Conception et mise en œuvre d'un Logiciel

Plan

- Le Génie Logiciel
- Modélisation, cycles de vie et méthodes
- De la programmation structurée à l'approche orientée objet
- UML
- UML : Etudes de cas

Le Génie Logiciel

❖ **Les Logiciels :**

Un logiciel ou une application est un ensemble de programmes, qui permet à un ordinateur ou à un système informatique d'assurer une tâche ou une fonction en particulier (exemple : logiciel de comptabilité, logiciel de gestion des prêts).

Les logiciels, suivant leur taille, peuvent être développés par une personne seule, une petite équipe, ou un ensemble d'équipes coordonnées. Le développement de grands logiciels par de grandes équipes pose d'importants problèmes de conception et de coordination. Or, le développement d'un logiciel est une phase absolument cruciale qui monopolise l'essentiel du coût d'un produit et conditionne sa réussite et sa pérennité.

Le Génie Logiciel

❖ Les Logiciels :

En 1995, une étude du Standish Group dressait un tableau accablant de la conduite des projets informatiques. Reposant sur un échantillon représentatif de 365 entreprises, totalisant 8 380 applications, cette étude établissait que :

- ➢ 16,2% seulement des projets étaient conformes aux prévisions initiales,
- ➢ 52,7% avaient subi des dépassements en coût et délai d'un facteur 2 à 3 avec diminution du nombre des fonctions offertes,
- ➢ 31,1% ont été purement abandonnés durant leur développement.

Pour les grandes entreprises, le taux de succès est de 9% seulement, 37% des projets sont arrêtés en cours de réalisation, 50% aboutissent hors délai et hors budget.

L'examen des causes de succès et d'échec est instructif : la plupart des échecs proviennent non de l'informatique, mais de la maîtrise d'ouvrage (i.e. le client). Pour ces raisons, le développement de logiciels dans un contexte professionnel suit souvent des règles strictes encadrant la conception et permettant le travail en groupe et la maintenance du code. Ainsi, une nouvelle discipline est née : **le génie logiciel**.

Le Génie Logiciel

❖ Le Génie Logiciel :

L'objectif du génie logiciel est d'optimiser le coût de développement du logiciel. L'importance d'une approche méthodologique s'est imposée à la suite de la crise de l'industrie du logiciel à la fin des années 1970. Cette crise de l'industrie du logiciel était principalement due à :

- l'augmentation des coûts ;
- les difficultés de maintenance et d'évolution ;
- la non-fiabilité ;
- le non-respect des spécifications ;
- le non-respect des délais.

Pour apporter une réponse à tous ces problèmes, le génie logiciel s'intéresse particulièrement à la manière dont le code source d'un logiciel est spécifié puis produit. Ainsi, le génie logiciel touche au cycle de vie des logiciels :

- l'analyse du besoin,
- l'élaboration des spécifications,
- la conceptualisation,
- le développement,
- la phase de test,
- la maintenance.

Le Génie Logiciel

❖ Qualité d'un Logiciel :

Validité : aptitude d'un produit logiciel à remplir exactement ses fonctions, définies par le cahier des charges et les spécifications.
Fiabilité ou robustesse : aptitude d'un produit logiciel à fonctionner dans des conditions anormales.
Extensibilité (maintenance) : facilité avec laquelle un logiciel se prête à sa maintenance, c'est-à-dire à une modification ou à une extension des fonctions qui lui sont demandées.
Réutilisabilité : aptitude d'un logiciel à être réutilisé, en tout ou en partie, dans de nouvelles applications.
Compatibilité :facilité avec laquelle un logiciel peut être combiné avec d'autres logiciels.
Efficacité : Utilisation optimale des ressources matérielles.
Portabilité :facilité avec laquelle un logiciel peut être transféré sous différents environnements matériels et logiciels.
Vérifiabilité : facilité de préparation des procédures de test.
Intégrité : aptitude d'un logiciel à protéger son code et ses données contre des accès non autorisés.
Facilité d'emploi : facilité d'apprentissage, d'utilisation, de préparation des données, d'interprétation des erreurs et de rattrapage en cas d'erreur d'utilisation.

Modélisation, cycles de vie et méthodes

❖ **Qu'est-ce qu'un modèle ? :**

Un modèle est une représentation abstraite et simplifiée (i.e. qui exclut certains détails), d'une entité (phénomène, processus, système, etc.) du monde réel en vue de le décrire, de l'expliquer ou de le prévoir. Modèle est synonyme de théorie, mais avec une connotation pratique : un modèle, c'est une théorie orientée vers l'action qu'elle doit servir.

Concrètement, un modèle permet de réduire la complexité d'un phénomène en éliminant les détails qui n'influencent pas son comportement de manière significative. Il reflète ce que le concepteur croit important pour la compréhension et la prédiction du phénomène modélisé. Les limites du phénomène modélisé dépendant des objectifs du modèle.

Modélisation, cycles de vie et méthodes

❖ **Pourquoi modéliser ? :**

Modéliser un système avant sa réalisation permet de mieux comprendre le fonctionnement du système. C'est également un bon moyen de maîtriser sa complexité et d'assurer sa cohérence. Un modèle est un langage commun, précis, qui est connu par tous les membres de l'équipe et il est donc, à ce titre, un vecteur privilégié pour communiquer. Cette communication est essentielle pour aboutir à une compréhension commune aux différentes parties prenantes (notamment entre la maîtrise d'ouvrage et la maîtrise d'œuvre informatique) et précise d'un problème donné.

Dans le domaine de l'ingénierie du logiciel, le modèle permet de mieux répartir les tâches et d'automatiser certaines d'entre elles. C'est également un facteur de réduction des coûts et des délais. Par exemple, les plateformes de modélisation savent maintenant exploiter les modèles pour faire de la génération de code (au moins au niveau du squelette) voire des allers-retours entre le code et le modèle sans perte d'information. Le modèle est enfin indispensable pour assurer un bon niveau de qualité et une maintenance efficace. En effet, une fois mise en production, l'application va devoir être maintenue, probablement par une autre équipe et, qui plus est, pas nécessairement de la même société que celle ayant créé l'application.

Le choix du modèle a donc une influence capitale sur les solutions obtenues. Les systèmes non triviaux sont mieux modélisés par un ensemble de modèles indépendants. Selon les modèles employés, la démarche de modélisation n'est pas la même.

Modélisation, cycles de vie et méthodes

❖ **Qui doit modéliser ? :**

La modélisation est souvent faite par la maîtrise d'œuvre informatique (MOE).

Il est préférable que la modélisation soit réalisée par la maîtrise d'ouvrage (MOA) de sorte que le métier soit maître de ses propres concepts. La MOE doit intervenir dans le modèle lorsque, après avoir défini les concepts du métier, on doit introduire les contraintes propres à la plateforme informatique.

Il est vrai que certains métiers, dont les priorités sont opérationnelles, ne disposent pas toujours de la capacité d'abstraction et de la rigueur conceptuelle nécessaires à la formalisation. La professionnalisation de la MOA a pour but de les doter de ces compétences. Cette professionnalisation réside essentiellement dans l'aptitude à modéliser le système d'information du métier : le maître mot est **modélisation**. Lorsque le modèle du système d'information est de bonne qualité, sobre, clair, stable, la maîtrise d'œuvre peut travailler dans de bonnes conditions. Lorsque cette professionnalisation a lieu, elle modifie les rapports avec l'informatique et déplace la frontière des responsabilités, ce qui contrarie parfois les informaticiens dans un premier temps, avant qu'ils n'en voient apparaître les bénéfices.

Modélisation, cycles de vie et méthodes

❖ **Maîtrise d'ouvrage et maîtrise d'œuvre ? :**

Maître d'ouvrage (MOA) : Le MOA est une personne morale (entreprise, direction, etc.), une entité de l'organisation. Ce n'est jamais une personne.

Maître d'œuvre (MOE) : Le MOE est une personne morale (entreprise, direction, etc.) garante de la bonne réalisation technique des solutions. Il a, lors de la conception du SI, un devoir de conseil vis-à-vis du MOA, car le SI doit tirer le meilleur parti des possibilités techniques.

Le MOA est client du MOE à qui il passe commande d'un produit nécessaire à son activité. Le MOE fournit ce produit ; soit il le réalise lui-même, soit il passe commande à un ou plusieurs fournisseurs (« entreprises ») qui élaborent le produit sous sa direction.

La relation MOA et MOE est définie par un contrat qui précise leurs engagements mutuels. Lorsque le produit est compliqué, il peut être nécessaire de faire appel à plusieurs fournisseurs.

Le MOE assure leur coordination ; il veille à la cohérence des fournitures et à leur compatibilité. Il coordonne l'action des fournisseurs en contrôlant la qualité technique, en assurant le respect des délais fixés par le MOA et en minimisant les risques.

Le MOE est responsable de la qualité technique de la solution. Il doit, avant toute livraison au MOA, procéder aux vérifications nécessaires.

Modélisation, cycles de vie et méthodes

❖ Le cycle de vie d'un logiciel :

Le cycle de vie d'un logiciel (en anglais software lifecycle), désigne toutes les étapes du développement d'un logiciel, de sa conception à sa disparition. L'objectif d'un tel découpage est de permettre de définir des jalons intermédiaires permettant la validation du développement logiciel, c'est-à-dire la conformité du logiciel avec les besoins exprimés, et la vérification du processus de développement, c'est-à-dire l'adéquation des méthodes mises en œuvre.

L'origine de ce découpage provient du constat que les erreurs ont un coût d'autant plus élevé qu'elles sont détectées tardivement dans le processus de réalisation. Le cycle de vie permet de détecter les erreurs au plus tôt et ainsi de maîtriser la qualité du logiciel, les délais de sa réalisation et les coûts associés.

Modélisation, cycles de vie et méthodes

❖ Le cycle de vie d'un logiciel :

Le cycle de vie du logiciel comprend généralement au minimum les étapes suivantes :

Analyse des besoins et faisabilité : c'est-à-dire l'expression, le recueil et la formalisation des besoins du demandeur (le client) et de l'ensemble des contraintes, puis l'estimation de la faisabilité de ces besoins ;

Spécifications ou conception générale : il s'agit de l'élaboration des spécifications de l'architecture générale du logiciel ;

Conception détaillée : cette étape consiste à définir précisément chaque sous-ensemble du logiciel ;

Codage (Implémentation ou programmation) : c'est la traduction dans un langage de programmation des fonctionnalités définies lors de phases de conception ;

Tests unitaires : ils permettent de vérifier individuellement que chaque sous-ensemble du logiciel est implémenté conformément aux spécifications ;

Intégration : l'objectif est de s'assurer de l'interfaçage des différents éléments (modules) du logiciel. Elle fait l'objet de tests d'intégration consignés dans un document ;

Modélisation, cycles de vie et méthodes

❖ **Le cycle de vie d'un logiciel :**

Qualification (ou recette) : c'est-à-dire la vérification de la conformité du logiciel aux spécifications initiales ;

Documentation : elle vise à produire les informations nécessaires pour l'utilisation du logiciel et pour des développements ultérieurs ;

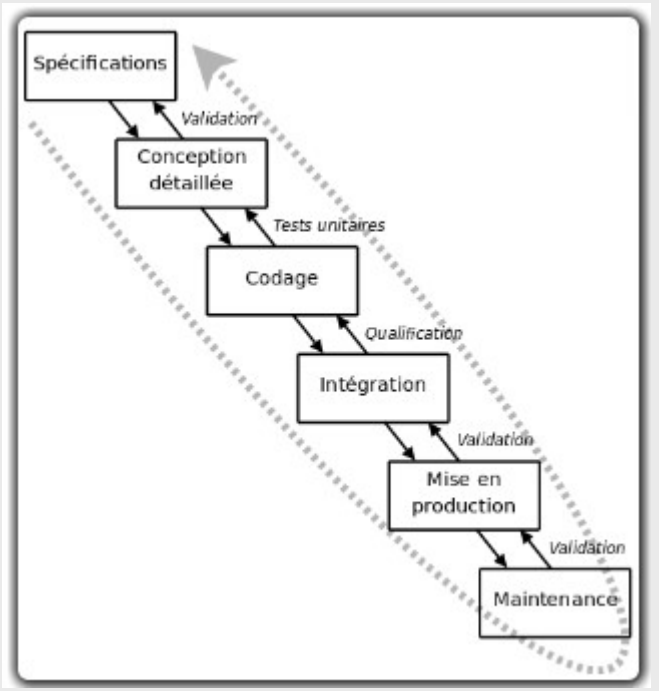
Mise en production : c'est le déploiement sur site du logiciel ;

Maintenance : elle comprend toutes les actions correctives (maintenance corrective) et évolutives (maintenance évolutive) sur le logiciel.

La séquence et la présence de chacune de ces activités dans le cycle de vie dépendent du choix d'un modèle de cycle de vie entre le client et l'équipe de développement. Le cycle de vie permet de prendre en compte, en plus des aspects techniques, l'organisation et les aspects humains.

Modélisation, cycles de vie et méthodes

❖ Modèle de cycle de vie en cascade :



Le modèle de cycle de vie en cascade a été mis au point dès 1966, puis formalisé aux alentours de 1970.

Dans ce modèle le principe est très simple : chaque phase se termine à une date précise par la production de certains documents ou logiciels.

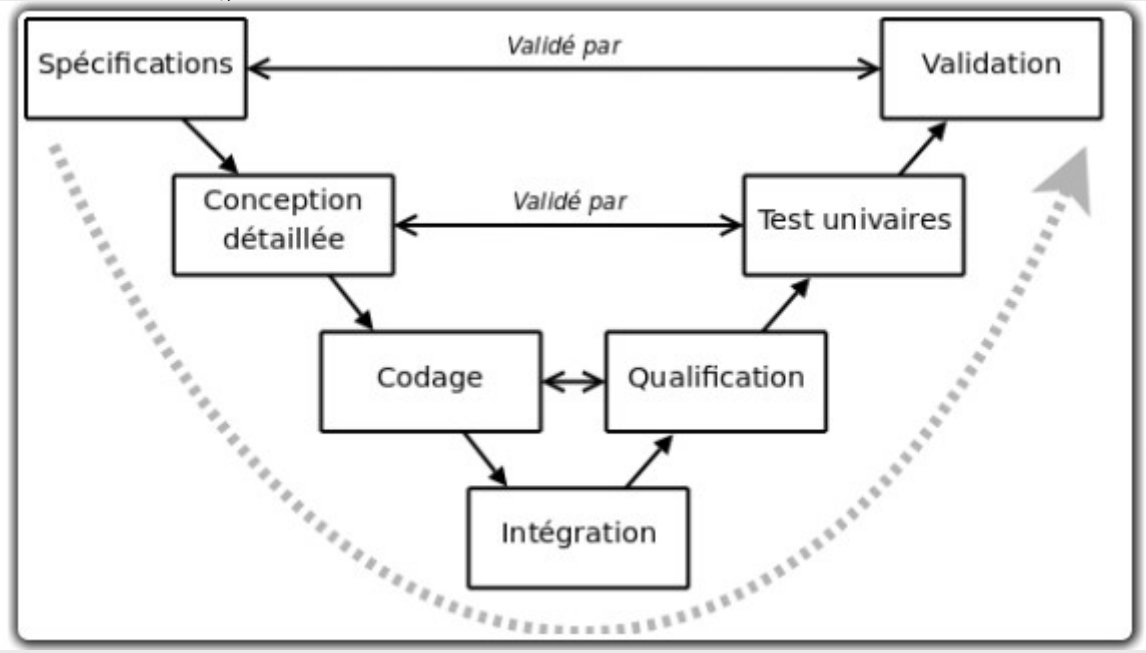
Les résultats sont définis sur la base des interactions entre étapes, ils sont soumis à une revue approfondie et on ne passe à la phase suivante que s'ils sont jugés satisfaisants.

Le modèle original ne comportait pas de possibilité de retour en arrière. Celle-ci a été rajoutée ultérieurement sur la base qu'une étape ne remet en cause que l'étape précédente, ce qui, dans la pratique, s'avère insuffisant.

L'inconvénient majeur du modèle de cycle de vie en cascade est que la vérification du bon fonctionnement du système est réalisée trop tardivement : lors de la phase d'intégration, ou pire, lors de la mise en ^production.

Modélisation, cycles de vie et méthodes

❖ **Modèle de cycle de vie en V :**



Le modèle en V demeure actuellement le cycle de vie le plus connu et certainement le plus utilisé. Il s'agit d'un modèle en cascade dans lequel le développement des tests et du logiciel sont effectués de manière synchrone.

Le principe de ce modèle est qu'avec toute décomposition doit être décrite la recomposition et que toute description d'un composant est accompagnée de tests qui permettront de s'assurer qu'il correspond à sa description.
Ceci rend explicite la préparation des dernières phases (validation-vérification) par les premières (construction du logiciel), et permet ainsi d'éviter un écueil bien connu de la spécification du logiciel : énoncer une propriété qu'il est impossible de vérifier objectivement après la réalisation.
Cependant, ce modèle souffre toujours du problème de la vérification trop tardive du bon fonctionnement du système.

Modélisation, cycles de vie et méthodes

❖ Modèle de cycle de vie en spirale :

Proposé en 1988, ce modèle est beaucoup plus général que le précédent. Il met l'accent sur l'activité d'analyse des risques : chaque cycle de la spirale se déroule en quatre phases :

1. Détermination, à partir des résultats des cycles précédents, ou de l'analyse préliminaire des besoins, des objectifs du cycle, des alternatives pour les atteindre et des contraintes ;
2. Analyse des risques, évaluation des alternatives et, éventuellement maquettage ;
3. Développement et vérification de la solution retenue, un modèle « classique » (cascade ou en V) peut être utilisé ici ;
4. Revue des résultats et vérification du cycle suivant.

L'analyse préliminaire est affinée au cours des premiers cycles. Le modèle utilise des maquettes exploratoires pour guider la phase de conception du cycle suivant. Le dernier cycle se termine par un processus de développement classique.

Modélisation, cycles de vie et méthodes

❖ Modèle par incrément :

Dans les modèles précédents, un logiciel est décomposé en composants développés séparément et intégrés à la fin du processus.

Dans les modèles par incrément un seul ensemble de composants est développé à la fois : des incréments viennent s'intégrer à un noyau de logiciel développé au préalable. Chaque incrément est développé selon l'un des modèles précédents.

Les avantages de ce type de modèle sont les suivants :

- chaque développement est moins complexe ;
- les intégrations sont progressives ;
- il est ainsi possible de livrer et de mettre en service chaque incrément ;
- il permet un meilleur lissage du temps et de l'effort de développement grâce à la possibilité de recouvrement (parallélisation) des différentes phases.

Les risques de ce type de modèle sont les suivants :

- remettre en cause les incréments précédents ou pire le noyau ;
- ne pas pouvoir intégrer de nouveaux incréments.
- Les noyaux, les incréments ainsi que leurs interactions doivent donc être spécifiés globalement, au début du projet.

Les incréments doivent être aussi indépendants que possible, fonctionnellement, mais aussi sur le plan du calendrier du développement.

Modélisation, cycles de vie et méthodes

❖ Méthodes d'analyse et de conception :

Les méthodes d'analyse et de conception fournissent une méthodologie et des notations standards qui aident à concevoir des logiciels de qualité. Il existe différentes manières pour classer ces méthodes, dont :

✓ **La distinction entre composition et décomposition :**

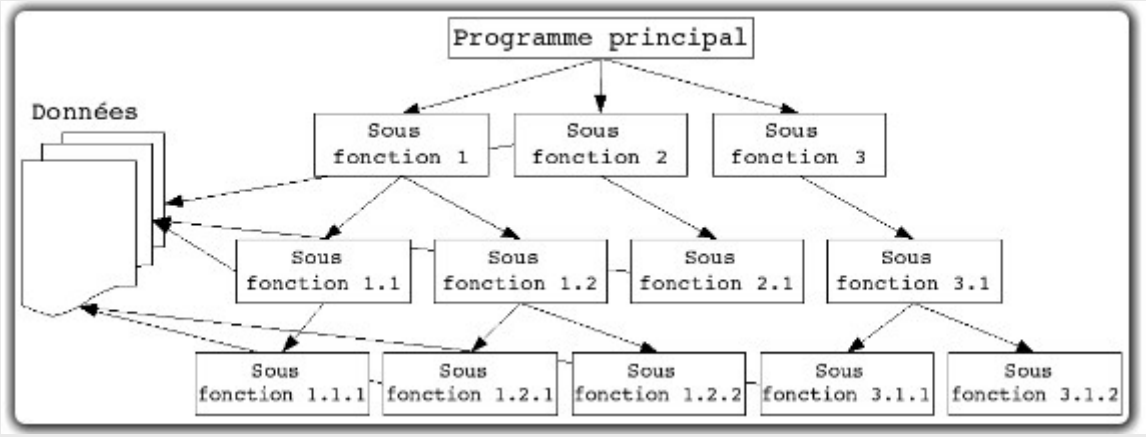
Elle met en opposition d'une part les méthodes ascendantes qui consistent à construire un logiciel par composition à partir de modules existants et, d'autre part, les méthodes descendantes qui décomposent récursivement le système jusqu'à arriver à des modules programmables simplement ;

✓ **La distinction entre fonctionnelle (dirigée par le traitement) et orientée objet :**

Dans la stratégie fonctionnelle (également qualifiée de structurée) un système est vu comme un ensemble hiérarchique d'unités en interaction, ayant chacune une fonction clairement définie. Les fonctions disposent d'un état local, mais le système a un état partagé, qui est centralisé et accessible par l'ensemble des fonctions. Les stratégies orientées objet considèrent qu'un système est un ensemble d'objets interagissant. Chaque objet dispose d'un ensemble d'attributs décrivant son état et l'état du système est décrit (de façon décentralisée) par l'état de l'ensemble.

De la programmation structurée à l'approche orientée objet

❖ Méthodes fonctionnelles ou structurées :



Les méthodes fonctionnelles (également qualifiées de méthodes structurées) trouvent leur origine dans les langages procéduraux. Elles mettent en évidence les fonctions à assurer et proposent une approche hiérarchique descendante et modulaire.

Ces méthodes utilisent intensivement les raffinements successifs pour produire des spécifications dont l'essentiel est sous forme de notation graphique en diagrammes de flots de données. Le plus haut niveau représente l'ensemble du problème (sous forme d'activité, de données ou de processus, selon la méthode). Chaque niveau est ensuite décomposé en respectant les entrées/sorties du niveau supérieur. La décomposition se poursuit jusqu'à arriver à des composants maîtrisables.

En résumé, l'architecture du système est dictée par la réponse au problème (i.e. **la fonction du système).**

De la programmation structurée à l'approche orientée objet

❖ L'approche orientée objet :

L'approche considère le logiciel comme **une collection d'objets** dissociés, identifiés et possédant des caractéristiques. Une caractéristique est soit un attribut (i.e. une donnée caractérisant l'état de l'objet), soit une entité comportementale de l'objet (i.e. une fonction). La fonctionnalité du logiciel émerge alors de l'interaction entre les différents objets qui le constituent. L'une des particularités de cette approche est qu'elle rapproche les données et leurs traitements associés au sein d'un unique objet.
Comme nous venons de le dire, un objet est caractérisé par plusieurs notions :

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, etc.) ;

Les attributs
il s'agit des données caractérisant l'objet. Ce sont des variables stockant des informations sur l'état de l'objet ;

Les méthodes
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.

En résumé, l'architecture du système est dictée par la structure du problème.

De la programmation structurée à l'approche orientée objet

❖ Concepts importants de l'approche objet :

Notion de classe :

Une classe est un type de données abstrait qui précise des caractéristiques (attributs et méthodes) communes à toute une famille d'objets et qui permet de créer (instancier) des objets possédant ces caractéristiques. Les autres concepts importants qu'il nous faut maintenant introduire sont l'encapsulation, l'héritage et l'agrégation.

Encapsulation

L'encapsulation consiste à masquer les détails d'implémentation d'un objet, en définissant une interface. L'interface est la vue externe d'un objet, elle définit les services accessibles (offerts) aux utilisateurs de l'objet.

L'encapsulation facilite l'évolution d'une application, car elle stabilise l'utilisation des objets : on peut modifier l'implémentation des attributs d'un objet sans modifier son interface, et donc la façon dont l'objet est utilisé.

L'encapsulation garantit l'intégrité des données, car elle permet d'interdire, ou de restreindre, l'accès direct aux attributs des objets.

De la programmation structurée à l'approche orientée objet

❖ Concepts importants de l'approche objet :

Héritage, spécialisation, généralisation et polymorphisme

L'héritage est un mécanisme de transmission des caractéristiques d'une classe (ses attributs et méthodes) vers une sous-classe. Une classe peut être spécialisée en d'autres classes, afin d'y ajouter des caractéristiques spécifiques ou d'en adapter certaines. Plusieurs classes peuvent être généralisées en une classe qui les factorise, afin de regrouper les caractéristiques communes d'un ensemble de classes.

Ainsi, la spécialisation et la généralisation permettent de construire des hiérarchies de classes. L'héritage peut être simple ou multiple. L'héritage évite la duplication et encourage la réutilisation.

Le polymorphisme représente la faculté d'une méthode à pouvoir s'appliquer à des objets de classes différentes. Le polymorphisme augmente la généricité, et donc la qualité du code.

Agrégation

Il s'agit d'une relation entre deux classes, spécifiant que les objets d'une classe sont des composants de l'autre classe. Une relation d'agrégation permet donc de définir des objets composés d'autres objets. L'agrégation permet donc d'assembler des objets de base, afin de construire des objets plus complexes.

De la programmation structurée à l'approche orientée objet

❖ Historique de la programmation par objets :

Les premiers langages de programmation qui ont utilisé des objets sont Simula I (1961-64) et Simula 67 (1967), conçus par les informaticiens norvégiens Ole-Johan Dahl et Kristan Nygaard. Simula 67 contenait déjà les objets, les classes, l'héritage, l'encapsulation, etc.

Alan Kay, du PARC de Xerox, avait utilisé Simula dans les années 1960. Il réalisa en 1976 Smalltalk qui reste, aux yeux de certains programmeurs, le meilleur langage de programmation par objets.

Bjarne Stroustrup a mis au point C++, une extension du langage C permettant la programmation orientée objet, aux Bell Labs d'AT&T en 1982. C++ deviendra le langage le plus utilisé par les programmeurs professionnels. Il arrivera à maturité en 1986, sa standardisation ANSI / ISO date de 1997.

Java est lancé par Sun en 1995. Comme il présente plus de sécurité que C++, il deviendra le langage favori de certains programmeurs professionnels.

Plan

- Le Génie Logiciel
- Modélisation, cycles de vie et méthodes
- De la programmation structurée à l'approche orientée objet
- UML
- UML : Etudes de cas

❖ Introduction :

La description de la programmation par objets a fait ressortir l'étendue du travail conceptuel nécessaire : définition des classes, de leurs relations, des attributs et méthodes, des interfaces, etc.

Pour programmer une application, il ne convient pas de se lancer tête baissée dans l'écriture du code : il faut d'abord organiser ses idées, les documenter, puis organiser la réalisation en définissant les modules et étapes de la réalisation. C'est cette démarche antérieure à l'écriture que l'on appelle modélisation ; son produit est un modèle.

Les spécifications fournies par la maîtrise d'ouvrage en programmation impérative étaient souvent floues : les articulations conceptuelles (structures de données, algorithmes de traitement) s'exprimant dans le vocabulaire de l'informatique, le modèle devait souvent être élaboré par celle-ci. L'approche objet permet en principe à la maîtrise d'ouvrage de s'exprimer de façon précise selon un vocabulaire qui, tout en transcrivant les besoins du métier, pourra être immédiatement compris par les informaticiens. En principe seulement, car la modélisation demande aux maîtrises d'ouvrage une compétence et un professionnalisme qui ne sont pas aujourd'hui répandus.

UML

❖ Histoire des modélisations par objets :

Les méthodes utilisées dans les années 1980 pour organiser la programmation impérative (notamment Merise) étaient fondées sur la modélisation séparée des données et des traitements. Lorsque la programmation par objets prend de l'importance au début des années 1990, la nécessité d'une méthode qui lui soit adaptée devient évidente.
Plus de cinquante méthodes apparaissent entre 1990 et 1995 (Booch, Classe-Relation, Fusion, HOOD, OMT, OOA, OOD, OOM, OOSE, etc.), mais aucune ne parvient à s'imposer.

En 1994, le consensus se fait autour de trois méthodes :

- OMT de James Rumbaugh (General Electric) fournit une représentation graphique des aspects statique, dynamique et fonctionnel d'un système ;
- OOD de Grady Booch, définie pour le Department of Defense, introduit le concept de paquetage (package) ;
- OOSE d'Ivar Jacobson (Ericsson) fonde l'analyse sur la description des besoins des utilisateurs (cas d'utilisation, ou use cases).

➔ UML (Unified Modeling Language) est né de cet effort de convergence. L'adjectif unified est là pour marquer qu'UML unifie, et donc remplace.

➔ En fait, et comme son nom l'indique, UML n'a pas l'ambition d'être exactement une méthode : c'est un langage.

UML

❖ Histoire des modélisations par objets :

L'unification a progressé par étapes. En 1995, Booch et Rumbaugh (et quelques autres) se sont mis d'accord pour construire une méthode unifiée, Unified Method 0.8 ; en 1996, Jacobson les a rejoints pour produire UML 0.9 (notez le remplacement du mot méthode par le mot langage, plus modeste).

Les acteurs les plus importants dans le monde du logiciel s'associent alors à l'effort (IBM, Microsoft, Oracle, DEC, HP, Rational, Unisys, etc.) et UML 1.0 est soumis à l'OMG. L'OMG adopte en novembre 1997 UML 1.1 comme langage de modélisation des systèmes d'information à objets. La version 2.1.1 d'UML a été créé en 2008 et les travaux d'amélioration se poursuivent.

UML est donc non seulement un outil intéressant, mais une norme qui s'impose en technologie à objets et à laquelle se sont rangés tous les grands acteurs du domaine, acteurs qui ont d'ailleurs contribué à son élaboration.

❖ UML en œuvre :

UML n'est pas une méthode (i.e. une description normative des étapes de la modélisation) : ses auteurs ont en effet estimé qu'il n'était pas opportun de définir une méthode en raison de la diversité des cas particuliers. Ils ont préféré se borner à définir un langage graphique qui permet de représenter et de communiquer les divers aspects d'un système d'information. Aux graphiques sont bien sûr associés des textes qui expliquent leur contenu.

UML est donc un métalangage, car il fournit les éléments permettant de construire le modèle qui, lui, sera le langage du projet.

Il est impossible de donner une représentation graphique complète d'un logiciel, ou de tout autre système complexe, de même qu'il est impossible de représenter entièrement une statue (à trois dimensions) par des photographies (à deux dimensions). Mais il est possible de donner sur un tel système **des vues partielles**, analogues chacune à une photographie d'une statue, et dont la conjonction donnera une idée utilisable en pratique sans risque d'erreur grave.

UML 2.0 comporte ainsi treize types de diagrammes représentant autant de vues distinctes pour représenter des concepts particuliers du système d'information. Ils se répartissent en deux grands groupes.

UML

❖ UML en œuvre :

Diagrammes structurels ou diagrammes statiques (UML Structure)

- ✓ diagramme de classes (Class diagram)
- ✓ diagramme d'objets (Object diagram)
- ✓ diagramme de composants (Component diagram)
- ✓ diagramme de déploiement (Deployment diagram)
- ✓ diagramme de paquetages (Package diagram)
- ✓ diagramme de structures composites (Composite structure diagram)

Diagrammes comportementaux ou diagrammes dynamiques (UML Behavior)

- ✓ diagramme de cas d'utilisation (Use case diagram)
- ✓ diagramme d'activités (Activity diagram)
- ✓ diagramme d'états-transitions (State machine diagram)
- ✓ Diagrammes d'interaction (Interaction diagram)
  - ▪ diagramme de séquence (Sequence diagram)
  - ▪ diagramme de communication (Communication diagram)
  - ▪ diagramme global d'interaction (Interaction overview diagram)
  - ▪ diagramme de temps (Timing diagram)

Ces diagrammes, d'une utilité variable selon les cas, **ne sont pas nécessairement tous produits** à l'occasion d'une modélisation. Les plus utiles pour la maîtrise d'ouvrage sont **les diagrammes d'activités, de cas d'utilisation, de classes, d'objets, de séquence et d'états-transitions.** Les diagrammes de composants, de déploiement et de communication sont surtout utiles pour la maîtrise d'œuvre à qui ils permettent de formaliser les contraintes de la réalisation et la solution technique.

UML :

DIAGRAMME DE CAS d'UTILISATION

DIAGRAMME DE CAS d'UTILISATION

GENERALITES :

- Le système existe pour servir ses utilisateurs
- Cas d'utilisation = Use cases
- Idée : description du comportement du système du point de vue de son utilisateur (facilite l'expression des besoins)

- Comportement = {Actions} + {Réactions}

DIAGRAMME DE CAS d'UTILISATION

GENERALITES :

- On part d'un scénario *(ex : un client achète un objet et paie sur internet)*
- Mais il peut y avoir des scénarios liés *ex*
  - *échec lors du paiement*
  - *Il s'agit d'un client régulier*
- Mais ces scénarios ont le même but : acheter un objet

⇨ Un cas d'utilisation est un ensemble de scénarios liés ensemble par un but commun d'un utilisateur.

- Acteur = entité externe qui agit sur le système

DIAGRAMME DE CAS d'UTILISATION

REPRESENTATION :

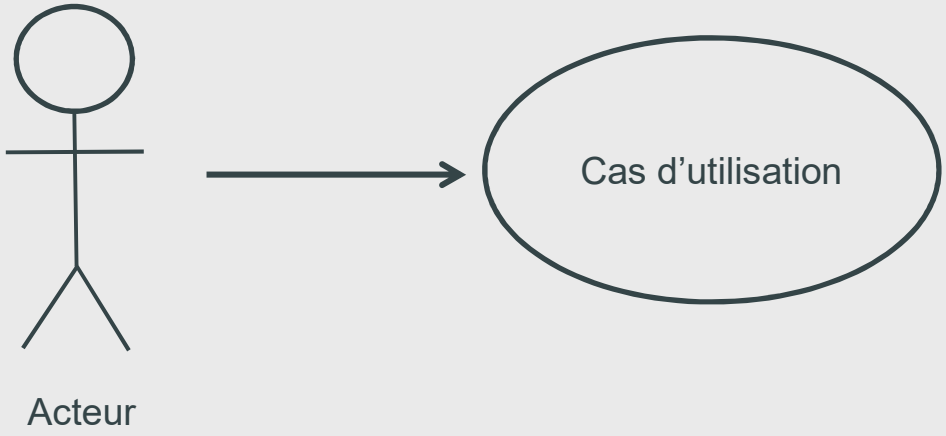


DIAGRAMME DE CAS d'UTILISATION

ACTEURS vs UTILISATEURS :

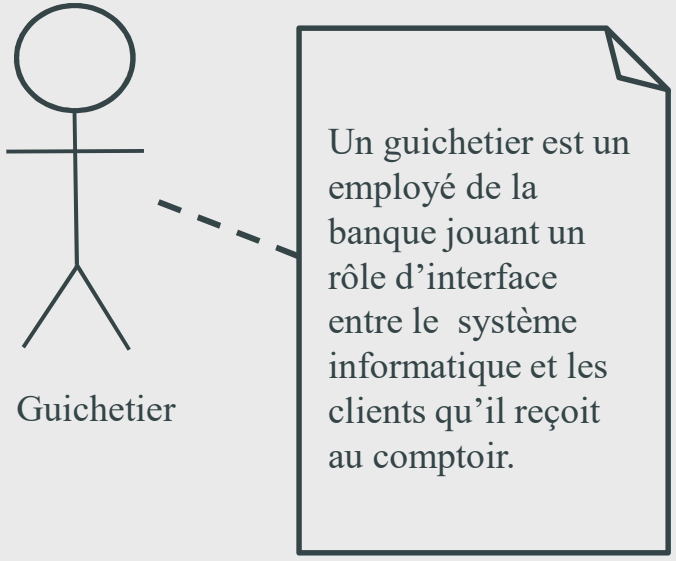
Ne pas confondre acteur et personne utilisant le système :

- Une même personne peut jouer plusieurs rôles
- Plusieurs personnes peuvent jouer un même rôle
- Un acteur n'est pas forcément une personne physique.

Types d'acteurs :

- Utilisateurs principaux
- Utilisateurs secondaires
- Périphériques externes
- Systèmes externes

DIAGRAMME DE CAS d'UTILISATION



DEFINITION DES ACTEURS :

Pour chaque acteur :

- choix d'un identificateur
- brève description (facultatif)

- Acteur principaux : utilisent le système
- Acteur secondaires : administrent le système

DIAGRAMME DE CAS d'UTILISATION

CAS D'UTILISATION : DEFINITIONS

Ensemble des actions réalisées par le système en réponse à une action d'un acteur

Les cas d'utilisation ne doivent pas se chevaucher

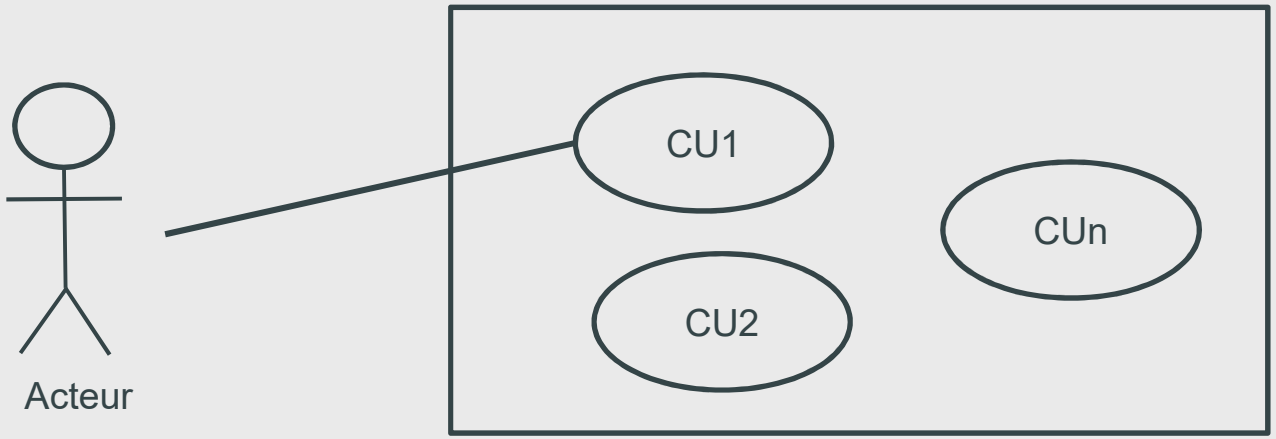


DIAGRAMME DE CAS d'UTILISATION

CAS D'UTILISATION

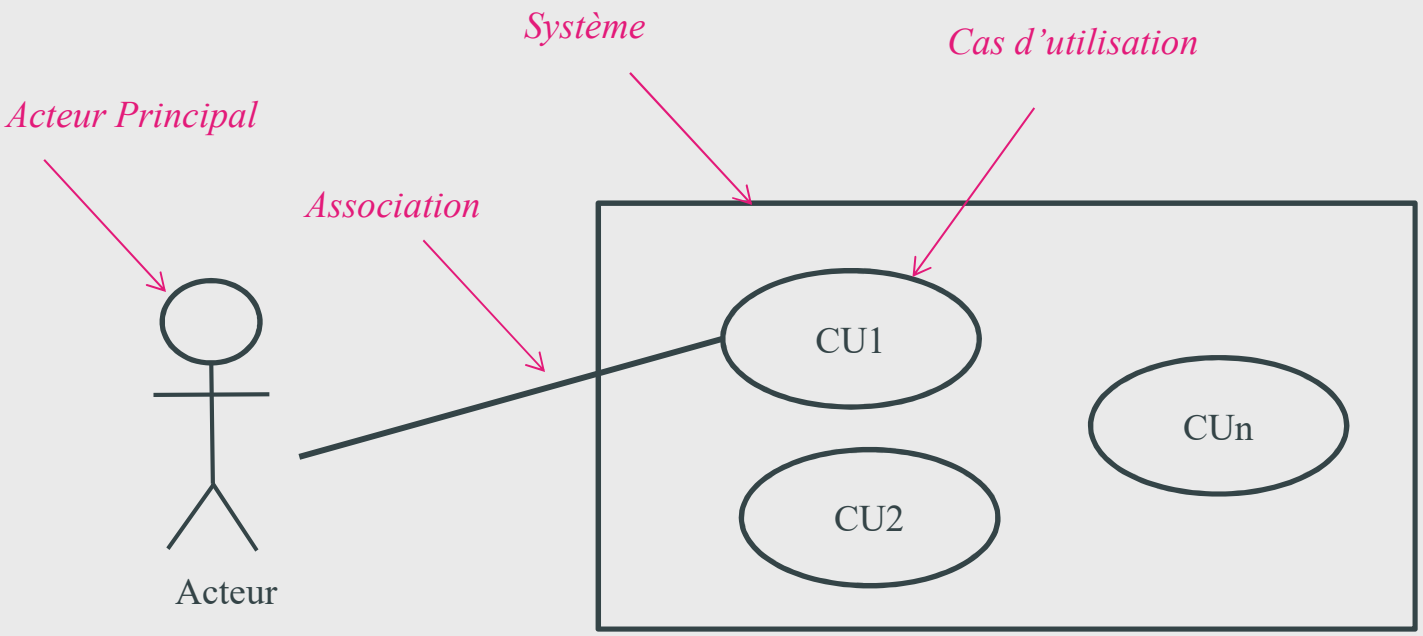


DIAGRAMME DE CAS d'UTILISATION

EXEMPLE

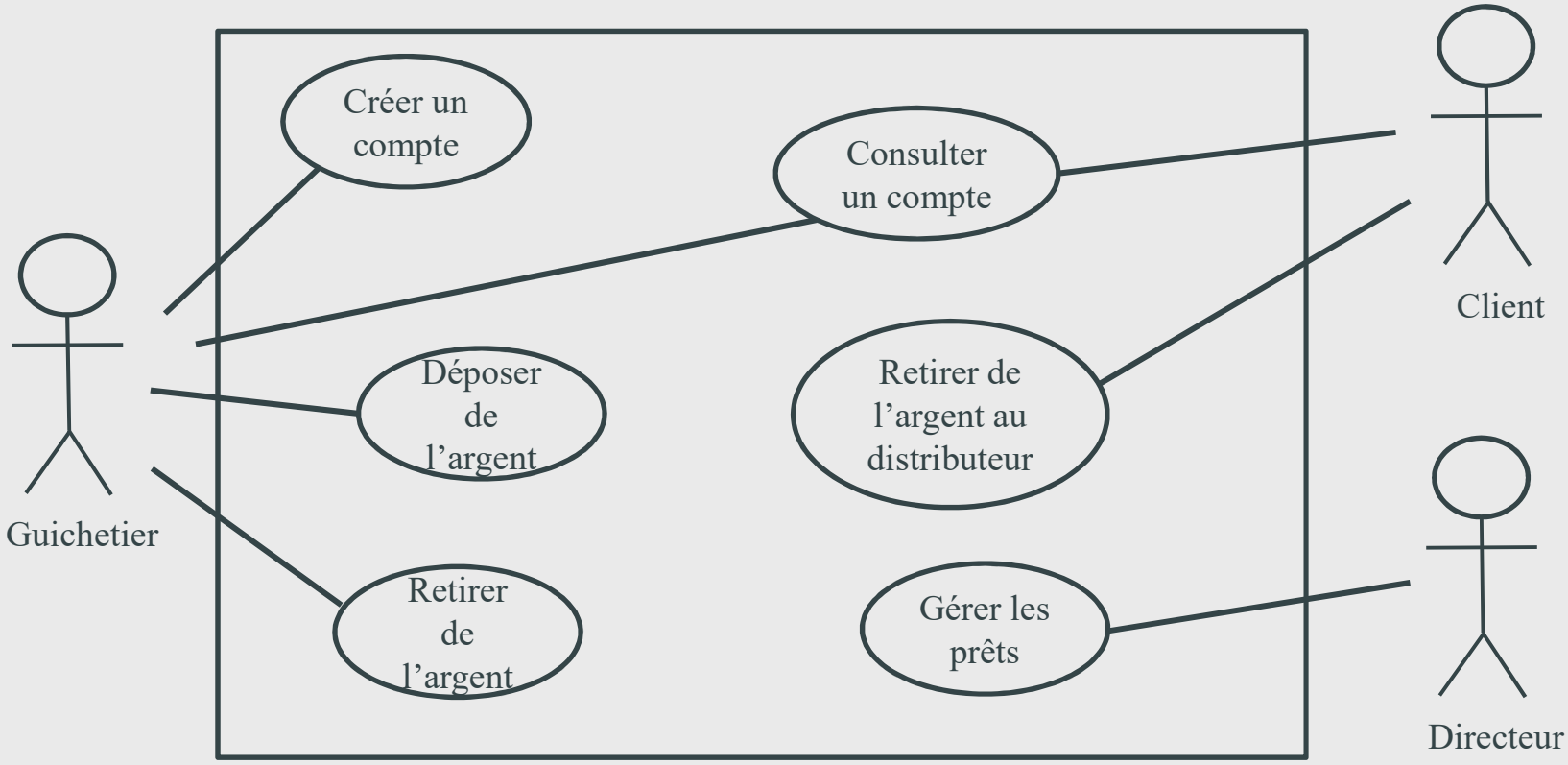


DIAGRAMME DE CAS d'UTILISATION

RELATIONS ENTRE CAS D'UTILISATIONS :

- Généralisation *(generalize)*
- Inclusion *(include)*
- Extension *(extend)*

DIAGRAMME DE CAS d'UTILISATION

RELATION GENERALIZE

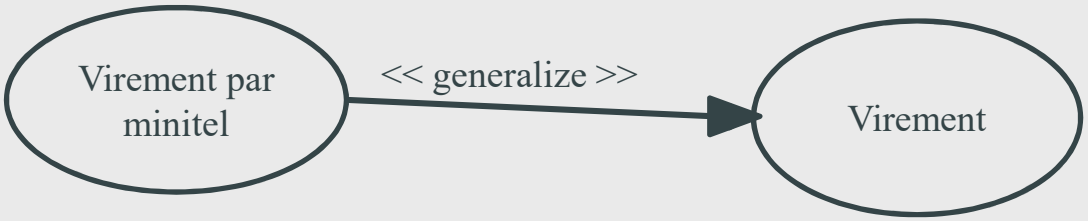


DIAGRAMME DE CAS d'UTILISATION

RELATION INCLUDE :

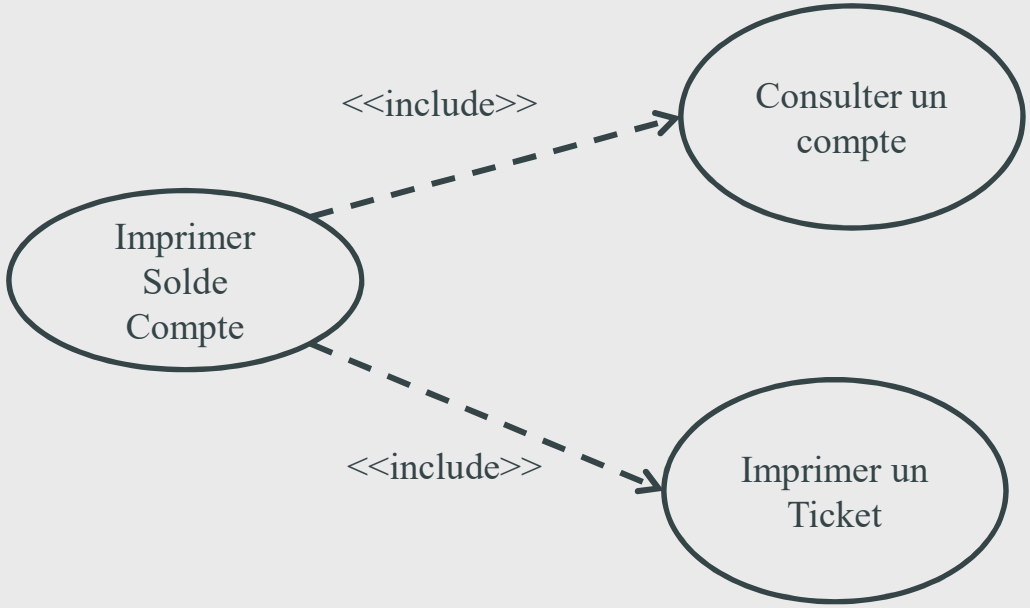


DIAGRAMME DE CAS d'UTILISATION

RELATION EXTEND :

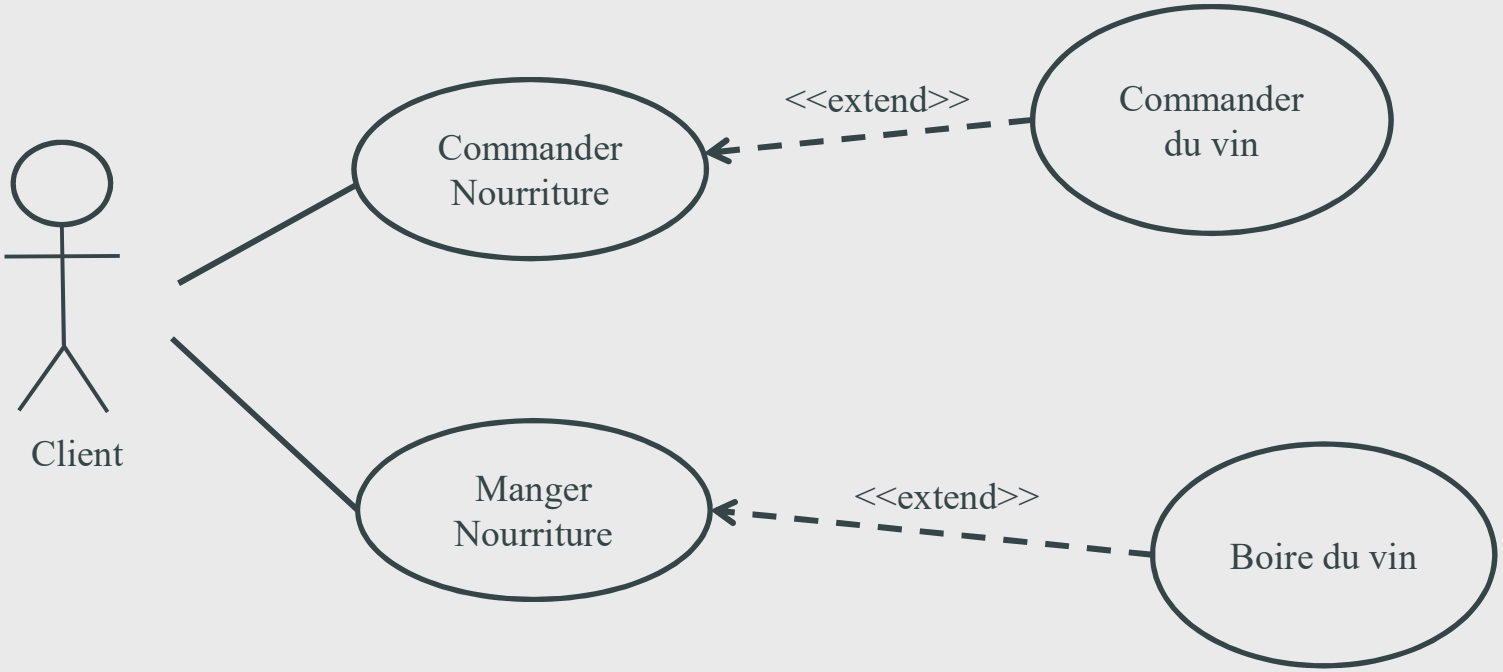


DIAGRAMME DE CAS d'UTILISATION

EN RESUME :

- Système = ensemble de cas d'utilisation
- Le système possède les cas d'utilisation mais pas les acteurs

- Un cas d'utilisation = ensemble de « chemins d'exécution » possibles

- Un scénario = un chemin particulier d'exécution
- Un scénario = Instance de cas d'utilisation

- Une instance d'acteur crée un scénario

DIAGRAMME DE CAS d'UTILISATION

QUAND L'UTILISER ?

- Outil appréciable pour aider à comprendre les prérequis fonctionnels d'un système.
- Utile dans les premières phases d'un projet.
- Précède les spécifications détaillées.

ASTUCES :

- S'aider des flux & des acteurs identifiés dans le diagramme de communication.
- Regrouper ces flux identifiés.
- Ne pas descendre trop bas dans la description.

DIAGRAMME DE CAS d'UTILISATION

ASTUCES :

- Impossible de décrire tous les scénarios
  - Sélection des scénarios optimaux : interaction la plus fréquente.
  - Sélection des scénarios dérivés : certaines alternatives intéressantes.

- Commencer par les diagrammes CU qui présentent :
  - Le plus d'enjeux / risque.
  - Les plus importants.

DIAGRAMME DE CAS d'UTILISATION

EXERCICES

DIAGRAMME DE CAS d'UTILISATION

Exercice 1:

- Dans un établissement scolaire, on désire gérer la réservation des salles de cours ainsi que du matériel pédagogique (ordinateur portable ou/et Vidéo projecteur). Seuls les enseignants sont habilités à effectuer des réservations (sous réserve de disponibilité de la salle ou du matériel). Le planning des salles peut quant à lui être consulté par tout le monde (enseignants et étudiants). Par contre, le récapitulatif horaire par enseignant (calculé à partir du planning des salles) ne peut être consulté que par les enseignants. Enfin, il existe pour chaque formation un enseignant responsable qui seul peut éditer le récapitulatif horaire pour l'ensemble de la formation.

  Modéliser cette situation par un diagramme de cas d'utilisation

DIAGRAMME DE CAS d'UTILISATION

Solution :

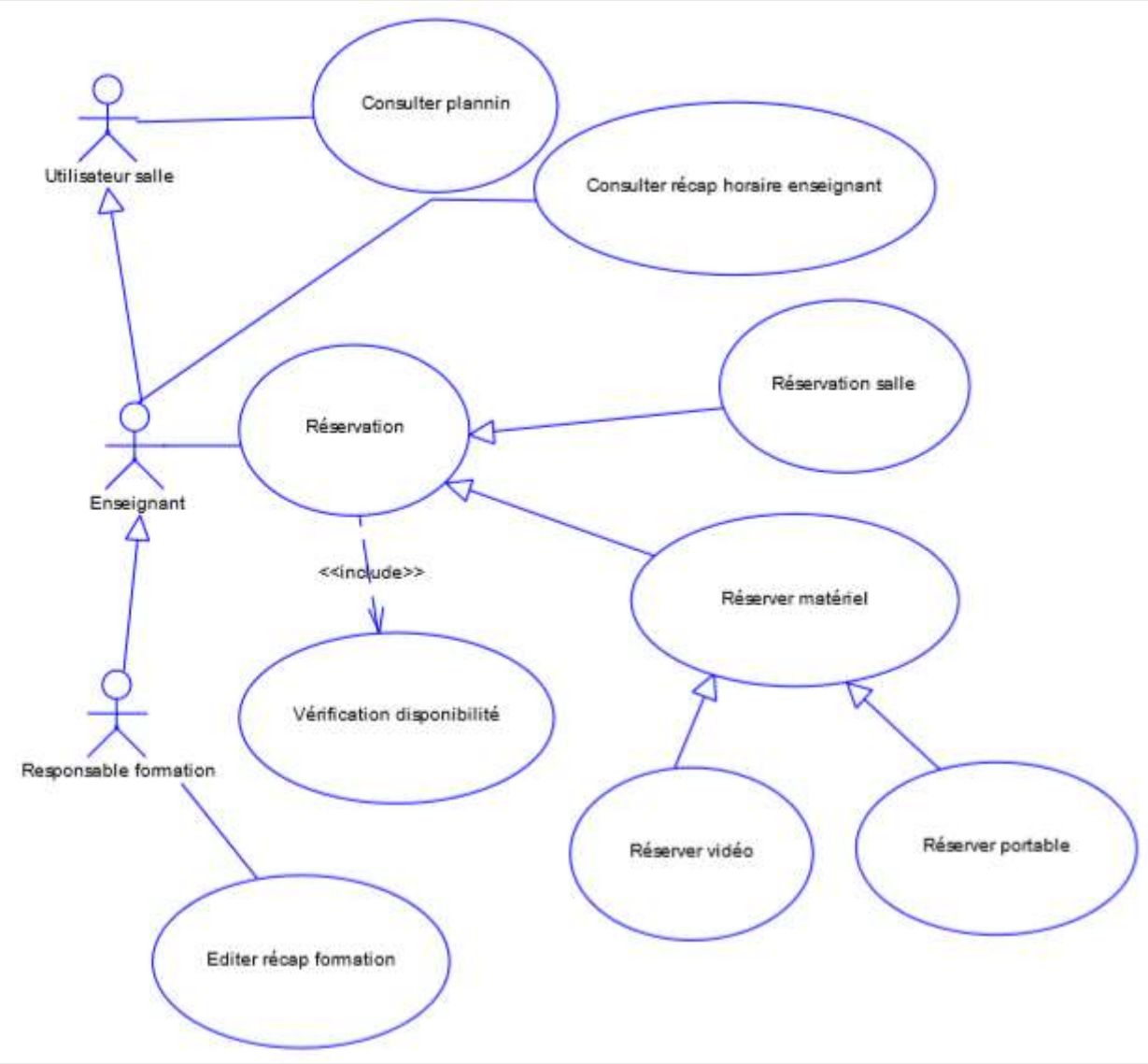


DIAGRAMME DE CAS d'UTILISATION

Exercice 2:

- On considère le système suivant de gestion d'un DAB (Distributeur automatique de billets) :
  - le distributeur délivre de l'argent à tout porteur de carte (carte Visa ou carte de la banque)
  - pour les clients de la banque, il permet :
    - la consultation du solde du compte
    - le dépôt d'argent (chèque ou numéraire)
  - toute transaction est sécurisée et nécessite par conséquent une authentification
  - dans le cas où une carte est avalée par le distributeur, un opérateur de maintenance se charge de la récupérer. C'est la même personne qui collecte également les dépôts d'argent et qui recharge le distributeur.

Modéliser cette situation par un diagramme de cas d'utilisation

DIAGRAMME DE CAS d'UTILISATION

Solution :

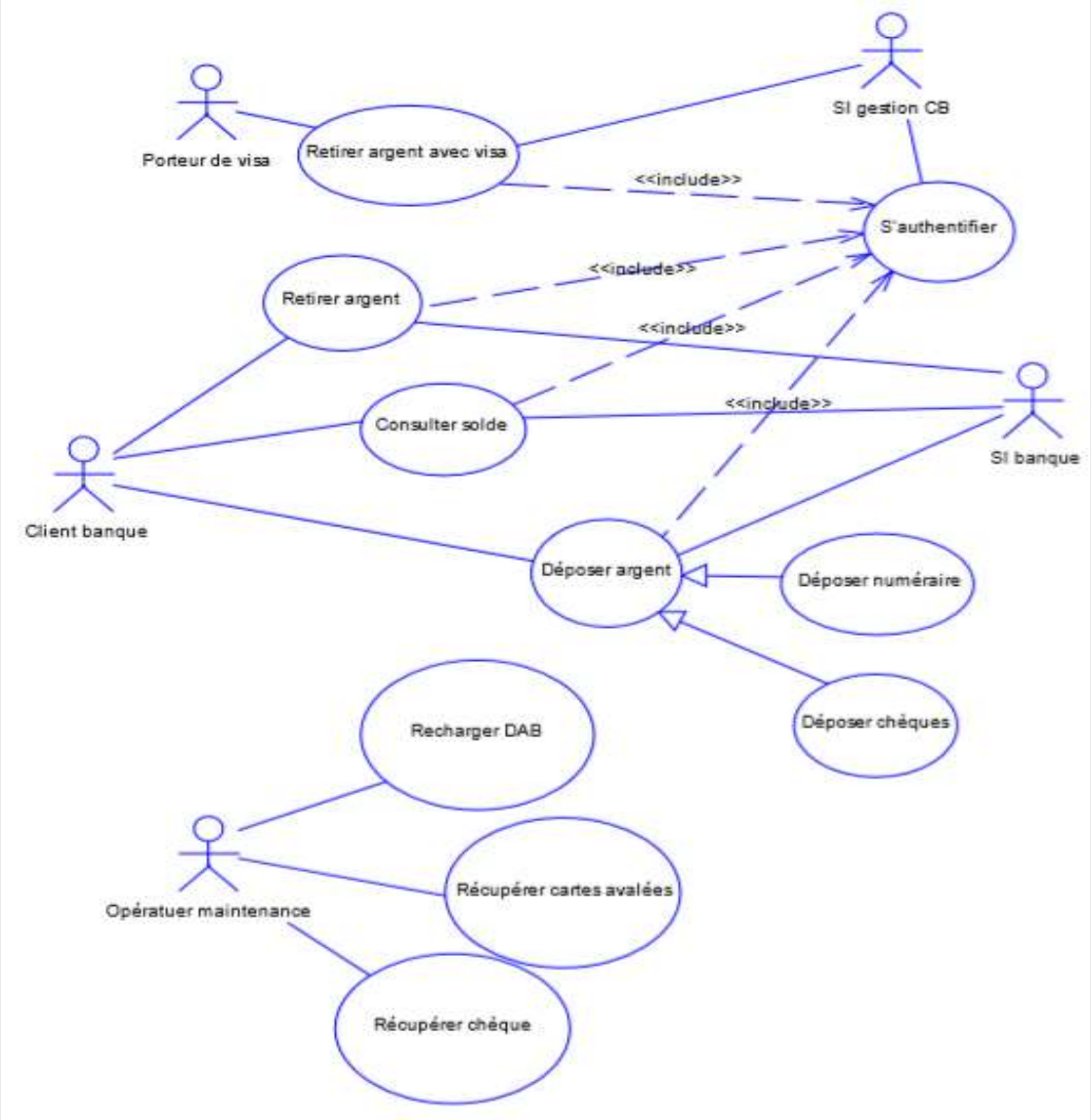


DIAGRAMME DE CAS d'UTILISATION

Exercice 3:

- Dans un magasin, le processus de vente est le suivant : le client entre, passe dans les rayons, demande éventuellement des renseignements ou procède à des essais, prend des articles (si le stock est suffisant), passe à la caisse où il règle ses achats (avec tout moyen de paiement accepté). Il peut éventuellement bénéficier d'une réduction.

Modéliser cette situation par un diagramme de cas d'utilisation

DIAGRAMME DE CAS d'UTILISATION

Solution :

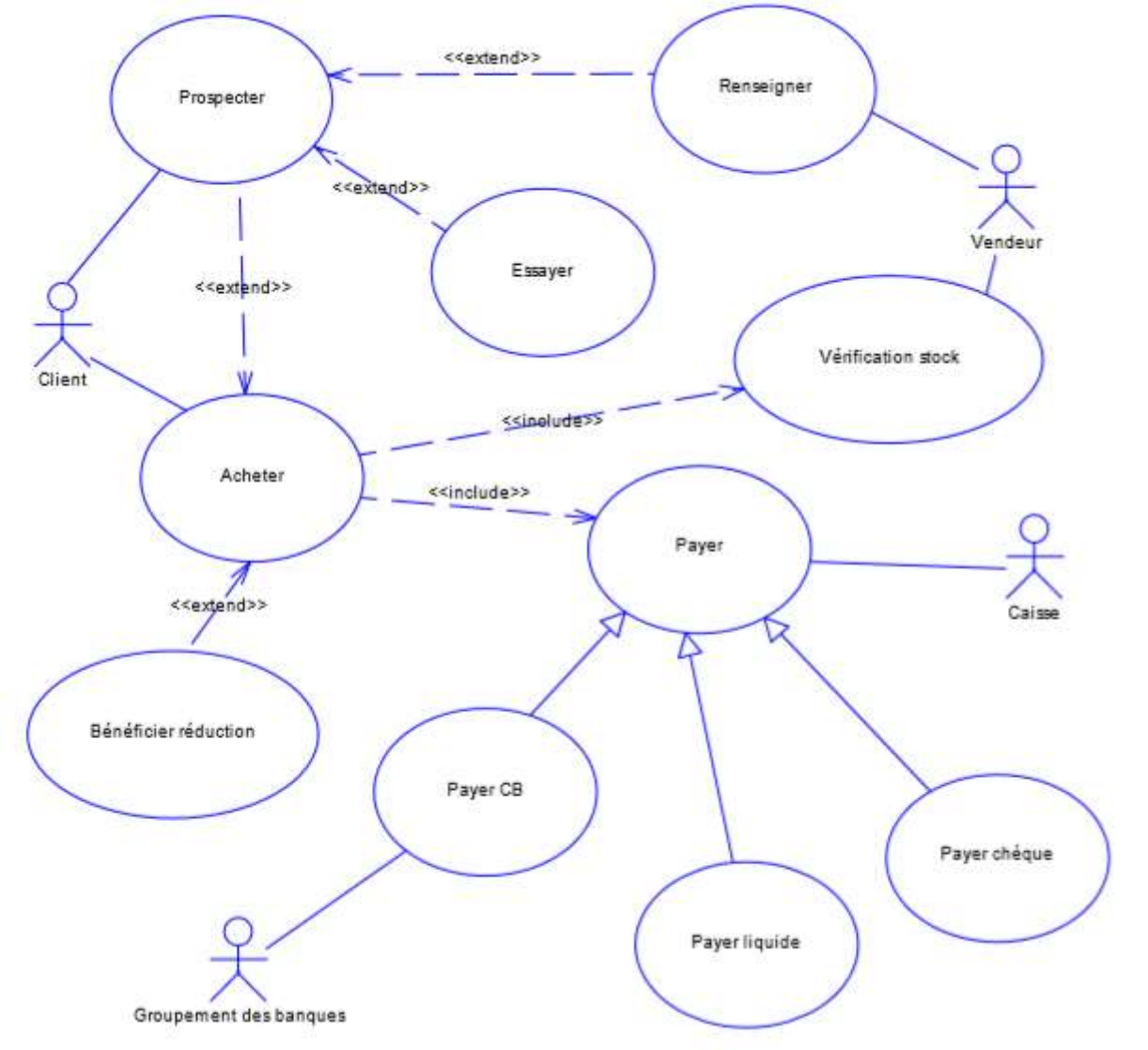


DIAGRAMME DE CAS d'UTILISATION

Fiche descriptive d'un cas d'utilisation :

On réalise une **fiche descriptive** pour chaque cas d'utilisation, de façon à raconter **l'histoire** du déroulement des différentes actions.

Cette fiche descriptive doit comporter 4 volets :

1. L'identification.
2. La description des scénarios.
3. La fin et les post-conditions.
4. Les compléments.

DIAGRAMME DE CAS d'UTILISATION

Le schéma suivant illustre la relation entre le diagramme des cas d'utilisation et la description textuelle d'un cas d'utilisation :

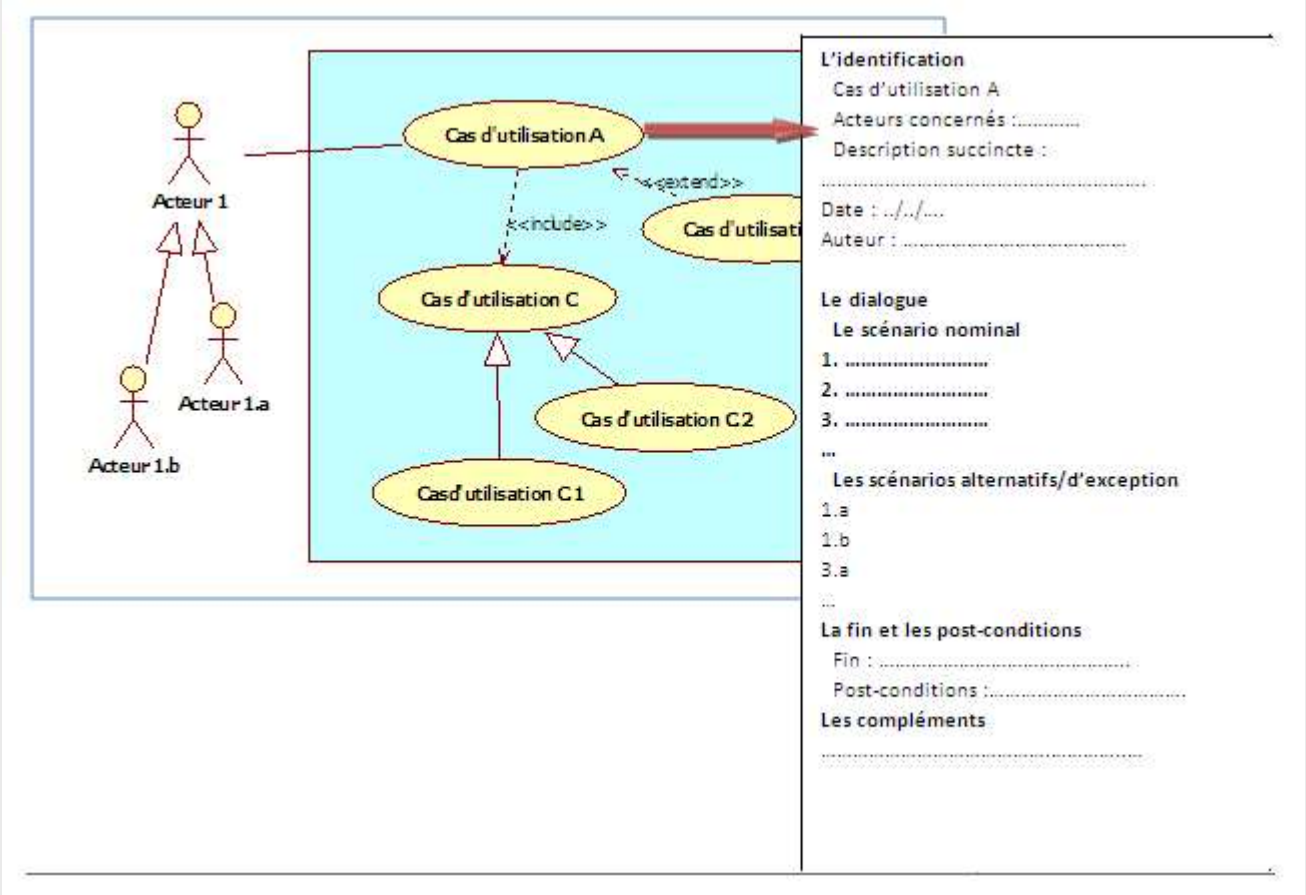


DIAGRAMME DE CAS d'UTILISATION

Description textuelle d'un cas d'utilisation simple :

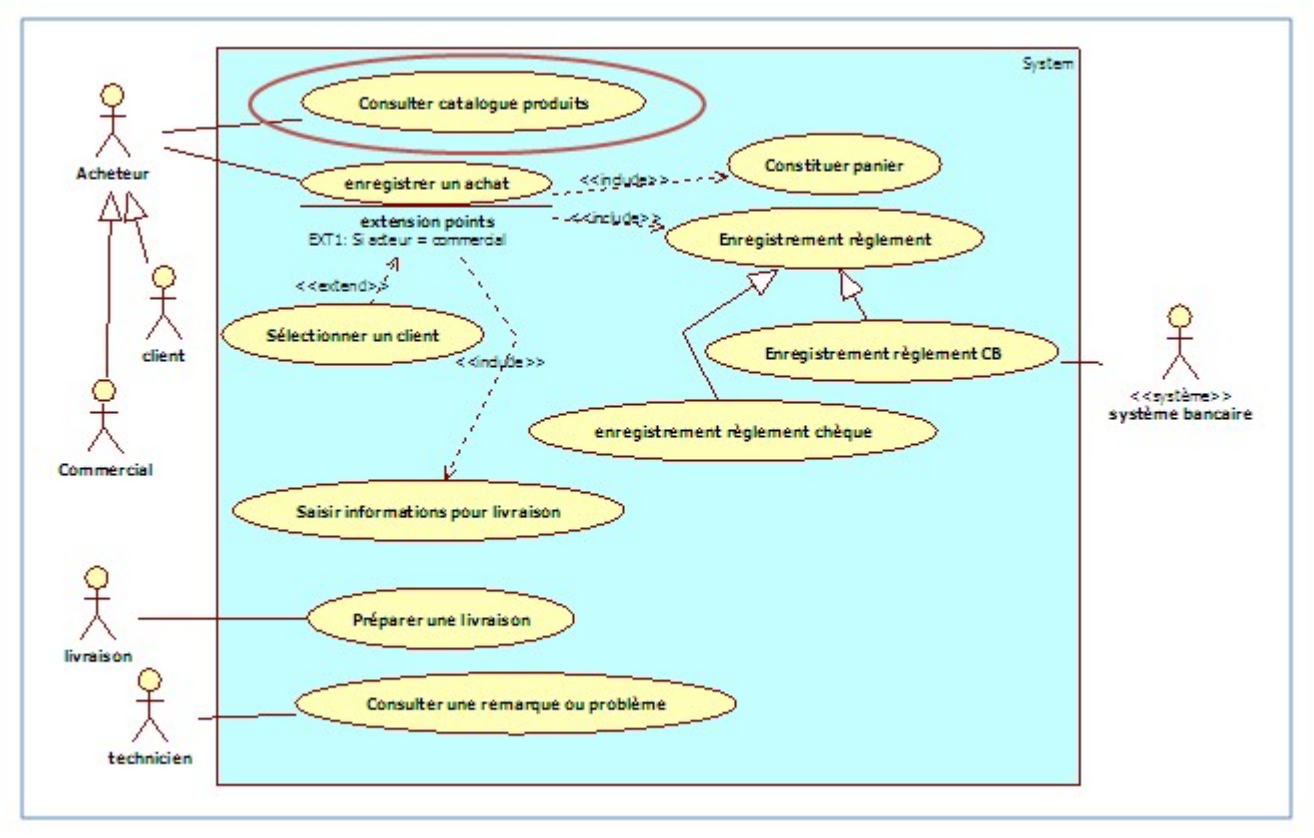


DIAGRAMME DE CAS d'UTILISATION

Volet 1 : l'identification

Dans le volet identification, on indique :

- Le numéro du cas d'utilisation, de façon aléatoire. Cela permet ensuite de les classer plus facilement ;
- le nom du cas d'utilisation (avec indication du package) ;
- l'acteur (ou les acteurs) s'il s'agit d'un cas d'utilisation principal ou le nom du cas d'utilisation principal lorsqu'il s'agit d'un cas d'utilisation interne ;
- une description succincte du cas d'utilisation ;
- la date de rédaction de la fiche et l'auteur, voire éventuellement les dates de mise à jour et les auteurs successifs ;
- les pré-conditions : il s'agit des conditions obligatoires au bon déroulement du cas d'utilisation. Par exemple, nous avons vu dans le chapitre précédent qu'il fallait obligatoirement s'authentifier avant même de pouvoir « Consulter le catalogue produit ». Dans le cas, le package « Authentification » est une pré-condition ;
- les événements à l'origine du démarrage du cas d'utilisation.

DIAGRAMME DE CAS d'UTILISATION

Volet 1 : l'identification

Voici donc comment la fiche descriptive débuterait pour notre cas d'utilisation « Consulter catalogue produit » :

Cas n° 1

Nom : Consulter catalogue produit (package « Gestion des achats »)
Acteur(s) : Acheteur (client ou commercial)
Description : La consultation du catalogue doit être possible pour un client ainsi que pour les commerciaux de l'entreprise.
Auteur : Carina Roels
Date(s) : 10/11/2013 (première rédaction)

Pré-conditions : L'utilisateur doit être authentifié en tant que client ou commercial (Cas d'utilisation « S'authentifier » – package « Authentification »)
Démarrage : L'utilisateur a demandé la page « Consultation catalogue »

DIAGRAMME DE CAS d'UTILISATION

Volet 2 : La description des scénarios (ou dialogue)

Nous allons maintenant décrire les scénarios qui explicitent la chronologie des actions qui seront réalisées par l'utilisateur et le système. Il existe 3 parties :

- **Le scénario nominal**
  Il s'agit ici de décrire le déroulement idéal des actions, où tout va pour le mieux.
- **Les scénarios alternatifs**
  Ici, il s'agit de décrire les éventuelles étapes différentes liées aux choix de l'utilisateur, par exemple. C'est le cas des étapes liées à des conditions.
- **Les scénarios d'exception**
  On parlera de scénario d'exception lorsque une étape du déroulement pourrait être perturbée à cause d'un événement anormal. Par exemple, lorsqu'une recherche de client ne trouve aucun client correspondant aux critères fournis.

DIAGRAMME DE CAS d'UTILISATION

Volet 2 : La description des scénarios (ou dialogue)

Le scénario nominal

1. **Le système** affiche une page contenant la liste les catégories de produits.
2. *L'utilisateur* sélectionne une des catégories.
3. **Le système** recherche les produits qui appartiennent à cette catégorie.
4. **Le système** affiche une description et une photo pour chaque produit trouvé.

5. *L'utilisateur* peut sélectionner un produit parmi ceux affichés.
6. **Le système** affiche les informations détaillées du produit choisi.
7. *L'utilisateur* peut ensuite quitter cette description détaillée.
8. **Le système** retourne à l'affichage des produits de la catégorie (retour à l'étape 4)

Les scénarios alternatifs

2.a *L'utilisateur* décide de quitter la consultation de la catégorie de produits choisie.
2.b *L'utilisateur* décide de quitter la consultation du catalogue.
5.a *L'utilisateur* décider de quitter la consultation de la catégorie de produits choisie.
5.b *L'utilisateur* décide de quitter la consultation du catalogue.
7.a *L'utilisateur* décide de quitter la consultation de la catégorie de produits choisie.
7.b *L'utilisateur* décide de quitter la consultation du catalogue.

DIAGRAMME DE CAS d'UTILISATION

Volet 3 : La fin et les post-conditions

La fin permet de récapituler toutes les situations d'arrêt du cas d'utilisation. Cela permet parfois de s'apercevoir que l'on n'a pas vu tous les cas de figure qui peuvent se présenter.

Par exemple, notre cas d'utilisation peut s'arrêter aux étapes 2, 5 et 7 car l'utilisateur peut décider de quitter la consultation du catalogue pour revenir à l'accueil par exemple.

Les post-conditions nous indiquent un résultat tangible qui est vérifiable après l'arrêt du cas d'utilisation et qui pourra témoigner du bon fonctionnement. Cela pourrait être une information qui a été enregistrée dans une base de données ou dans un fichier, ou encore un message envoyé par mail, etc.

Par exemple, ici il n'y en a pas car la consultation du catalogue ne donne pas lieu à l'enregistrement d'informations dans un fichier ou une base de données.

Voici le volet 3 de notre cas d'utilisation « Consulter catalogue produits ».

Fin : Scénario nominal : aux étapes 2, 5 ou 7, sur décision de l'utilisateur
Post-conditions : Aucun

DIAGRAMME DE CAS d'UTILISATION

Volet 4 : Les compléments

Les compléments peuvent porter sur des sujets variés :

- l'ergonomie ;
- la performance attendue ;
- des contraintes (techniques ou non) à respecter ;
- des problèmes non résolus (ou questions à poser au client et aux futurs utilisateurs).

Ergonomie

L'affichage des produits d'une catégorie devra se faire par groupe de 15 produits. Toutefois, afin d'éviter à l'utilisateur d'avoir à demander trop de pages, il devra être possible de choisir des pages avec 30, 45 ou 60 produits.

Performance attendue

La recherche des produits, après sélection de la catégorie, doit se faire de façon à afficher la page des produits en moins de 10 secondes.

Problèmes non résolus

Nous avons fait la description basée sur l'information que les produits appartiennent à une catégorie. Est-ce qu'il existe des sous-catégories ?
Si tel est le cas, la description devra être revue.

Est-ce que la consultation du catalogue doit être possible uniquement par catégorie ou est-ce qu'on doit prévoir d'autres critères de recherche de produits ?

Doit-on prévoir un affichage trié sur des critères choisis par l'utilisateur (par exemple : par tranche de prix, par disponibilité, etc) ?