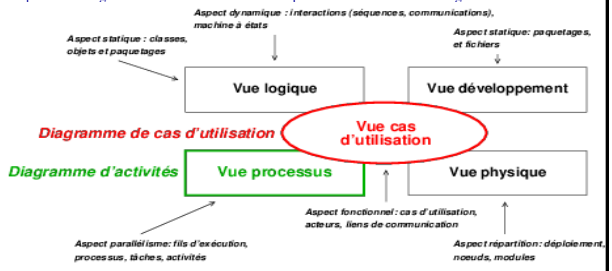
université PARIS-SACLAY

IV
Le diagramme de cas d'utilisation

Modèle de l'analyse

- Vue cas d'utilisation = ce que fait le système, les fonctionnalités
- Vue processus = ce que fait le système, les règles de gestion

L'élément pivot du modèle du système informatique est la vue cas d'utilisation. Cette vue est la plus proche du client et des utilisateurs finaux. C'est elle que ces derniers comprennent le mieux. Elle est complétée par la vue processus qui décrit comment le système répond aux stimulus externes. Cette vue processus présente les processus métier, par exemple les règles de gestion d'un système d'information. La première vue est composée des diagrammes de cas d'utilisation alors que la seconde contient les diagrammes d'activité.

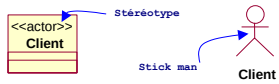


Le diagramme de cas d'utilisation

- Ce diagramme permet une description, en prenant le **point de vue de l'utilisateur**, du système à construire.
- Pas d'aspects techniques.
- Les deux concepts de base du diagramme :
  - l'acteur,
  - Le cas d'utilisation.

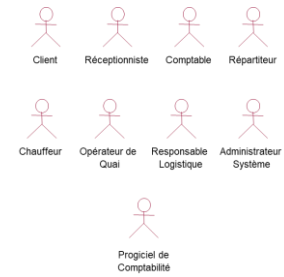
Acteur - définition

- Un acteur est une **entité externe** au système :
  - qui attend un ou plusieurs services du système,
  - à qui le système fournit une interface d'accès,
  - qui interagit avec le système par échange de messages.
- C'est une personne ou un autre système
- Les acteurs sont décrits par une abstraction ne retenant que le rôle qu'ils jouent vis à vis du système
- Notation :



Acteur - définition

- Exemples d'acteurs :



- Tous, humains ou système, interagissent avec le système considéré !

Acteurs vs Utilisateurs

- On ne doit pas raisonner en terme d'entité physique, mais en terme de **rôle** que l'entité physique joue
- Un acteur représente un rôle joué par un utilisateur qui interagit avec le système
- Exemple du monopoly :
  - La même personne physique peut jouer le rôle de plusieurs acteurs (joueur, banquier)
  - Plusieurs personnes peuvent également jouer le même rôle, et donc agir comme le même acteur (tous les joueurs)
  - le responsable d'un magasin est parfois responsable du magasin, parfois client, … → il joue plusieurs rôles.

Cas d'utilisation - définition

- Un cas d'utilisation modélise un **service rendu** par le système
- En anglais, on parle de « use case », on emploie aussi ce terme en français
- Il exprime les interactions entre les acteurs et le système
- Il apporte une valeur ajoutée "notable" aux acteurs concernés
- Règle de nommage : verbe (+ complément)
- Notation :

Cas d'utilisation, enchaînements et scénarios

- **Cas d'utilisation** : représente un cas en général, une représentation générale et synthétique d'un ensemble de scénarios similaires décrits sous la forme d'enchaînements

  Exemple : un joueur joue un coup en lançant 2 dés

- **Enchainement** : succession d'étapes qui se réalisent lorsqu'un acteur déclenche un cas d'utilisation.

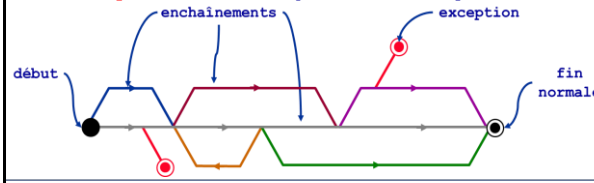
  Exemple : un joueur joue un coup, les 2 dés ont une valeur identique, le joueur peut ensuite rejouer un autre coup

- **Scénario** : exécution d'un ou plusieurs enchaînements, joignant le début du cas d'utilisation à une fin normale ou non.

  Exemple : le joueur Pierre joue : il obtient 7 avec les dés, se déplace rue de Belleville et achète la propriété

Cas d'utilisation vs scénario

- Un cas d'utilisation contient en général plusieurs enchainements.
- Pour décrire lisiblement un cas d'utilisation, on distinguera les trois types d'enchaînements :
  - **Nominal** : déroulement normal du cas d'utilisation
  - **Alternatif** : embranchements dans la séquence nominale
  - **Exception** : interviennent quand une erreur se produit



Pré et post conditions

- Pré-conditions : conditions obligatoires pour jouer un enchaînement du cas d'utilisation.
- Exemple : l'enchaînement « Faire un virement bancaire » a pour pré-condition : l'acteur doit être authentifié
- Post-conditions : changement intervenu dans le système considéré entre le début et la fin d'un enchaînement.
- Exemple : l'enchaînement « Faire une demande de prêt » a deux post-conditions :
  - La création dans le système d'un nouveau dossier de prêt,
  - L'envoi d'un mail de confirmation au client.

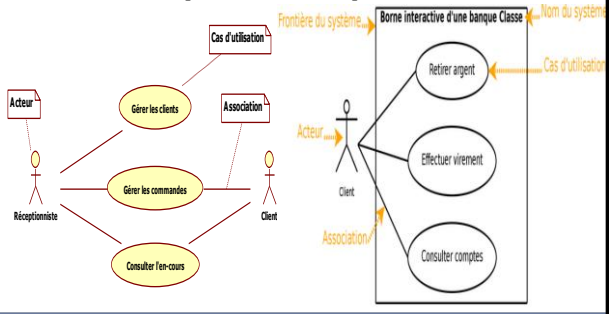
Intérêt des cas d'utilisation

- Un moyen de déterminer le **but** et le **périmètre** d'un système
- Utilisé par les utilisateurs finaux pour exprimer leur attentes et leur besoins ➔ permet d'impliquer les utilisateurs dès les premiers stades du développement
- Support de communication entre les équipes et les clients
- Découpage du système global en **grandes tâches** qui pourront être réparties entre les équipes de développement
  - **UC (Use Case) Permet de concevoir les Interfaces Homme-Machine**
  - Constitue une base pour les tests fonctionnels

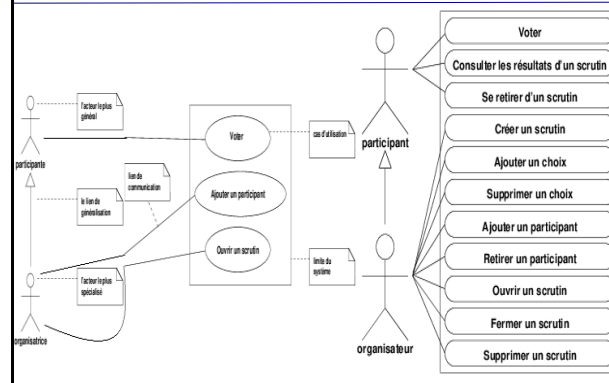
Exemple de diagramme de cas d'utilisation

Un acteur peut être associé à plusieurs cas d'utilisation.

Un cas d'utilisation peut faire intervenir plusieurs acteurs.



Exemple de diagramme de cas d'utilisation

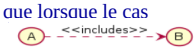


Relations entre cas d'utilisation

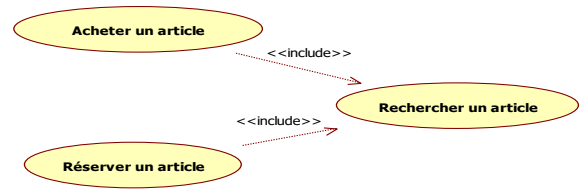
- UML définit trois types de relations standardisées entre cas d'utilisation :
  - une relation d'inclusion,
  - une relation d'extension,
  - une relation de généralisation/spécialisation.
- On peut également généraliser les acteurs

Relations entre cas d'utilisation

- <<include>> permet d'incorporer le comportement d'un autre cas d'utilisation.
- un cas d'utilisation **A** inclut (Include) un cas d'utilisation **B** c'est à dire que ce cas d'utilisation **A** ne se réalise que lorsque le cas d'utilisation **B** se réalise.

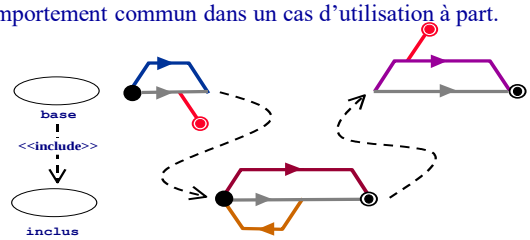


- Le cas de base en incorpore explicitement un autre, à un endroit spécifié dans sa description.



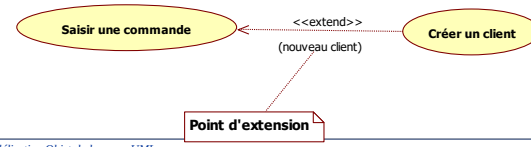
Relations entre cas d'utilisation

- La relation <<include>> se traduit de la façon suivante en terme d'enchaînements :
- On utilise fréquemment cette relation pour éviter de décrire plusieurs fois le même enchaînement, en factorisant le comportement commun dans un cas d'utilisation à part.



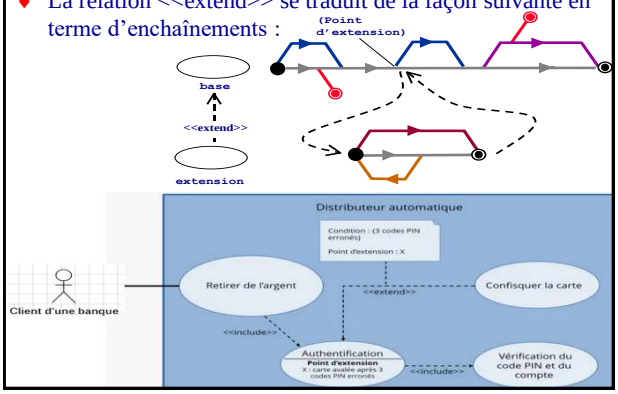
Relations entre cas d'utilisation

- <<extend>> permet de modéliser la partie d'un cas d'utilisation considérée comme facultative dans le comportement du système.
- On utilise principalement cette relation pour séparer le comportement optionnel (les variantes) du comportement obligatoire
- Le cas de base peut fonctionner seul, mais il peut aussi être complété par un autre, sous certaines conditions, et uniquement à certains points particuliers de son flot d'événements appelés points d'extension.



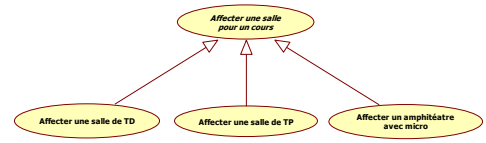
Relations entre cas d'utilisation

- La relation <<extend>> se traduit de la façon suivante en terme d'enchaînements :



Relations entre cas d'utilisation

- Les cas d'utilisation peuvent être hiérarchisés par généralisation.
- Les cas d'utilisation descendants héritent de la sémantique de leur parent. Ils peuvent comprendre des interactions spécifiques supplémentaires, ou modifier les interactions héritées.



Relations entre cas d'utilisation GAB

On utilise le comportement optionnel (les variantes) du comportement obligatoire

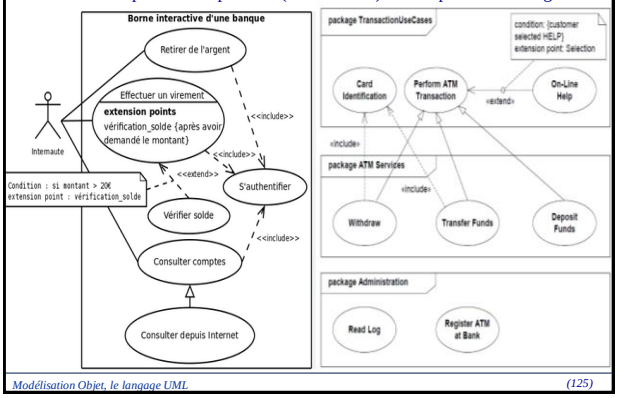


Diagramme de cas d'utilisation : Agence de voyage

Une agence de voyage organise des voyages où l'hébergement se fait en hôtel.
Le client doit disposer d'un taxi quand il arrive à la gare pour se rendre à l'hôtel.
Certains clients demandent à l'agent de voyage d'établir une facture détaillée.
Le voyage se fait soit par avion, soit par train.

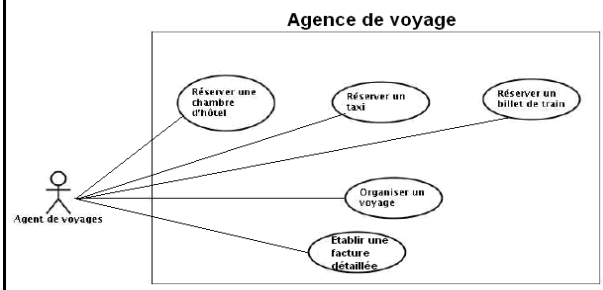
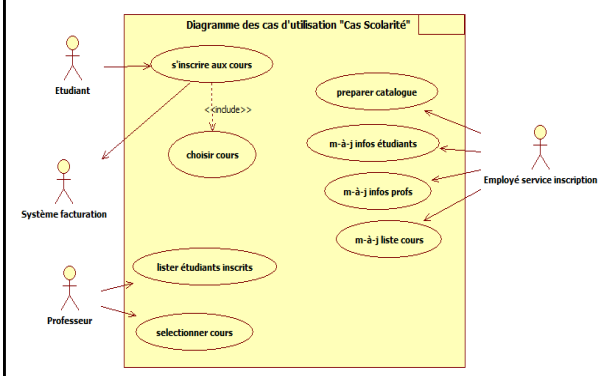


Diagramme de cas d'utilisation : Gestion de scolarité



Description textuelle pour les cas d'utilisation : Gestion de scolarité

♦ **UC1 :« s'inscrire aux cours » :**

Titre: s'inscrire aux cours

Résumé: Ce cas d'utilisation permet à l'étudiant de sélectionner cours et modifier son choix.

Acteurs:

Principal : Etudiant

Secondaire : Système de facturation

Pré conditions	Le catalogue contient des cours ;
Scénario nominal	1. l'étudiant obtient le catalogue 2. l'étudiant fait un choix 3. le système de facturation, sur la base des choix effectués par l'étudiant, calcule les droits d'inscription 4. l'étudiant s'acquitte des frais.
Post conditions	L'étudiant est inscrit dans les cours de son choix

Description textuelle pour les cas d'utilisation : Gestion de scolarité

- **UC2 :« lister les étudiants inscrits » :**

Titre: lister les étudiants inscrits

Résumé: Ce cas d'utilisation permet au professeur de se renseigner sur la liste des étudiants inscrits

Acteurs: Principal : Professeur

- **UC3 :« préparer le catalogue des cours » :**

Titre: préparer le catalogue des cours

Résumé: Ce cas d'utilisation permet à l'employé du service d'inscription de mettre à jour le catalogue

Acteurs: Principal : employé service d'inscription

- **UC4 :« sélectionner cours » :**

Titre: sélectionner cours

Résumé: Ce cas d'utilisation permet au professeur de choisir les cours qu'ils sont disposés à assurer

Acteurs: Principal : Professeur

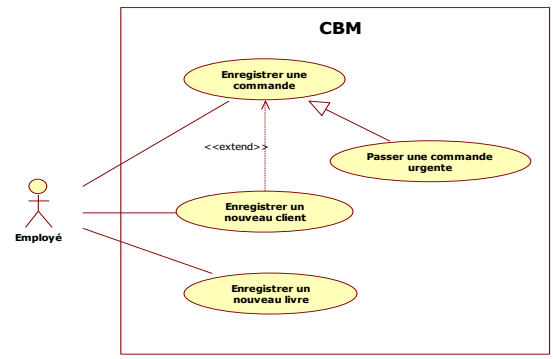
Spécification

- La description textuelle n'est pas normalisée par UML !
- Convergence vers un modèle standard :
  - Sommaire d'identification
    inclut titre, but, résumé, dates, version, responsable, acteurs…
  - Description des enchaînements
    décrit les enchaînements nominaux, les enchaînements alternatifs, les exceptions, mais aussi les préconditions, et les postconditions.
  - Exigences fonctionnelles
  - Besoins d'IHM
    ajoute éventuellement les contraintes d'interface homme-machine
  - Contraintes non-fonctionnelles
    ajoute éventuellement les informations suivantes : fréquence, volumétrie, disponibilité, fiabilité, intégrité, confidentialité, performances, concurrence, etc.
- On peut aussi réaliser des diagrammes

Exemple : La CBM

- La CBM (Computer Books by Mail) est une société de distribution d'ouvrages d'informatique qui agit comme intermédiaire entre les librairies et les éditeurs.
- Elle prend des commandes en provenance des libraires, s'approvisionne (à prix réduit) auprès des éditeurs concernés et livre ses clients à réception des ouvrages
- Il n'y a donc pas de stockage de livres.
- Seules les commandes des clients solvables sont prises en compte.
- Les commandes « urgentes » font l'objet d'un traitement particulier.
- La CBM désire mettre en place un Système Informatique lui permettant de gérer les libraires et les livres, et d'enregistrer les commandes.

Diagramme de cas d'utilisation CBM

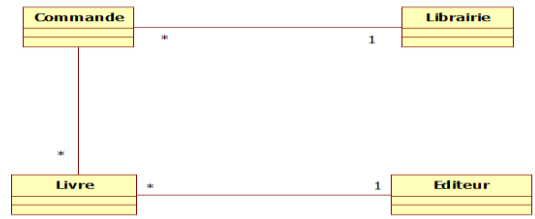


Spécification textuelle du cas « *Enregistrer une commande* »

Acteur : l'employé de la coopérative
Objectif : enregistrer une commande de livres
Précondition : le libraire existe
Enchainement nominal :
1 - l'employé sélectionne le libraire et vérifie sa solvabilité
2 - l'employé vérifie l'existence des livres
3 - l'employé précise la quantité pour chaque livre
4 – L'employé confirme la commande
Postcondition : une nouvelle commande est créée.
Enchainement d'exception 1 :
1a - le libraire n'est pas solvable
1b - le système alerte l'employé et lui propose d'arrêter l'enregistrement
Enchainement d'exception 2 :
2a - un des livres n'existe pas
2b – Le système édite une lettre qui pourra être envoyée au libraire. La commande est placée en attente.

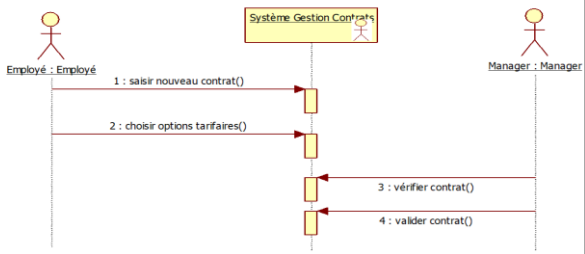
Diagrammes complémentaires

- On peut ajouter à chaque cas d'utilisation un diagramme de classes simplifié, appelé Diagramme de Classes Participantes (DCP)



Diagrammes complémentaires

- Autres diagrammes possibles pour compléter la description d'un cas d'utilisation :
  - Diagramme d'activités
  - Diagramme de séquence (vision boîte noire)



Pour finir : quelques pièges à éviter

- Des cas d'utilisation trop petits et trop nombreux
  - Jacobson : pas plus de 20 UC !
  - Larman : test du patron
  - Test de la taille
- Trop d'importance au diagramme
  - Pas trop de relations entre UC !
- Décomposition fonctionnelle
  - Garder le point de vue de l'utilisateur et pas le point de vue interne !
- Confusion entre processus métier et UC
  - Pas d'interactions entre acteurs directement !
  - Se concentrer sur les actions à automatiser !

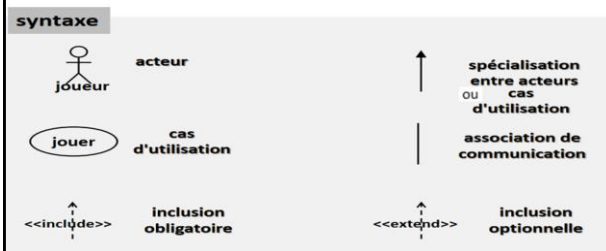
cas d'utilisation

- Le diagramme de cas d'utilisation correspond à :

→ une représentation globale (schéma) et détaillée textuelle (tableau) des activités d'un acteur (vision du système par l'acteur)

→ l'ensemble des opérations possibles entre cet acteur et le système

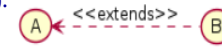
→une base de travail pour construire les interfaces graphiques



Relations entre cas d'utilisation

- *Inclusion* : B est une partie obligatoire de A et on lit *A inclut B* (dans le sens de la flèche)

- *Extension* : B est une partie optionnelle de A et on lit *B étend A* (dans le sens de la flèche).



- *Généralisation* : le cas A est une généralisation du cas B et on lit *B est une sorte de A*.

Les flèches en pointillés dénotent en fait une relation de *dépendance*, et les mentions *includes* et *extends* sont des *stéréotypes* et à ce titre placés entre guillemets.

Exemple de schéma UC = Use Case

Schématisation de l'expression des besoins

Ce diagramme se présente sous 2 formes :
→ schéma regroupant les activités d'un acteur
→ tableau détaillant les actions liées à une activité

Un joueur se connecte.

Une fois connecté, il choisit un personnage après avoir éventuellement consulté le catalogue des personnages.

Il peut alors entrer dans l'arène et jouer (se déplacer, attaquer les adversaires).

Il peut aussi, pendant le jeu, dialoguer avec les autres joueurs (chat).

À tout moment (connecté ou non), le joueur peut consulter l'aide du jeu.

Tableau des opérations d'un cas d'utilisation

Le tableau détaille les opérations d'un cas d'utilisation (un cercle).
Il sert de base à la construction des interfaces.

Cas d'utilisation	Se connecter
Acteur	joueur
Év. déclencheur	néant
Intérêts	Accéder au serveur afin d'être authentifié et accéder au jeu
Pré-conditions	Serveur actif
Post-conditions	Connexion établie
Scénario nominal	1. l'utilisateur saisit l'adresse IP du serveur 2. l'utilisateur demande une connexion au serveur 3. le serveur répond
Extensions	non
Contraintes	2a. L'adresse IP n'est pas remplie : aller en 1 3a. Le serveur retourne un message d'erreur : aller en 1 3b. Le serveur ne répond pas : aller en 1

Description graphique des cas d'utilisation

Pour documenter les cas d'utilisation, **la description textuelle est indispensable**, car elle seule permet de communiquer facilement avec les utilisateurs et s'entendre sur la terminologie métier employée dans le cahier des charges ou l'expression des besoins.

En revanche, le texte présente des désavantages puisqu'il est difficile de montrer comment les enchaînements se succèdent, ou à quel moment les acteurs secondaires sont sollicités. En outre, la maintenance des évolutions s'avère souvent fastidieuse. Il est donc recommandé de compléter la description textuelle par un ou plusieurs diagrammes dynamiques UML.

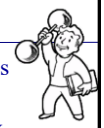
Description dynamique d'un cas d'utilisation

Pour les cas d'utilisation, on peut utiliser **le diagramme d'activité** car les utilisateurs le comprennent d'autant plus facilement qu'il paraît ressembler à organigramme traditionnel, ou un algorithme d'un programme.

Pour les scénarios particuliers, le **diagramme de séquence** est une bonne solution. Car il permet de démontrer les interactions entre l'acteur principal (positionné à gauche), puis un objet unique représentant le système à modéliser (boite noir) et des éventuels acteurs secondaires sollicités durant le scénario à droite du système.

Avec les intéressants ajouts au diagramme de séquence apportés par UML 2, en particulier les cadres d'interactions (avec les opérateurs loop, opt et alt par exemple), ainsi que la possibilité de référencer une interaction décrite par un autre Use Case.

Exercice 3



Représenter par un diagramme de cas d'utilisation les éléments de l'énoncé suivant :

- Le système de déclaration des impôts en ligne permet aux contribuables :
  - De saisir toutes les informations relatives à leurs revenus de l'année précédente,
  - De mettre à jour leurs informations administratives (adresse postale, …).
- La déclaration en ligne n'est accessible qu'aux contribuables authentifiés, au moyen de leur n° fiscal et de leur mot de passe.
- Lors de la saisie, le contribuable choisit de réaliser une déclaration simplifiée ou une déclaration complète.
- La déclaration d'impôts d'une année donnée n'est possible qu'après ouverture du service par les services fiscaux.

Les diagrammes un par un

Université d'Évry-Val d'Essonne

université PARIS-SACLAY

V

Le diagramme de séquence
Le diagramme de communication

Objectifs

- Décrire des scénarios particuliers
- Capturer l'ordre des interactions entre les parties du système (objets, composants, acteurs, …)
- Décrire les interactions déclenchées lorsqu'un scénario d'un cas d'utilisation est exécuté
- Montrer la dynamique d'un système. On a dit qu'une application objet rendait des services grâce à la collaboration et à l'échange de messages entre objets, ici on va montrer ces échanges de messages.
- Rappel : scénario = instance d'enchaînement d'un UC.
- Montre les échanges de messages permettant de fournir les services attendus.

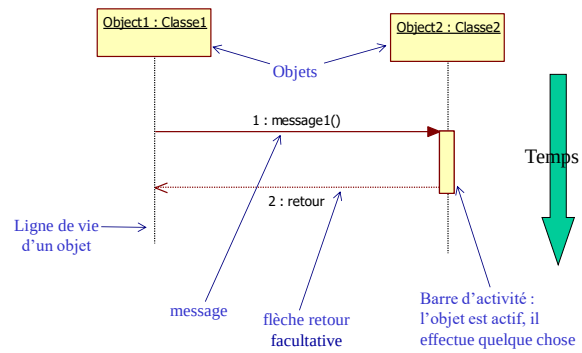
Diagramme de séquence / diagramme de communication

- Ces diagrammes comportent :
  - des objets dans une situation donnée (instances)
  - les messages échangés entre les objets
- Les objets sont les instances des classes identifiées et décrites dans le diagramme de classes.
- On travaille sur un scénario → sur des instances particulières de classes.

Diagramme de séquence / diagramme de communication

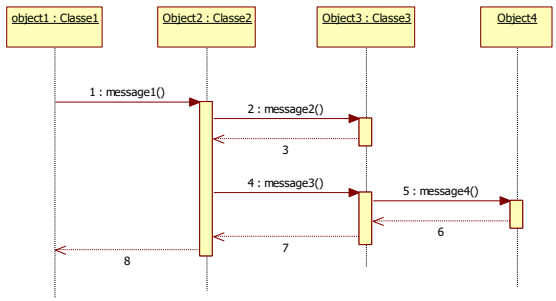
- UML propose deux types de diagrammes pour modéliser la collaboration entre les objets du système :
  - Le diagramme de séquences,
  - Le diagramme de communication
- Ces deux diagrammes sont regroupés sous le terme de diagrammes d'interactions.
- Nous nous focaliserons dans un premier temps sur le diagramme de séquences.

Diagramme de séquence en détail



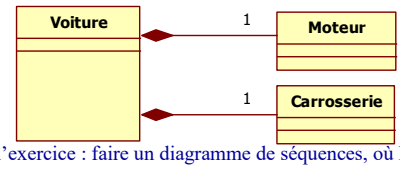
Messages imbriqués

- Un message peut conduire à l'envoi d'un ou plusieurs messages par le récepteur
- C'est par la collaboration des objets que le système rend les services attendus !



Exemple

- Comment calculer le poids d'une voiture, égal au poids du moteur plus le poids de la carrosserie ?

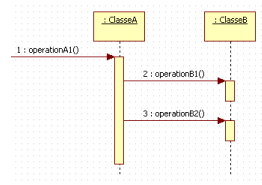


Solution de l'exercice : faire un diagramme de séquences, où l'on voit un acteur appeler la voiture, qui va envoyer à Moteur et à Carroserie le message calculerPoids().

La voiture ne va pas faire le calcul complet toute seule ! Elle va s'adresser au moteur et à la carrosserie.

Diagrammes de séquences et diagramme de classes

- Le diagramme de classes présente une vue statique du système.
- Le diagramme de séquences présente une vue dynamique du système.



Transposition en java :

```
public class ClasseA
{
public objetB = new ClasseB();
public void operationA1() {
        objetB.operationB1();
        objetB.operationB2();
}
}

Public class ClasseB
{
Public void operationB1() {}
Public void operationB2() {}
}
```


Messages synchrones/asynchrones

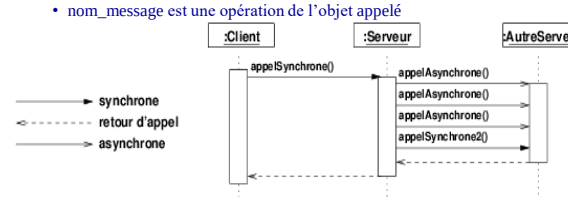
- Message synchrone : l'émetteur est bloqué pendant le traitement et attend le retour du récepteur du message

l'expéditeur est bloqué pendant le traitement Le retour d'appel est optionnel (implicite)

- Message asynchrone : l'émetteur n'est pas bloqué, il continue l'exécution de ses traitements

l'expéditeur continue son exécution pendant le traitement du message, exemple : envoi d'un mail, d'une impression.

- attribut = nomMessage(arguments) : type_retour
  - attribut peut être un attribut de l'objet appelant ou une variable locale
  - nom_message est une opération de l'objet appelé



Syntaxe et types de messages

L'interaction la plus petite est l'événement. Un événement est une interaction pendant laquelle « quelque chose » arrive. Les événements sont les constituants de base des messages, aussi appelés « signaux » en automatique. Un message est constitué d'un événement d'émission chez l'appelant et d'un événement de réception chez l'appelé, et possède une signature : « attribut = nom_message(arguments) : type_retour ». attribut peut être un attribut de l'objet appelant ou une variable locale. L'opération nom_message est une opération de l'objet appelé qui retourne un objet de type type_retour qui est affecté à l'attribut de l'objet appelant ou à la variable locale. La signature du message respecte la signature de l'opération appelée. La notation UML des diagrammes de séquence n'oblige pas à renseigner tous les éléments des prototypes des messages. Ainsi, les premiers diagrammes de séquence de l'analyse indiquent par exemple uniquement les noms des opérations. Ensuite, les mêmes diagrammes sont raffinés pour y ajouter les arguments et les types de retour, puis les attributs des classes appelantes ou variables locales recevant les valeurs de retour.

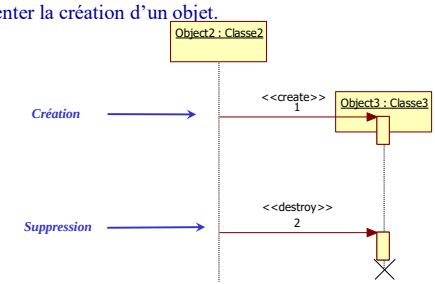
Syntaxe et types de messages

Un message synchrone est une invocation d'opération bloquant l'appelant jusqu'à ce que l'appelé effectue le traitement et retourne l'appel, que ce dernier contienne ou non une valeur de retour.
Un retour d'appel est un message que vous pouvez représenter à la fin de la barre d'activation de l'appelé pour indiquer que le traitement est terminé et que la valeur de retour est passée à l'appelant qui peut reprendre son traitement après la fin de l'appel synchrone. Comme la barre d'activation, la représentation de l'appel de retour est optionnelle.

Un message asynchrone est initié par l'appelant qui continue son exécution car il n'attend pas le retour de l'appel. L'appelant peut par exemple émettre à destination d'un même objet appelé une salve de messages asynchrones avant un message synchrone demandant le résultat de tous les traitements précédents. Cela lui permet d'effectuer aussi des traitements en parallèle, et dans cet exemple, cela diminue d'autant le nombre de messages échangés (car il n'y a qu'un seul retour d'appel). Il est important de noter qu'un message asynchrone ne possède pas de valeur de retour.

Messages de création/destruction

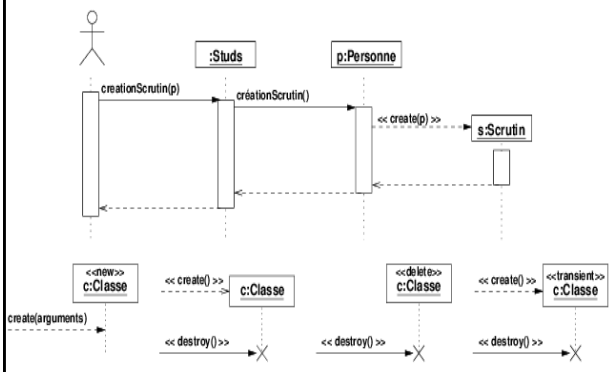
- Rappel : les objets naissent, vivent et meurent
- Attention : certains AGL, comme celui que j'ai utilisé pour ce slide, ne supporte pas la technique du rectangle de titre surbaissé pour représenter la création d'un objet.



Création et suppression d'objets

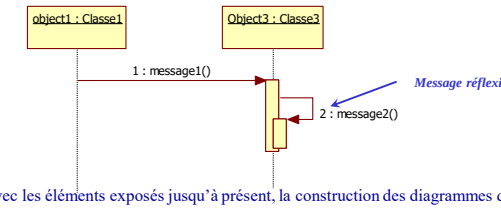
Certains objets vivent pendant tout le diagramme, d'autres sont créés et/ou meurent pendant la séquence. Pour montrer qu'un participant est créé lors de la séquence, vous pouvez soit placer l'objet en haut du diagramme en y ajoutant le stéréotype «new» et utiliser un message synchrone appelant l'opération create(arguments) , soit placer l'objet plus bas dans le diagramme au niveau du message synchrone de création et utiliser le stéréotype «create(arguments)» pour nommer ce message synchrone. Le message correspondant à l'opération de création d'un objet est un message particulier. L'opération est dans la suite nommée un constructeur. Notez que l'objet en question n'existe pas avant le message, c'est-à-dire avant sa création, et que l'opération utilisée ne possède pas de type de retour ; c'est une seconde particularité.
Par analogie avec la création d'un objet dans un diagramme de séquence, la destruction d'un objet pendant une séquence est modélisée soit en plaçant l'objet en haut du diagramme en y ajoutant le stéréotype «delete» , la destruction étant repérée par un message synchrone appelant l'opération destroy() , soit en plaçant l'objet plus bas dans le diagramme au niveau du message synchrone de suppression.
Enfin, un objet est dit transitoire (en anglais, transient) lorsqu'il est créé puis détruit durant la même séquence. Le stéréotype de l'objet est alors « «transient» ».

Création et suppression d'objets



Messages réflexifs

- Envoi d'un message d'un objet à lui-même



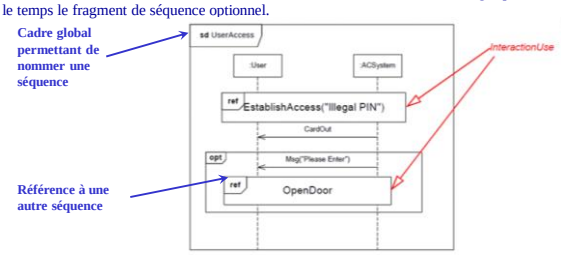
Avec les éléments exposés jusqu'à présent, la construction des diagrammes de séquence devient vite laborieuse car les diagrammes se complexifient rapidement. Pour ce faire, UML propose une notion de bloc appelé « fragment de séquence » permettant d'inclure dans un rectangle des sous-parties de diagrammes de séquence. Un fragment de séquence indique dans la cartouche situé dans le coin haut à gauche le nom de l'opérateur du fragment. Une autre raison de l'intérêt des fragments est la possibilité d'exprimer les instructions de choix, les itérations... des algorithmes classiques. Nous présentons dans cette diapositive deux premiers types de fragments de séquence.

Fragments : Cadres d'interaction et fragments combinés

les diagrammes de séquence pour structurer les interactions complexes

Le fragment de séquence ref permet d'inclure une sous-séquence du diagramme de séquence, la sous-séquence étant décrite dans un autre diagramme de séquence.

Le fragment de séquence opt présente une sous-séquence exécutée si une condition de garde est vraie. Les termes de la condition sont souvent des valeurs de retour des messages précédant dans le temps le fragment de séquence optionnel.

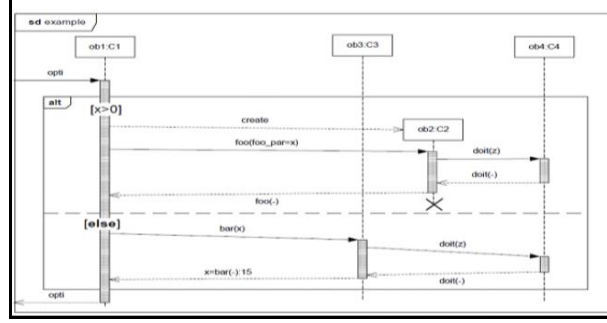


ref : sous-séquence détaillée dans un autre diagramme de séquence

opt : sous-séquence optionnelle exécutée si condition de garde est vraie

Fragments (UML2)

- De la même façon, les fragments combinés permettent d'enrichir un diagramme de séquence.
- Exemple : fragment de type **alt (**alternative if then else) :
- Autres fragments utiles : **opt** (partie optionnelle), **loop** (boucle tant que).



Utilisation du diagramme de séquence

- Dans la vue fonctionnelle du système
  - Diagramme complémentaire de description d'un cas d'utilisation
  - Décrire un scénario d'un cas d'utilisation : décrire les interactions entre le système et son environnement (vision boîte noire)
  - Appelé diagramme de séquence « système »
- Dans l'analyse détaillée et la conception OBJET du système
  - Décrire les interactions internes du système : les interactions entre les objets

Diagramme de séquence GAB : Description textuelle des cas d'utilisation Retirer de l'argent GAB

Flots des événements pour le cas d'utilisation:
Retirer l'argent avec la carte Visa

Une fois les cas d'utilisation identifiés, il faut encore les décrire!
Sommaire d'identification:
Titre: **Retirer de l'argent avec une carte Visa**
Résumé: Ce cas d'utilisation permet à un porteur de carte Visa, qui n'est pas client de la banque, de retirer de l'argent, si son crédit hebdomadaire le permet.
Acteurs: Porteur de CB Visa, SA Visa

Pré conditions	La caisse du GB est alimentée. Aucune carte bancaire ne se trouve dans le lecteur
Scénario nominal	1. **Le porteur de CB visa introduit sa carte Visa dans le lecteur de cartes du GB** 2. Le GB vérifie que la carte introduite est bien une carte Visa. 3. Le GB demande au porteur de CB Visa de saisir son code d'identification. 4. **Le porteur de CB Visa saisit son code d'identification.** 5. Le GB compare le code d'identification avec celui qui est codé sur la puce de la carte. 6. Le GB demande une autorisation au système d'autorisation VISA. 7. Le système d'autorisation VISA donne son accord et indique le solde hebdomadaire 8. Le GB demande au porteur de CB Visa de saisir le montant désiré du retrait. 9. **Le porteur de CB visa saisit le montant désiré de retrait.** 10. Le GB contrôle le montant demandé par rapport au solde hebdomadaire. 11. Le GB demande au porteur de CB Visa s'il veut un ticket. 12. **Le porteur de CB répond OK.** 13. Le GB rend sa carte au porteur de CB Visa. 14. **Le porteur de CB Visa reprend sa carte.** 15. Le GB délivre les billets et un ticket. 16. **Le porteur de CB Visa reprend les billets et le ticket.**

Diagramme de séquence GAB : Description textuelle des cas d'utilisation Retirer de l'argent GAB

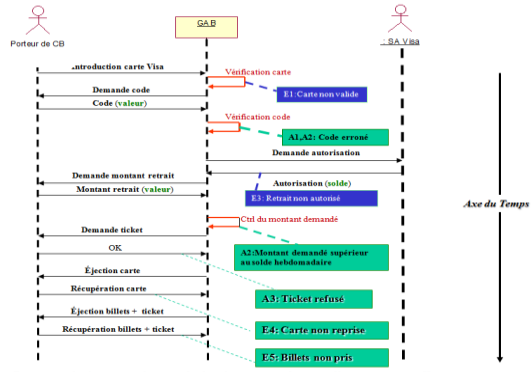


Diagramme de séquence système enrichi du scénario nominal « retirer l'argent avec carte Visa »

Description des enchaînements: Description textuelle des cas d'utilisation Retirer de l'argent GAB

Enchaînements alternatifs	**A1 : Code d'identification provisoirement erroné** L'enchaînement A1 démarre au point 5 du scénario 6. Le GB indique au client que le code est erroné, pour la première ou la deuxième fois. 7. Le GB enregistre l'échec sur la carte Le scénario reprend au point 3. **A2 : Montant demandé supérieur au solde hebdomadaire** L'enchaînement A2 démarre au point 10 du scénario nominal. 11. Le GB indique au client que le montant demandé est supérieur au solde hebdomadaire. Le scénario nominal reprend au point 3. **A3 :Ticket refusé** L'enchaînement A3 démarre au point 11 du scénario nominal. 12. Le porteur de CB Visa refuse le ticket. 13. Le GB rend sa carte au porteur de CB visa . 14. Le porteur de CB Visa reprend sa carte. 15. Le GB délivre les billets. 16. Le porteur de CB Visa prend les billets.
Cas d'exception	**E1 : carte non-valide** L'enchaînement E1 démarre au point 2 du scénario nominal. 3. Le GB indique au porteur que la carte n'est pas valide (illisible, périmée, etc..), la confisque, le cas d'utilisation est terminé.

Description des enchaînements: Description textuelle des cas d'utilisation Retirer de l'argent GAB

Cas d'exception	**E2 : Code d'identification définitivement erroné** L'enchaînement E2 démarre au point 2 du scénario nominal. 6. Le GB indique au client que le code est erroné, pour la troisième fois. 7. Le GB confisque la carte. 8. Le système d'autorisation VISA est informé ;le cas d'utilisation est terminé. **E3 : Retrait non autorisé** L'enchaînement E3 démarre au point 6 du scénario nominal. 7. Le système d'autorisation VISA interdit tout retrait. 8. Le GB éjecte la carte ; le cas d'utilisation est terminé. **E4 : Carte non reprise** L'enchaînement E4 démarre au point 13 du scénario nominal. 14. Au bout de 30 secondes, le GB confisque la carte. 15. Le système d'autorisation VISA est informé ; le cas d'utilisation est terminé. **E5 : Billet non pris** L'enchaînement E5 démarre au point 15 du scénario nominal. 16. Au bout de 30 secondes, le GB reprend les billets . 17. Le système d'autorisation VISA est informé ; le cas d'utilisation est terminé
Postconditions	La caisse de GB contient moins de billets qu'au début du cas d'utilisation (le nombre de billets manquants est fonction du montant du retrait).

On peut éventuellement compléter la description du Cas d'Utilisation avec les deux paragraphes optionnels suivants:

1- Besoins d'LH.M:

Les dispositifs d'entrée/Sortie à la disposition du porteur de CB Visa doivent être:

- Un **lecteur** de carte bancaire.
- Un **clavier** numérique (pour le code), avec des touches « validation », « correction » et « annulation ».
- un **écran** pour l'affichage des messages du GAB.

Schématisation des messages entre objet

Quand l'utilisateur se connecte, l'objet joueur et sa boule sont créés côté serveur. Quand une flèche est utilisée, l'objet joueur demande au serveur de se déplacer. Le serveur retourne l'affichage de la nouvelle position. Quand la touche espace est utilisée, l'objet joueur demande au serveur d'attaquer. Le serveur retourne l'affichage de la boule qui est lancée. Le serveur contrôle la vie des joueurs et fait mourir ceux qui sont à 0.

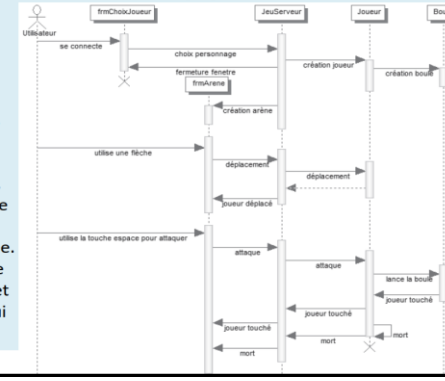


Diagramme de séquence

Le diagramme de séquences est :

→la représentation temporelle des messages entre objets

→ le cycle de vie d'un ensemble d'objets liés dans un cas d'utilisation

→la représentation des méthodes qui agissent entre objets

syntaxe

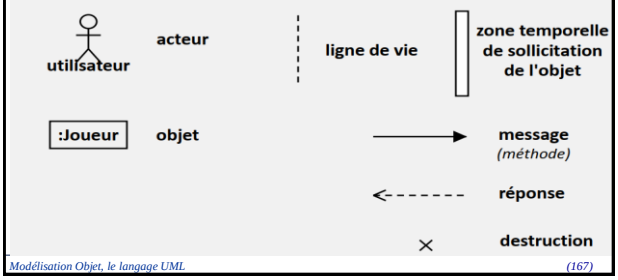


Diagramme de séquence

Le diagramme de séquences permet de lister les méthodes publiques, leur classe et leur rôle

Voici un exemple de méthodes correspondant aux premiers messages échangés dans le diagramme de séquences précédent.

Nom méthode	Classe	Rôle méthode
choixPersonnage()	frmChoixJoueur	Avertit JeuServeur du choix d'un personnage et donc de l'arrivée d'un nouveau joueur.
creationJoueur()	JeuServeur	Crée une nouvelle instance de Joueur.
creationBoule()	Joueur	Crée une nouvelle instance de Boule (dès qu'un joueur est créé, sa boule est créée).
fermetureChoixJoueur()	JeuServeur	Détruit l'objet frmChoixJoueur.
creationArene()	JeuServeur	Crée une instance de frmArene
Deplacement()	frmArene	Avertit JeuServeur d'un déplacement.
Deplacement()	JeuServeur	Utilise Joueur pour enregistrer le déplacement. Joueur retourne la nouvelle position.
...		

Diagramme de communication

- Diagramme de séquences : l'accent est mis sur l'ordre temporel des interactions
- Diagramme de communication : l'accent est mis sur l'examen des interactions vis-à-vis des liens entre objets.

- Équivalents en UML1, quelques nouveautés sur le diagramme de séquences en UML2, non disponibles sur le diagramme de communication
- Les diagrammes de séquence et de communication sont tellement similaires que de nombreux outils de modélisation permettent de transformer l'un en l'autre et vice-versa.

Diagramme de communication

- Focalisation sur les liens entre objets d'une interaction

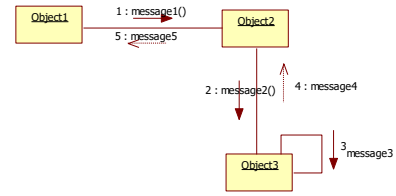


Diagramme de communication

- Exemple :

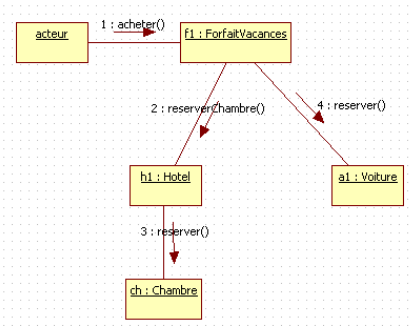
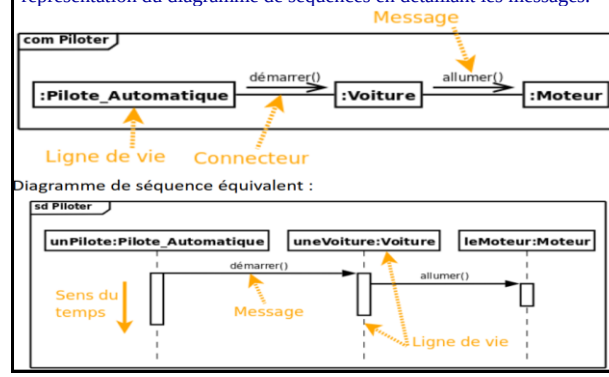


Diagramme de communication

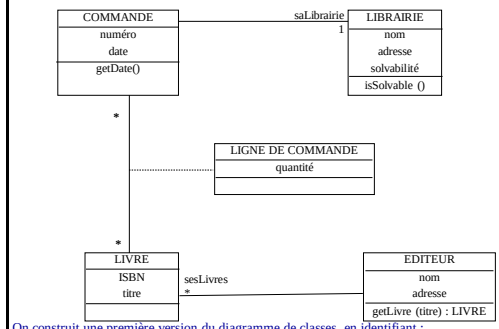
Ce diagramme représente la coopération entre objets. C'est une autre représentation du diagramme de séquences en détaillant les messages.



Exemple : La CBM

- Description du système :
  - Reprendre l'énoncé de l'exercice sur les cas d'utilisation.
- Objectif :
  - Réaliser un diagramme de classes pour le système CBM,
  - Réaliser un diagramme de séquence objet lorsque qu'un utilisateur demande les détails de toutes les commandes d'une librairie donnée.

CBM : diagramme de classes

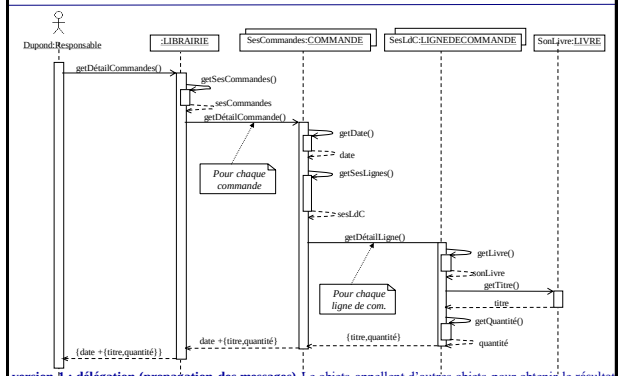


On construit une première version du diagramme de classes, en identifiant :
- Les classes, les propriétés des classes, les relations,
- Les premières méthodes nées de la lecture de la description du système.

CBM : un diagramme de séquence

- Diagramme de séquence : « Communiquer les détails de toutes les commandes d'une librairie donnée »
- Ce diagramme correspond à la méthode ***getDétailCommandes()*** de la classe LIBRAIRIE
- On fait l'analyse/conception de la méthode getDétailsCommande()
- Cela semble intéressant car a priori met en jeu de nombreuses interactions entre objets.

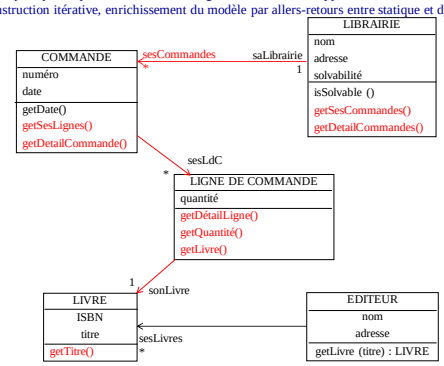
CBM : diagramme de séquence V1



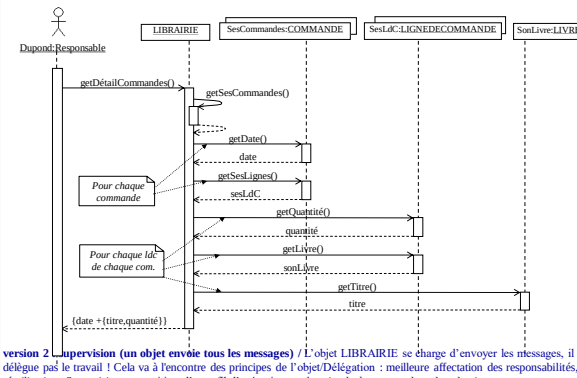
version 1 : délégation (propagation des messages) Le objets appellent d'autres objets pour obtenir le résultat
On s'adresse systématiquement à l'objet le mieux « placé » pour réaliser le traitement.

CBM : diagramme de classes complété

L'analyse dynamique vient enrichir le diagramme de classes. Apparition des méthodes.
Construction itérative, enrichissement du modèle par allers-retours entre statique et dynamique

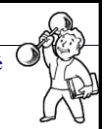


CBM : diagramme de séquence V2

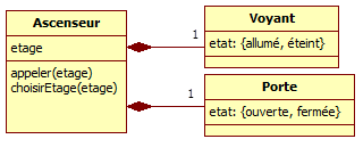


version 2 : supervision (un objet envoie tous les messages) / L'objet LIBRAIRIE se charge d'envoyer les messages, il ne délègue pas le travail ! Cela va à l'encontre des principes de l'objet/Délégation : meilleure affectation des responsabilités, réutilisation. Supervision : apparition d'un « fil d'exécution », plus simple à comprendre, plus classique.

Exercice 4



- Soit la modélisation (simplifiée) d'un ascenseur, composé d'une porte et d'un voyant :

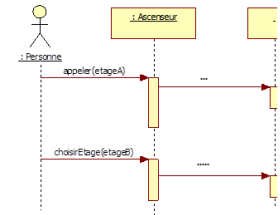


Note : les éléments présentés entre { } à droite d'un attribut correspondent à la liste des valeurs possibles de l'attribut.

- Lorsqu'une personne appelle l'ascenseur pour se rendre à un étage, le voyant s'allume, puis si l'ascenseur se trouve à un étage différent de l'étage de la personne, la porte se ferme, l'ascenseur se déplace jusqu'à l'étage de la personne et la porte s'ouvre.
- Ensuite, lorsqu'une personne sélectionne un étage, la porte se ferme, l'ascenseur se déplace à l'étage désiré, puis la porte s'ouvre et le voyant s'éteint.

Exercice 4

- Représenter par un diagramme de séquences « objet » les interactions déclenchées lorsqu'une personne présente à l'étage « etageA » appelle l'ascenseur pour se rendre à l'étage « etageB ». On complétera pour ce faire le diagramme suivant :



- Compléter le diagramme de classes à partir du diagramme de séquences réalisé.

Les diagrammes un par un

VI

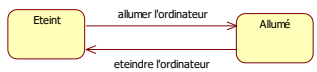
Le diagramme d'états-transitions

Diagramme d'états

- Vue synthétique du fonctionnement dynamique d'un objet
- Description du comportement d'un objet tout au long de son cycle de vie :
  - Description de tous les états possibles d'un **unique** objet à travers l'**ensemble** des cas d'utilisation dans lequel il est impliqué
  - Utile pour les objets qui ont un **comportement complexe**. Un diagramme d'états est alors réalisé pour la classe qui décrit ces objets au comportement complexe.
  - Un diagramme d'état est associé à une et une seule classe.
- Attention : toutes les classes n'ont pas besoin d'un diagramme d'états, mais seules les plus complexes en terme de métier (impliquées dans le plus grand nombre de scénarios ou de cas d'utilisation).
- Le diagramme d'état permet une vue "orthogonale" par rapport aux UC.

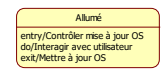
Diagramme d'états

- Éléments fondamentaux du diagramme :
  - Les états : représente une situation dans la vie d'un objet pendant laquelle il satisfait une certaine condition, exécute certaines activités, attend certains évènements.
  - Les transitions entre états : pour marquer le changement d'état d'un objet.
  - Les évènements (ou déclencheurs) qui provoquent des changements d'état.
- Exemple : diagramme d'état d'un ordinateur :



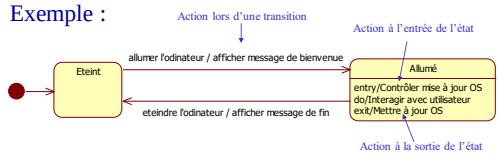
Les états en détail

- Plusieurs types d'états :
  - État initial (un seul par diagramme) :
  - État final (aucun ou plusieurs possibles) :
  - État intermédiaire (plusieurs possibles) : Nom de l'état
- Contenu d'un état intermédiaire :
  - le nom
  - l'activité attachée à cet état
  - les actions réalisées pendant cet état
- Exemple : nom d'état « Allumé » de l'ordinateur :



Actions

- Opération instantanée (durée négligeable) toujours intégralement réalisée.
- Exécutée :
  - lors d'une transition : $event_i$ / $action_i$
  - à l'entrée dans un état : **entry** / $action_i$
  - à la sortie d'un état : **exit** / $action_i$
  - interne, sans changer d'état : $event_i$ / $action_i$
- Exemple :



Activités

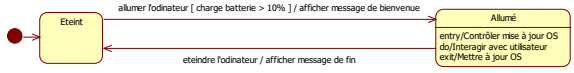
A la différence d'une action :

- Opération qui nécessite un certain temps d'exécution.
- Peut être interrompue à chaque instant.
- Exécutée entre l'entrée et la sortie de l'état.
- Notation : **do** / activité
- "do" : c'est pour une activité, pas une action.
- Exemple :

mots-clés "entry" et "exit" : évènements prédéfinis par UML.
On peut spécifier n'importe quel autre évènement.
"do" : c'est pour une activité, pas une action.
A la différence d'une action, une activité nécessite un certain temps et peut être interrompue par un évènement.

Transitions

- Passage unidirectionnel et instantané d'un état à un autre état
- Déclenchée :
  - par un **événement,**
  - automatiquement à la fin d'une **activité** (transition automatique).
- **Condition de garde** : condition booléenne qui autorise ou bloque la transition
- **Action** : réalisée lors du changement d'état
- Syntaxe complète : **<Événement> [<Garde>] / <Action>**
- Sur une transition: évènement, garde et action sont facultatifs.
- Exemple :



Transitions

- Deux types d'évènements particuliers :
  - Le passage du temps (time event) : représente une durée décomptée à partir de l'entrée dans l'état courant
    Notation : after(expression représentant une durée)
  - Un changement interne à l'objet (change event) : sans réception de message (souvent le temps)
    Notation : when(expression booléenne)
- Exemple : diagramme d'états d'un réveil :

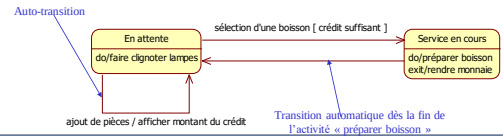
Transitions

- **Transition automatique** : lorsqu'il n'y a pas de nom d'événement sur une transition, il est sous-entendu que la transition aura lieu dès la fin de l'activité.
- **Auto-transition** : transition d'un état vers lui-même

Auto-transition : permet de spécifier que certains évènements interrompent l'activité mais ne modifient pas l'état.

Exemple : classe Commande. Etat "en cours". L'évènement "ajout article" est associé à une auto-transition, ce qui permet d'exécuter l' action "exit/mettre à jour montant total" à chaque ajout d'article.

- Exemple : diagramme d'états du distributeur de boissons, On peut pour rendre plus lisible le diagramme mettre comme évènement la fin de l'activité ("fin préparation boisson" au lieu de transition automatique) :

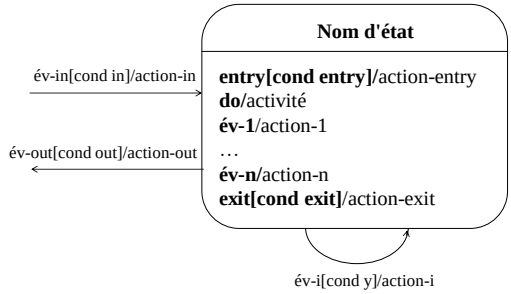


Exercice 5

- Représenter sous forme de diagramme d'états le fonctionnement d'un lecteur de DVD, dont voici la description :
  - Le lecteur possède un tiroir qui peut recevoir plusieurs disques.
  - A la fin de la lecture d'un disque, le lecteur démarre la lecture du disque suivant.
  - A la fin du dernier disque, le lecteur s'arrête.
  - L'utilisateur peut lancer la lecture des disques, arrêter la lecture ou mettre en pause le lecteur.

Forme générale d'un état

En résumé, voici les principales notations associées au diagramme d'états. Il y en a d'autres, plus complexes.

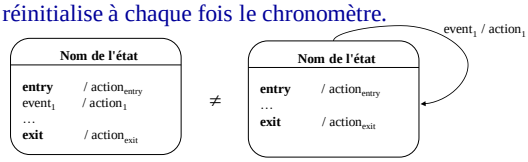


Ordonnancement des actions

- En entrée
  - Action sur la transition d'entrée
  - Action d'entrée
  - Activité associée à l'état
- En interne
  - Interruption de l'activité en cours (contexte sauvé)
  - Action interne
  - Reprise de l'activité
- En sortie
  - Interruption de l'activité en cours (contexte perdu)
  - Action de sortie
  - Action sur la transition de sortie
- Auto-transition
  - Interruption de l'activité en cours (contexte perdu)
  - Action de sortie
  - Action sur l'auto-transition
  - Action d'entrée
  - Activité associée à l'état

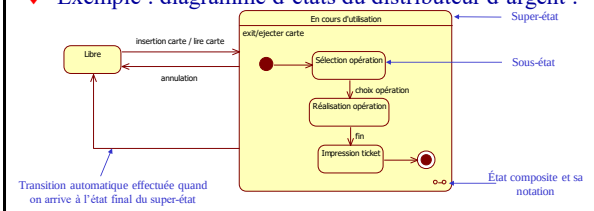
Auto-transition / Action interne

- Action interne, le contexte de l'activité est préservé (on ne sort pas de l'état)
- Auto-transition : le contexte est réinitialisé (on sort et on re-rentre dans l'état)
- Exemple d'activité : sonner pendant 5 minutes. Si on sort de l'état et que y entre à nouveau, alors on réinitialise à chaque fois le chronomètre.



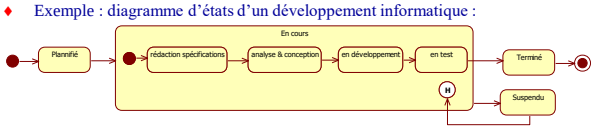
Hiérarchie d'états

- Permet de structurer les diagrammes complexes.
- Factorisation des actions, activités et transitions dans un super-état ou état composé.
- Décomposition d'un état en sous-états. Un sous état hérite des transitions du parent.
- Exemple : diagramme d'états du distributeur d'argent :



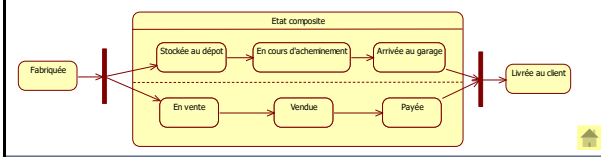
Pseudo-état historique

- Quand une transition sort d'un super-état :
  - le dernier sous-état atteint n'est plus connu,
  - une nouvelle entrée dans le super état redémarre au premier sous-état.
- Un pseudo-état historique permet de mémoriser le dernier sous-état.
  Les transitions peuvent avoir pour cible la frontière d'un état composite et sont équivalentes à une transition ayant pour cible l'état initial de l'état composite. Une transition ayant pour source la frontière d'un état composite est équivalente à une transition qui s'applique à tout sous-état de l'état composite source. Cette relation est transitive : la transition est franchissable depuis tout état imbriqué, quelle que soit sa profondeur.
- Notation : Ⓗ
- Exemple : diagramme d'états d'un développement informatique :



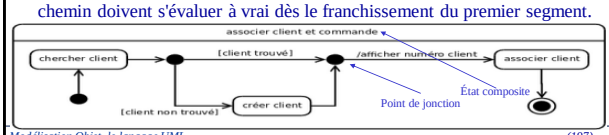
Comportements concurrents, plusieurs états en parallèle

- Un objet peut se trouver dans plusieurs états à la fois. On crée alors un état composé de plusieurs **régions** concurrentes.
- Chaque région possède son propre diagramme d'état.
- Lorsqu'un objet est dans l'état composite, il se trouve dans un état de chaque diagramme d'état.
- **Exemple** : dans une compagnie d'assurance, un dossier peut avoir pour états : {à traiter, dossier complet, dossier incomplet, dossier tarifé} et {en attente décision, en cours de contrôle, contrôle OK, contrôle KO}, synchronisation puis {dossier en paiement, dossier refusé}.
- **Autre exemple** de plusieurs états en parallèle: diagramme d'état d'une automobile



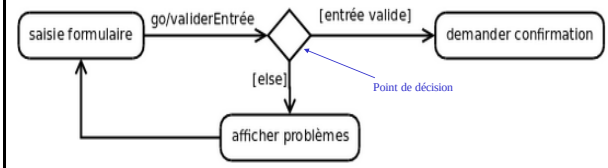
Point de jonction

- Les points de jonction sont un artefact graphique qui permet de partager des segments de transition, l'objectif étant d'aboutir à une notation plus compacte ou plus lisible des chemins alternatifs.
- Un point de jonction peut avoir plusieurs segments de transition entrante et plusieurs segments de transition sortante. Par contre, il ne peut avoir d'activité interne ni des transitions sortantes dotées de déclencheurs d'événements.
- Il ne s'agit pas d'un état qui peut être actif au cours d'un laps de temps fini. Lorsqu'un chemin passant par un point de jonction est emprunté (donc lorsque la transition associée est déclenchée) toutes les gardes le long de ce chemin doivent s'évaluer à vrai dès le franchissement du premier segment.



Point de décision

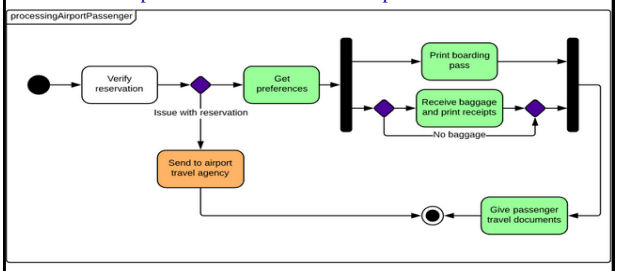
- Un point de décision possède une entrée et au moins deux sorties. Contrairement à un point de jonction, les gardes situées après le point de décision sont évaluées au moment où il est atteint. Cela permet de baser le choix sur des résultats obtenus en franchissant le segment avant le point de choix. Si, quand le point de décision est atteint, aucun segment en aval n'est franchissable, c'est que le modèle est mal formé.
- Il est possible d'utiliser une garde particulière, notée [else], sur un des segments en aval d'un point de choix. Ce segment n'est franchissable que si les gardes des autres segments sont toutes fausses. L'utilisation d'une clause [else] est recommandée après un point de décision, car elle garantit un modèle bien formé.



Région représentant un flot d'exécution

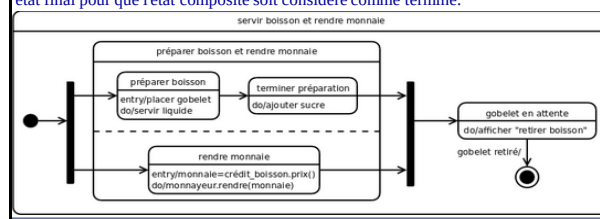
Exemple de diagramme d'état transition du processus d'enregistrement dans un aéroport

L'exemple suivant simplifie les étapes d'enregistrement dans un aéroport. Pour les compagnies aériennes, un diagramme d'état-transition permet de rationaliser les processus et d'éliminer les étapes inutiles.



Région représentant un flot d'exécution

Les diagrammes d'états-transitions permettent de décrire efficacement les mécanismes concurrents grâce à l'utilisation d'états orthogonaux. Un état orthogonal est un état composite comportant plus d'une région, chaque région représentant un flot d'exécution. Graphiquement, dans un état orthogonal, les différentes régions sont séparées par un trait horizontal en pointillé allant du bord gauche au bord droit de l'état composite.
Une transition qui atteint la bordure d'un état composite orthogonal est équivalente à une transition qui atteint les états initiaux de toutes ses régions concurrentes.
Toutes les régions concurrentes d'un état composite orthogonal doivent atteindre leur état final pour que l'état composite soit considéré comme terminé.



Schématisation de la vie d'une instance

Ce diagramme représente les états pris par un objet au cours de sa vie.

Une fois le joueur créé, le choix du personnage va permettre de valoriser les propriétés concernées. Le joueur entre dans l'arène et peut ainsi accéder aux méthodes qui vont lui permettre de se déplacer et/ou d'attaquer. Le joueur peut aussi être touché ce qui va diminuer sa vie.
Si la vie arrive à 0, le joueur meurt. L'objet est détruit.

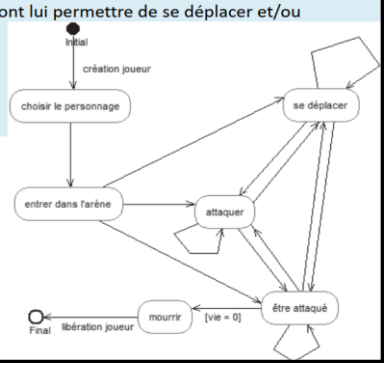


Diagramme d'états

Le diagramme d'états est :
→ un schéma qui ne concerne qu'un objet précis (une instance)
→ une représentation des différents états possibles de cet objet
→ une intégration de la notion de temps (étapes de vie d'un objet)

Point de jonction

Les points de jonction sont un artefact graphique (un pseudoétat en l'occurrence) qui permet de partager des segments de transition, l'objectif étant d'aboutir à une notation plus compacte ou plus lisible des chemins alternatifs.
Un point de jonction peut avoir plusieurs segments de transition entrante et plusieurs segments de transition sortante. Par contre, il ne peut avoir d'activité interne ni des transitions sortantes dotées de déclencheurs d'événements.
Il ne s'agit pas d'un état qui peut être actif au cours d'un laps de temps fini.
Lorsqu'un chemin passant par un point de jonction est emprunté (donc lorsque la transition associée est déclenchée) toutes les gardes le long de ce chemin doivent s'évaluer à vrai dès le franchissement du premier segment.

Exercice 6

- Représenter par un diagramme d'états les états que peut prendre un individu du point de vue de l'INSEE :
  - vivant,
  - décédé,
  - mineur,
  - majeur,
  - célibataire,
  - marié,
  - veuf,
  - divorcé.

Les diagrammes un par un

Université d'Évry-Val d'Essonne

université PARIS-SACLAY

VII

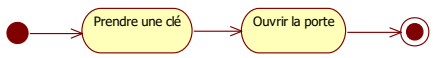
Le diagramme d'activités

Diagramme d'activités

- Permet la modélisation dynamique d'un système.
- Mettre l'accent sur les enchaînements et les conditions pour exécuter et coordonner des actions.
- Utilisation :
  - Décrire un processus métier,
  - Décrire un cas d'utilisation,
  - Décrire un algorithme ou une méthode.

Éléments de base du diagramme

- Une **action** modélise une étape dans l'exécution d'un flot (algorithme ou processus).
- Les actions sont reliées par des **transitions**, en général automatiques.
- Exemple :



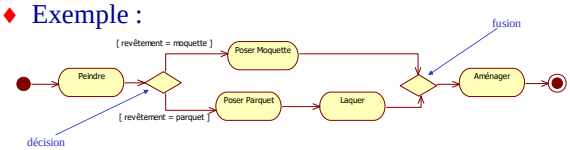
- Noter la similitude avec le diagramme d'états !

Décisions et fusions

- Une **décision** modélise un choix entre plusieurs flots.
- Une **fusion** rassemble plusieurs flots alternatifs.

Équivalent d'un **OU** dans un texte.

- Notation : ◇
- Exemple :

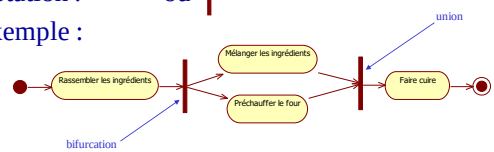


Bifurcation et union

- Une **bifurcation** (ou débranchement) modélise une séparation d'un flot en plusieurs flots concurrents.
- Une **union** (ou jointure) synchronise des flots multiples.

Équivalent d'un **ET** dans un texte.

- Notation : ——— ou |
- Exemple :

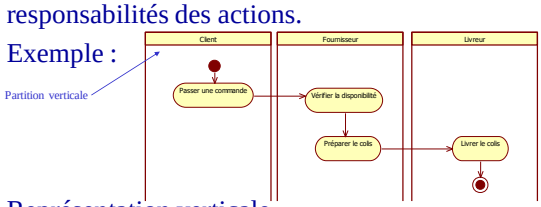


Objets

- Il est possible de faire apparaître des **objets** dans le diagramme.
- L'**état** de l'objet peut être indiqué si l'objet change d'état durant l'exécution du diagramme.
- Permet de montrer la circulation et l'utilisation d'objets au sein de l'activité.
- Exemple :

Partitions

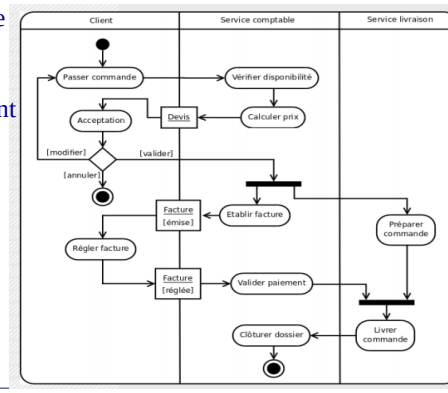
- Les **partitions** (ou couloirs) sont utilisées pour structurer le diagramme en opérant des regroupements.
- Souvent utilisées pour représenter les acteurs ou les responsabilités des actions.
- Exemple :



- Représentation verticale

Partitions

Ce diagramme représente les règles d'enchaînement des activités par rapport aux différents acteurs :



Partitions

- Graphiquement, les partitions sont représentées par des lignes continues. Elles peuvent prendre la forme d'un tableau. De plus, les nœuds d'activités doivent appartenir à une seule et unique partition et les transitions peuvent passer à travers les frontières des partitions.

Exemple de diagramme d'activité GAB : retrait et dépôt d'argent

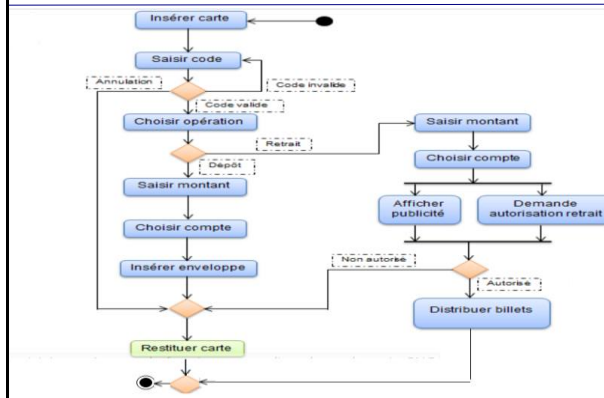


Diagramme d'activités GAB : retrait d'argent

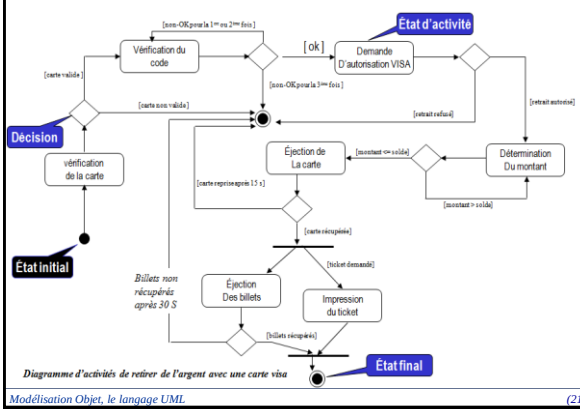


Diagramme d'activité illustre ces différents nœuds de contrôle. Il décrit le fonctionnement d'une prise de commande

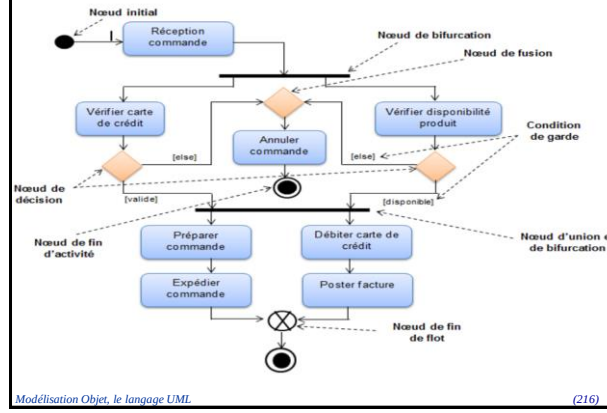


Diagramme d'activité créer un scrutin par un acteur organisateur

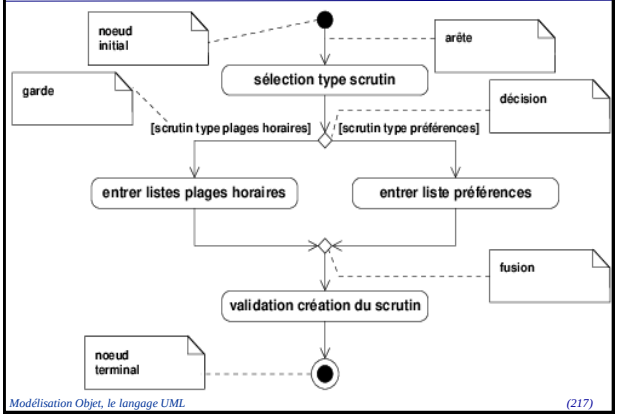
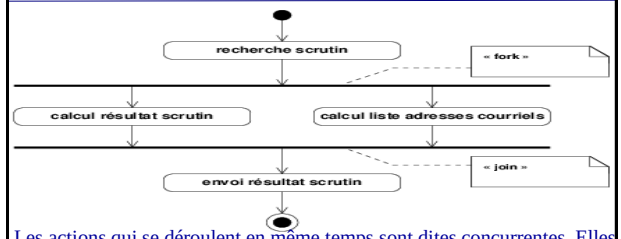


Diagramme d'activité envoyer résultat de scrutin



Les actions qui se déroulent en même temps sont dites concurrentes. Elles sont modélisées entre deux traits épais comme des chemins parallèles, le premier de ces traits étant appelé « fork » et le second « join ». Les séquences d'actions en parallèle débutent en même temps, sont toutes exécutées, mais par définition ne finissent pas au même instant. La fin de l'action join intervient lorsque toutes les actions en parallèle se terminent; join est donc un point de synchronisation.

Nœud initial / Nœud final

- **Nœud initial**

C'est un nœud de contrôle à partir duquel le flot débute lorsque l'activité enveloppante est invoquée. Une activité peut avoir plusieurs nœuds initiaux et un nœud a un arc sortant et pas d'arc entrant. Il est représenté par un petit cercle plein

- **Nœud final**

Un nœud final est un nœud de contrôle possédant un ou plusieurs arcs entrants et aucun arc sortant. Il y a deux type de nœuds finaux: nœud de fin d'activité et nœud de fin de flot. Lorsque l'un des arcs d'un nœud de fin d'activité est activé, l'exécution de l'activité enveloppante s'arrête et tous les nœuds ou flots actifs appartenant de cette activité est abandonné. Si l'activité a été appelé par un appel synchrone, un message *Reply* qui contient les valeurs sortantes est transféré en retour à l'appelant. Un nœud de fin d'activité est représenté par un cercle contenant un petit cercle plein,

Un flot est terminé lorsqu'un flot d'exécution atteint un nœud de fin de flot. Mais cette fin de flot n'a pas d'incidence sur les autres flots actifs de l'activité enveloppante. Un nœud de fin de flot est représenté par un cercle vide barré d'une croix.

Nœud d'union

C'est un nœud de contrôle qui synchronise des flots multiples. Il possède plusieurs arcs entrants et un seul arc sortant. Lorsque tous les arcs entrants sont activés, l'arc sortant l'est aussi également. Un nœud d'union est aussi représenté par un trait plein épais. (cf. au diagramme suivant. Enfin il est possible de fusionner un nœud de bifurcation et un nœud d'union, possédant donc plusieurs arcs entrants et sortants.

Nœud de bifurcation

Un nœud de bifurcation est un nœud de contrôle qui sépare un flot ou plusieurs flots concurrents. Il possède donc un arc entrant ou plusieurs arcs sortants. Il est souvent accordé avec un nœud d'union pour équilibrer la concurrence

Nœud de fusion

Un nœud de fusion est un nœud de contrôle rassemblant plusieurs flots alternatifs entrants en un seul flot sortant. On ne l'utilise pas pour synchroniser des flots concurrents mais pour accepter un flot parmi plusieurs. Un nœud de fusion est représenté par un losange comme le nœud de décision.

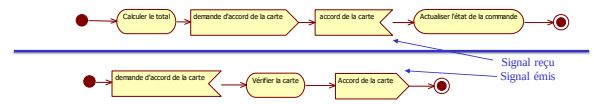
Graphiquement, il est possible de fusionner un nœud de fusion et un nœud de décision, c'est-à-dire de posséder plusieurs arcs entrants et sortants. Il est aussi possible de fusionner un nœud de décision ou de fusion avec un autre nœud. Mais pour mieux mettre en évidence un branchement conditionnel, il est préférable d'utiliser un nœud de décision.

Nœud de décision

C'est un nœud de contrôle qui permet de faire un choix entre plusieurs flots sortants. Il possède un arc entrant et plusieurs arcs sortants, accompagnés de conditions de garde pour conditionner le choix. Quand le nœud de décision est atteint et qu'aucun arc en aval n'est franchissable (ce qui veut dire qu'aucune condition est vraie), cela signifie que le modèle est mal formé. C'est pour cela que l'utilisation d'une garde (else) est recommandée après un nœud de décision, car elle garantit un modèle bien formé. En effet, la condition de garde est validée si et seulement si toutes les autres gardes des transitions ayant la même source sont fausses. Lorsque plusieurs arcs sont franchissables (c'est-à-dire que plusieurs conditions de garde sont vraies), l'un d'entre eux est retenu et ce choix est non déterministe. Un nœud de décision est représenté par un losange.

Signaux

- Les **signaux** permettent de représenter des interactions avec des participants externes (acteurs, systèmes, autres activités,…).
- Un **signal reçu** déclenche une action du diagramme. Un **signal émis** est un signal envoyé à un participant externe.
- Exemple : interactions entre 2 diagrammes d'activités:



Utilisation du diagramme d'activité

- Le diagramme d'activité est plus facile à lire et à comprendre que les autres diagrammes. Il permet donc de communiquer avec des acteurs « non techniques ».
- Il s'utilise à différents niveaux :
  - [1] Modélisation d'un processus métier (BPM),
  - [2] Modélisation d'un cas d'utilisation,
  - [3] Modélisation du comportement interne d'une méthode (algorithme).

Exercice 7

- Réaliser un diagramme d'activités pour décrire le passage au guichet d'enregistrement d'un vol dans un aéroport.

1. Lorsqu'un passager se présente au guichet d'enregistrement, l'hôtesse lui demande son billet et une pièce d'identité (PI).
2. Elle vérifie alors que le billet correspond bien au vol en cours d'enregistrement, et la correspondance entre le nom écrit sur le billet et le nom écrit sur la PI.
3. En cas de problème, le passager peut être réorienté vers l'agence de voyage qui se trouve dans l'aéroport.
4. Si tout est correct, l'hôtesse demande les préférences au passager (placement dans l'avion, …).
5. Puis, si le passager est muni de bagages, elle prend les bagages et édite un reçu à destination du passager.
6. En parallèle, la carte d'enregistrement est imprimée.
7. Enfin la carte d'enregistrement est remise au passager, ainsi qu'une documentation sur la compagnie aérienne.

Les diagrammes un par un

Université d'Évry-Val d'Essonne

université PARIS-SACLAY

VIII
Le diagramme de composants

Diagrammes de la vue développement

- Composant = partie de logiciel réutilisable et remplaçable
- Paquetage = espace de nommage d'éléments de modélisation

Lors du développement d'un système logiciel, il est rare de passer directement des exigences aux classes logicielles codées. Il est nécessaire de regrouper le développement de plusieurs classes et d'organiser le développement par groupes de classes. Les composants et les paquetages sont deux concepts de regroupement. Les composants sont des parties de logiciel réutilisables et remplaçables, selon l'approche COTS (en anglais, Components Off The Shelf, pour composants sur étagère). Les paquetages organisent des éléments, pas uniquement des classes, dans des espaces de nommages différents et permettent de gérer la complexité d'un logiciel de grande taille.

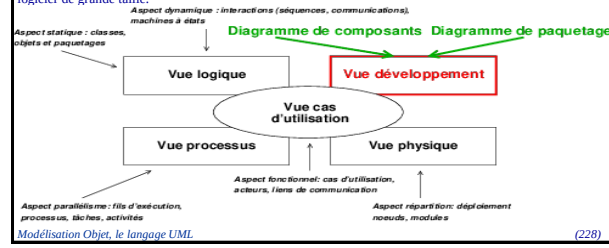


Diagramme de composants

- Le diagramme de composants présente l'**architecture applicative** statique du système, alors que le diagramme de classes présente la structure logique du système.
- Il permet de montrer les **composants** du système et leurs **dépendances** dans leur environnement d'implémentation.
- Les composants permettent d'organiser un système en « morceaux » logiciels.
- Ce diagramme représente l'organisation des ressources **au niveau logique.** C'est la représentation des unités (bases de données, fichiers, librairies...) dont l'assemblage compose l'application

Composant

- Un composant est un élément encapsulé, réutilisable et remplaçable du logiciel.
- Ce sont des « briques de construction », qui permettent en les combinant de réaliser un système.
- Notation : 

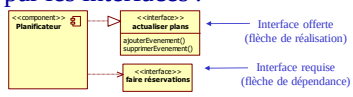
ou 

- Exemples : enregistreur d'évènements, éditeur PDF, convertisseur de format, …

Interfaces de composants

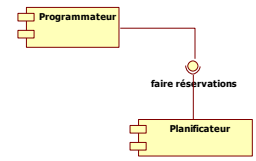
- Les interfaces d'un composant définissent les comportements offerts à d'autres composants (interfaces offertes) ou attendus d'autres composants (interfaces requises).
- Notation : 

- Autre notation possible, avec affichage des opérations offertes par les interfaces :



Dépendances entre composants

- Les dépendances entre composants peuvent être matérialisées par les interfaces :



- Ou bien par le lien de dépendance standard UML :

Conception d'un composant

- Un composant doit fournir des **services** bien précis, qui doivent être cohérents et génériques pour être réutilisables dans différents contextes.
- Le comportement interne, réalisé par un ensemble de classes, est masqué aux autres composants ; seules les interfaces sont visibles.
- On peut substituer un composant à un autre si les interfaces sont identiques.
- Le « découpage » d'un système en composants se fait en recherchant les éléments qui sont employés fréquemment dans le système.

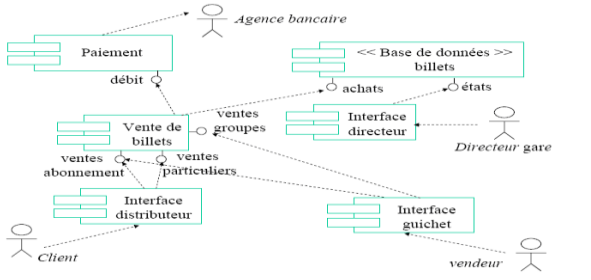
Vues d'un composant

- **Vue boîte noire** : montre l'aspect extérieur d'un composant : interfaces fournies et requises, liens avec d'autres composants.
  → pour se concentrer sur les problèmes d'architecture applicative générale du système.
- **Vue boîte blanche** : montre les classes, les interfaces et les autres composants inclus.
  → Pour montrer comment un composant rend ses services au travers des classes qu'il utilise.

Diagramme de composant : Vente des billets de train

- **But** : modéliser le processus de vente de billets de train.

- Une interface distributeur est mise à la disposition des clients soit pour des ventes particuliers ou pour des ventes abonnement.
- Une interface guichet permet au vendeur d'effectuer des ventes groupes ou des ventes abonnement.
- Le paiement des billets s'effectue par carte bancaire.
- Le directeur de la gare peut consulter les états des billets en accèdent à la base de données via une interface.



Les diagrammes un par un

Université d'Évry-Val d'Essonne

université PARIS-SACLAY

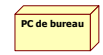
IX

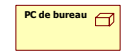
Le diagramme de déploiement

Diagramme de déploiement

- Le diagramme de déploiement montre la **configuration physique** des différents matériels qui participent à l'exécution du système, et montre la répartition des composants sur ces matériels.
- Utile :
  - dans les premières phases du projet pour montrer une esquisse grossière du système,
  - à la fin du développement pour servir de base au guide d'installation par exemple.

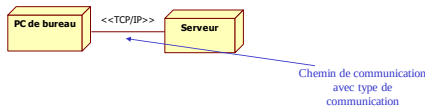
Noeud

- Un **nœud** est une ressource matérielle ou logicielle qui peut héberger des logiciels ou des fichiers associés : hôte, disque, etc.
- Notation : 

ou 

- Exemples :
  - nœuds matériels : serveur, PC de bureau, disque dur
  - nœuds logiciels : système d'exploitation, serveur web, et un SGBDR.

Chemin de communication

- Un nœud peut avoir besoin de communiquer avec d'autres nœuds.
- Les **chemins de communication** sont utilisés pour spécifier que des nœuds communiquent les uns avec les autres à l'exécution.
- Notation : 

Artefact

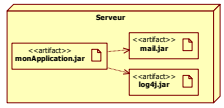
- Un **artéfact** est un fichier physique qui s'exécute ou est utilisé par le système. Les librairies, les fichiers de propriétés et les fichiers exécutables.
- Notation : 

ou 

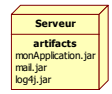
- Exemples :
  - fichiers exécutables (.exe, .jar),
  - fichiers de bibliothèque (.dll),
  - fichiers source (.java),
  - fichiers de configuration (.xml, .properties).

Artefacts, nœuds et composants

- Un **artéfact** est déployé sur un **nœud** : il réside sur (ou est installé sur) le nœud.
- Un artefact peut dépendre d'autres artefacts pour s'exécuter.
- Exemple :



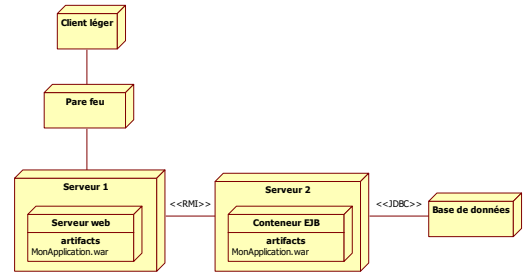
ou



- Un artefact peut être la représentation physique d'un **composant**.
- Exemple :

Exemple

- Diagramme de déploiement d'une architecture J2EE :



Ce diagramme représente l'organisation physique de la distribution des composants logiciels de l'application.

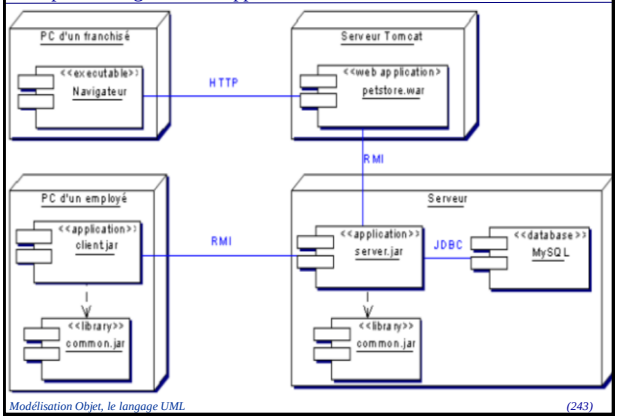


Diagramme de déploiement : Nœud et liaison de communication

Ce modèle définit le diagramme de l'architecture matérielle du système

- Il représente les différentes machines et les logiciels
- Il montre les liens de communication entre ces différentes entités

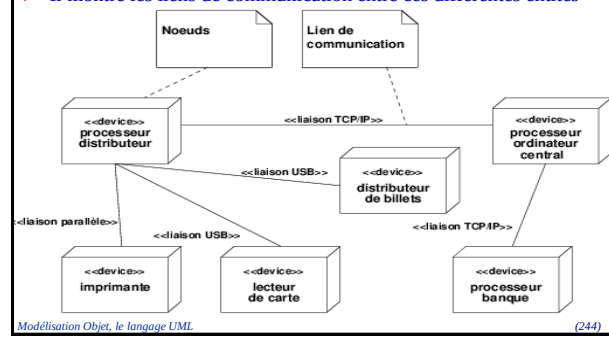


Diagramme de déploiement : Artefact et composant

- La diapositive montre trois façons de représenter un artefact : avec le stéréotype «artefact» et un icone, avec seulement le stéréotype «artefact», et sans stéréotype et sans icone.
- Le déploiement d'un artefact peut être montré de deux manières différentes : soit l'artefact est placé dans le nœud, soit l'artefact et le nœud sont reliés par une association stéréotypée «deploy».
- Enfin, le diagramme de déploiement peut modéliser par une association stéréotypée «manifest» le fait qu'un artefact est « empaqueté » dans un composant lui-même déployé sur

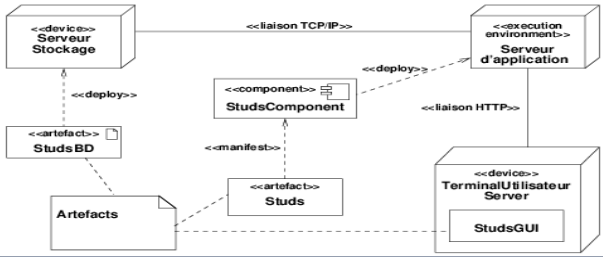
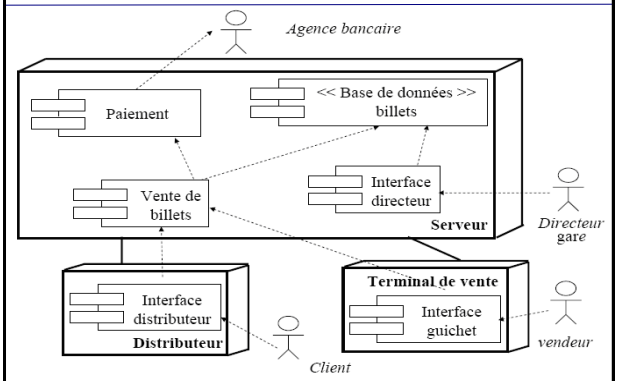


Diagramme de déploiement : vente de billets de train



Les diagrammes un par un

Université d'Évry-Val d'Essonne

université PARIS-SACLAY

X

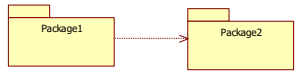
Le diagramme de paquetages

Paquetage

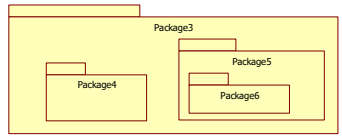
- Problème : un système peut contenir plusieurs centaines de classes ou d'éléments de modélisation. Comment organiser ces classes et ces diagrammes ?
- Solution : rassembler les classes dans des groupes logiques.
- Le **paquetage** est un regroupement d'éléments UML, par exemple un regroupement de classes et d'autres diagrammes.
- Un paquetage peut aussi être un regroupement d'autres éléments comme les cas d'utilisation.

Diagramme de paquetages

- Le diagramme de paquetages permet de représenter les paquetages et leurs dépendances.
- Notation :

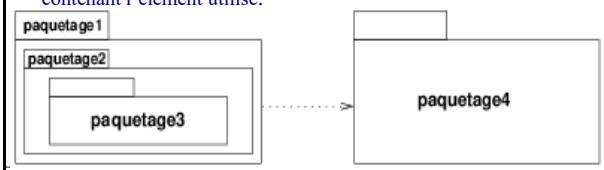


- Il est possible d'imbriquer des paquetages dans d'autres paquetages :



Relation entre paquetages

- Imbrication de paquetages = imbrication d'espaces de nommage
- Dépendance entre paquetages
- Les paquetages peuvent contenir d'autres paquetages pour construire une imbrication d'espaces de nommage. Si un élément d'un paquetage, par exemple paquetage3 dans la diapositive, utilise un élément d'un autre paquetage, par exemple du paquetage4, alors le premier paquetage dépend du second. La dépendance est notée par une flèche pointillée allant du paquetage utilisateur vers le paquetage contenant l'élément utilisé.



Espaces de noms

- Pour qu'un élément utilise un élément d'un autre paquetage, il doit spécifier où il se trouve, en précisant son nom complet, sous la forme :

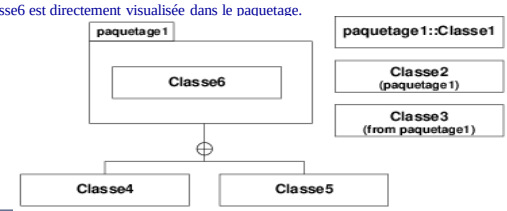
  nomPaquetage::nomClasse

- Le nom complet permet de distinguer 2 classes de même nom mais de paquetages différents.
- Le nom complet permet de rendre unique un élément.

Paquetage, espace de nommage

Un paquetage est représenté par un rectangle avec une cartouche rectangulaire en haut à gauche. Lorsque le diagramme montre des éléments de modélisation (dans la diapositive, une classe) dans e paquetage, le nom du paquetage est indiqué dans la cartouche. Le diagramme de la diapositive montre les différentes notations pour indiquer qu'une classe est contenue dans un paquetage :

- Classe1 possède dans son nom le nom du paquetage, la hiérarchie des paquetages étant indiquée par le symbole « :: » ;
- la notation pour Classe2 possède en dessous de son nom le nom du paquetage entre parenthèse
- la notation pour Classe3 utilise une notation proche de celle utilisée pour Classe2 ;
- les classes Classe4 et Classe5 sont reliées au paquetage ;
- Classe6 est directement visualisée dans le paquetage.



Visibilité d'un élément

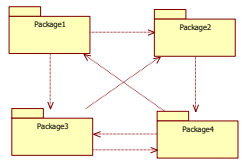
- Les éléments peuvent avoir une visibilité publique ou privée:
- **Visibilité publique** : visible et accessible de l'extérieur du paquetage.
- Notation : +
- **Visibilité privée** : non visible et non disponible de l'extérieur du paquetage.
- Notation : -

Architecture logique d'un système

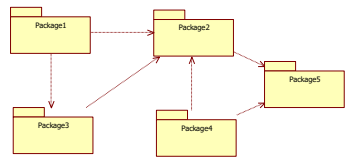
- Les **dépendances** entre paquetages sont issues des dépendances entre éléments des paquetages.
- De trop nombreuses dépendances entre paquetages conduisent à un système fragile et peu évolutif.
- Objectif de l'**architecture** : limiter les dépendances entre paquetages.
- Si une classe A du paquetage 1 dépend d'une classe B du paquetage 2, alors le paquetage 1 dépend du paquetage 2.

Architecture logique d'un système

- Exemples :
  - Mauvais !



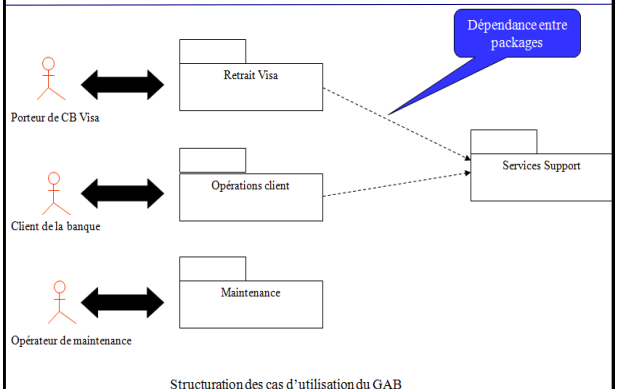
  - Beaucoup mieux !



Les conseils de l'architecte

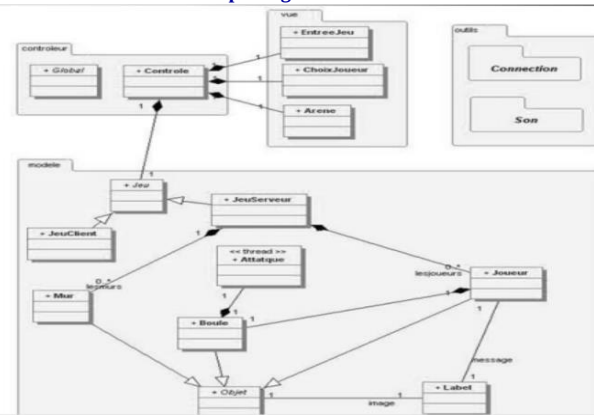
- Faire des regroupements en assurant une cohésion forte à l'intérieur des paquetages et un couplage faible entre les paquetages.
- Éviter les dépendances circulaires.
- Aller dans le sens de la stabilité.

Structuration en Package



Structuration des cas d'utilisation du GAB

Liens entre les différents packages



Conclusion

- Modèle pour appréhender la réalisation d'un système informatique
- Diagrammes UML qui permettent d'aider à la réflexion, à la discussion (clients, architectes, développeurs, etc.)
- Pas de solution unique mais un ensemble de solutions plus ou moins acceptables suivant les contraintes du client, et les logiciels et matériels disponibles
- Une solution acceptable est obtenue par itérations successives

Conception d'une application pour la gestion d'une agence de location de voiture

- **Information utile**: Une bonne solution de développer un site web avec intelligence artificielle ou de booster votre plateforme web existante

Le diagramme de classe est composé de 3 classes :

- **Client** : Comportant comme attributs les informations relatives aux clients avec les méthodes d'accès à ces derniers
- **Voiture** : Comportant comme attributs les informations relatives aux voitures avec les méthodes d'accès à ces derniers.
- **Manager** : Comportant comme attributs les informations relatives aux Managers avec encore une fois les méthodes d'accès à ces derniers
- Il y a aussi une association Location entre Client et Voiture porteuse de la date et la durée de location.

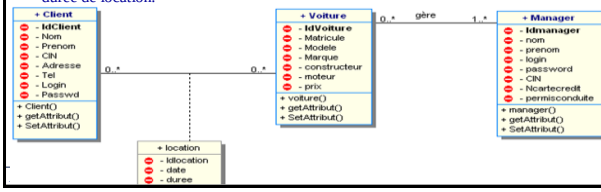


Diagramme de déploiement: Agence de location de voiture

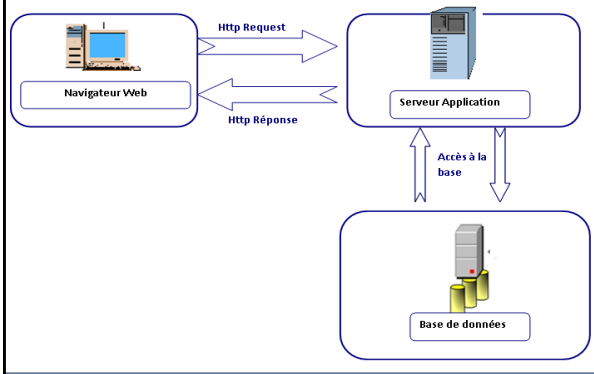


Diagramme de Package : Agence de location de voiture

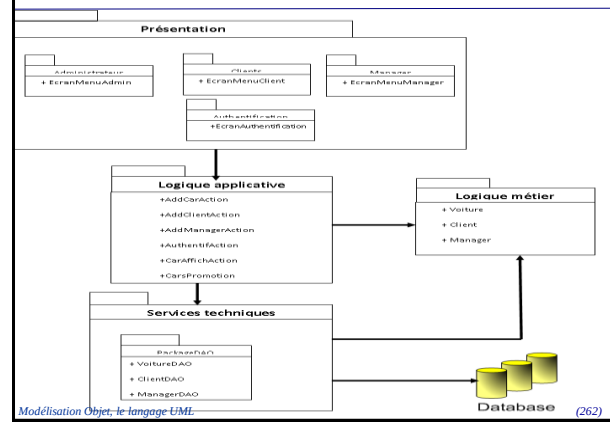


Diagramme de Package : Agence de location de voiture

- Pour garantir à notre application la facilité d'extension et de modification ultérieure on a adopté une architecture organisée en cinq couche (5 tiers):

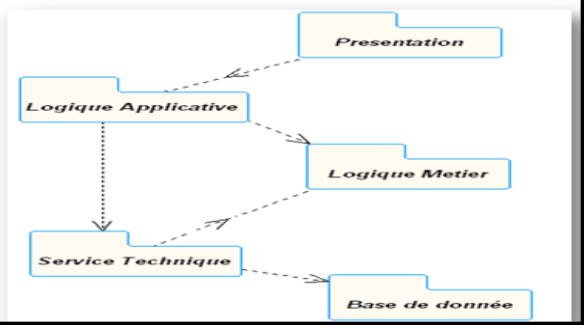


Diagramme de Package : Agence de location de voiture

- **Couche présentation :** elle incorpore toute la logique présentation de l'application : pages HTML et JSP. Elle interagit avec la couche logique applicative.
- **Couche logique applicative :** elle regroupe un ensemble d'objets « contrôleurs » qui sont des objets artificiels entre les objets graphiques et les objets métiers, elle connaît l'interface des objets de la couche métiers et joue le rôle de « façade » vis-à-vis de la couche présentation. cette tache est assurée par des SERVLETS.
- **Couche logique métiers :** elle englobe les objets métiers constituant le système étudié (Voiture, Client, Location, Manager).
- **Couche services techniques :** elle assure l'accès à la base de données. Elle incorpore les classes d'accès aux données (CarDAO, ClientDAO, ManagerDAO, LocationDAO, JDBCDAO).
- **Couche Base données :** Ce niveau abrite les données nécessaires à l'application. Il s'agit de la base de données installée sur le SGBDR (exemple MySQL).
- Cette architecture va permettre de répondre au critère d'évolutivité :

- modifier l'interface de l'application sans devoir modifier les objets métier.
- Les objets graphiques ne connaissent pas la couche logique.
- Changer de mécanisme de stockage sans avoir a retouché l'interface ni les règles métier.
- Modularité, réutilisation et maintenabilité.

Exercice de révision sur les cas d'utilisation

La Direction des Ressources Humaines (DRH) d'une grande entreprise a décidé de mettre en place un système informatique pour gérer les demandes de mobilité de son personnel dans les différentes agences de l'entreprise.

Avec le futur système, les employés pourront renseigner leurs informations personnelles (qualifications, langues pratiquées, diplômes avec date d'obtention et mention), et faire des demandes de mobilité.

Faire une demande de mobilité consiste à indiquer les zones géographiques souhaitées et la période de validité de la demande.

Lorsqu'un employé fait une demande de mobilité, il doit avoir mis à jour ses informations personnelles depuis moins de 3 mois, sinon le système lui demandera de les mettre à jour. D'autre part, si l'employé a déjà une demande de mobilité en cours, alors le système doit l'avertir et lui demander s'il désire annuler ou non sa demande en cours avant de saisir sa nouvelle demande.

Chaque demande de mobilité est traitée par le directeur de l'agence de l'employé, qui valide ou refuse la demande. S'il la valide, la demande est traitée par un employé de la DRH qui recherche et sélectionne une ou plusieurs agences susceptibles d'accueillir l'employé. Chaque directeur d'agence sélectionnée doit alors, dans un délai d'une semaine, étudier le profil de l'employé et accepter ou refuser l'employé pour son agence.

Exercice de révision sur les cas d'utilisation

Lorsque plusieurs agences acceptent un employé, c'est au responsable de la DRH de traiter la demande et sélectionner l'agence dans laquelle l'employé sera affecté.

Un directeur d'agence peut renseigner les profils recherchés pour son agence : indiquer les qualifications recherchées, et pour chacune d'elles le niveau d'expertise (débutant, confirmé, expert).

1. Réaliser un diagramme de cas d'utilisation présentant les acteurs et les cas d'utilisation du système

2. Décrire le cas d'utilisation de la demande de mobilité par l'employé