Faculté des Mathématiques et de l'Informatique
Département de l'Informatique

Module : Analyse et Conception des Systèmes Informatiques

Master2 cryptographie et Sécurité

Enseignant : Betta Mohamed

Année universitaire 2020/2021

Contenu du module

- Introduction à la programmation objet
- Origines et Objectifs d'UML
- Axe fonctionnel
- Cas d'utilisations (uses cases)
- Relation entre les cas d'utilisations
- Axe dynamique
    - Diagramme de séquences
    - Diagramme de collaboration
    - Diagrammes d'états
    - Diagrammes d'activités

Contenu du module (suite)

- Démarche de construction du logiciel avec UML
  - Introduction au cycle de vie logiciel
  - Construction d'un modèle d'analyse
  - Construction des cas d'utilisations
  - Processus de construction du diagramme de classes
  - Diagrammes dynamiques
  - Construction des diagrammes d'états
- Validation et vérification
- Etude de cas

Références

1. Pascal Roquer. UML in Practice : the Art of Modeling Software Systems demonstrated through worked Examples and Solutions. John Wiley and Sons. 2003.
2. Jim Arlono and Ila Neustadt. UML and the Unified Process Practical Object-Oriented Analysis and Design. Addisson Wesley.2002.
3. Bhuvan Unhelkor. Software Engineering with UML. Taylor and Francis. 2018.
4. Ian Sommerville. Software Engineering tenth edition. Pearson Education Limited. 2016.
5. Pascal Roques. UML in Proctice. John Xilley & Sons LTD ; 2204;

Introduction à la programmation objet

- Motivations pour la programmation objet
- Notion d'objet
- notion de classe
- Mécanisme d'héritage

Un rappel sur ces notions sera présenté en salle de cours (présentiel).

Ces notions sont considérés acquises dans la formation licence.

Pourquoi des méthodes d'analyse et de conception ?

- L'évolution du matériel.
- L'évolution des systèmes.
- L'évolution des langages de programmation et
- La complexité croissante des logiciels.

Classement des méthodes

Différentes manières de classement existent:

- **Analyse descendante** (décomposition)
- **Analyse ascendante (composition)** : on part des modules déjà existants.

Comme on a :

- **Des méthodes fonctionnelles** (dirigées par les traitements) c'est à dire on donne l'importance aux traitements au détriment des données.
- **Méthodes dirigées par les données** qui donnent de l'importance aux données au détriment des traitements (bases de données) et
- **Les méthodes orientées objets** qui donnent la même importance aux données et aux traitements. Elles sont influencées par le développement du langage ADA et les langages de programmation orientés objet.

Analyse et Conception Orientée-Objet

Pour l'analyse. ces méthodes examinent un problème en mettant en évidence les classes et les objets correspondant sous forme de composants indépendants qui interagissent selon des modalités bien définies. Le résultat de cette analyse peut servir de base à une conception orientée objet.

Dans la plupart des méthodes orientées objet. l'étude d'un problème est réalisée suivant trois aspects :

- **Aspect statique (descriptif)** : identification des propriétés des objets et leurs liaisons avec les autres objets.
- **Aspect dynamique (comportement des objets)** : les différents états par lesquels les objets passent et les événements qui déclenchent ces changements d'états.
- **Aspect fonctionnel** dans lequel on doit préciser les fonctions réalisées par les objets par l'intermédiaire des méthodes.

Le résultat de la conception est un modèle qui représente la réalité. Il peut être textuel. mathématique ou graphique.

UML (Unified Modeling Language)

UML est un langage pour **spécifier. visualiser. construire et documenter** les artefacts logiciels.

Il permet d'unifier les approches :

- Schaler/Mellor
- Jacobson
- Booch
- Good and Yourdon
- OMT
- .....

Pourquoi cette unification ?

- Les méthodes sont spécialisées ou adaptées a une démarche particulière (secteur industriel. matériel embarqué …etc.)
- Ces méthodes sont développées indépendamment les unes des autres donc elles sont redondantes et incompatibles entre elles lorsqu'elles font appel à des notations ou terminologies différentes
- Problème de maintenance

Différence entre analyse et conception

La différence entre l'analyse et la conception dans UML se situe au niveau de détails dans les diagrammes utilisés.

Analyse: niveau comprenant la vue fonctionnelle ainsi qu'un sous-ensemble des vues statique et dynamiques excluant les diagrammes de composant. déploiement et collaboration.

Conception : elle s'accentue sur les diagrammes de collaboration et les détails de conception des diagrammes de classes et introduit aussi les diagrammes de composants et de déploiement.

Diagramme de Classe: Analyse et Conception

Analyse : nommer les classes. quelques attributs et méthodes.

Conception : tous les attributs et méthodes doivent apparaitre de façon détaillée avec tous les types de paramètres et types de retour.

Les différentes vues dans UML

UML utilise 14 diagrammes è savoir **cas d'utilisation**. **classe**. **objet**. composant. déploiement. **séquences**. **collaboration**. **états**. **activités**. structure composée. paquetage. temps et profile

- **Vue fonctionnelle** : diagramme de cas d'utilisation. diagramme de séquences.
- **Vue statique (produit une structure du système)** : diagramme de classes. diagramme de paquetage (regroupement de classes fortement liées)
- **Vue dynamique** (produit une vue comportementale du système): diagrammes s'activités.

Remarques:

- les diagrammes de collaboration sont spécifiques à la conception et à la l'implémentation.
- Les diagrammes de déploiement (indique pour quelle architecture seront déployés les différents processus qui réalisent l'application).

Structurel vs comportemental et statique vs dynamique

	structurel	**comportemental**
statique	Classe. paquetage. composant. déploiement. profil	Cas d'utilisations. activité. vue d'ensemble d'interaction
dynamique	Objet. structure composite	Machine d'états. séquence. communication. temps

DIAGRAMMES UML

Structurel vs comportemental et statique vs dynamique (suite)

- **Structurel vs comportemental**

L'aspect structurel d'un diagramme illustre la façon dont le système est organisée. par contre l'aspect comportemental modélise le flux du système.

Par exemple. l'aspect structurel montre comment les classes sont reliées dans un digramme de classes et l'aspect comportemental montre la façon par laquelle les utilisateurs interagissent avec le système à travers les cas d'utilisation el les activités.

- **Statique vs dynamique**

C'est la dépendance temps dans le modèle qui différencie l'aspect statique de l'aspect dynamique. Dans l'aspect statique. pas de concept de temps ni de mouvement. si un changement de temps existe (ou évènement dans le temps) c'est un aspect dynamique.

Cas d'utilisations (uses cases)

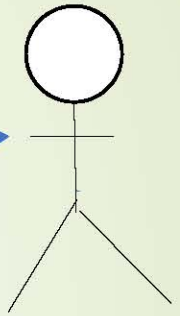
- Les diagrammes des cas d'utilisations montrent les interactions des utilisateurs avec les composants du système.
- Ils montrent un ensemble de cas d'utilisations et comment les acteurs peuvent les utilisés.

- Un cas d'utilisation est la spécification des séquences d'actions. y compris des séquences d'erreurs et variantes de séquences. que le système. sous/système ou les classes peuvent effectuées en interagissant avec les acteurs externes.

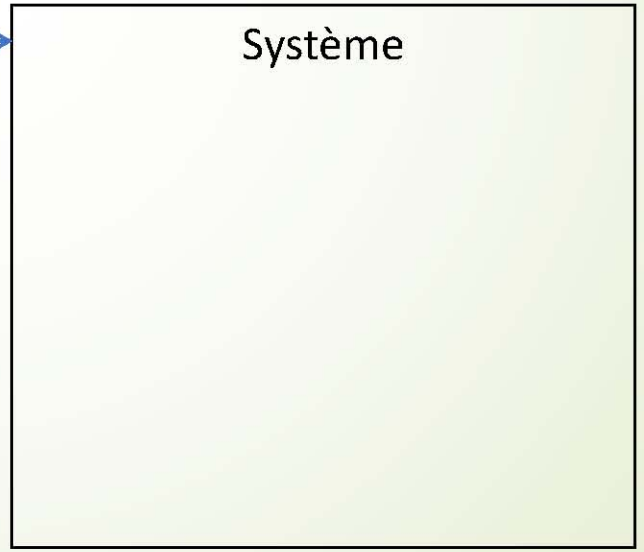
Cas d'utilisations (uses cases)

Trois types d'élément sont utilisés:

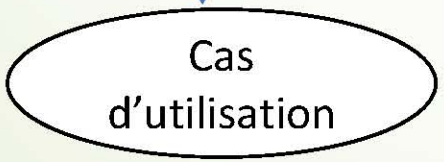
1. Acteur
2. Système
3. Cas d'utilisation



Personne
Système ou
temps



Délimiter les
frontières du
système



Verbe

Cas d'utilisations (suite)

- Un acteur est une entité externe qui interagit avec le système à un niveau pour représenter la simulation d'un évènement possible (business process) dans le système.
- Un acteur initie un cas d'utilisation.
- Un acteur peut être : personne. classe. outil software. temps ..etc.

Remarque : la granularité des cas d'utilisation est variable. Elle dépend de la situation qu'on modélise.

Relations entre les cas d'utilisations

- **Include**

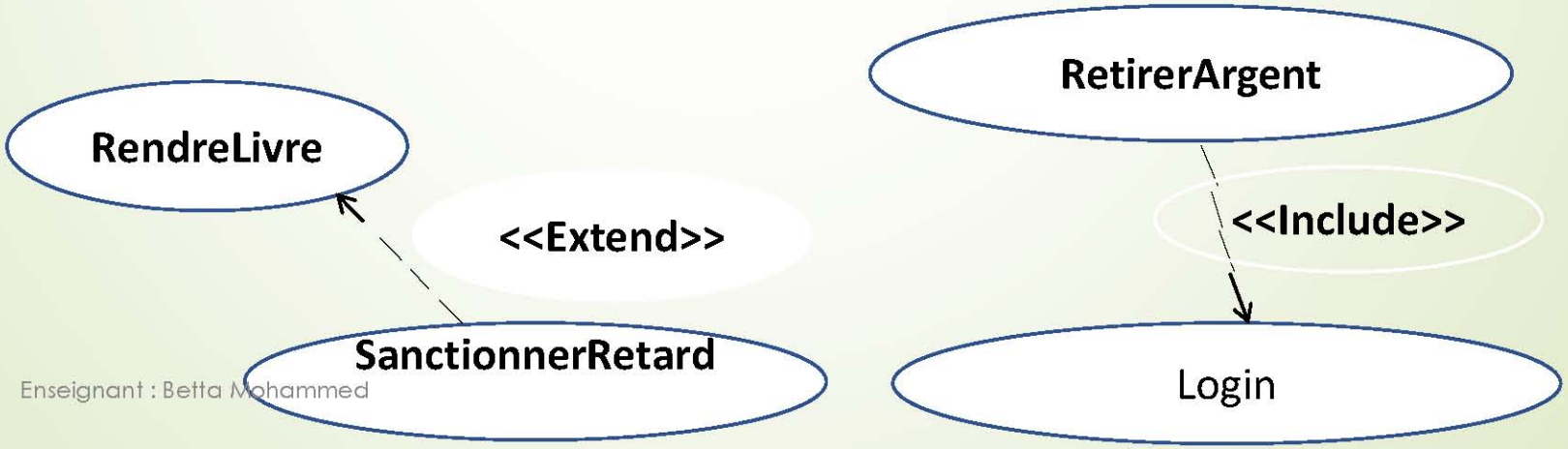
C'est l'inclusion obligatoire d'un cas d'inclusion dans un autre.

exemple: pour le **retrait** d'argent on doit s'**authentifier** .

- **Extend**

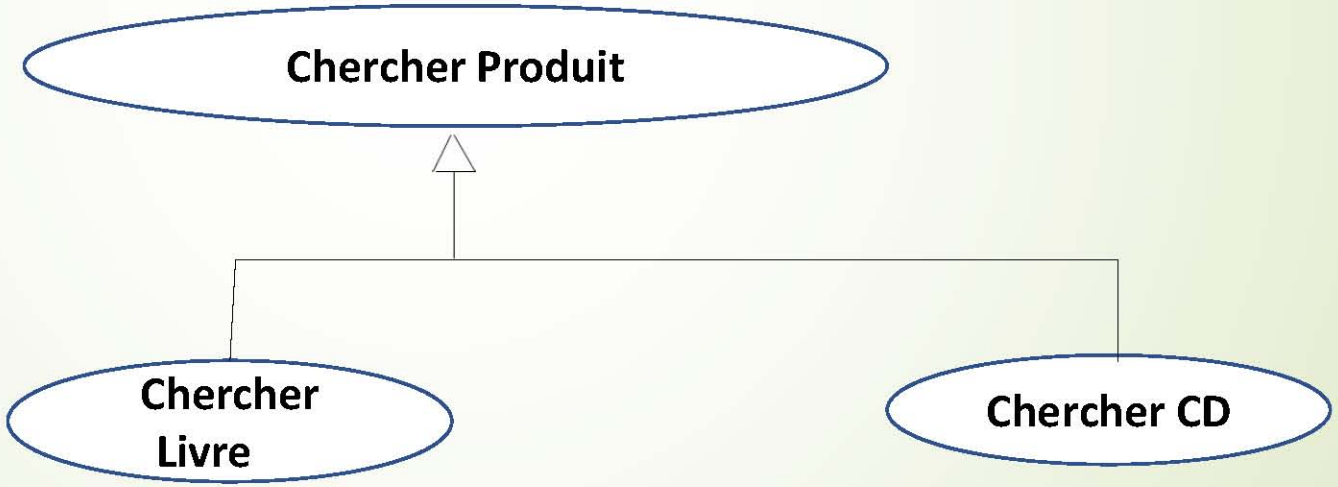
C'est l'extension d'un cas d'utilisation pour un autre. extend est facultatif

Exemple **RendreLivre** peut s'étendre par **sanctionnerRetard**.



Généralisation des Cas d'utilisations

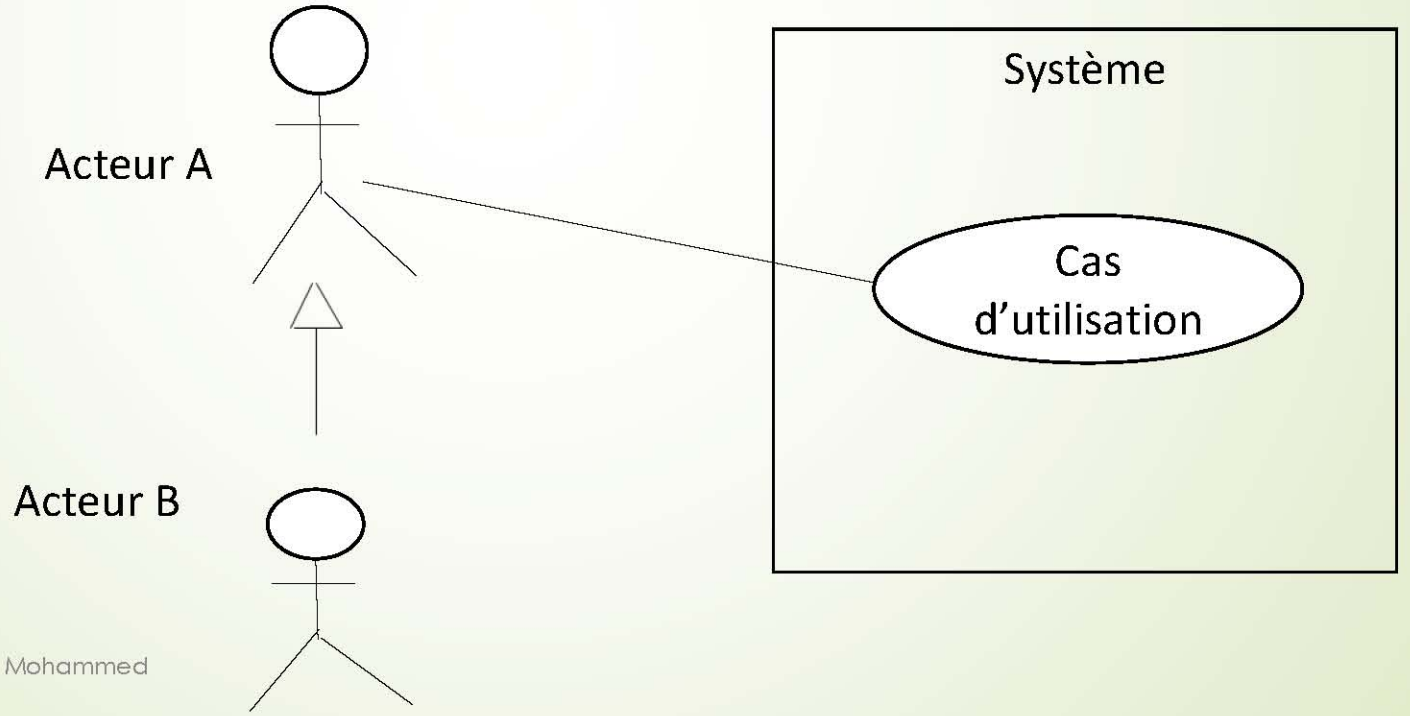
- La généralisation des cas d'utilisations est utilisée lorsque un ou plusieurs cas d'utilisations sont des spécialisations d'un cas plus général.
- C'est la factorisation du comportement d'un ou de plusieurs des cas d'utilisations en un cas d'utilisation parent.



Relation entre les acteurs

- La relation de généralisation peut être utilisée entre les acteurs.

L'acteur utilise la cas d'utilisation. c'est un héritage.



Identification des acteurs

Pour identifier un acteur, on doit répondre aux questions suivantes:

1. Qui utilise ou interagi avec le système? (question principale)

1. Quel rôle joue-t-il dans l'interaction?
2. Qui installe le système?
3. Qui maintient le système?
4. Quels sont les autres systèmes qui interagissent avec ce système?
5. Qui obtient et fournit des informations au système?
6. Quelque chose se passe à une heure fixe?

Modélisation des acteurs

Pour modéliser les acteurs, il faut prendre en considération les points suivants:

1. Les acteurs sont toujours externes au système.
2. Les acteurs interagissent directement avec le système.
3. Les acteurs représentent les rôles que les personnes ou les choses jouent par rapport au système et non des personnes ou choses spécifiques.
4. Un acteur peut jouer plusieurs rôles en relation avec le système.
5. Chaque acteur a besoin d'un nom court et significatif.

Identification des cas d'utilisations

- Un cas d'utilisations décrit le comportement que le système expose au profil d'un ou de plusieurs acteurs.
- le cas d'utilisations est toujours activé par un acteur.

pour identifier un cas d'utilisation, on doit répondre aux questions suivantes:

1. Comment chaque acteur utilise le système?
2. Que fait le système pour chaque acteur?