Université Antonine

Modélisation Logicielle

Année Académique 2016-2017

B.P. 40016 Hadat-Baabda, LIBAN
Tél. + 961 5 92 40 73 - 74 - 76 | **Fax** + 961 5 92 48 15
contact@upa.edu.lb | **www**.upa.edu.lb | **@**uAntonine

Les acquis d'apprentissage du cours

A la fin de ce cours, l'apprenant sera capable de :

- Déterminer les objets à partir d'un scénario donné
- Réutiliser les classes et les fonctions et faire les dépendances nécessaires entre les différentes classes et objets
- Déterminer les acteurs ainsi que les cas d'utilisation des acteurs
- Illustrer l'enchaînement et le déroulement des scénarios

Un modèle

Un *modèle* est une représentation simplifiée d'un procesus ou d'un système, permettant de le décrire et de l'expliquer.

Un *modèle* est la théorie d'une action qui va avoir lieu

Un modèle

Modéliser un système avant sa réalisation permet de mieux comprendre le fonctionnement du système.

Un *modèle* est un langage commun, il est donc un vecteur facilitant la communication avec les membres de l'équipe chargés à travailler sur le même système

Un modèle

Dans le domaine de l'ingénierie du logiciel, le modèle permet de mieux répartir et visualiser les tâches.

Il s'agit aussi d'un facteur de réduction des coûts et des délais

Certaines outils/plateformes de modélisation exploitent le modèle et génère directement le code (niveau squelette des classes) à partir du modèle conçu (exemple : Objecteering)

Le cycle de vie d'un logiciel

Le cycle de vie d'un logiciel désigne toutes les étapes de développement d'un logiciel, de sa conception à sa disparition.

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

- **Définition des objectifs** : définir la finalité du projet et sa stratégie globale
- **Analyse des besoins et faisabilité** : recueillir et formaliser les besoins du clients (demandeur du système)
- **Spécifications ou conception générale** : spécifier l'architecture générale du logiciel.
- **Conception détaillée** : définir précisément chaque sous-ensemble du logiciel

Le cycle de vie d'un logiciel

- **Codage (Implémentation ou Programmation) :** traduire dans un langage de programmation des fonctionnalités définies lors des phases de conception
- **Tests unitaires**: vérifier que le sous-ensemble du logiciel est implémenté conformément aux spécifications
- **Intégration** : S'assurer de l'interfaçage des différents éléments (modules) du logiciel
- Qualification : vérifier la conformité du logiciel aux spécifications initiales
- **Documentation**: produire les informations nécessaires pour l'utilisation du logiciel
- Mise en production : déployer le système implémenté
- **Maintenance**: Appliquer des actions correctives ou évolutive sur le logiciel

Histoire des modélisations par **objets**

- Les méthodes de modélisation par objets apparaissent entre 1990 et 1195 (50+ méthodes)
- En 1994, un consensus se fait autour de 3 méthodes:
  - OMT de James Rumbaugh fournit une représentation graphique des aspects statique, dynamique et fonctionnel d'un système
  - OOD de Grady Booch introduit le concept de paquetage (package)
  - OOSE d'Ivar Jacobson fonde l'analyse sur la description des besoins des utilisateurs (cas d'utilisation)

Histoire des modélisations par **objets**

- Ces trois méthodes ont été convergées pour définir une méthode commune
  - Le UML – Unified Modeling Language
- *Unified* signifie que les méthodes ont été unifiées
- Le UML est un langage de modélisation
- Progrès -> Unified Method 0.8 (1995), UML 0.9 (1996), UML 1.0 (1997), UML 2.1 (2008)

Les diagrammes d'UML

- 13 diagrammes regroupés dans deux grands ensembles.
  - Les diagrammes **structurels**

  Ces diagrammes, au nombre de six, représente l'aspect statique d'un système (classes, objets, composants…).

  - Les diagrammes de **comportement**

  Ces diagrammes représentent la partie dynamique d'un système réagissant aux événements et permettant de produire les résultats attendus par les utilisateurs.

LES DIAGRAMMES STRUCTURELS

Diagramme de classes **(DCL)**	Représente la description statique du système en intégrant dans chaque classe la partie dédiée aux données et celle consacrée aux traitements
Diagramme d'objets **(DOB)**	Représente les instances des classes et des liens entre instances.
Diagramme de composants **(DCP)**	Représente les différents constituants du logiciel au niveau de l'implémentation d'un Système
Diagramme de déploiements **(DPL)**	Décrit l'architecture technique d'un système avec une vue centrée sur la répartition des composants dans la configuration d'exploitation.
Diagramme de paquetages **(DPA)**	Donne une vue d'ensemble du système structuré en paquetage. Chaque paquetage représente un ensemble homogène d'éléments du système (classes, composants…).
Diagramme de structure composites **(DSC)**	Décrit la structure interne d'un ensemble complexe composé par exemple de classes ou d'objets et de composants techniques.

Les diagrammes de comportement

Diagramme des cas d'utilisations **(DCU)**	Ce diagramme est destiné à représenter les **besoins** des utilisateurs par rapport au système.
Diagramme d'état-transitions **(DET)**	Représente les différents **états** des objets en réaction aux événements.
Diagramme d'activités **(DAC)**	Représente une vision des **enchaînements** des activités propres à une opération ou à un cas d'utilisation.
Diagramme de séquences **(DSE)**	Permet de décrire les scénarios de chaque cas d'utilisation en mettant l'accent sur la **chronologie** des opérations en interaction avec les objets.
Diagramme de communications **(DCO)**	Ce diagramme est une autre représentation des scénarios des cas d'utilisation qui met plus l'accent sur les objets et les **messages** échangés.
Diagramme global d'interactions **(DGI)**	Ce diagramme fournit une vue générale des interactions décrites dans le diagramme de séquence et des **flots** de contrôle décrits dans le diagramme d'activités.
Diagramme de temps **(DTP)**	Ce diagramme permet de représenter les états et les interactions d'objets dans un contexte où le **temps** a une forte influence sur le comportement du système à gérer.

UML dans un processus de développement

- **Analyse des besoins**
  - Déterminer ce que les clients et les utilisateurs attendent du logiciel à développer
  - Techniques UML utilisées:
    - Les cas d'utilisation (**DCU**)
    - Un diagramme de classes (**DCL**)
    - Un diagramme d'activités (**DAC**)
    - Un diagramme d'états (**DET**)

UML dans un processus de développement

- **Conception**
  - Diagrammes plus techniques
  - Techniques UML utilisées:
    - Les diagrammes de classes (**DCL**)
    - Les diagrammes de séquences (**DSE**)
    - Les diagrammes de paquetages (**DPA**)
    - Les diagrammes d'états (**DET**)
    - Les diagrammes de déploiement (**DPL**)

UML dans un processus de développement

- **Documentation**
  - Obtenir une vue globale du système
  - Documentation détaillée générée à partir du code
  - Une première étape pour le lecteur, avant qu'il ne s'engage dans les détails du code.
  - Techniques UML utilisées:
    - Diagramme de paquetages (**DPA**)
    - Diagramme de déploiement (**DPL**)
    - Diagramme de classes (**DCL**)
    - Diagramme d'états (**DET**)
    - Diagramme d'activités (**DAC**)

Objet et Classe

- Un **objet** représente une entité qui se caractérise par l'ensemble suivant:
  - propriétés (attributs)
  - états significatifs.
  - un comportement.

- Une **classe** est l'abstraction d'un ensemble **d'objets** qui possèdent une structure identique (liste des **attributs**) et un même comportement (liste des **opérations**)

→ Exemple

Encapsulation et Interface

- Encapsulation: Regroupement des **attributs** et **opérations** dans une même classe. Ils ne sont accessibles qu'à partir d'opérations définies dans la **classe**.
- Le principe d'encapsulation renforce l'autonomie et l'indépendance de chaque classe et renforce le concept de **réutilisabilité**.
- Interface: L'ensemble des opérations d'une classe rendu **visible** aux autres classes.

→ Exemple

Association et agrégation

- L'association représente une **relation** entre plusieurs classes.
- L'agrégation est une **forme particulière d'association** entre plusieurs classes. Elle exprime le fait qu'une classe est composée d'une ou plusieurs autres classes.

→ Exemple

Généralisation et spécialisation

- La généralisation de classes consiste à factoriser dans une classe, appelée **superclasse**, les attributs et/ou opérations des classes considérées.
- Elle permet de réaliser une **hiérarchie** des classes.
- La spécialisation représente la démarche inverse de la généralisation puisqu'elle consiste à créer à partir d'une classe, plusieurs classes spécialisées.
- La généralisation-spécialisation est un des mécanismes les plus importants de l'approche objet qui facilite la réutilisation des classes.

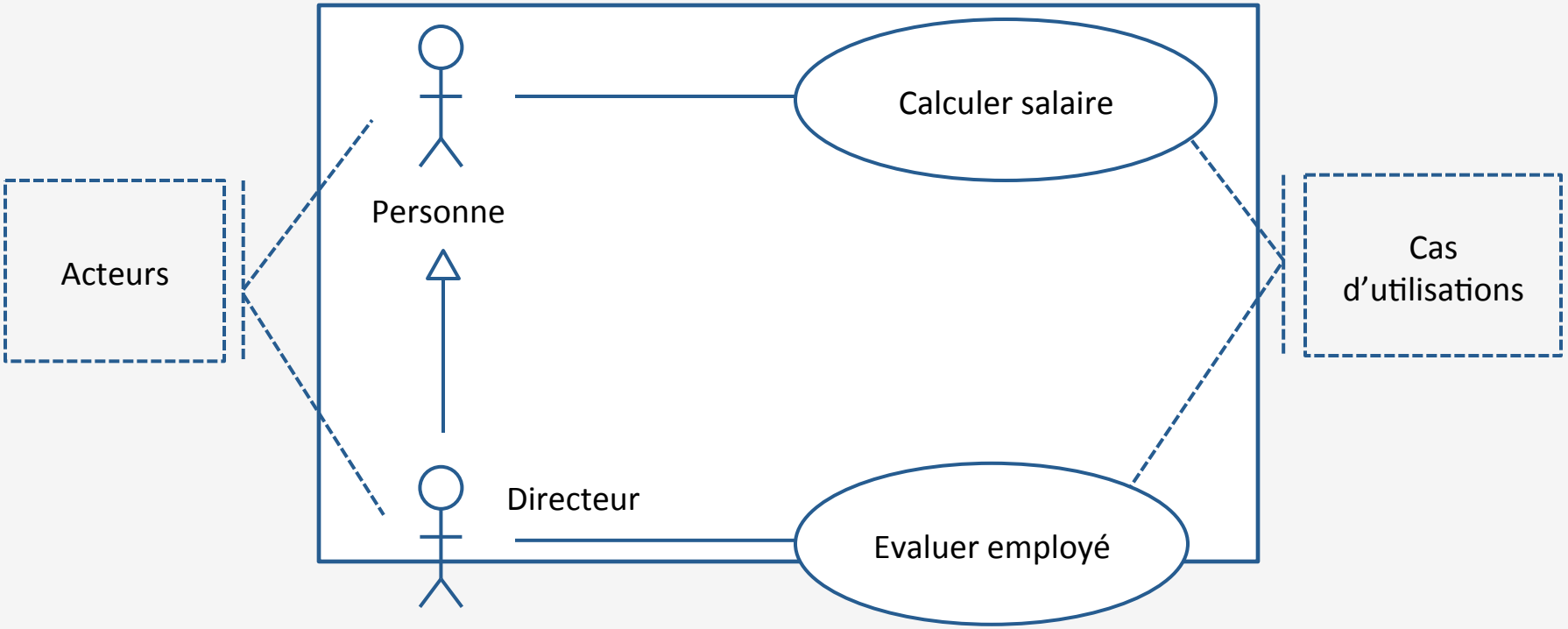
→ Exemple

Polymorphisme

- Le polymorphisme est la capacité donnée à une même opération de s'exécuter **différemment** suivant le **contexte** de la classe où elle se trouve.
- Ainsi une opération définie dans une superclasse peut s'exécuter de manière différente selon la **sous-classe** où elle est **héritée**.

→ Exemple

Exemple de diagramme de cas d'utilisation (DCU)



Exemple de diagramme de classe **(DCL)**