IFT2255 - Génie logiciel

Modélisation

Bruno Dufour, Houari Sahraoui, Julie Vachon
{dufour,sahraouh,vachon}@iro.umontreal.ca

Qu'est-ce que la modélisation ?

- Activité consistant à créer une **représentation simplifiée** (modèle) d'un phénomène ou d'un système
  - permet d'étudier son déroulement ou son fonctionnement
  - peut être accompli avec différents degrés de formalisme

Degrés de formalisme

- Descriptions informelles
  - ex: description en langue naturelle, croquis, etc.
- Descriptions semi-formelles
  - Notation graphique dont la sémantique n'est pas formellement définie
  - ex: UML
- Descriptions formelles
  - Spécifications algébriques, spécifications logiques, réseaux de Petri, langages de programmation, etc.

Utilité de la modélisation

- Approfondir la compréhension du problème
- Réduire la complexité du problème par l'abstraction
- Réunir et visualiser un ensemble de détails choisis
- Favoriser la communication avec les autres membres de l'équipe de développement, les utilisateurs, etc.
- Documenter

UML

Unified Modelling Language (UML)

- Une norme OMG (Object Management Group)
- Un langage visuel de modélisation rigoureux et non formel
- Un support de communication qui facilite la représentation et la compréhension
  - Notation graphique : facilite la comparaison et l'évaluation de solutions
  - Notation rigoureuse : limite les ambiguïtés et les incompréhensions
  - Notation abstraite : indépendante des langages de programmation, domaines d'application et processus de développement

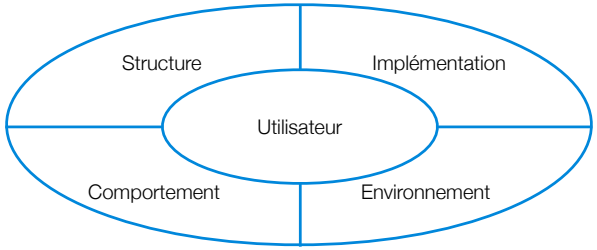
UML comme cadre d'analyse

- L'analyse et la conception se font graduellement par l'élaboration et le raffinement de modèles
  - Pas de barrière stricte entre analyse et conception
  - Les modèles d'analyse et de conception ne diffèrent que par leur niveau d'abstraction (ajout de détails)
- Approprié pour une approche de développement incrémentale et itérative

Diagrammes UML et vues

- Modèle UML: Ensemble de diagrammes décrivant le logiciel développé
  - Le modèle UML d'un logiciel peut être étudié sous différentes perspectives ou **vues**
    - ex: vue comportementale, vue structurelle
  - Différents types de diagrammes peuvent être regroupés pour offrir différentes vues

Vues UML



Vue utilisateur

- Définit les buts et objectifs des utilisateurs du logiciel
- Définit les besoins (services & contraintes) requis par la solution
- Vue unificatrice des autres vues en ce sens qu'elle sert de référence à leur validation

Vue structurelle

- Décrit les aspects statiques représentant la structure du logiciel.
- Identifie les éléments du domaine (classes, attributs, paquetages) et les relations (association, compositions, dépendance) entre eux

Vue comportementale

- Décrit les aspects dynamiques et le comportement du logiciel
- Spécifie les interactions et collaborations entre éléments du système
- Montre la décomposition du logiciel en termes de processus, d'interactions entre processus, de synchronisation et de communication entre activités

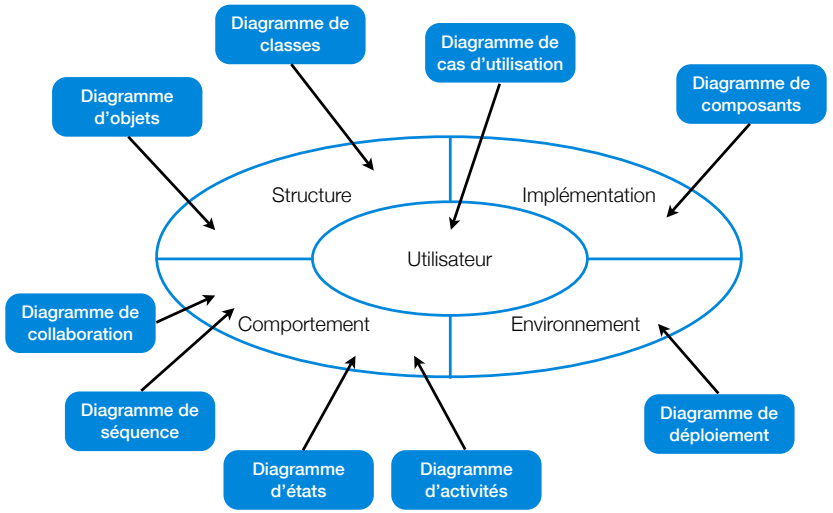
Vue implémentation

- Décrit les aspects de structure et de comportement de la solution
- Décrit la réalisation, l'organisation en composants, les contraintes de développement

Vue environnementale

- Décrit les aspects de structure et de comportement du domaine dans lequel la solution est réalisée
- Décrit les ressources matérielles (disposition, nature, performance) et comment elles sont utilisées par le logiciel

Diagrammes UML



Diagrammes UML

- Deux grandes catégories de diagrammes
  - Diagrammes décrivant les **aspects statiques**
  - Diagrammes décrivant les **aspects dynamiques**

Diagrammes statiques

- **Diagramme de classes** : décrit les “données” du logiciel (structure, traitements, relations, contraintes, rôles)
- **Diagramme d’objets** : diagramme de classes instanciées utilisé pour illustrer un exemple particulier
- **Diagramme de composants** : montre l'architecture physique du logiciel et l'affectation des objets aux différents composants de cette architecture
- **Diagramme de déploiement** : montre la configuration des différents composants à l'exécution et leur distribution

Diagrammes dynamique

- **Diagramme de cas d’utilisation** : montre à un haut niveau d'abstraction une collection de cas d'utilisation caractérisant le comportement de tout le logiciel
- **Diagramme de séquence** : montre l'échange de messages entre objets en fonction du temps
- **Diagramme de collaboration** : s'intéresse à la structure de collaboration entre les objets (séquence, itération, concurrence)
- **Diagramme d’états** : permet de décrire le comportement dynamique d'un objet (changements d’états)
- **Diagramme d’activités** : montre l'ensemble des traitements associés à une classe, une opération ou à un cas d'utilisation (workflow)

Concepts OO (rappel)

Objet

- Caractéristiques
  - État : données enregistrées dans l’objet
  - Opérations : comportement supporté par l’objet
  - Identité : permet de distinguer différents objets

Classe

- Description générale d'un ensemble d'objets
  - Attributs (état)
  - Opérations (comportements)
- Partie publique : interface
- Partie cachée : implémentation
- Instance : objet créé à partir de la classe

Envoi de message (*message passing*)

- Communication entre deux instances
  - Instance émettrice
  - Nom de l'opération
  - Arguments et paramètres de sortie

Héritage (*inheritance*)

- Relation entre deux classe
  - Enrichissement
  - Substitution
- Par exemple, en Java

```
class FileInputStream extends InputStream
{
    ...
}
```


Portée

- Les membres (attributs, opérations) peuvent appartenir
  - aux instances : attributs / méthodes d'instances
  - à la classe : attributs / méthodes de classes
    - `static` en Java

Polymorphisme

- Permet de manipuler différents types de données à l'aide d'une interface uniforme
- Par exemple, en Java, une instance de la class `ArrayList` peut être manipulée comme `List` ou `Collection`

Liaison dynamique (*dynamic dispatch*)

- Un même message peut déclencher des opérations différentes selon le type de l'instance auquel il est destiné
  - La méthode spécifique qui est appelée n'est déterminée qu'au moment de l'exécution

Surcharge (*overloading*)

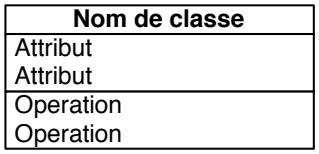
- Dans une classe, un même nom est employé pour définir deux opérations de signatures différentes
- Par exemple, en Java :
  - `String.valueOf(boolean b)`
  - `String.valueOf(char c)`
  - `String.valueOf(Object obj)`

Diagrammes de classes

Diagramme de classes

- Permet de décrire la structure statique du logiciel
  - **Classes** : attributs, opérations
  - **Relations** : associations, agrégations, compositions, héritage, etc.

Représentation des classes



Attributs

`[visibilité] [portée] <nom> : <type> [= <valeur_initiale>]`

- Type
  - primitif : `boolean, real, integer, string, date`
  - Types définis par une classe ou interface
  - NB: un modèle ne devrait pas comporter d'attribut de type classe, ces attributs (références sur des instances) sont représentés par des associations
- Portée
  - $ ou nom souligné: attribut de classe, sinon attribut d'instance

Visibilité

- Représentée par un des 4 symboles : +, –, # ou ~
  - Public (+) : élément visible par toutes les instances de toutes les classes
  - Private (–) : élément visible que par les instances de la classe
  - Protected (#) : élément visible par toutes les instances de la classe et de ses sous-classes
  - Package (~) : élément visible par les classes du même paquetage

Opérations


```
[visibilité] [portée] <nom> (<params>) : <type_resultat>
```

- Paramètres
  - `<direction> <nom> : <type> [= valeur_initiale]`
  - `<direction> := in | out | inout`

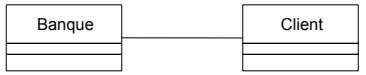
Classe - exemple

Rectangle
- length: integer - width: integer - pos: Point ~ rectangleCount : integer
+ display(canvas:Canvas) : void + remove() : void + setPosition(newPos:Point) : void

Relations

Association

- Lien / connexion entre deux instances
  - Habituellement implémentée à l'aide d'attributs d'instances
  - Représentée visuellement par une ligne entre deux classes

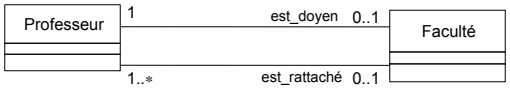


- Spécification
  - Cardinalité
  - Rôle(s)

Cardinalité

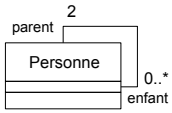
- Précise le nombre d'instances participantes
- *min*..*max*
  - *min* et *max* sont des entiers
  - *max* peut être non borné (∗)
- Par défault, 1 si aucune cardinalité n'est spécifiée
- Exemples
  - 6 : implique exactement six instances
  - 1..5 : entre une et cinq instances
  - 2..∗ : implique deux ou plusieurs instances
  - ∗ : équivalent à 0..∗

Cardinalité



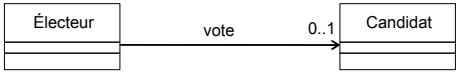
Noms de rôles

- Spécifie la fonction d'une classe (de ses instances) pour une association
  - Chaque extrémité peut être annotée avec un **rôle** (optionnel)
- Information indispensable pour les associations réflexives



Navigabilité restreinte

- Par défaut, une association est bidirectionnelle
- Pour réduire la portée de l'association et indiquer que les instances d'une classe ne peuvent être identifiées par les instances de l'autre, la navigabilité est restreinte



  - À partir d'un électeur, on peut directement identifier le candidat pour lequel il a voté.
  - À partir d'un candidat, on ne peut pas retrouver directement les électeurs qui ont voté pour lui.

Agrégation

- Association qui exprime une union marquée et une relation de subordination entre deux instances (agrégat, instance agrégée)
- Inclusion « possède » ou « a pour partie »
- Une instance agrégée peut être impliqué dans plus d'une association
  - N'implique pas d'unicité
- Une instance agrégée peut exister sans son agrégat et inversement, i.e., les cycles de vie de l'agrégat et de l'instance agrégée sont indépendants
  - N'implique pas de dépendance existentielle

Agrégation - exemple

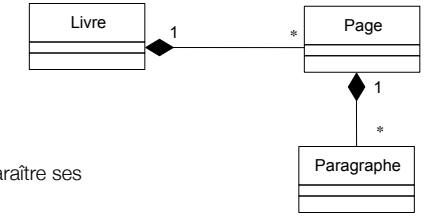
Composition

- Agrégation forte entre deux instances (un composite et un composant)
- Inclusion « est composé (physiquement) de »
- À un même moment, un composant ne peut appartenir qu'à un seul composite
  - Unicité exigée
- Si le composite est détruit (ou copié), ses composants le sont aussi
  - Implique une dépendance existentielle
  - Les composants peuvent être détruits avant le composite

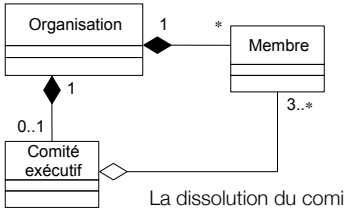
Composition - exemple

Composition vs association

La destruction d'un livre entraîne la destruction de ses pages et des paragraphes de texte qu'elles contiennent

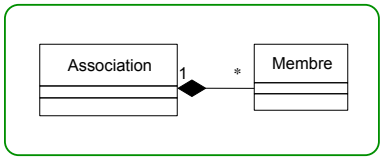


La fin d'une organisation fait disparaître ses membres et son comité exécutif.

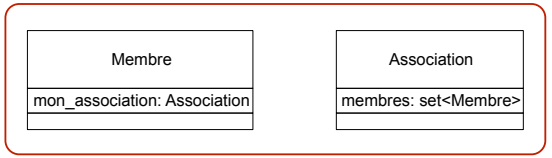


La dissolution du comité n'engendre pas l'exclusion de ses membres.

Modélisation vs implémentation

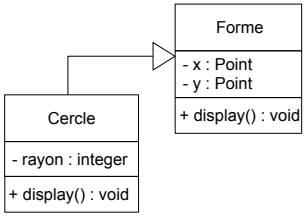


et non pas



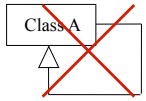
Généralisation

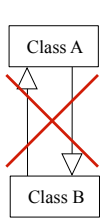
- Relation d'héritage entre un élément de description générale et un élément plus spécifique (cohérent)
  - "est un(e)"

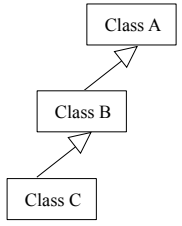


Généralisation

- Relation
  - Non réflexive
  - Non symétrique
  - Transitive

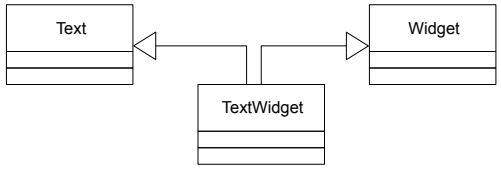






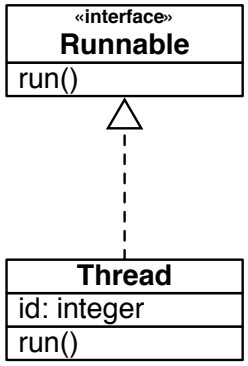
Type d'héritage

- Une super-classe peut avoir plusieurs sous-classes
  - héritage (généralisation) simple
- Une sous-classe peut avoir plusieurs super-classes
  - héritage (généralisation) **multiple**
  - le langage d'implémentation peut ne pas supporter l'héritage multiple



Autres concepts

Réalisation



Notes