DI GALLO Frédéric

Méthodologie des systèmes d'information - UML

Cours du Cycle Probatoire

Cours dispensé par Annick Lassus.
CNAM ANGOULEME 2000-2001

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

METHODOLOGIES DES SYSTEMES D'INFORMATION :

UML

UP - LE PROCESSUS UNIFIE

APPROCHE DU LANGAGE UML

INTRODUCTION AU LANGAGE UML

METHODOLOGIE – CNAM ANGOULEME 2000-2001

UP - LE PROCESSUS UNIFIE

Comparaison des méthodologies UP et Merise:

UP	MERISE
Cycle de vie itératif et incrémental Méthode générique	Séquentiel

I. Le processus de développement : nouvelle approche

Dans une démarche traditionnelle, le processus de développement était caractérisé par :

- Un processus de type séquentiel : développement organisé en phases qui regroupent des étapes, elles mêmes décomposées en tâche.
- Les niveaux de découpage coïncident : la fin d'une phase correspond à la conclusion de ses étapes, qui elles mêmes se terminent avec l'accomplissement des tâches qui les composent.

Dans une approche objet tout change :

- Le processus est de type itératif ;
- Les découpages ne coïncident pas : les activités (taches, phases, étapes, etc…) se déroulent dans plusieurs dimensions.

La maîtrise des processus de développement implique pourtant une organisation et un suivi des activités : c'est ce à quoi s'attachent les différentes méthodes qui s'appuient sur l'utilisation du langage UML pour modéliser un système d'information.

UP (Unified Process) est une méthode générique de développement de logiciel. Générique signifie qu'il est nécessaire d'adapter UP au contexte du projet, de l'équipe, du domaine et/ou de l'organisation (*exemple: R.UP ou X.UP*). C'est, entre parenthèses, plus ou moins vrai pour toute méthode, qu'elle se définisse elle-même comme générique ou pas.

Il existe donc un certain nombre de méthodes issues de UP.

II. Le processus unifié : cadre général

Le processus unifié est un processus de développement logiciel : il regroupe les activités à mener pour transformer les besoins d'un utilisateur en système logiciel.

Caractéristiques essentielles du processus unifié :

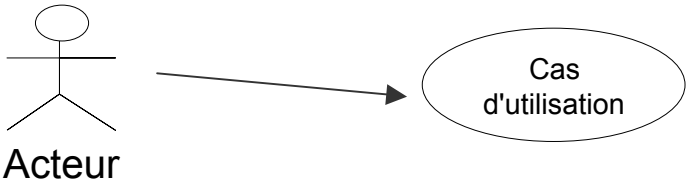
- **Le processus unifié est à base de composants**,
- **Le processus unifié utilise le langage UML** (ensemble d'outils et de diagramme),
- **Le processus unifié est piloté par les cas d'utilisation**,
- **Centré sur l'architecture**,
- **Itératif et incrémental**.

III. Le processus unifié est piloté par les cas d'utilisation

3.1) Présentation

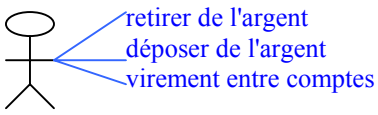
L'objectif principal d'un système logiciel est de rendre service à ses utilisateurs ; il faut par conséquent bien comprendre les désirs et les besoins des futurs utilisateurs. **Le processus de développement sera donc centré sur l'utilisateur**. Le terme utilisateur ne désigne pas seulement les utilisateurs humains mais également les autres systèmes. L'utilisateur représente donc une personne ou une chose dialoguant avec le système en cours de développement.

Ce type d'interaction est appelé cas d'utilisation.

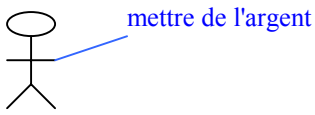


Les cas d'utilisation font apparaître les besoins fonctionnels et leur ensemble constitue le modèle des cas d'utilisation qui décrit les fonctionnalités complètes du système.

3.2) Exemple: guichet de banque



client de la banque



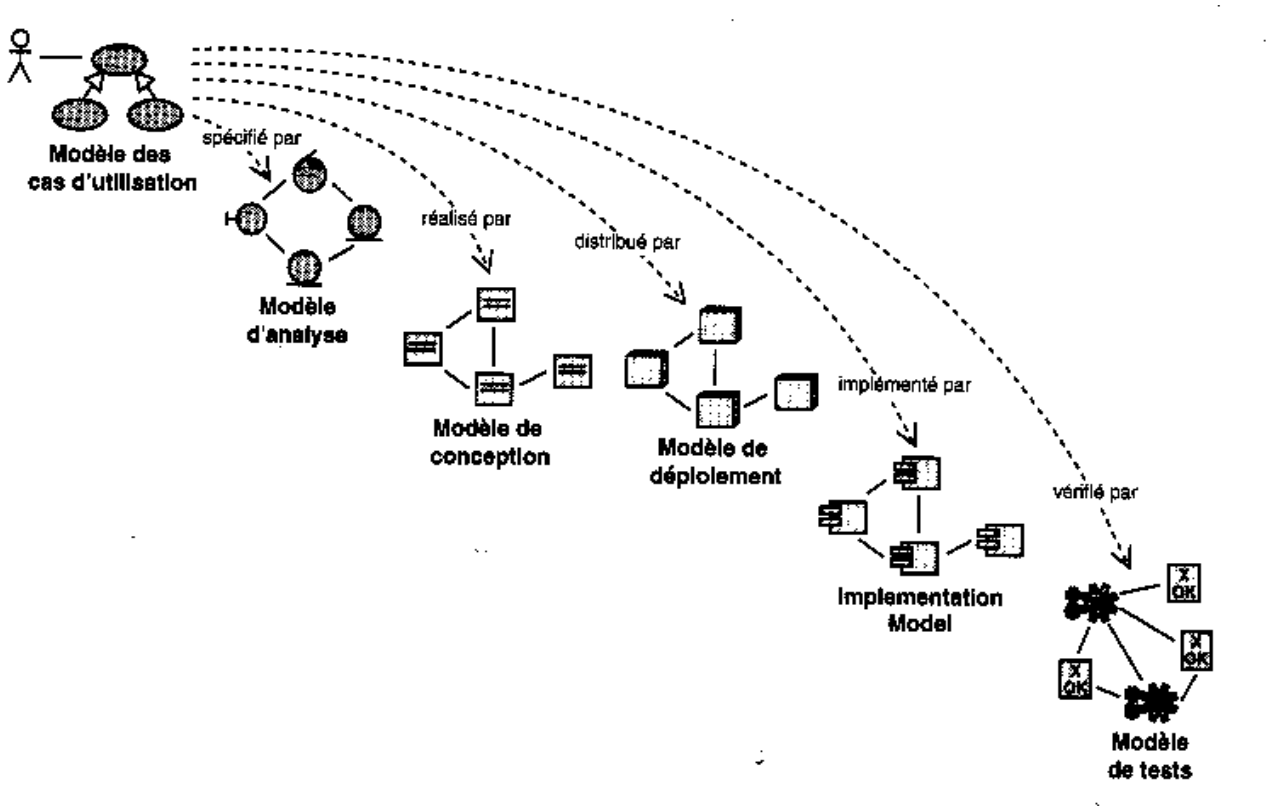
employé de la banque

On va se demander, en premier, quels sont les utilisateurs du système (pas forcément des utilisateurs physiques, mais plutôt des rôles). Ici, l'employé est aussi un client de la banque. On a donc une personne physique pour deux rôles. Nous ne sommes pas dans une approche de type fonctionnelle mais une approche pilotée par des cas d'utilisation.

3.3) Stratégie des cas d'utilisation

Que doit pouvoir faire le système pour chaque utilisateur ?

Les cas d'utilisation ne sont pas un simple outil de spécification des besoins du système. Ils vont complètement guider le processus de développement à travers l'utilisation de modèles basés sur l'utilisation du langage UML



- A partir du modèle des cas d'utilisation, les développeurs créent une série de modèles de conception et d'implémentation réalisant les cas d'utilisation.
- Chacun des modèles successifs est ensuite révisé pour en contrôler la conformité par rapport au modèle des cas d'utilisation.
- Enfin, les testeurs testent l'implémentation pour s'assurer que les composants du modèle d'implémentation mettent correctement en œuvre les cas d'utilisation.

Les cas d'utilisation garantissent la cohérence du processus de développement du système. S'il est vrai que les cas d'utilisation guident le processus de développement, ils ne sont pas sélectionnés de façon isolée, mais doivent absolument être développés "en tandem" avec l'architecture du système.

IV. Le processus unifié est centré sur l'architecture

Dès le démarrage du processus, on aura une vue sur l'architecture à mettre en place. L'architecture d'un système logiciel peut être décrite comme les différentes vues du système qui doit être construit. L'architecture logicielle équivaut aux aspects statiques et dynamiques les plus significatifs du système. L'architecture émerge des besoins de l'entreprise, tels qu'ils sont exprimés par les utilisateurs et autres intervenants et tels qu'ils sont reflétés par les cas d'utilisation.

Elle subit également l'influence d'autres facteurs :

- la plate-forme sur laquelle devra s'exécuter le système ;
- les briques de bases réutilisables disponibles pour le développement ;
- les considérations de déploiement, les systèmes existants et les besoins non fonctionnels (performance, fiabilité..)

4.1) Liens entre cas d'utilisation et architecture ?

Tout produit est à la fois forme et fonction. Les cas d'utilisation doivent une fois réalisés, trouver leur place dans l'architecture. L'architecture doit prévoir la réalisation de tous les cas d'utilisation. L'architecture et les cas d'utilisation doivent évoluer de façon concomitante.

4.2) Marche à suivre :

- L'architecte crée une ébauche grossière de l'architecture, en partant de l'aspect qui n'est pas propre aux cas d'utilisation (plate forme..). Bien que cette partie de l'architecture soit indépendante des cas d'utilisation. L'architecte doit avoir une compréhension globale de ceux ci avant d'en esquisser l'architecture.

- Il travaille ensuite, sur un sous ensemble des cas d'utilisations identifiés, ceux qui représentent les fonctions essentielles du système en cours de développement.

- L'architecture se dévoile peu à peu, au rythme de la spécification et de la maturation des cas d'utilisation, qui favorisent, à leur tour, le développement d'un nombre croissant de cas d'utilisation.

Ce processus se poursuit jusqu'à ce que l'architecture soit jugée stable.

V. Le processus unifié est itératif et incrémental

Le développement d'un produit logiciel destiné à la commercialisation est une vaste entreprise qui peut s'étendre sur plusieurs mois. On ne va pas tout développer d'un coup. On peut découper le travail en plusieurs parties qui sont autant de mini projets. Chacun d'entre eux représentant une itération qui donne lieu à un incrément.

Une itération désigne la succession des étapes de l'enchaînement d'activités, tandis qu'un incrément correspond à une avancée dans les différents stades de développement.

Le choix de ce qui doit être implémenté au cours d'une itération repose sur deux facteurs :

- Une itération prend en compte un certain nombre de cas d'utilisation qui ensemble, améliorent l'utilisabilité du produit à un certain stade de développement.
- L'itération traite en priorité les risques majeurs.

Un incrément constitue souvent un additif.

A chaque itération, les développeurs identifient et spécifient les cas d'utilisations pertinents, créent une conception en se laissant guider par l'architecture choisie, implémentent cette conception sous forme de composants et vérifie que ceux ci sont conformes aux cas d'utilisation. Dés qu'une itération répond aux objectifs fixés le développement passe à l'itération suivante.

Pour rentabiliser le développement il faut sélectionner les itérations nécessaires pour atteindre les objectifs du projet. Ces itérations devront se succéder dans un ordre logique.

Un projet réussi suivra un déroulement direct, établi dés le début par les développeurs et dont ils ne s'éloigneront que de façon très marginale. L'élimination des problèmes imprévus fait partie des objectifs de réduction des risques.

5.1) Avantages d'un processus itératif contrôlé

- Permet de limiter les coûts, en termes de risques, aux strictes dépenses liées à une itération.

- Permet de limiter les risques de retard de mise sur le marché du produit développé (identification des problèmes dès les premiers stades de développement et non en phase de test comme avec l'approche « classique »).

- Permet d'accélérer le rythme de développement grâce à des objectifs clairs et à court terme.

- Permet de prendre en compte le fait que les besoins des utilisateurs et les exigences correspondantes ne peuvent être intégralement définis à l'avance et se dégagent peu à peu des itérations successives

L'architecture fournit la structure qui servira de cadre au travail effectué au cours des itérations, tandis que les cas d'utilisation définissent les objectifs et orientent le travail de chaque itération. Il ne faut donc pas mésestimer l'un des trois concepts.

VI. Le cycle de vie du processus unifié

Le processus unifié répète un certain nombre de fois une série de cycles.

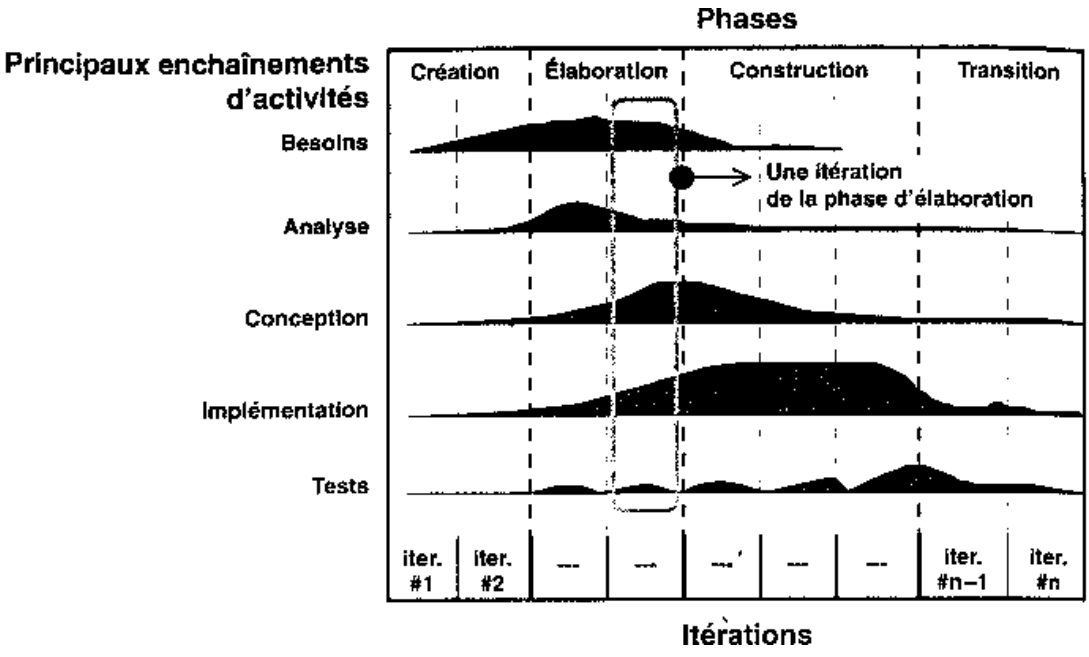
Tout cycle se conclut par la livraison d'une version du produit aux clients et s'articule en 4 phases : création, élaboration, construction et transition, chacune d'entre elles se subdivisant à son tour en itérations.

Chaque cycle se traduit par une nouvelle version du système. Ce produit se compose d'un corps de code source réparti sur plusieurs composants pouvant être compilés et exécutés et s'accompagne de manuels et de produits associés. Pour mener efficacement le cycle, les développeurs ont besoin de construire toutes les représentations du produit logiciel :

Modèle des cas d'utilisation	Expose les cas d'utilisation et leurs relations avec les utilisateurs
Modèle d'analyse	Détaille les cas d'utilisation et procède à une première répartition du comportement du système entre divers objets
Modèle de conception	Définit la structure statique du système sous forme de sous système, classes et interfaces ; Définit les cas d'utilisation réalisés sous forme de collaborations entre les sous systèmes les classes et les interfaces
Modèle d'implémentation	Intègre les composants (code source) et la correspondance entre les classes et les composants
Modèle de déploiement	Définit les nœuds physiques des ordinateurs et l'affectation de ces composants sur ces nœuds.
Modèle de test	Décrit les cas de test vérifiant les cas d'utilisation
Représentation de l'architecture	Description de l'architecture

Tous ces modèles sont liés. Ensemble, ils représentent le système comme un tout. Les éléments de chacun des modèles présentent des dépendances de traçabilité ; ce qui facilite la compréhension et les modifications ultérieures.

6.1) Présentation du cycle de vie de UP

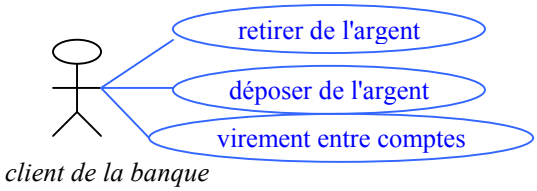


Phase	Description et Enchaînement d'activités
Phase de création	Traduit une idée en vision de produit fini et présente une étude de rentabilité pour ce produit - Que va faire le système pour les utilisateurs ? - A quoi peut ressembler l'architecture d'un tel système ? - Quels sont l'organisation et les coûts du développement de ce produit ? On fait apparaître les principaux cas d'utilisation. L'architecture est provisoire, identification des risques majeurs et planification de la phase d'élaboration.
Phase d'élaboration	Permet de préciser la plupart des cas d'utilisation et de concevoir l'architecture du système. L'architecture doit être exprimée sous forme de vue de chacun des modèles. Emergence d'une **architecture de référence**. A l'issue de cette phase, le chef de projet doit être en mesure de prévoir les activités et d'estimer les ressources nécessaires à l'achèvement du projet.
Phase de construction	Moment où l'on construit le produit. L'architecture de référence se métamorphose en produit complet, elle est maintenant stable. Le produit contient tous les cas d'utilisation que les chefs de projet, en accord avec les utilisateurs ont décidé de mettre au point pour cette version. Celle ci doit encore avoir des anomalies qui peuvent être en partie résolue lors de la phase de transition.
Phase de transition	Le produit est en version bêta. Un groupe d'utilisateurs essaye le produit et détecte les anomalies et défauts. Cette phase suppose des activités comme la fabrication, la formation des utilisateurs clients, la mise en œuvre d'un service d'assistance et la correction des anomalies constatées.(où le report de leur correction à la version suivante)

6.2) Exemple sur les différents modèles

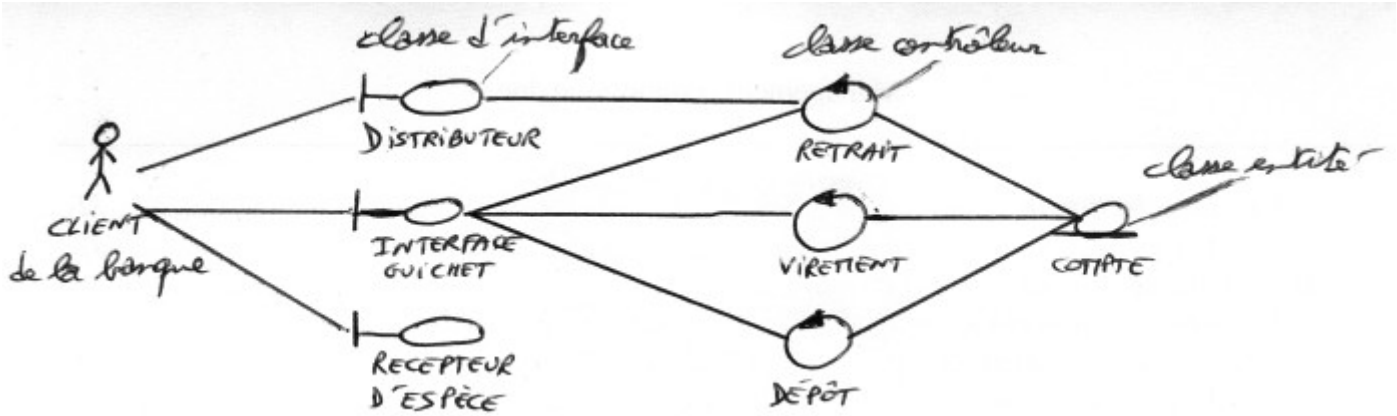
Cas du guichet de banque…

a) Diagramme de cas d'utilisation:

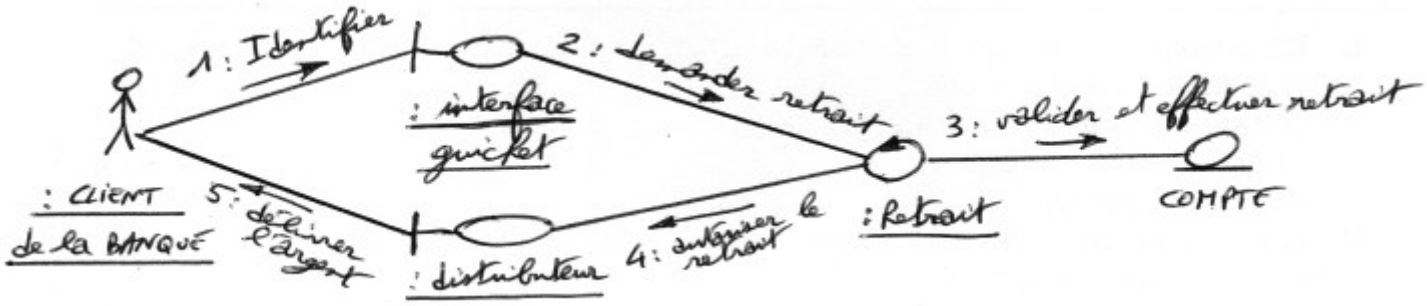


On va essayer de faire apparaître des cas nominaux (classiques) et des scénarios d'exception.

b) Modèle d'analyse: réalisation d'un cas d'utilisation



c) Diagramme de collaboration



VII. Conclusion : un processus intégré

Le processus unifié est basé sur des composants. Il utilise UML et est basé sur les cas d'utilisation, l'architecture et le développement incrémental. Pour mettre en pratique ces idées il faut recourir à un processus multi-facettes prenant en considération les cycles, les phases, les enchaînements d'activités, la réduction des risques, le contrôle qualité, la gestion de projet et la gestion de configuration. Le processus unifié a mis en place un cadre général (framework) intégrant chacune de ces facettes.

Approche du langage UML

APPROCHE DU LANGAGE UML

APPROCHE DU LANGAGE UML

I. Les méthodes objet et la genèse d'UML

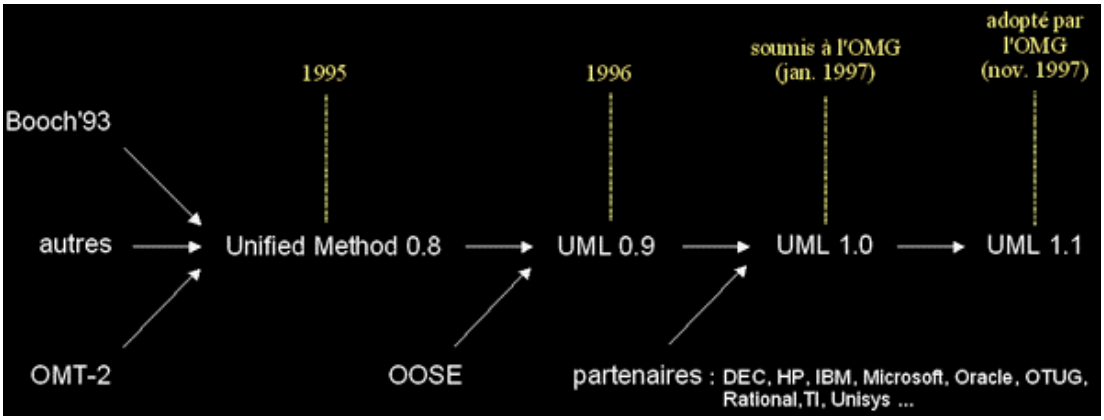
1.1) Méthodes ?

- **Les premières méthodes d'analyse (années 70)**
  Découpe cartésienne (fonctionnelle et hiérarchique) d'un système.

- **L'approche systémique (années 80)**
  Modélisation des données + modélisation des traitements (Merise, Axial, IE...).

- **L'émergence des méthodes objet (1990-1995)**
  Prise de conscience de l'importance d'une méthode spécifiquement objet: comment structurer un système sans centrer l'analyse uniquement sur les données ou uniquement sur les traitements (mais sur les deux) ? Plus de 50 méthodes objet sont apparues durant cette période (Booch, Classe-Relation, Fusion, HOOD, OMT, OOA, OOD, OOM, OOSE...)! Aucune méthode ne s'est réellement imposée.

- **Les premiers consensus (1995)**
  - **OMT** (James Rumbaugh) : vues statiques, dynamiques et fonctionnelles d'un système. Issue du centre de R&D de General Electric. Notation graphique riche et lisible.
  - **OOD** (Grady Booch) : vues logiques et physiques du système. Définie pour le DOD, afin de rationaliser de développement d'applications ADA, puis C++. Ne couvre pas la phase d'analyse dans ses 1ères versions (préconise SADT). Introduit le concept de package (élément d'organisation des modèles).
  - **OOSE** (Ivar Jacobson) : couvre tout le cycle de développement. Issue d'un centre de développement d'Ericsson, en Suède. La méthodologie repose sur l'analyse des besoins des utilisateurs.
- **L'unification et la normalisation des méthodes (1995-1997)**
  - **UML** (Unified Modeling Langage), la fusion et synthèse des méthodes dominantes :



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

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

>>> UML n'est pas une mode, c'est un investissement fiable !

1.2) A quoi sert UML ?

- **UML n'est pas une méthode ou un processus !**
  - Si l'on parle de méthode objet pour UML, c'est par abus de langage !
  - Ce constat vaut aussi pour OMT ou d'autres techniques / langages de modélisation.
  - Une méthode propose aussi un processus, qui régit notamment l'enchaînement des activités de production d'une entreprise.
  - UML a été pensé pour permettre de modéliser les activités de l'entreprise, pas pour les régir (ce n'est pas CMM ou SPICE).
  - Un processus de développement logiciel universel est une utopie :
    - Impossible de prendre en compte toutes les organisations et cultures d'entreprises.
    - Un processus est adapté (donc très lié) au domaine d'activité de l'entreprise.
    - Même si un processus constitue un cadre général, il faut l'adapter de manière précise au contexte de l'entreprise.

- **UML est un langage pseudo-formel**
  - UML est fondé sur un métamodèle, qui définit :
    - les éléments de modélisation (les concepts manipulés par le langage),
    - la sémantique de ces éléments (leur définition et le sens de leur utilisation).
  - Un métamodèle est une description très formelle de tous les concepts d'un langage. Il limite les ambiguïtés et encourage la construction d'outils.
  - Le métamodèle d'UML permet de classer les concepts du langage (selon leur niveau d'abstraction ou domaine d'application) et expose sa structure.
  - Le métamodèle UML est lui-même décrit par un méta-métamodèle (OMG-MOF).
  - UML propose aussi une notation, qui permet de représenter graphiquement les éléments de modélisation du métamodèle.
  - Cette notation graphique est le support du langage UML.

- **UML cadre l'analyse objet, en offrant :**
  - différentes vues (perspectives) complémentaires d'un système, qui guident l'utilisation des concept objets,
  - plusieurs niveaux d'abstraction, qui permettent de mieux contrôler la complexité dans l'expression des solutions objets.

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

1.3) Les points forts d'UML

- **UML est un langage formel et normalisé**
  - gain de précision
  - gage de stabilité
  - encourage l'utilisation d'outils

- **UML est un support de communication performant**
  - Il cadre l'analyse.
  - Il facilite la compréhension de représentations abstraites complexes.
  - Son caractère polyvalent et sa souplesse en font un langage universel.

1.4) Les points faibles d'UML

- **La mise en pratique d'UML nécessite un apprentissage et passe par une période d'adaptation.**

Même si l'Espéranto est une utopie, la nécessité de s'accorder sur des modes d'expression communs est vitale en informatique. UML n 'est pas à l'origine des concepts objets, mais en constitue une étape majeure, car il unifie les différentes approches et en donne une définition plus formelle.

- **Le processus (non couvert par UML) est une autre clé de la réussite d'un projet.**

Or, l'intégration d'UML dans un processus n'est pas triviale et améliorer un processus est un tâche complexe et longue. Les auteurs d'UML sont tout à fait conscients de l'importance du processus, mais l'acceptabilité industrielle de la modélisation objet passe d'abord par la disponibilité d'un langage d'analyse objet performant et standard.

II. Caractéristiques de la méthode UML

2.1) UML est basé sur un méta-modèle

UML est un moyen d'exprimer des modèles objet en faisant abstraction de leur implémentation, c'est-à-dire que le modèle fourni par UML est valable pour n'importe quel langage de programmation. UML est un langage qui s'appuie sur un métamodèle, un modèle de plus haut niveau qui définit les éléments d'UML (les concepts utilisables) et leur sémantique (leur signification et leur mode d'utilisation). Le métamodèle permet de se placer à un niveau d'abstraction supérieur car il est étudié pour être plus générique que le modèle qu'il permet de construire. Le métamodèle d'UML en fait un langage formel possédant les caractéristiques suivantes:

- un langage sans ambiguïtés
- un langage universel pouvant servir de support pour tout langage orienté objet
- un moyen de définir la structure d'un programme
- une représentation visuelle permettant la communication entre acteurs d'un même projet
- une notation graphique simple, compréhensible même par des non informaticiens

Le métamodèle permet de donner des bases solides et rigoureuses à ce langage graphique, dont la représentations graphiques ne sont là que pour véhiculer des concepts de réalisation.

2.2) UML: visualisation complète d'un système

UML offre une manière élégante de représenter le système selon différentes vues complémentaires grâce aux diagrammes.

Lorsqu'une entreprise désire un logiciel, elle le réalise parfois en interne, mais le fait plus généralement réaliser par une société de services. Dans un cas comme dans l'autre il est nécessaire de définir l'ensemble des fonctionnalités que le logiciel doit possèder. Le demandeur du logiciel n'a parfois pas de compétences particulières en informatique et exprime donc ses souhaits sous forme d'un **CdCF** (*Cahier des Charges Fonctionnelles*), c'est-à-dire un document décrivant sous forme textuelle l'ensemble des particularités que le logiciel doit possèder, les conditions qu'il doit remplir (système(s) d'exploitation visé(s)), les écueils à éviter, ainsi que les délais impartis, éventuellement des clauses sur le coût, les langages à utiliser, ...

Le CdCF est ainsi distribué à différentes sociétés de services (dans le cas d'une sous-traitance) sous forme d'un appel d'offre, auquel les sociétés vont répondre par un coût, un délai, ...

Lorsqu'une société obtient le marché et qu'elle décide (si elle a le choix) d'opter pour un langage orienté objet, il lui faut dans un premier temps créer un modèle (c'est là qu'intervient UML) afin:

- de présenter au client la façon de laquelle elle compte développer le logiciel
- d'accorder tous les acteurs du projet (une application de grande envergure est généralement réalisée par modules développés par différentes équipes)

Ainsi, si le modèle ne convient pas au client, il sera "simple" à modifier, contrairement à une application directement implémentée (qui aurait mobilisé beaucoup plus de personnel, pendant une période plus longue), ce qui signifie une perte d'argent beaucoup moins importante pour la société de services, ainsi qu'une meilleure probabilité de rendre dans les temps (on parle généralement de *dead line*) une application conforme aux exigences du client (si l'application se conforme au modèle présenté au client, celui-ci peut difficilement contester la validité du logiciel).

III. Introduction à la notation UML

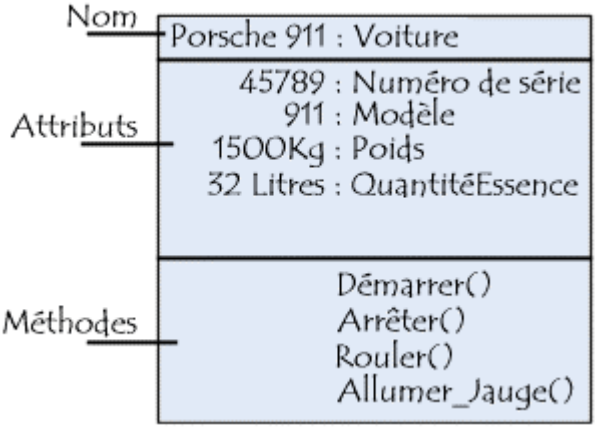
UML (*Unified Modeling Language*, que l'on peut traduire par "*langage de modélisation unifié*) est une notation permettant de modéliser un problème de façon standard. Ce langage est né de la fusion de plusieurs méthodes existant auparavant, et est devenu désormais la référence en terme de modélisation objet, à un tel point que sa connaissance est souvent nécessaire pour obtenir un poste de développeur objet.

3.1) La notion d'objet

La programmation orientée objet consiste à modéliser informatiquement un ensemble d'éléments d'une partie du monde réel (que l'on appelle *domaine*) en un ensemble d'entités informatiques. Ces entités informatiques sont appelées *objet*. Il s'agit de données informatiques regroupant les principales caractéristiques des éléments du monde réel (taille, la couleur, ...). La difficulté de cette modélisation consiste à créer une représentation abstraite, sous forme d'objets, d'entités ayant une existence matérielle (chien, voiture, ampoule, ...) ou bien virtuelle (sécurité sociale, temps, ...).

3.2) Les méthodes objet

La modélisation objet consiste à créer une représentation informatique des éléments du monde réel auxquels on s'intéresse, sans se préoccuper de l'implémentation, ce qui signifie *indépendamment d'un langage de programmation*. Il s'agit donc de déterminer les objets présents et d'isoler leurs données et les fonctions qui les utilisent.



Cette méthode représente un langage permettant de spécifier, représenter et construire les composantes d'un système informatique. Avec la méthode UML, un objet est par exemple représenté de la façon suivante:

3.3) Intérêt d'une méthode objet

Les langages orientés objet constituent chacun une manière spécifique d'implémenter le paradigme objet. Ainsi, une méthode objet permet de définir le problème à haut niveau sans rentrer dans les spécificités d'un langage. Il représente ainsi un outil permettant de définir un problème de façon graphique, afin par exemple de le présenter à tous les acteurs d'un projet (n'étant pas forcément des experts en un langage de programmation).

De plus, le fait de programmer à l'aide d'un langage orienté objet ne fait pas d'un programmer un concepteur objet. En effet il est tout à fait possible de produire un code syntaxiquement juste sans pour autant adopter une approche objet. Ainsi la programmation orientée objet implique en premier lieu une conception abstraite d'un modèle objet (c'est le rôle de la méthode objet), en second plan l'implémentation à l'aide d'un langage orientée objet (tel que C++/Java/...)

Une méthode objet est donc d'une part une méthode d'analyse du problème (afin de couvrir toutes les facettes du problèmes), d'autre part un langage permettant une représentation standard stricte des concepts abstraits (la modélisation) afin de constituer un langage commun.

3.4) La normalisation OMG

Ainsi, il est nécessaire qu'une méthode objet soit définie de manière rigoureuse et unique afin de lever les ambiguïtés. De nombreuses méthodes objet ont été définies, mais aucune n'a su s'imposer en raison du manque de standardisation. C'est pourquoi l'ensemble des acteurs du monde informatique a fondé en 1989 l'**OMG** (*Object Management Group*), une organisation à but non lucratif, dont le but est de mettre au point des standards garantissant la compatibilité entre des applications programmées à l'aide de langages objet et fonctionnant sur des réseaux hétérogènes (de différents types). A partir de 1997, UML est devenue une norme de l'OMG, ce qui lui a permis de s'imposer en tant que méthode de développement objet et être reconnue et utilisée par de nombreuses entreprises.

L'OMG est un organisme à but non lucratif, créé en 1989 à l'initiative de grandes sociétés (HP, Sun, Unisys, American Airlines, Philips...). Aujourd'hui, l'OMG fédère plus de 850 acteurs du monde informatique. Son rôle est de promouvoir des standards qui garantissent l'interopérabilité entre applications orientées objet, développées sur des réseaux hétérogènes.

L'OMG propose notamment l'architecture CORBA (Common Object Request Broker Architecture), un modèle standard pour la construction d'applications à objets distribués (répartis sur un réseau). Pour rester simple, on peut considérer CORBA comme une généralisation de l'architecture clients/serveurs aux objets.

Au centre de l'architecture CORBA, un routeur de messages (ORB : Object Request Broker) permet à des objets clients d'envoyer des requêtes et de recevoir des réponses, sans avoir à se préoccuper des détails techniques propres à l'infrastructure du réseau. L'ORB se charge de transmettre les requêtes aux objets concernés, où qu'ils se trouvent (dans le même processus, dans un autre, sur un autre nœud du réseau...). Le bus CORBA (dont l'ORB est le composant central) permet d'assurer la transparence des invocations de méthodes ; les requêtes aux objets semblent toujours être locales.

De plus, l'ORB assure une coopération entre objets qui est indépendante des langages de programmation. Le modèle CORBA permet en effet de se focaliser sur les interfaces des objets, qu'on exprime en IDL (Interface Definition Language). L'IDL décrit de manière standard l'interface d'un objet, en faisant abstraction des détails qui relèvent de son d'implémentation. Avec CORBA, il n'est donc pas nécessaire de savoir comment les objets sont codés pour utiliser leurs services. L'utilisation d'IDL comme langage pivot dans la construction d'une architecture, permet de gérer l'hétérogénéité. Cette approche, qui consiste à masquer les couches techniques du réseau (système d'exploitation, processeur, langage de programmation...), permet d'assurer l'interopérabilité de toute application à objets distribués, conforme au modèle CORBA.

CORBA fait partie d'une vision globale de la construction d'applications réparties, appelée OMA (Object Management Architecture) et définie par l'OMG. Sans rentrer dans les détails, on peut résumer cette vision par la volonté de **favoriser l'essor industriel des technologies objet**, en offrant un ensemble de **solutions technologiques non propriétaires**, qui suppriment les clivages techniques.

UML a été adopté (normalisé) par l'OMG et intégré à l'OMA, car il participe à cette vision et parce qu'il répond à la "philosophie" OMG.

IV. Modéliser avec UML

4.1) Qu'est-ce qu'un modèle ?

Un modèle est une abstraction de la réalité

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

Un modèle est une vue subjective mais pertinente de la réalité

Un modèle définit une frontière entre la réalité et la perspective de l'observateur. Ce n'est pas "la réalité", mais une vue très subjective de la réalité.

Bien qu'un modèle ne représente pas une réalité absolue, un modèle reflète des aspects importants de la réalité, il en donne donc une vue juste et pertinente.

Quelques exemples de modèles

Modèle météorologique: à partir de données d'observation (satellite ...), permet de prévoir les conditions climatiques pour les jours à venir.

Modèle économique: peut par exemple permettre de simuler l'évolution de cours boursiers en fonction d'hypothèses macro-économiques (évolution du chômage, taux de croissance...).

Modèle démographique: définit la composition d'un panel d'une population et son comportement, dans le but de fiabiliser des études statistiques, d'augmenter l'impact de démarches commerciales, etc...

Caractéristiques fondamentales des modèles

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

4.2) Comment modéliser avec UML ?

UML est un langage qui permet de représenter des modèles, mais il ne définit pas le processus d'élaboration des modèles! Cependant, dans le cadre de la modélisation d'une application informatique, les auteurs d'UML préconisent d'utiliser une démarche :

- itérative et incrémentale,
- guidée par les besoins des utilisateurs du système,
- centrée sur l'architecture logicielle.

D'après les auteurs d'UML, un processus de développement qui possède ces qualités devrait favoriser la réussite d'un projet.

Une démarche itérative et incrémentale ?

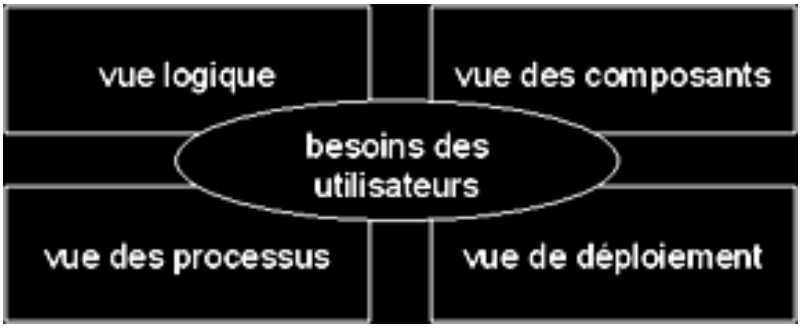
L'idée est simple : pour modéliser (comprendre et représenter) un système complexe, il vaut mieux s'y prendre en plusieurs fois, en affinant son analyse par étapes. Cette démarche devrait aussi s'appliquer au cycle de développement dans son ensemble, en favorisant le prototypage. Le but est de mieux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

Une démarche pilotée par les besoins des utilisateurs ?

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

Une démarche centrée sur l'architecture ?

Une architecture adaptée est la clé de voûte du succès d'un développement. Elle décrit des choix stratégiques qui déterminent en grande partie les qualités du logiciel (adaptabilité, performances, fiabilité...). **Ph. Kruchten propose différentes perspectives, indépendantes et complémentaires, qui permettent de définir un modèle d'architecture (publication IEEE, 1995).** Cette vue ("4+1") a fortement inspiré UML :



Définir une architecture avec UML (détail de la "vue 4+1")

- **La vue logique**

Cette vue de haut niveau se concentre sur l'abstraction et l'encapsulation, elle modélise les éléments et mécanismes principaux du système. Elle identifie les éléments du domaine, ainsi que les relations et interactions entre ces éléments : les éléments du domaine sont liés au(x) métier(s) de l'entreprise, ils sont indispensables à la mission du système, ils gagnent à être réutilisés (ils représentent un savoir-faire). Cette vue organise aussi (selon des critères purement logiques), les éléments du domaine en *"catégories"* : pour répartir les tâches dans les équipes, regrouper ce qui peut être générique, isoler ce qui est propre à une version donnée, etc...

- **La vue des composants**

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

- **La vue des processus**

Cette vue est très importante dans les environnements multitâches ; elle montre : La décomposition du système en terme de processus (tâches); les interactions entre les processus (leur communication); la synchronisation et la communication des activités parallèles (threads).

- **La vue de déploiement**

Cette vue très importante dans les environnements distribués, décrit les ressources matérielles et la répartition du logiciel dans ces ressources : la disposition et nature physique des matériels, ainsi que leurs performances, l'implantation des modules principaux sur les nœuds du réseau, les exigences en terme de performances (temps de réponse, tolérance aux fautes et pannes...).

- **La vue des besoins des utilisateurs**

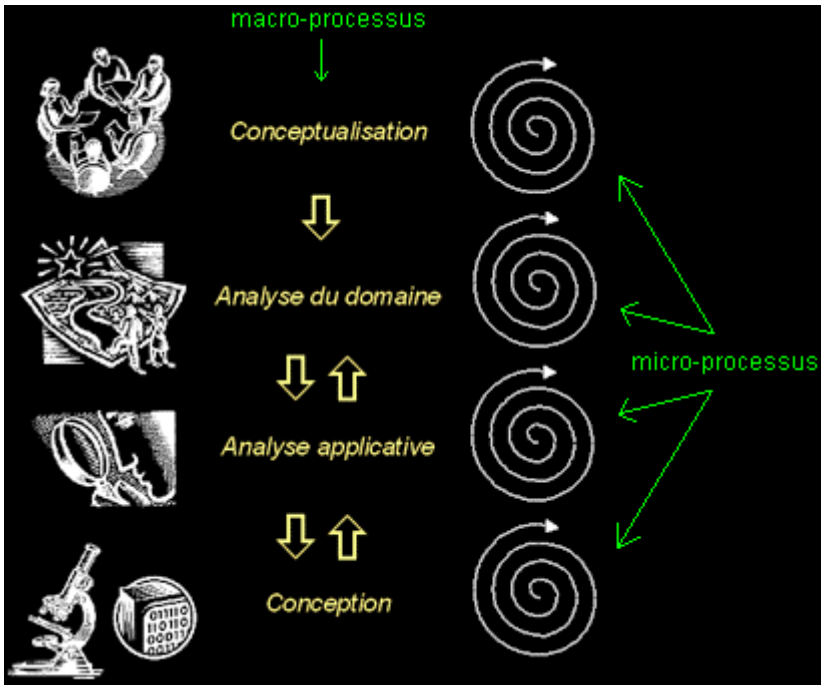
Cette vue (dont le nom exact est "vue des cas d'utilisation"), guide toutes les autres. Dessiner le plan (l'architecture) d'un système informatique n'est pas suffisant, il faut le justifier ! Cette vue définit les besoins des clients du système et centre la définition de l'architecture du système sur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

Résumons la démarche...

Modéliser une application ? Mais comme UML n'est pas un processus... Quelle démarche utiliser ? Trouver un "bon" modèle est une tâche difficile mais capitale !

1. Optez pour une approche itérative et incrémentale.
2. Centrez votre démarche sur l'analyse des besoins des utilisateurs.
3. Prenez grand soin à la définition de l'architecture de votre application (l'approche "4+1" permet de mieux la cerner).

OK OK , mais en pratique ? ien qu'UML n'est pas un processus, il facilite une démarche d'analyse itérative et incrémentale, basée sur les niveaux d'abstraction. Les niveaux d'abstraction permettent de structurer les modèles. Un micro-processus régit les itérations à niveau d'abstraction constant. Un macro-processus régit le passage de niveau à niveau. Une démarche incrémentale consiste à construire les modèles de spécification et de conception en plusieurs étapes (cible = catégories). Le schéma ci-dessous montre les niveaux d'abstraction principaux, qu'on peut identifier dans un processus de développement logiciel :



Elaboration plutôt que transformation

UML opte pour l'élaboration des modèles, plutôt que pour une approche qui impose une barrière stricte entre analyse et conception : Les modèles d'analyse et de conception ne diffèrent que par leur niveau de détail, il n'y a pas de différence dans les concepts utilisés. UML n'introduit pas d'éléments de modélisation propres à une activité (analyse, conception...); le langage reste le même à tous les niveaux d'abstraction. Cette approche simplificatrice facilite le passage entre les niveaux d'abstraction. L'élaboration encourage une approche non linéaire (les "retours en arrière" entre niveaux d'abstraction différents sont facilités). La traçabilité entre modèles de niveaux différents est assurée par l'unicité du langage.

Détail des différents niveaux d'abstraction (phases du macro-processus)

- **Conceptualisation**

L'entrée de l'analyse à ce niveau, est le dossier d'expression des besoins client. A ce niveau d'abstraction, on doit capturer les besoins principaux des utilisateurs. Il ne faut pas chercher l'exhaustivité, mais clarifier, filtrer et organiser les besoins ! Le but de la conceptualisation est de définir le contour du système à modéliser (de spécifier le "quoi"), de capturer les fonctionnalités principales du système, afin d'en fournir une meilleure compréhension (le modèle produit sert d'interface entre les acteurs du projet), de fournir une base à la planification du projet.

- **Analyse du domaine**

L'entrée de l'analyse à ce niveau, est le modèle des besoins clients (les "cas d'utilisation" UML). Il s'agit de modéliser les éléments et mécanismes principaux du système. On identifie les éléments du domaine, ainsi que les relations et interactions entre ces éléments : les éléments du domaine sont liés au(x) métier(s) de l'entreprise, ils sont indispensables à la mission du système, ils gagnent à être réutilisés (ils représentent un savoir-faire). A ce stade, on organise aussi (selon des critères purement logiques), les éléments du domaine en *"catégories"* : pour répartir les tâches dans les équipes, regrouper ce qui peut être générique, etc...

- **Analyse applicative**

A ce niveau, on modélise les aspects informatiques du système, sans pour autant rentrer dans les détails d'implémentation. Les interfaces des éléments de modélisation sont définis (cf. encapsulation). Les relations entre les éléments des modèles sont définies. Les éléments de modélisation utilisés peuvent être propres à une version du système.

- **Conception**

On y modélise tous les rouages d'implémentation et on détaille tous les éléments de modélisation issus des niveaux supérieurs. Les modèles sont optimisés, car destinés à être implémentés.

Activités des micro-processus d'analyse (niveau d'abstraction constant)

A chaque niveau d'abstraction, un micro-processus régit la construction des modèles. UML ne régit pas les activités des micro-processus, c'est le principe d'abstraction qui permet l'élaboration itérative et incrémentale des modèles. Exemple de micro-processus de construction d'un modèle :

- identifiez les classes (d'objets) :

recherchez les classes candidates (différentes suivant le niveau d'abstraction) à l'aide de diagrammes d'objets (ébauches), filtrez les classes redondantes, trop spécifiques ou vagues, qui ne représentent qu'une opération ou un attribut, documentez les caractéristiques des classes retenues (persistances, nombre maximum d'instances, etc.).

- identifiez les associations entre classes / interactions entre objets (instances) :

recherchez les connexions sémantiques et les relations d'utilisation, documentez les relations (nom, cardinalités, contraintes, rôles des classes...), en spécification, filtrez les relations instables ou d'implémentation, définissez la dynamique des relations entre objets (les interactions entre instances de classes et les activités associées).

- identifiez les attributs et les opérations des classes :

recherchez les attributs dans les modèles dynamiques (recherchez les données qui caractérisent les états des objets), filtrez les attributs complexes (faites-en des objets) et au niveau spécification, ne représentez pas les valeurs internes propres aux mécanismes d'implémentation, recherchez les opérations parmi les activités et actions des objets (ne pas rentrer dans le détail au niveau spécification).

- optimisez les modèles :

choisissez vos critères d'optimisation (généricité, évolutivité, précision, lisibilité, simplicité...), utilisez la généralisation et la spécialisation (classification), documentez et détaillez vos modèles, encapsulez.

- validez les modèles :

vérifiez la cohérence, la complétude et l'homogénéité des modèles, confrontez les modèles à la critique (comité de relecture...).

Synthèse de la démarche

Modéliser une application n'est pas une activité linéaire.
Il s'agit d'une tâche très complexe, qui nécessite une approche :

- **itérative et incrémentale** (grâce aux niveaux d'abstraction), car il est plus efficace de construire et valider par étapes, ce qui est difficile à cerner et maîtriser,
- **centrée sur l'architecture** (définie par la vue "4+1"), car il s'agit de la clé de voûte qui conditionne la plupart des qualités d'une application informatique,
- **guidée par la prise en compte des besoins des utilisateurs** (à l'aide des cas d'utilisation), car ce qui motive l'existence même du système à concevoir, c'est la satisfaction des besoins de ses utilisateurs.

4.3) Modélisation UML

Qu'est-ce qu'un modèle ?

La modélisation consiste à créer une représentation simplifiée d'un problème: **le modèle**.

Grâce au modèle il es possible de représenter simplement un problème, un concept et le simuler. La modélisation comporte deux composantes:

- L'analyse, c'est-à-dire l'étude du problème
- la conception, soit la mise au point d'une solution au problème

Le modèle constitue ainsi une représentation possible du système pour un point de vue donné.

La modélisation UML

Le méta-modèle UML fournit une panoplie d'outils permettant de représenter l'ensemble des) éléments du monde objet (classes, objets, ...) ainsi que les liens qui les relie. Toutefois, étant donné qu'une seule représentation est trop subjective, UML fournit un moyen astucieux permettant de représenter diverses projections d'une même représentation grâce aux **vues**. Une vue est constitué d'un ou plusieurs **diagrammes**.

On distingue deux types de vues:

- **Les vues statiques**, c'est-à-dire représentant le système physiquement
  - diagrammes d'objets
  - diagrammes de classes
  - diagrammes de cas d'utilisation
  - diagrammes de composants
  - diagrammes de déploiement
- **Les vues dynamiques**, montrant le fonctionnement du système
  - diagrammes de séquence
  - diagrammes de collaoration
  - diagrammes d'états-transitions
  - diagrammes d'activités
  -

Les cas d'utilisation

Les cas d'utilisation (en anglais *use cases*) permettent de représenter le fonctionnement du système vis-à-vis de l'utilisateur, c'est donc une vue du système dans son environnement extérieur.

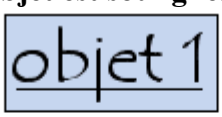
Modélisation des classes et objets

➢ ***Modélisation d'un objet***

La modélisation objet consiste à créer une représentation abstraite, sous forme d'objets, d'entités ayant une existence matérielle (arbre, personne, téléphone, ...) ou bien virtuelle (sécurité sociale, compte bancaire, ...). Un objet est caractérisé par plusieurs notions:

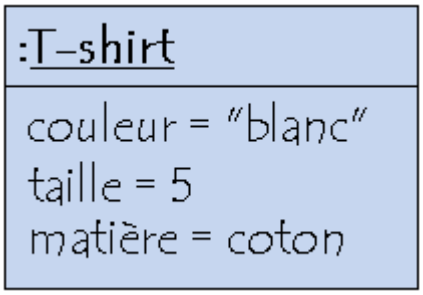
- **Les attributs** (on parle parfois de propriétés): Il s'agit des données caractérisant l'objet. Ce sont des variables stockant des informations d'état de l'objet
- **Les méthodes** (appelées parfois fonctions membres): Les méthodes d'un objet caractérisent son comportement, c'est-à-dire l'ensemble des actions (appelées opérations) que l'objet est à même de réaliser. Ces opérations permettent de faire réagir l'objet aux sollicitations extérieures (ou d'agir sur les autres objets). De plus, les opérations sont étroitement liées aux attributs, car leurs actions peuvent dépendre des valeurs des attributs, ou bien les modifier
- **L'identité**: L'objet possède une identité, qui permet de le distinguer des autres objets, indépendamment de son état. On construit généralement cette identité grâce à un identifiant découlant naturellement du problème (par exemple un produit pourra être repéré par un code, une voiture par un numéro de série, ...)

UML propose une manière de représenter les objets de façon graphique, sous forme de **rectangle, dans lequel le nom de l'objet est souligné**.



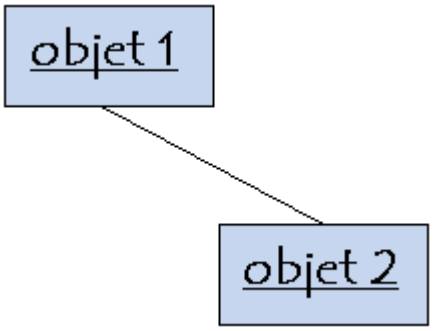
- ***Les attributs d'un objet***

Les attributs (propriétés) d'un objet représentent ses caractéristiques. L'ensemble des valeurs des attributs d'un objet constituent son état. UML propose de représenter les attributs d'un objet et les valeurs associées de la manière suivante:

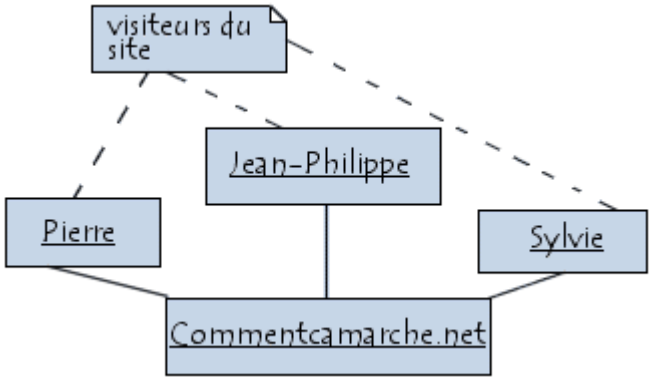


- ***Les liens entre objets***

Dans l'approche objet, les objets ne sont pas des corps inertes isolés. Bien au contraire, même s'ils possèdent leurs caractéristiques propres par l'intermédiaire des valeurs de leurs attributs (ce qui constitue ce que l'on appelle l'état), ils ont la possibilité d'interagir entre-eux grâce à leurs méthodes. UML propose de représenter les liens qui existent entre les objets grâce à un trait continu.



Un lien existe dès lors qu'un objet possède une visibilité vis-à-vis d'un autre, c'est-à-dire lorsqu'un objet a un rapport direct avec un autre (appartenance, utilisation, communication, ...). Il est de plus possible d'ajouter des commentaires sur le modèle sous forme de notes. Une note est représentée par un rectangle dont le coin supérieur droit est replié. Pour relier une note à un objet il faut utiliser un trait discontinu (en pointillés).



Modélisation d'une classe

On appelle classe la structure d'un objet, c'est-à-dire la déclaration de l'ensemble des entités qui composeront un objet. Un objet est donc "issu" d'une classe, c'est le produit qui sort d'un moule. En réalité on dit qu'un objet est une **instanciation** d'une classe, c'est la raison pour laquelle on pourra parler indifféramment d'objet ou d'instance (éventuellement d'occurence).

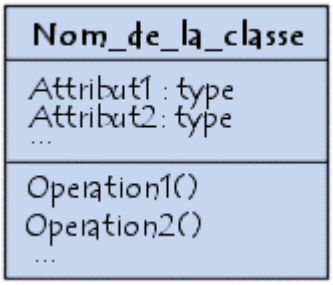
Une classe est composée:

- d'attributs: il s'agit des données, dont les valeurs représentent l'état de l'objet
- Les méthodes : il s'agit des opérations applicables aux objets

Si on définit la classe *voiture*, les objets *Peugeot 406*, *Volkswagen Golf* seront des instanciations de cette classe. Il pourra éventuellement exister plusieurs objets *Peugeot 406*, différenciés par leur numéro de série.

Mieux: deux instanciations de classes pourront avoir tous leurs attributs égaux sans pour autant être un seul et même objet (c'est la différence entre *état* et *identité*). C'est le cas dans le monde réel, deux T-shirts peuvent être strictement identique (avoir le même état) et pourtant ils sont distincts (ils ont chacun leur identité propre). D'ailleurs en les mélangeant il serait impossible de les distinguer...

Une classe se représente avec UML sous forme d'un rectangle divisé en trois sections. Le premier contient le nom donné à la classe (non souligné). Les attributs d'une classe sont définis par un nom, un type (éventuellement une valeur par défaut, c'est-à-dire une valeur affectée à la propriété lors de l'instanciation) dans le second compartiment. Les opérations sont répertoriées dans le troisième volet du rectangle.



La visibilité des attributs

L'un des principaux concepts du paradigme objet est l'encapsulation. L'encapsulation est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure en cachant l'implémentation de l'objet, c'est-à-dire en empêchant l'accès aux données par un autre moyen que les services proposés. L'encapsulation permet donc de garantir l'intégrité des données contenues dans l'objet.

En effet, la programmation orientée objet permet de cacher l'implémentation dun objet en ne lui permettant d'accéder aux attributs uniquement par l'intermédiaire de méthodes crées à cet effet (on parle d'interface). Il est ainsi possible de définir des niveaux d'encapsulation, afin de définir le type de classe ayant accès aux attributs.

On parle de niveau de visibilité des éléments de la classe. Ces niveaux de visibilité définissent les droits d'accès aux données selon que l'on y accède par une méthode de la classe elle-même, d'une classe héritière, ou bien d'une classe quelconque. Il existe trois niveaux de visibilité.

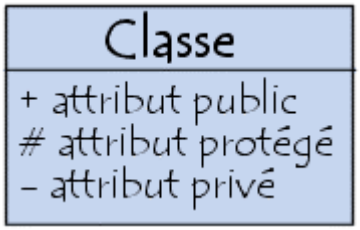
- **publique**: Les fonctions de toutes les classes peuvent accéder aux données ou aux méthodes d'une classe définie avec le niveau de visibilité *public*. Il s'agit du plus bas niveau de protection des données

- **protégée**: l'accès aux données est réservé aux fonctions des classes héritières, c'est-à-dire par les fonctions membres de la classe ainsi que des classes dérivées

- **privée**: l'accès aux données est limité aux méthodes de la classe elle-même. Il s'agit du niveau de protection des données le plus élevé

La notation UML permet de représenter le niveau de visibilité des attributs de façon graphique en faisant précéder le nom de chaque attribut par un caractère représentant la visibilité:

- + défini un attribut public
- # défini un attribut protégé
- - défini un attribut privé



Introduction au langage UML

INTRODUCTION AU LANGAGE UML

METHODOLOGIE – CNAM ANGOULEME 2000-2001

INTRODUCTION AU LANGAGE UML

UML est un langage de modélisation avec plusieurs objectifs qui en font un véritable outil de communication:

- Comprendre et décrire les besoins,
- Spécifier (un système),
- Etablir l'architecture logicielle.

Les diagrammes d'UML vont mettre en place deux parties:

Statique (structure)	Dynamique (comportement)
Diagramme de cas d'utilisation	Diagramme de collaboration
Diagramme de classe	Diagramme de séquence
Diagramme à objet	Diagramme état/ transition
Diagramme de composants	Diagramme d'activités
Diagramme de déploiement	

I. Les cas d'utilisation

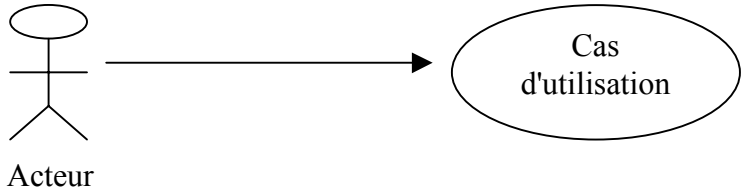
1.1) Objectifs des cas d'utilisation

Les cas d'utilisation sont une technique de description du système étudié privilégiant le point de vue de l'utilisateur. **Il s'agit de la solution UML pour représenter le modèle conceptuel.** Les cas d'utilisation décrivent sous la forme d'actions et de réactions, le comportement d'un système du point de vue d'un utilisateur. Les cas d'utilisation servent à structurer les besoins des utilisateurs et les objectifs correspondants du système.

Un cas d'utilisation est une manière spécifique d'utiliser un système. C'est l'image d'une fonctionnalité du système, déclenchée en réponse à la stimulation d'un acteur externe.

Les cas d'utilisation apportent une solution au problème de la **détermination et de la compréhension des besoins**. En effet fréquemment, des besoins se contredisent, des oublis et des imprécisions sont réalisés et ainsi l'analyse du système part sur de mauvaises bases. Les cas d'utilisation recentrent l'expression des besoins sur les utilisateurs en partant du principe qu'un système est avant tout construit pour ses utilisateurs. Les cas d'utilisation permettent aux utilisateurs de structurer et d'articuler leurs désirs ; ils les obligent à définir la manière dont ils voudraient interagir avec le système, à préciser quelles informations ils entendent échanger et à décrire ce qui doit être fait pour obtenir le résultat escompté.

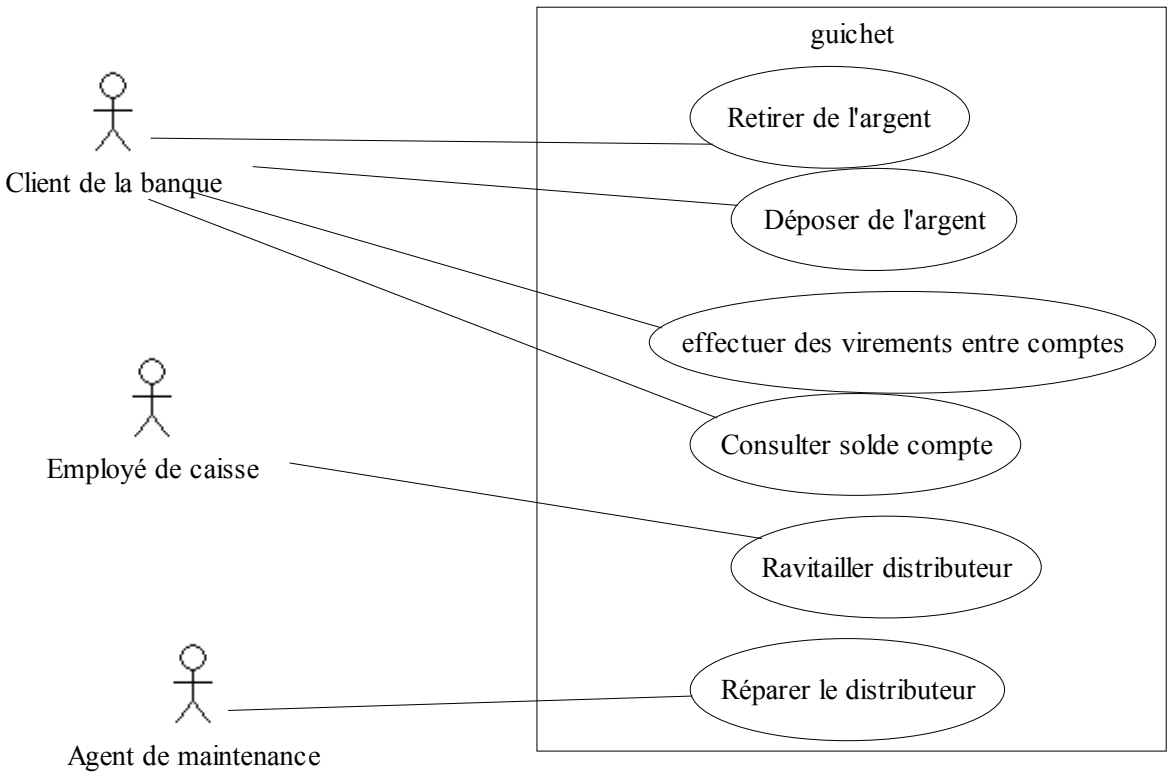
1.2) Éléments constitutifs des cas d'utilisation



- **Acteur :** entité externe qui agit sur le système ; Le terme acteur ne désigne pas seulement les utilisateurs humains mais également les autres systèmes. les acteurs sont des classificateurs qui représentent des rôles au travers d'une certaine utilisation (cas) et non pas des personnes physiques. Ce sont des acteurs types.

- **Cas d'utilisation :** ensemble d'actions réalisées par le système en réponse à une action d'un acteur.
  - les cas d'utilisation peuvent être structurés,
  - les cas d'utilisation peuvent être organisés en paquetages,
  - l'ensemble des cas d'utilisation décrit les objectifs du système.

Exemple : Diagramme de cas d'utilisation d'un guichet automatique bancaire



1.3) Description des cas d'utilisation

« Un cas d'utilisation spécifie une séquence d'interactions, avec ses variantes, entre les acteurs et le système, produisant un résultat satisfaisant pour un acteur particulier. ». On peut donc considérer un cas d'utilisation comme une abstraction de plusieurs chemins d'exécution d'une utilisation du système. Pour décrire un cas d'utilisation, il nous faut décrire un maximum de chemins d'exécution possibles pour la séquence d'actions correspondant à ce cas. On étudiera donc un certain nombre de scénarios d'un cas d'utilisation.

Un scénario est donc une instance de cas d'utilisation, un flot d'évènement.

A chaque fois qu'une instance d'un acteur déclenche un cas d'utilisation, un scénario est créé : ce scénario suivra un chemin particulier dans la description d'un cas d'utilisation. Un scénario ne contient pas de branche du type « si la condition X est vraie alors faire Y », car pendant l'exécution la condition est soit vraie, soit fausse mais elle aura toujours une valeur.

Bien sûr il est impossible de décrire un cas d'utilisation en écrivant tous les scénarios, l'exhaustivité est difficile à atteindre, mais il faut répertorier les scénarios les plus représentatifs afin de gérer les risques. On recherchera donc le ou les scénarios nominaux et les principaux cas d'exceptions.

Nous aurons donc deux niveaux de description :

- Description générale d'un cas d'utilisation reprenant les divers chemins pouvant être réunis en un même cas.
- Description des scénarios les plus pertinents.

a) **Exemple** : Cas d'utilisation Retirer de l'argent

1- Le client de la banque s'identifie
2- Le système lui propose les différents comptes sur lesquels il peut effectuer un retrait
3- Le client choisit le compte à débiter et spécifie le montant du retrait
4- Le système vérifie si le retrait est autorisé, si oui il déduit le montant du compte et délivre l'argent, sinon il renvoie un message de rejet de l'opération

b) **Exemple** : scénario du cas d'utilisation :Retirer de l'argent ; retrait autorisé

Acteur déclencheur : M. Martin

Ce scénario commence par
1- M. Martin s'identifie
2- Le système lui propose les différents comptes sur lesquels il peut effectuer un retrait
3- M. Martin choisit son compte chèque et demande 200F
/*Le système vérifie que le retrait est autorisé*/
4- Le système délivre deux billets de 100F et demande à ce que M. Martin les saisisse.

Ce scénario se termine par
5- M. Martin prend l'argent.

1.4) Structuration des cas d'utilisation

Après avoir identifié les acteurs et les cas d'utilisation, il est utile de restructurer l'ensemble des cas d'utilisation que l'on a fait apparaître afin de rechercher les comportements partagés, les cas particuliers et les généralisations.

Les relations possibles entre cas d'utilisation : UML définit trois types de relations standardisées entre cas d'utilisation, détaillées ci-après :

- Une relation d'inclusion : formalisée par la dépendance **« include »**
- Une relation d'extension : formalisée par la dépendance **« extend »**
- Une relation de généralisation / spécialisation

Etapes de construction:

1. Les acteurs
2. Pour chaque acteur, recherche des cas d'utilisation,
3. Structuration des cas d'utilisation pour faire apparaître les comportements partagés (relation d'inclusion), les cas particuliers ou options (relation d'extension), les généralisations / spécialisations.

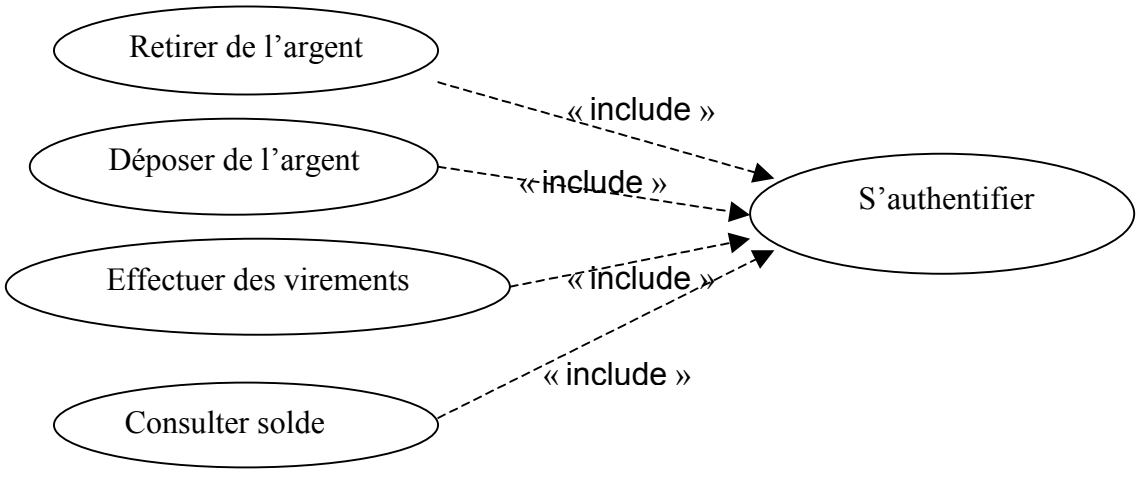
a) La relation d'inclusion :

Lors de la description des cas d'utilisation, il apparaît qu'il existe des sous-ensembles communs à plusieurs cas d'utilisation, il convient donc de factoriser ces fonctionnalités en créant de nouveaux cas d'utilisation qui seront utilisés par les cas d'utilisation qui les avaient en commun.

Un cas d'utilisation A utilise un cas d'utilisation B signifie que :

- une instance de A va engendrer une instance de B et l'exécuter,
- A dépend de B,
- B n'existe pas tout seul et A n'existe pas sans B

Exemple :



Les cas de base « Déposer de l'argent », « Retirer de l'argent », « Effectuer des virements entre comptes » et « Consulter solde » incorporent de façon explicite le cas d'utilisation « S'authentifier », à un endroit spécifié dans leurs enchaînements.

Remarquez que dans une relation « include », le cas d'utilisation de base utilise systématiquement les enchaînements provenant du cas inclus.

On utilise cette relation pour éviter de décrire plusieurs fois un même enchaînement d'actions. Ainsi on est amené à **factoriser** un comportement commun à plusieurs cas d'utilisation dans un cas d'utilisation à part.

b) La relation d'extension

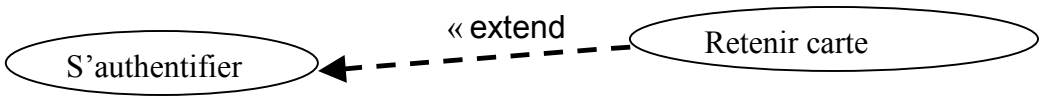
La relation stéréotypée <<extend>> permet d'étendre les interactions et donc les fonctions décrites par les interactions. Le cas de base peut fonctionner tout seul, mais il peut également être complété par un autre, sous certaines conditions, et uniquement à certains points particuliers de son flot d'évènements (point d'insertion). On utilise principalement cette relation pour séparer le comportement optionnel (les variantes) du comportement obligatoire.

Considérons l'exemple : le cas d'utilisation B étend le cas d'utilisation A signifie que :

- Une instance de A va engendrer une instance de B et l'exécuter sous certaines conditions, B sait où s'insérer dans A.
- B connaît A et non l'inverse, c'est à dire que B dépend de A
- B n'existe pas tout seul et A existe sans B

La relation <<extend>> montre une possibilité d'exécution d'interactions qui augmenteront les fonctionnalités du cas étendu, mais **de façon optionnelle**, non obligatoire, alors que la relation <<include>> suppose une **obligation** d'exécution des interactions.

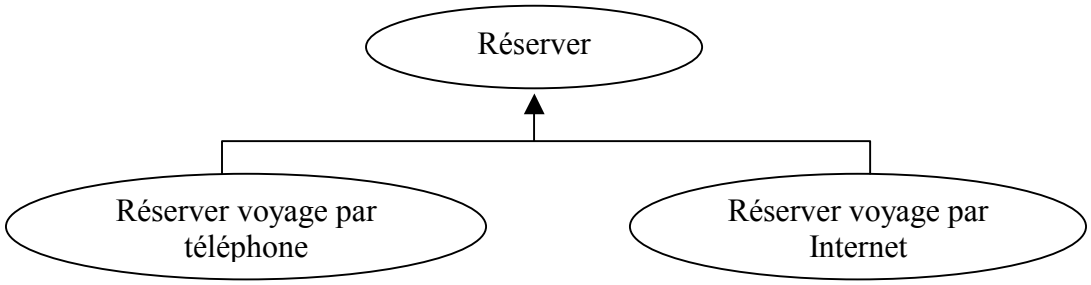
Exemple :



c) Relation de généralisation entre cas d'utilisation

La relation d'héritage ou de *généralisation* entre cas est plus subtile. La version 1.1 de UML ne distinguait d'ailleurs pas <<extend>> et généralisation. Cette relation est à prendre au sens classique de *spécialisation*, inhérent à 'héritage. Ici, la généralisation peut être vue aussi comme un "polymorphisme" de cas.

Exemple : Dans un système d'agence de voyage



Dans un système d'agence de voyage, un acteur touriste peut participer à un cas de base qui est "Réserver voyage", qui suppose par exemple des interactions basiques au comptoir de l'agence. Une réservation peut aussi être réalisée par téléphone ou internet. On voit qu'il ne s'agit pas de relation <<extend>> car la réservation par Internet n'étend pas les interactions ni les fonctionnalités du cas 'Réserver voyage". Elle les traduit différemment. Les interactions Internet ne sont pas une option des interactions comptoir. Par contre les deux cas "Réserver voyage" et "Réserver voyage par Internet" sont liés : la réservation par Internet est un cas particulier de réservation. De façon générale en objet, une situation de cas particulier se traduit par une relation de généralisation.

ATTENTION : On peut également généraliser les acteurs

1.6) Notion de paquetage

Les modèles UML produits dans le cadre d'un projet sont parfois d'une superficie trop importante pour être facilement utilisés. La notion de paquetage ou « package » est une technique qui permet de mettre en œuvre ce partitionnement des modèles tout en préservant la cohérence de l'ensemble.

Un paquetage est un ensemble d'éléments de modélisation : des classes, des associations, des objets, des composants…. *Dans le cas du guichet bancaire on pourrait faire deux diagrammes de cas d'utilisation, un pour le paquetage utilisation du guichet et un autre pour le paquetage maintenance.*

1.7) Exercice TVServices (parties I et II)

TVServices est une société qui met à disposition de ses clients un ensemble de services relatifs à la télévision (accès à des chaînes thématiques, contrôle des utilisations et des utilisateurs, programme électronique détaillé des chaînes accessibles...). Ces services sont commercialisés sous le nom d' « IntelliTélé »

Chaque client dispose d'un boîtier électronique situé entre l'antenne satellite et l'installation de télévisualisation (écrans, magnétoscopes...). C'est cet équipement intermédiaire, appelé ci-après "boîtier 'NS", constamment en veille et relié au réseau téléphonique, qui permet au client: le paiement des services par carte bancaire, et le décodage des images reçues.

Et à TVServices, la diffusion automatique des programmes et magazines électroniques (et des publicités ;-); en chargeant les mémoires des boîtiers avec les programmes et magazines, et l'autorisation d'accès aux services; TVS mémorise les informations relatives aux autorisations d'accès aux services (fonction des paiements reçus) dans les boîtiers.

Partie 1: Diagramme de cas d'utilisation - vue de l'administrateur

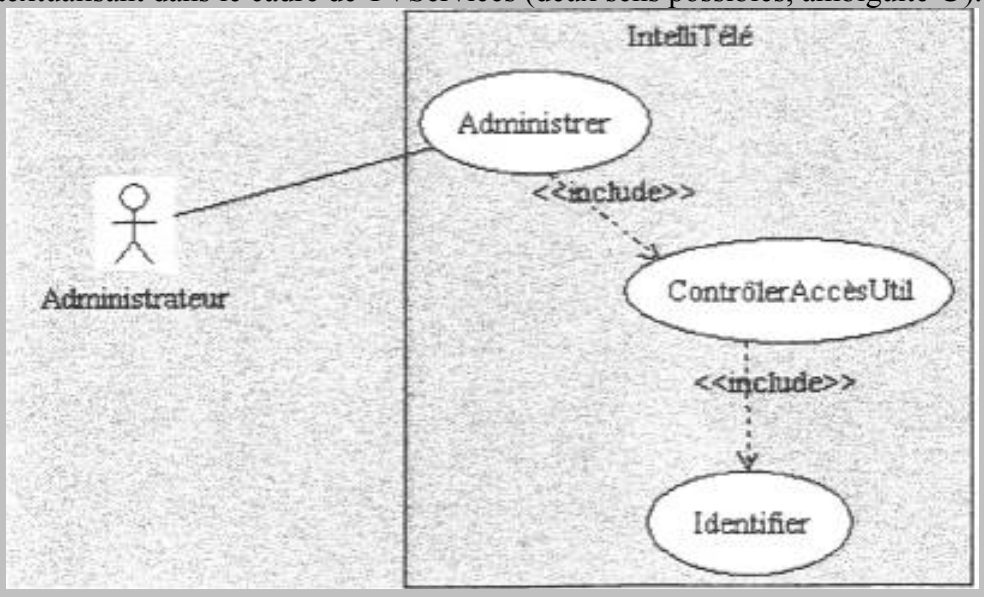
Voici un extrait de la brochure commerciale proposant les contrats 'IntelliTélé'

« Une fois le contrat signé et l'abonnement aux services de votre choix payé vous recevrez par l'intermédiaire d'un installateur agréé, un "boîtier TVS" préprogrammé avec les autorisations d'accès correspondant à votre contrat.

Lors de l'installation, un compte "administrateur" (typiquement, pour une installation familiale, un des parents) est créé qui aura pour tâche de déclarer les futurs utilisateurs du système (identificateur et mot de passe) ainsi que d'administrer leurs droits (types d'émission autorisés, plages horaires autorisées, durée maximale hebdomadaire de visualisation autorisée appelée "crédit hebdomadaire").»

Travail à faire:

I.l) Expliquer le diagramme ci-dessous par un texte décrivant les informations qu'il contient en les contextualisant dans le cadre de TVServices (deux sens possibles, ambiguïté ☹).



Voici une description possible du diagramme proposé:

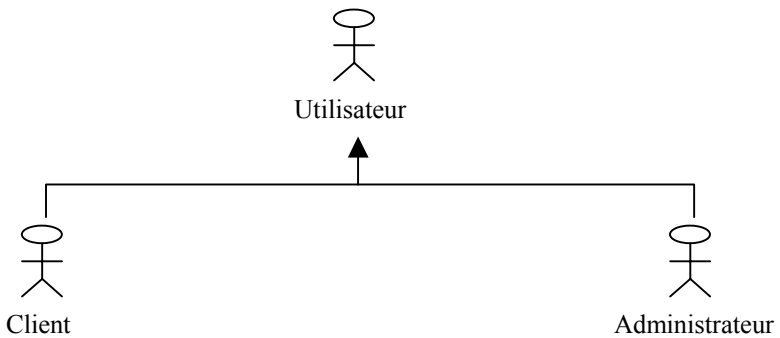
L'accès aux fonctions d'administration est protégé, le contrôle s'effectuant sur l'identificateur et le mot de passe de l'administrateur. Ce qui explique que l'exécution du cas d'utilisation 'Administrer' nécessite celle du cas d'utilisation 'ContrôleAccèsUtil', dont l'exécution nécessite celle du cas d'utilisation 'Identifier'.

Le découpage fonctionnel de l'administration en création des utilisateurs, contrôle d'accès des utilisateurs... ne me semble pas pertinent à ce niveau de description avec l'approche 'cas d'utilisation'.

I.2) Quel peut-être l'intérêt de séparer les cas d'utilisation "ContrôleAccèsUtil" et "Identifier" du cas d'utilisation "Administrer"?

Il peut y avoir intérêt à séparer les cas d'utilisation "ContrôleAccèsUtil" et "Identifier" du cas d'utilisation "Administrer" si les deux premiers sont utilisés par ('inclus dans') d'autres cas d'utilisation; on les a en quelque sorte factorisés.

I.3) Représenter par un fragment de diagramme de cas d'utilisation le fait que parmi les acteurs sollicitant le système, "Client" et "Administrateur" sont tous deux "Utilisateur".



Partie II: Diagramme de cas d'utilisation - vue de l'utilisateur

Voici un extrait de la description faite ci-dessous par un des premiers clients:

« Une "IntelliTélé" est un téléviseur-enregistreur, elle (ou 'il' je ne sais pas...☺) permet de regarder des émissions télédiffusées, et aussi de les enregistrer. C'est un système qui, via un réseau, connaît les possibilités pour lesquelles on a payé. Par exemple, on peut acheter des droits de réception de certaines chaînes payantes ou s'abonner à un service qui envoie régulièrement une analyse détaillée des programmes des principales chaînes.

Pour utiliser une "IntelliTélé", on dispose d'un terminal, comme une télécommande, avec un petit écran tactile, qu'on manipule avec un stylet. Dans ce terminal on peut glisser si nécessaire une carte à puce, par exemple pour payer un service.

Pour pouvoir utiliser "l'IntelliTélé", il faut s'identifier. Un utilisateur autorisé dispose de droits ; c'est moi qui ai défini les droits d'accès des enfants. Je leur ai interdit des plages horaires et certaines chaînes. Il me reste à interdire certains types de programme (je sais que c'est possible car ils disent que le boîtier connaît la catégorie de chaque émission). Mais c'est vrai je ne vous ai pas encore parlé du boîtier; tenez, voici la présentation de TVServices, la société qui commercialise "L'IntelliTélé",... » (Voir la partie I).

Travail à faire:

En imaginant utiliser une IntelliTélé comme vous le faites (peut-être) de votre récepteur TV/magnétoscope, et en tenant compte de la description ci-dessus, produire un diagramme de cas d'utilisation de l'IntelliTélé vu de l'utilisateur ☺ acteur générique comme défini en 1.3. ci-avant.

Scénario du cas d'utilisation: Utiliser la TV; Enregistrement direct.
Acteur déclencheur: Mr Martin
Ce scénario commence par:

1. Mr Martin s'identifie

/* Le système lui donne l'autorisation d'accès */

2. Mr Martin décide d'enregistrer une émission sur M6 en direct

Ce scénario se termine par

3. Mr Martin éteint la TV

Scénario du cas d'utilisation: Utiliser la TV; Accès à une chaîne refusée.
Acteur déclencheur: Mr Martin
Ce scénario commence par:

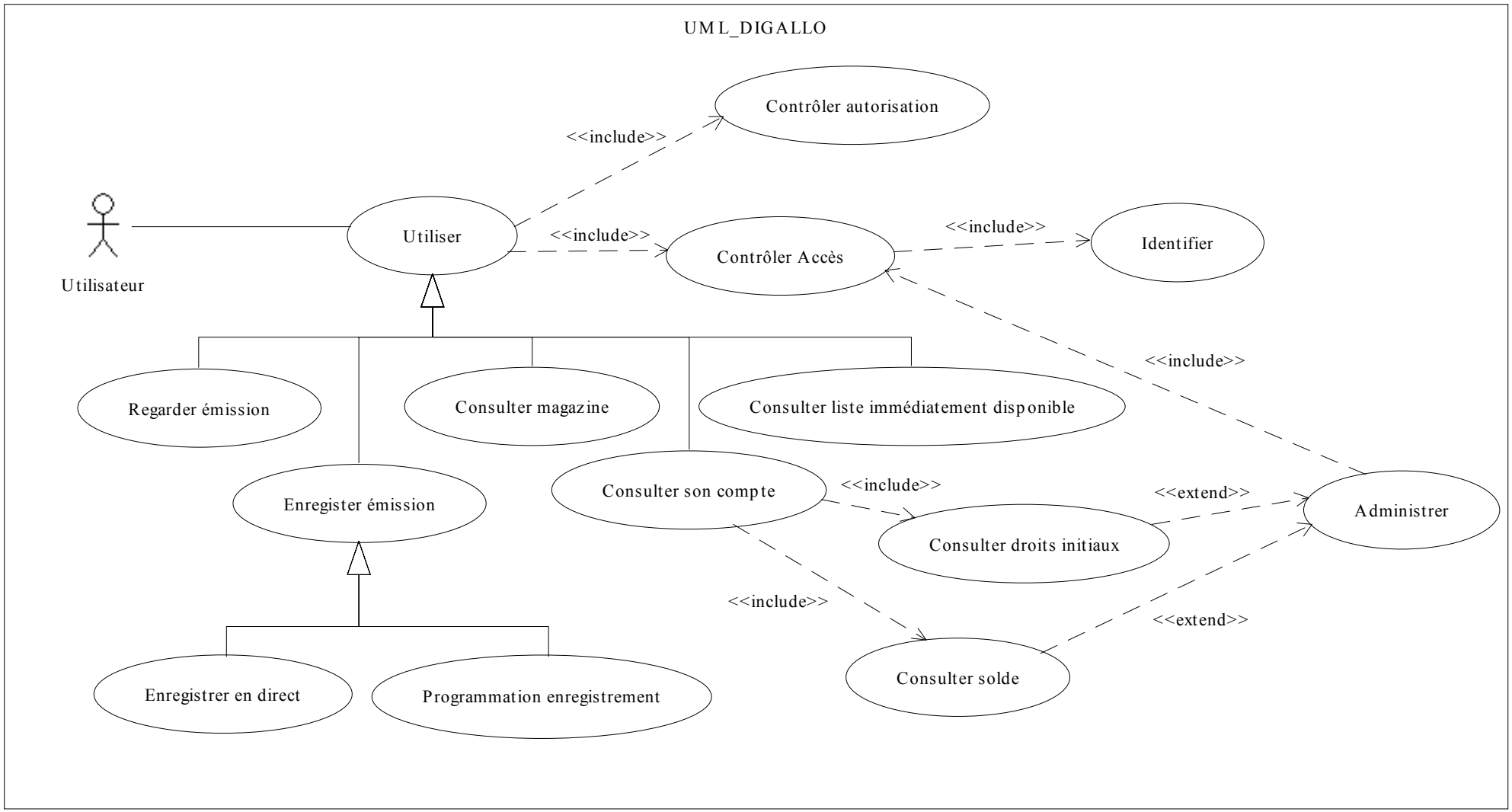
1. Mr Martin s'identifie
2. Mr Martin décide de regarder une émission d'informations sur France3
3. Le système lui indique qu'il n'a pas les droits d'accès sur cette chaîne.

Ce scénario se termine par

4. Mr Martin éteint la TV

Scénario du cas d'utilisation: Utiliser la TV; Crédit de temps restant de 5 minutes.
Acteur déclencheur: La fille de Mr Martin
Ce scénario commence par:

1. La fille de Mr Martin s'identifie
2. Le système la reconnaît et lui affiche qui ne lui reste que 5 minutes de crédit
3. Elle choisit de regarder un dessin animé
4. au bout de 5 minutes, la TV s'éteint toute seule.


UML_DIGALLO
Contrôler autorisation
<<include>>
Utiliser
<<include>>
Contrôler Accès
<<include>>
Identifier
Utilisateur
<<include>>
Regarder émission
Consulter magazine
Consulter liste immédiatement disponible
Enregister émission
Consulter son compte
<<include>>
<<extend>>
Administrer
Consulter droits initiaux
<<include>>
<<extend>>
Consulter solde
Enregistrer en direct
Programmation enregistrement

II. Le diagramme de classes

Les diagrammes de classes expriment de manière générale la structure statique d'un système, en termes de classes et de relations entre ces classes. Une classe permet de décrire un ensemble d'objets (attributs et comportement), tandis qu'une relation ou association permet de faire apparaître des liens entre ces objets. On peut donc dire :

- un objet est une instance de classe,
- un lien est une instance de relation

Le diagramme de classe est un modèle permettant de décrire de manière abstraite et générale les liens entre objets.

UML permet de définir trois types de stéréotypes pour les classes :

a) **les classes « frontière »**(interface): classes qui servent à modéliser les interactions entre le système et ses acteurs.
b) **les classes « contrôle »** : classes qui servent à représenter la coordination, le séquencement, les transactions et le contrôle d'autres objets.
c) **les classes « entité »** : classes qui servent à modéliser les informations durables et persistantes.

Dans un premier temps c'est à cette dernière catégorie de classes que nous allons nous intéresser. Le diagramme de classe va être un outil nous permettant de représenter le modèle du domaine. Le modèle du domaine saisit les éléments les plus importants pour comprendre le contexte du système :

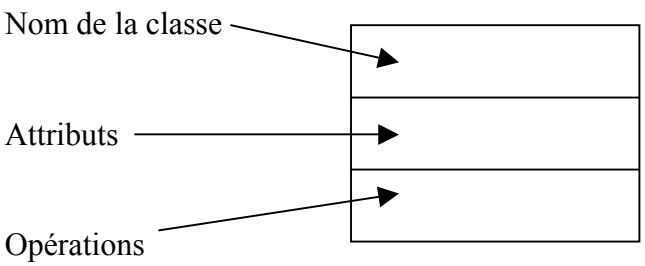
- les objets métiers (mis en œuvre dans une activité professionnelle tels que les commandes, les contrats..) ,
- les concepts du domaine à modéliser dont le système doit garder une trace,
- les évènements s'étant produits ou devant se produire qui déclencheront un certain comportement des objets.

2.1) Les classes

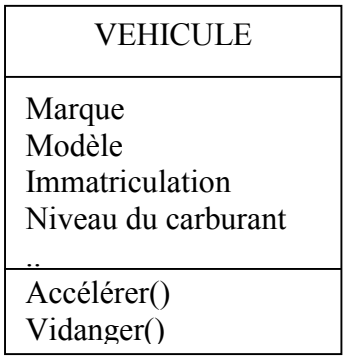
a) Définition

Description d'un ensemble d'objets partageant la même sémantique, ainsi que les mêmes attributs, opérations et relations.

b) Représentation



Exemple :



Une classe correspond à un concept global d'information et se compose d'un ensemble d'informations élémentaires, appelées attributs de la classe qui servent à la décrire.

UML définit trois niveaux de visibilité pour les attributs et les opérations :

- Public qui rend l'élément visible à tous les clients de la classe,
- Protégé qui rend l'élément visible aux sous classes de la classe,
- Privé qui rend l'élément visible à la classe seule.

Un attribut est caractérisé par un nom et par un format.

c) Syntaxe

Nom_attribut : type_attribut = valeur_par_défaut

Dans un premier temps on ne retiendra comme attribut que des données non calculées, cependant par commodité de gestion on pourra garder des attributs pouvant être construits à partir d'autres attributs (on rajoutera / devant l'attribut dit dérivé).

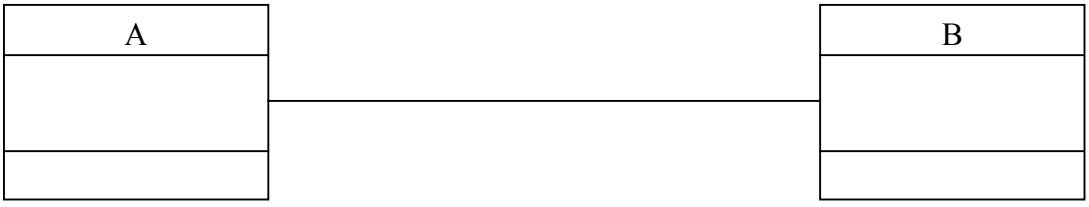
d) Opérations

La définition d'une classe est complétée par l'ensemble des opérations qu'elle peut exécuter. Une opération est une fonctionnalité assurée par la classe.

Le niveau de détail à retenir pour décrire les opérations est fonction du niveau d'avancement de l'étude.

2.2) Les associations

Une association représente une relation structurelle entre classes d'objets. La plupart des associations sont binaires, c'est à dire qu'elles connectent deux classes. On représente une association en traçant une ligne entre les classes associées.

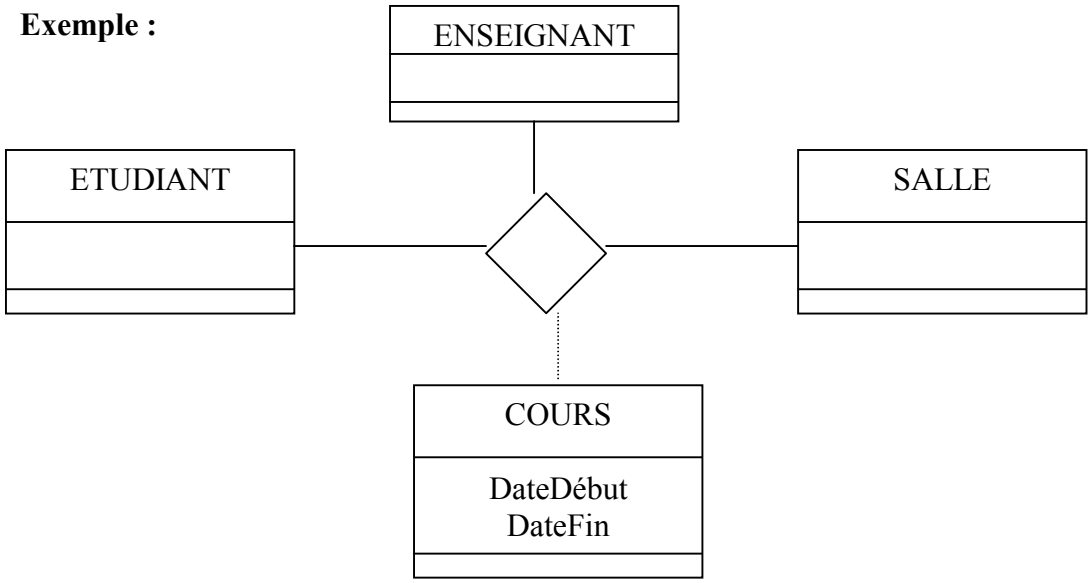


a) Arité des associations

On appelle arité d'une association le nombre de classes qui participent à l'association.

Il arrive en effet que l'information que l'on veut représenter nécessite la mise en relation de plus de deux classes. Ces associations n'aires peuvent se représenter au moyen d'un losange sur lequel arrivent les différents brins de l'association.

Exemple :

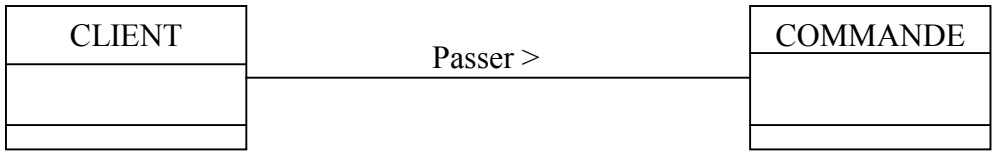


b) Rôle

Les extrémités d'une association sont appelées rôles et peuvent porter un nom. Le rôle décrit comment une classe voit une autre classe au travers d'une association.

c) Identification des associations

Les associations peuvent être nommées afin de faciliter la compréhension des modèles. Il est d'usage de nommer les associations par une forme verbale. On peut également préciser le sens de lecture par le biais d'un petit triangle dirigé vers la classe désignée par la forme verbale.



d) Identification des rôles

Un rôle est nommé au moyen d'une forme nominale.

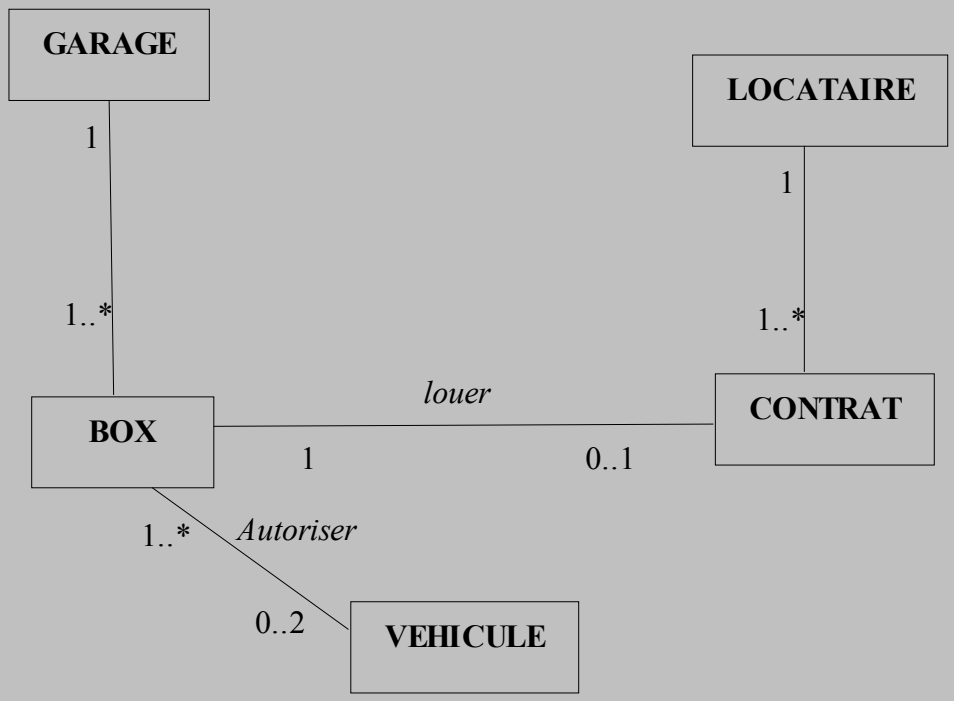
On utilise rarement les deux modes de description d'association simultanément, mais on cherche celui le plus adapté à la situation à décrire.

e) Multiplicité des associations

Chaque rôle peut porter une multiplicité montrant combien d'objets de la classe considérée (celle qui joue ce rôle) peuvent être liés à une instance de l'autre classe par l'association. La multiplicité est représentée sous la forme d'un couple de cardinalités.

1..1 noté 1	Un et un seul
0..1	Zéro ou un
0..* noté *	De Zéro à n
1..*	De un à n
n..m	De n à m

➢ ***Exercice 2 : Lecture de multiplicité***



Travail à faire :

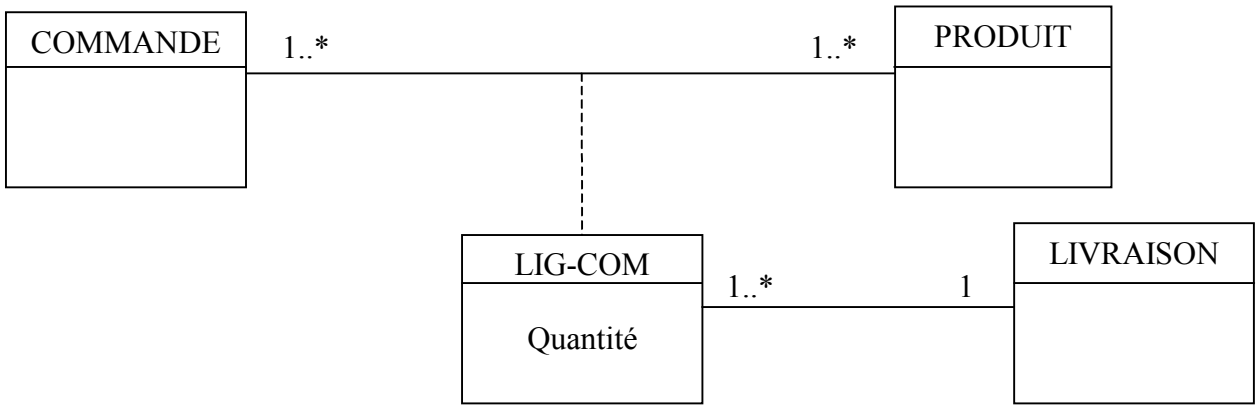
Faites une description de ce diagramme de classe en explicitant en une phrase chacune des multiplicités.

Un box peut être loué par au maximum un seul contrat ou peut rester non loué.
Un contrat concerne la location d'un seul box à la fois.
Un box peut être vide ou contenir au maximum 2 véhicule.
Un véhicule est autorisé à aller dans un box (au minimum) ou plus.
Un contrat ne concerne qu'un seul locataire.
Un locataire peut souscrire plusieurs contrat mais doit en avoir souscrit au moins un.

f) Les classes associations :

Il peut arriver que l'on ait besoin de garder des informations (attributs ou opérations) propres à une association. Une classe de ce type est appelée classe association.

Exemple :



Ici, NumCommande + RéfProduit → Quantité

Une classe association est une classe comme une autre qui peut entretenir des relations avec d'autres classes

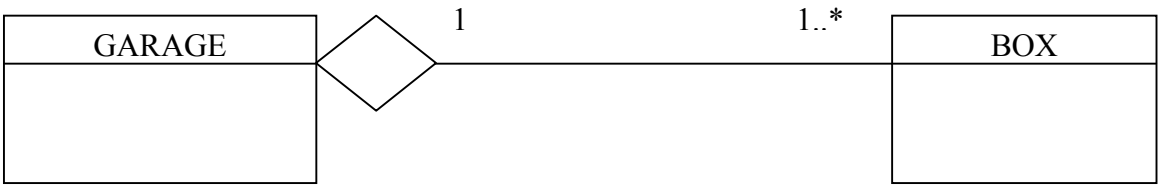
g) Agrégation

Une agrégation est un type particulier d'association. Elle traduit la volonté de renforcer la dépendance entre les classes. C'est une association non symétrique dans laquelle une des extrémités joue un rôle prédominant par rapport à l'autre extrémité.

Les critères suivants impliquent une agrégation :

- une classe fait partie d'une autre classe,
- une action sur une classe implique une action sur une autre classe,
- les objets d'une classe sont subordonnés aux objets d'une autre classe.

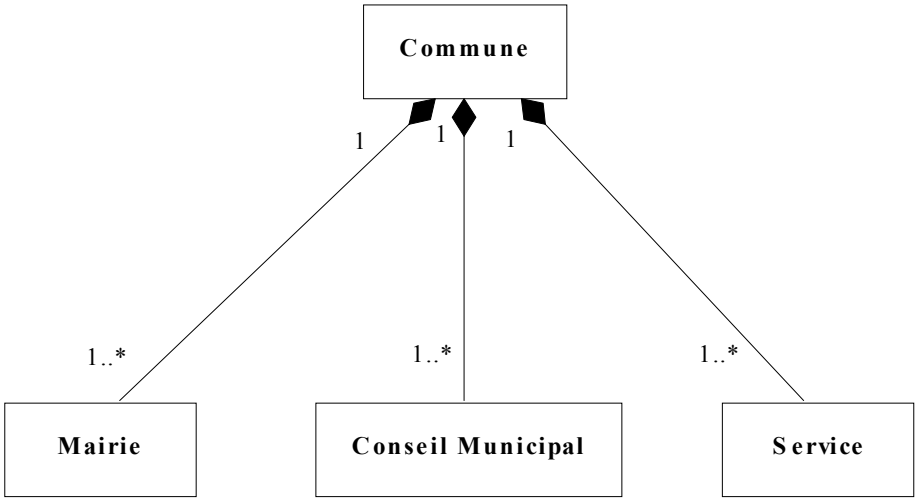
Attention : l'inverse n'est pas toujours vrai ; l'agrégation n'implique pas nécessairement tous les critères ci-dessus.



Un agrégat peut être multiple.

h) La composition

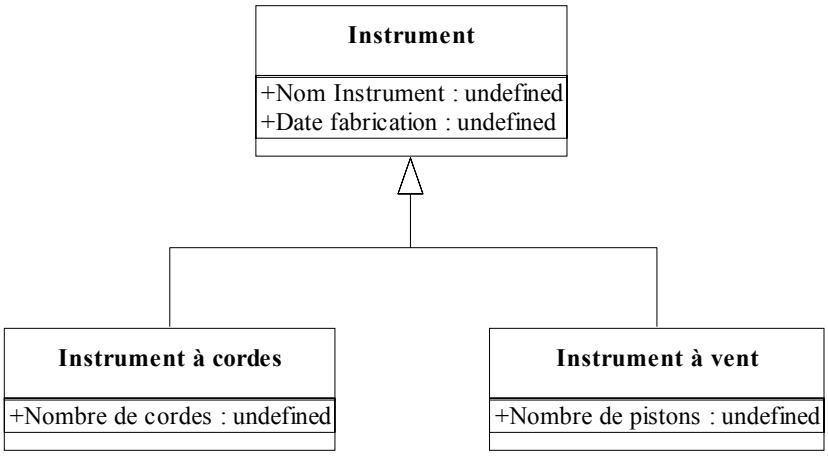
La composition est un cas particulier d'agrégation dans laquelle la vie des composants est liée à celle de l'agrégat. Dans la composition, l'agrégat ne peut être multiple. La composition se représente par un losange noir.



Une composition est une association contraignante : la suppression d'un objet agrégat entraîne la suppression des objets agrégés.

i) Généralisation :

UML emploie le terme de généralisation pour désigner la relation de classification entre un élément plus général et un élément plus spécifique. La relation de généralisation signifie « est un » ou « est une sorte de ».



La classe 'Instrument' est une classe générique, elle porte les attributs communs à tous les instruments. La classe 'Instrument à cordes' est une classe spécialisée qui porte les attributs spécifiques à ce type d'instrument.

Une classe spécialisée peut avoir des relations avec d'autres classes.

III. Le diagramme de collaboration

Nous avons jusqu'à présent étudié la statique (la structure) du système à modéliser à travers le diagramme des cas d'utilisation et le diagramme de classe. Nous allons maintenant passer à l'étude de la dynamique (le comportement) du système. Pour cela nous allons chercher à mettre en évidence les interactions entre objets, ainsi que les messages échangés.

Le diagramme de collaboration permet de mettre en évidence les interactions entre les différents objets du système étudié, ainsi que les messages qu'ils échangent entre eux.

3.1) Interaction

La séquence d'actions d'un scénario d'un cas d'utilisation débute lorsqu'un acteur invoque ce cas d'utilisation en envoyant une forme quelconque de message au système. En fait ce n'est pas le système dans son ensemble qui va recevoir le message mais un objet frontière (ou interface). L'objet frontière va envoyer à son tour un message à un autre objet, de sorte que les objets concernés vont dialoguer pour réaliser ce cas d'utilisation ou plus précisément **un scénario** de ce cas d'utilisation.

A l'aide du diagramme de collaboration, nous illustrons donc l'interaction entre objets en créant des liens entre ces objets et en associant des messages à ces liens. Le nom d'un message doit évoquer l'intention de l'objet appelant lors de l'interaction avec l'objet associé.

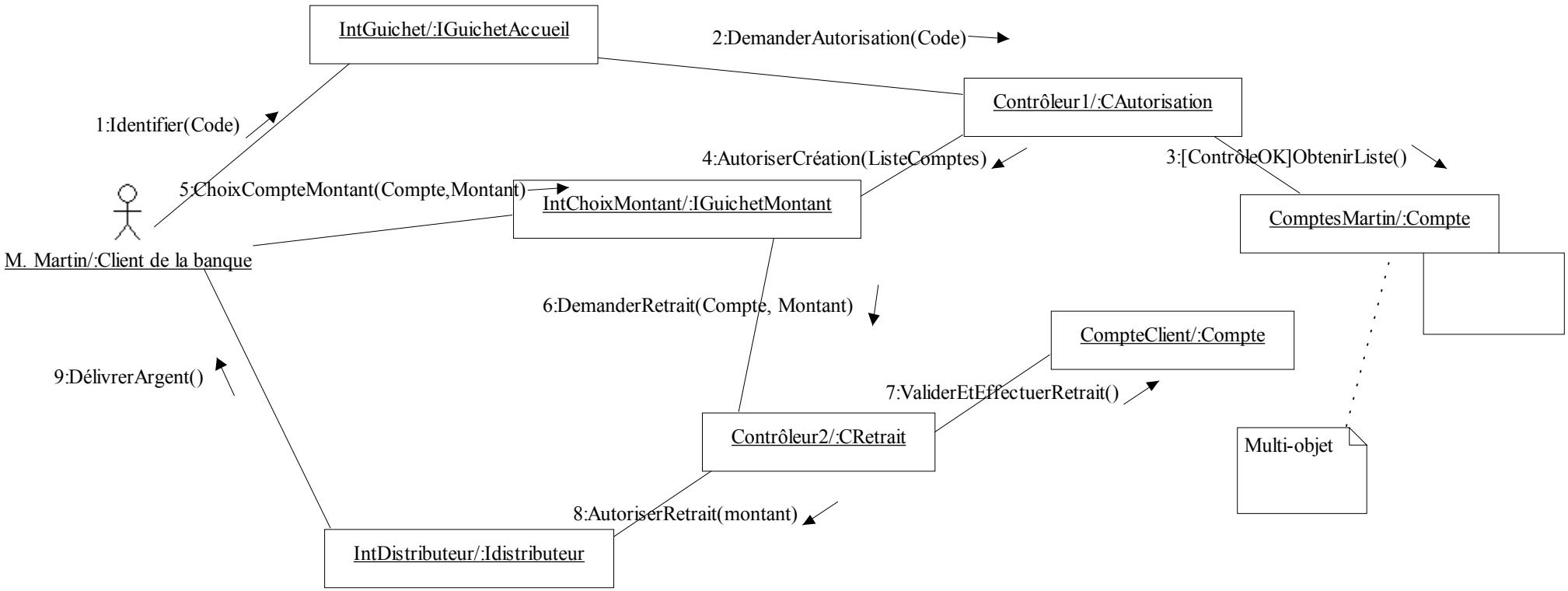
3.2) De nouveaux stéréotypes de classe

Ce diagramme de collaboration va nous permettre de compléter le modèle d'analyse commencé avec le diagramme de classe en ajoutant de nouvelles classes. Dans une première version du diagramme de classe nous nous sommes limités à l'études des **classes entités**. Nous allons voir à l'aide du diagramme de collaboration qu'il est nécessaire d'avoir recours à d'autres types de classes pour gérer les interactions entre objets du système :

- **Les classes frontières (ou interfaces) :** servent à modéliser les interactions entre le système et ses acteurs ;
- **Les classes de contrôle** : ces classes encapsulent souvent le contrôle lié à un cas d'utilisation, ce qui implique qu'un objet de contrôle est créé au démarrage d'une instance de cas d'utilisation et prend fin à l'issue de ce scénario. Il y a néanmoins des exceptions : lorsqu'un objet de contrôle participe à la réalisation de plusieurs cas d'utilisation, lorsque plusieurs objets de contrôle (issus de différentes classes de contrôle) participent à la réalisation du cas d'utilisation, et enfin lorsqu'une réalisation de cas d'utilisation ne nécessite aucun objet de contrôle.

Un diagramme de collaboration permet de décrire les interactions entre objets intervenant dans la réalisation d'un scénario d'un cas d'utilisation.

Exemple : Utilisation d'un diagramme de collaboration pour décrire la réalisation du scénario « retrait autorisé » du cas d'utilisation « retirer de l'argent »



Le diagramme de collaboration peut être complété par du texte décrivant de quelle façon les objets dialoguent pour effectuer le scénario du cas d'utilisation. C'est une description de scénario un peu plus précise.

Exemple : **scénario du cas d'utilisation :Retirer de l'argent ; retrait autorisé**

Acteur déclencheur : M. Martin

Ce scénario commence par

1. M. Martin active l'objet Interface Guichet et s'identifie
2. L'interface Guichet demande au contrôleurAutorisation si le code est valide. C'est le cas, le contrôleurAutorisation demande les comptes de M. Martin (le message est donc envoyé à plusieurs objets et non à un seul.) Il demande ensuite la création d'une instance de l'interface IguichetMontant IntChoixMontant présentant la liste des comptes de M.Martin et demandant le montant.
3. M. Martin choisit son compte chèque et demande 200F (message envoyé à l'objet IntChoixMontant)

   /*Le système vérifie que le retrait est autorisé*/
4. L'Objet IntChoixMontant transmet la demande de retrait du compte et du montant choisis par M. Martin à un deuxième objet contrôleur chargé de vérifier que M. Martin peut retirer cet argent sur son compte. L'objet contrôleur2 confirme en demandant à l'objet CompteClient de valider la requête et, si celle-ci est correcte, de débiter le compte. L'objet contrôleur2 autorise ensuite l'objet IntDistributeur à délivrer le montant demandé par M. Martin. Enfin M. Martin reçoit le montant demandé.

Ce scénario se termine par

5. M. Martin prend l'argent.

3.3) Les Messages :

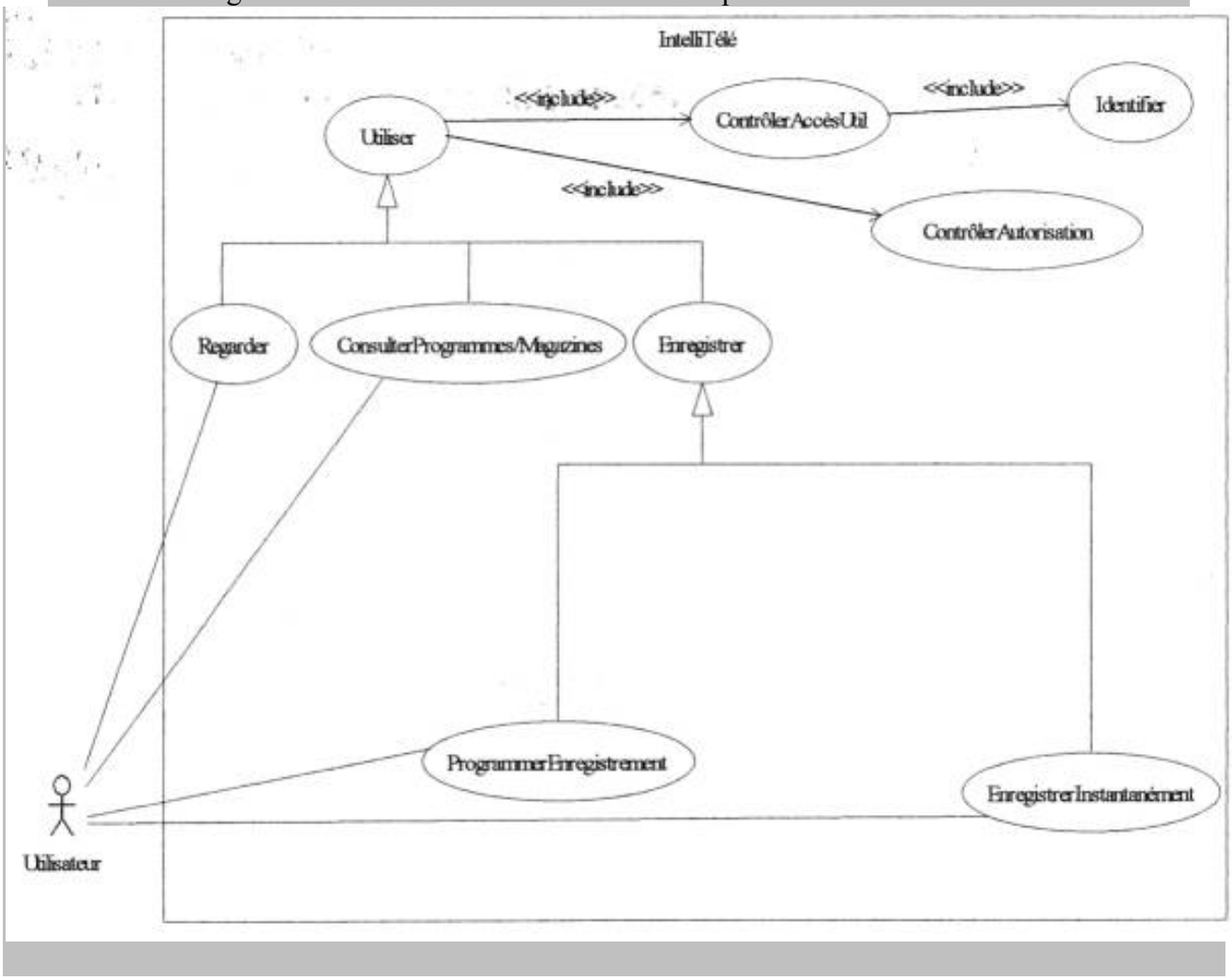
Les messages sont les seuls moyens de communication entre les objets. Ils sont donc positionnés sur le lien entre deux objets.

- Un *nom* est associé au message pour faciliter son identification.
- La *séquence* permet de préciser l'ordre d'émission des messages.
- Un message peut être complété par un ou plusieurs *arguments* :le message 5 ChoixCompteMontant a pour argument le compte et le montant choisis par le client.
- Certains messages peuvent solliciter un *résultat.* On peut représenter ce cas par deux messages : le premier fait la demande, le second apporte la réponse. Toutefois, lorsque le message en retour est immédiatement attendu, pour simplifier les diagrammes on peut les omettre.
- L'émission d'un message peut être soumise à une *garde :* .Le contrôle doit être OK pour demander la liste des comptes d'un client.

3.4) Exercice TVServices (parties III et IV)

TvServices est une société qui met à disposition de ses clients un ensemble de services relatifs à la télévision (accès à des chaînes thématiques, contrôle des utilisations et des utilisateurs, programme électronique détaillé des chaînes accessibles...). Ces services sont commercialisés sous le nom d' « IntelliTélé »Chaque client dispose d'un boîtier électronique situé entre l'antenne satellite et l'installation de télévisualisation (écrans, magnétoscopes...). C'est cet équipement intermédiaire, appelé ci-après "boîtier TVS", constamment en veille et relié au réseau téléphonique, qui permet au client le paiement des services par carte bancaire, et le décodage des images reçues; et à TvServices, la diffusion automatique des programmes et magazines électroniques (et des publicités ;-); en chargeant les mémoires des boîtiers avec les programmes et magazines, et l'autorisation d'accès aux services; TVS mémorise les informations relatives aux autorisations d'accès aux services (fonction des paiements reçus) dans les boîtiers.

Voici le diagramme de cas d'utilisation résultat de la partie II



Partie III: Diagramme de classe métier pour le cas d'utilisation « regarder »

Définitions* et *contraintes complétant la description de l'installation donnée en Partie I:

Les chaînes proposent des émissions.

Une émission peut être l'objet de plusieurs diffusions. Une émission a une certaine durée.

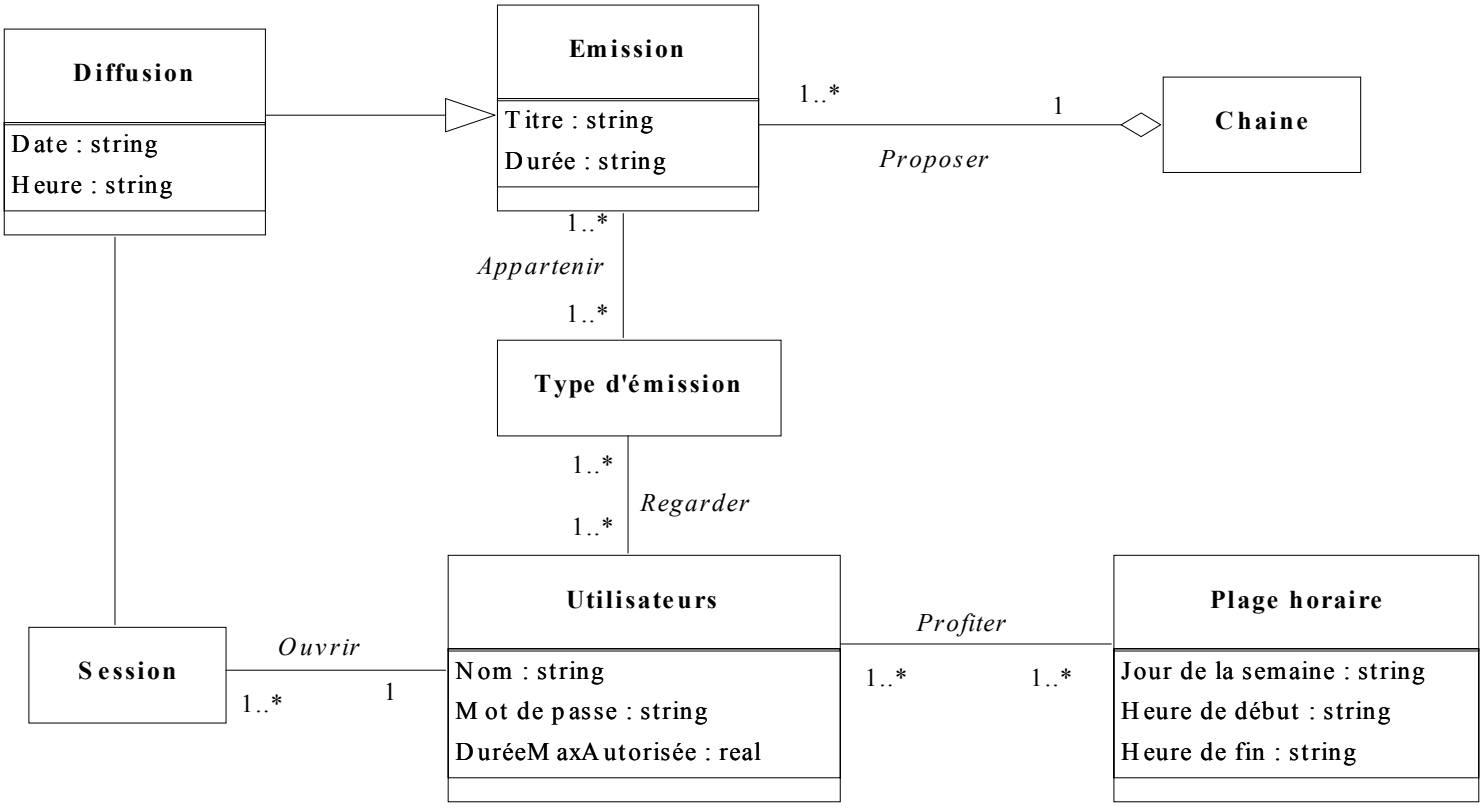
Une diffusion a une certaine date/heure de début.

Une plage horaire est relative à un certain jour de la semaine, elle commence et se termine à des heures rondes.

Une session est l'utilisation du système par un utilisateur, d'une certaine date/heure de début à une certaine date/heure de fin, au cours de laquelle il peut zapper d'une chaîne à une autre.

Une session n'est autorisée que 51 l'utilisateur qui la déclenche est un utilisateur autorisé pour le type d'émission, correspondant à la diffusion regardée, Si l'heure de début de session appartient à une plage horaire autorisée pour cet utilisateur, et Si le crédit hebdomadaire de cet utilisateur n'est pas atteint.

Proposer une première version du diagramme de classe faisant apparaître les classes entités et les attributs nécessaires à la vision statique de ce diagramme de classe.



Partie IV: Diagramme de collaboration d'un scénario du cas d'util. «regarder»

Description du cas d'utilisation "Regarder"

Ce cas d'utilisation commence par une demande d'ouverture de session faite par l'utilisateur.

Le système crée une session "en cours d'ouverture" et demande à l'utilisateur de s'identifier et de donner son mot de passe.

L'utilisateur saisit son identificateur et le mot de passe associé.

Le système contrôle ces informations et vérifie que la plage courante est une plage autorisée pour l'utilisateur; il autorise l'ouverture de session.

Le système propose parmi les chaînes accessibles (fonction de l'abonnement) au client, parmi les émissions en cours de diffusion, celles qui sont autorisées à l'utilisateur qui a déclenché la session.

L'utilisateur en choisit une.

Le système envoie les images correspondantes au téléviseur.

…

L'utilisateur arrête le téléviseur.

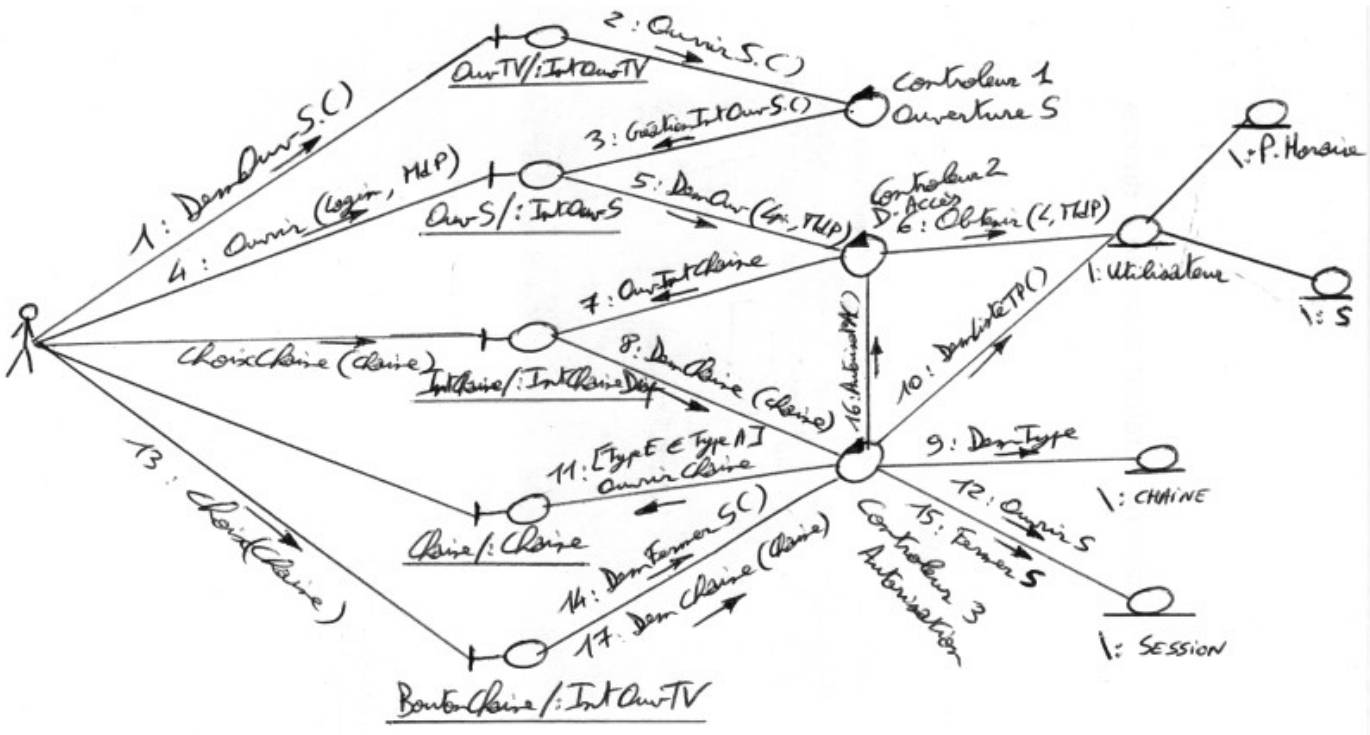
Le système temporise 30 secondes; au bout des 30 secondes, le téléviseur restant arrêté, le système ferme la session.

Ainsi se clôt ce cas d'utilisation.

IV.1) Ecrire un scénario nominal et un scénario d'exception du cas d'utilisation «regarder».

IV.2) Choisir un des 2 scénarios et réaliser le diagramme de collaboration correspondant.

IV.3) Modifier le diagramme de classes en faisant apparaître les classes entités et contrôleurs ainsi que les attributs nécessaires à la réalisation de ce cas d'utilisation.



3.5) TP de synthèse: Création d'un site Web

L'IWTS (International Waste Treatment Society), association internationale de chercheurs scientifiques, a pour objet de promouvoir des actions de recherche sur le traitement des déchets. Elle compte différents collèges : celui des universitaires et des étudiants en doctorat, celui des ingénieurs des administrations, celui des laboratoires, celui des industriels producteurs... Pour améliorer sa communication et sa réactivité, cette association souhaite fonder une communauté virtuelle internationale, elle désire donc mettre en place un site web orienté vers tous ses partenaires et permettant d'effectuer des transactions.

a) L'adhésion:

La procédure d'adhésion d'un membre à l'association doit se faire en ligne. Une personne peut demander à devenir adhérente à condition d'être parrainée par deux membres en place. Le demandeur connecté à la partie du site de l'IWTS accessible au public, remplit un formulaire de demande d'adhésion. La première partie du formulaire demande une @dresse électronique et les références des deux parrains. Après contrôle de l'identité des parrains une clé est fournie au demandeur (dans sa boîte aux lettres électronique) qui pourra ainsi accéder au formulaire d'adhésion complet. L'association souhaite connaître; l'identité du demandeur, son adresse personnelle, son diplôme le plus important, son adresse professionnelle, sa profession (par exemple ingénieur d'études) et sa fonction dans l'organisation (par exemple directeur de laboratoire), ses thèmes préférés de recherche et les récompenses obtenues (par exemple Awards 2000 de la meilleure publication sur le thème du lagunage...). Le demandeur choisit les groupes thématiques auquel il souhaite appartenir; il doit fournir également sa photo numérisée et les références de deux articles dont il est l'auteur; ces articles doivent avoir paru dans des revues scientifiques répertoriées, ou être disponibles sur un site, auquel cas c'est l'adresse réticulaire (URL) des articles qu'il fournit.Un des membres du conseil (composé d'un président, d'un vice président, et d'un nombre de membres de chaque collège proportionnel au nombre d'adhérents de ce collège) qui dirige l'association examine la demande d'adhésion, notamment le contenu scientifique des deux articles cités par le demandeur et contrôle le parrainage. Si le conseil apprécie le dossier du demandeur, un collège lui est proposé et il peut devenir membre. S'il confirme sa demande, le demandeur (admis) signale le mode de paiement de l'adhésion qu'il souhaite. Le tarif dépend du collège du membre. Le membre admis peut, après s'être acquitté du paiement de bénéficier des informations privées de l'association. L'enregistrement du paiement est effectué par le secrétariat. Il fait alors partie de la liste de diffusion du groupe thématique auquel il a choisi d'appartenir; il pourra dialoguer avec les autres membres lorsqu'il aura signé la charte déontologique des listes. Cela peut se faire à tout moment notamment lors de la confirmation de demande d'adhésion.

b) La publication des articles:

La revue de l'association est publiée pour ses membres sur le web tous les trois mois. Les membres ont deux mois entre chaque revue pour proposer des articles. Un article possède un titre, un thème et des mots clefs. La sélection des articles est décidée par le conseil, après avis de deux lecteurs du groupe thématique correspondant, choisis par le conseil. Un article peut être accepté, refusé ou proposé après amendements, pour une relecture. L'avis, après la deuxième lecture est définitif. L'auteur de l'article reçoit une évaluation de l'article où sont notées les remarques précises des lecteurs, ceux ci restent anonymes. Un résumé de chaque article publié sera accessible sur la partie publique du site web, à titre d'apéritif, avec ses mots clefs.

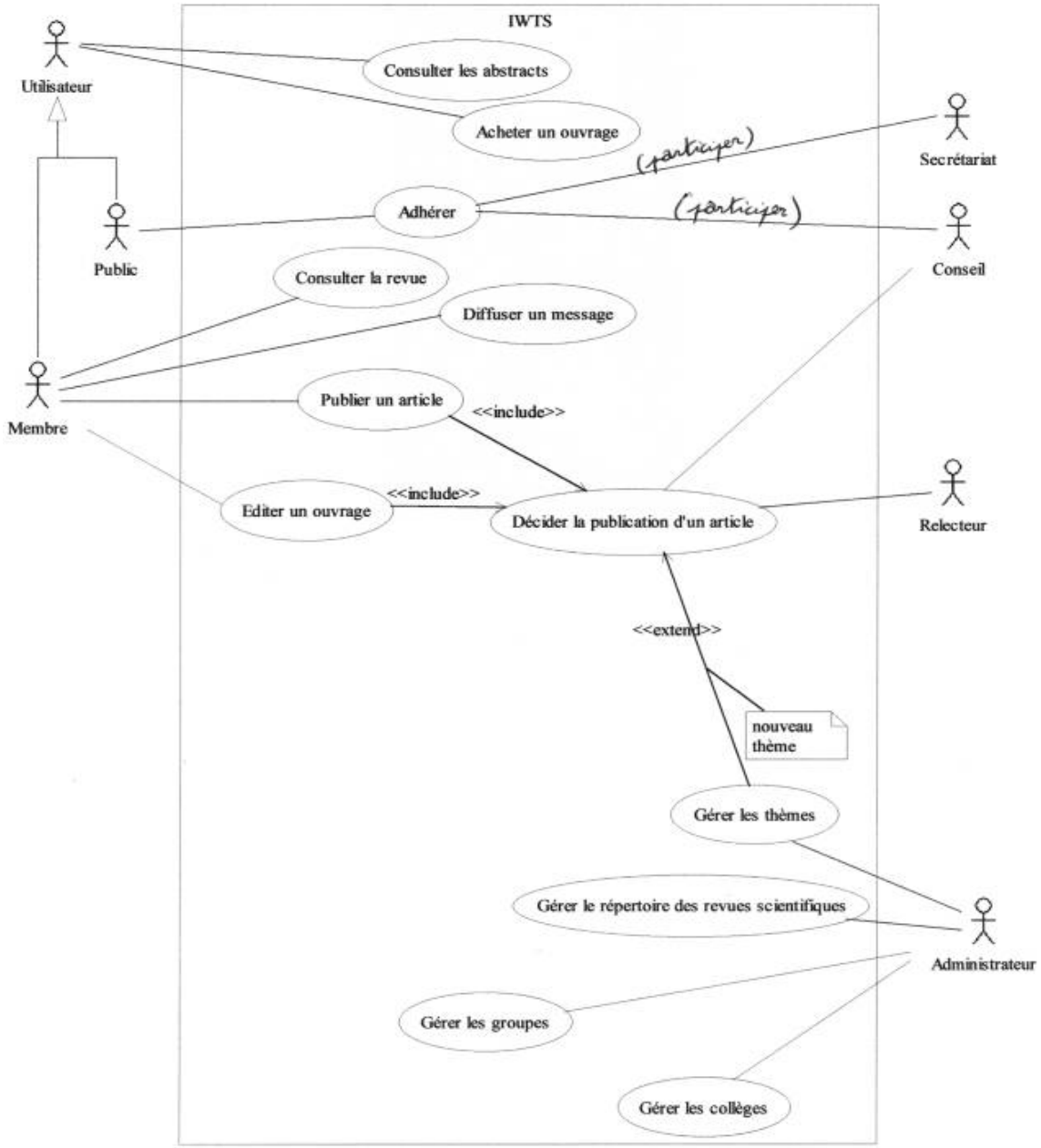
c) La vente des ouvrages:

Les membres de l'association réalisent parfois[1] des ouvrages spécifiques (CDROM de photos par exemple).Ces ouvrages sont édités par l'association à condition qu'ils demeurent scientifiques et non publicitaires (le traitement de l'acceptation d'un ouvrage est le même que celui d'un article sans que ne se pose le problème de délai). Ces ouvrages sont alors mis en vente pour tout public sur le web (une répartition des bénéfices entre auteur et éditeur est alors calculée).

Travail à faire:

1- A partir d'une lecture analytique du texte ci-dessus, produisez une première version du **diagramme des cas d'utilisation du système "site web de l'IWTS"**. Veillez à respecter les frontières du système. *Une des approches possibles consiste à passer au crible les substantifs (ou groupes nominaux) ; ils sont candidats à être des objets du système ou des **acteurs extérieurs**, utilisateurs du système. Les verbes (formes verbales), quand à eux, marquent un aspect dynamique, un lien entre objets, un comportement du système ou un **service** qu'il rend.*

Pour produire le diagramme des cas d'utilisation, on sélectionnera les acteurs et les services.


IWTS
Utilisateur
Consulter les abstracts
Acheter un ouvrage
Secrétariat
(participer)
Adhérer
(participer)
Public
Conseil
Consulter la revue
Diffuser un message
Membre
Publier un article
<<include>>
<<include>>
Editer un ouvrage
Décider la publication d'un article
Relecteur
<<extend>>
nouveau thème
Gérer les thèmes
Gérer le répertoire des revues scientifiques
Administrateur
Gérer les groupes
Gérer les collèges

2- Écrire une **description du cas d'utilisation "Adhérer".**

Cas d'utilisation "Adhérer" ; acteur déclencheur : "Public"; acteurs participants: "Conseil" et "Secrétariat"

Ce cas commence par

La réception d'une demande d'adhésion émanant d'un acteur "Public".

Le serveur envoie un formulaire de demande d'adhésion, explicitant la procédure.

Le demandeur expédie le formulaire rempli (nom & prénom, @dresse électronique), nommant ses deux parrains.

/* Le serveur contrôle l'identité des parrains et veille à la non redondance du nom du demandeur. */

Le serveur expédie dans la boîte (mail) du demandeur une clé valide. Le demandeur renouvelle sa demande d'adhésion.

Le serveur envoie un formulaire d'adhésion en ligne.

/* Le demandeur complète le formulaire avec la clé d'authentification qui lui a été fournie, son adresse personnelle, son diplôme le plus important, son adresse professionnelle, sa profession et sa fonction dans l'organisation dont il fait partie, ses thèmes préférés de recherche et les groupes thématiques auxquels il souhaite appartenir. */

Le demandeur transmet le formulaire et un dossier comportant une photo (magnétique) au format requis, et les références de deux articles dont il est l'auteur parues dans une revue ou publiées sur un site.

Le "Conseil" (un membre autorisé), demande à accéder au dossier Le serveur lui fournit le dossier.

Le "Conseil" propose ou non l'adhésion à un collège.

/* Le serveur enregistre l'avis du conseil. */

Le serveur signale l'avis du conseil au demandeur.

Lorsque l'avis est positif, le demandeur peut confirmer sa demande d'adhésion au collège proposé, signaler son mode de paiement de la cotisation correspondante, et s'il souhaite utiliser la liste de diffusion, en signer la charte.

Le "Secrétariat" enregistre le paiement de la cotisation.

/* Le demandeur devient membre effectif. */

Ce cas se termine par l'enregistrement du paiement de l'adhésion (interaction 13), ou par le signalement au demandeur (interaction n^0 11) du rejet de sa demande par le conseil.

Écrire un **scénario** "Adhésion au collège des laboratoires", instance du cas d'utilisation "Adhérer".

Scénario du cas d'utilisation adhérer: "adhésion au collège des laboratoires"

L'acteur "Public" Pierre Aver (PA), connecté au site, demande à accéder à l'adhésion en ligne.

Le serveur lui envoie un formulaire de demande d'adhésion, explicitant la procédure.

P.A. s'identifie (Pierre Aver, Pierre.Aver@eigsi.fr), nomme ses deux parrains (Lagun Hadj et Hincine R.) et expédie le début du formulaire.

/* Le serveur contrôle l'identité des parrains (0K) et veille à la non redondance du nom du demandeur (0K). */

Le serveur expédie à l'@ Pierre.Aver@eigsi.fr la clé LMU3288. PA renouvelle sa demande d'adhésion

Le serveur lui envoie un formulaire d'adhésion en ligne.

/* PA complète le formulaire avec la clé qui lui a été fournie (LMU3268), son adresse personnelle (18, Rue des Roses Trémières à Dampierre sur Boutonne, France...), son diplôme le plus important (Ph D en biologie), son adresse professionnelle (...), sa profession (ingénieur

d'études) et sa fonction dans l'organisation dont il fait partie (directeur du laboratoire de biologie appliquée), ses thèmes préférés de recherche (l'utilisation des algues dans le processus de lagunage et les groupes thématiques auxquels il souhaite appartenir (déca ntation & traitements biologiques). */

PA transmet le formulaire incluant sa photo (magnétique), et les références de deux articles dont il est l'auteur publiées sur le site des Techniques de l'Ingénieur.

Le serveur émet un message vers la bal du conseil.

Georges Cerbère (GC), membre du conseil, demande à accéder au dossier.

Le serveur lui fournit le dossier.

GC propose l'adhésion au collège des laboratoires. /* Le serveur enregistre l'avis du conseil. */

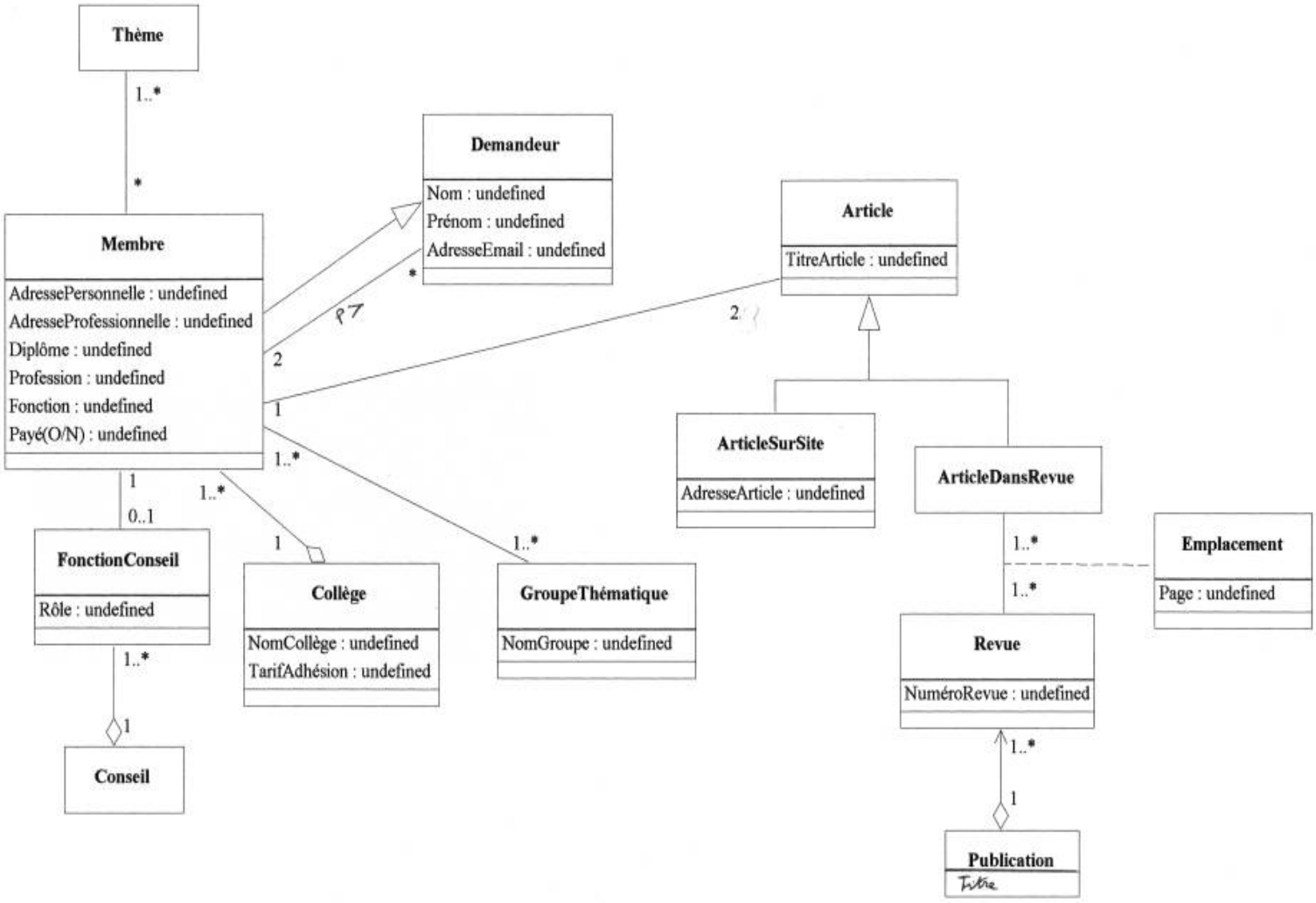
Le serveur signale l'avis du conseil à PA.

PA confirme sa demande d'adhésion au collège des laboratoires, signale son mode de paiement de la cotisation par chèque bancaire, et signe la charte afin d'utiliser les listes de diffusion relatives aux thèmes qu'il a choisi.

Le "Secrétariat" enregistre le paiement de la cotisation. /* PA devient membre effectif */

Travail à faire:

1- En examinant en détail le texte de description du cas d'utilisation « adhérer », produire une première version du diagramme de classe du domaine de cette application.



3- Proposer un diagramme de collaboration du scénario "adhésion au collège des laboratoires"