Introduction à UML

Cours PROG
ESIR Informatique-Télécom

Rémi Cozot

UMR IRISA

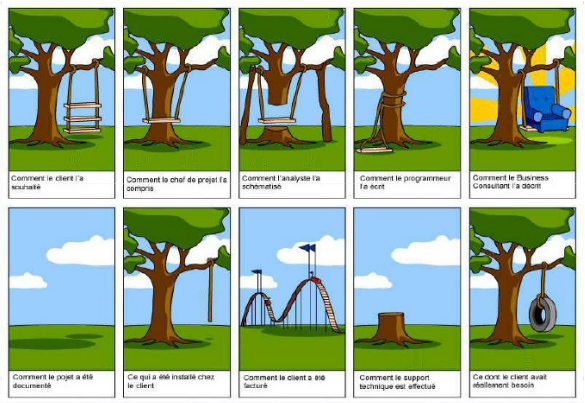
Plan

- Introduction
- Modélisation : classification relation
- UML : les diagrammes
- Cas d'utilisation
- Diagramme de Classes
- États transitions
- Séquences
- Conclusion

Introduction

- Projet informatique ou autre
  - Besoin de spécifier
    - Les besoins
    - La solution envisagée
  - Solutions
    - Cahier des charges
      - Laisse trop de place à l'interprétation
  - Besoin d'un outil formel
    - Généraliste

Introduction



Introduction

- Spécifier, modéliser
  - Pour qui ?
  - Pour le client
    - Comment exprimer son besoin
  - Pour l'informaticien
    - Pour concevoir la solution répondant au besoin
- Si l'outil n'est pas commun entre le client et le fournisseur, cela entraine des erreurs
  - Outil : vocabulaire, schéma, etc.
  - UML : cadre formel « simple »

Plan

- Introduction
- Modélisation : classification relation
- UML : les diagrammes
- Cas d'utilisation
- Diagramme de Classes
- États transitions
- Séquences
- Conclusion

Modélisation : classification relations

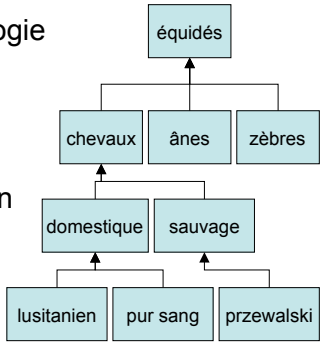
- Représentation un « problème »
  - Lister les « acteurs »
  - Mettre en évidence les relations
- Des acteurs partagent des propriétés communes
  - Généraliser / spécialiser : classifier
  - Approches communes aux sciences

Modélisation : classification relations

- Exemple : bibliothèque
  - Lister les acteurs, généraliser
    - Livre, usagé
    - Emprunt
    - Date d'emprunt, de retour
  - Relations
    - Usagé (Recherche) livre
    - Usagé (Emprunte) livre
    - Usagé (Rend) livre
    - Emprunt (A) date d'emprunt
    - Emprunt (A) date de retour

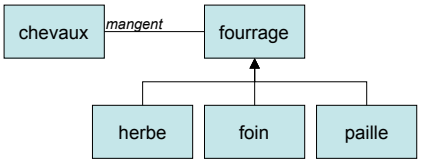
Modélisation : classification relations

- Classification
  - Exemple biologie / zoologie
    - Équidés
      - Chevaux
      - Ânes
      - Zèbres
  - Classification en fonction du but
    - Sauvage
    - Domestique



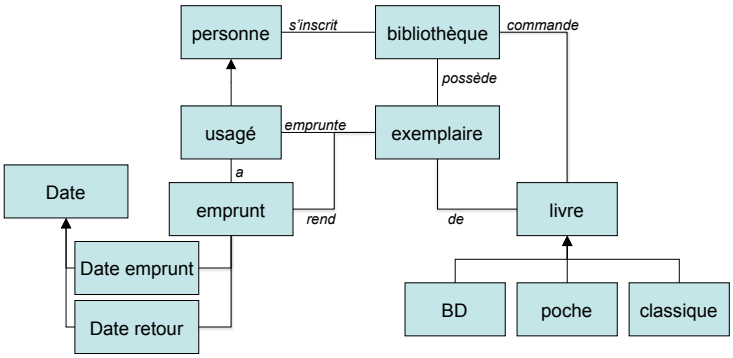
Modélisation : classification relations

- Relation autres que la classification
  - Exemple
    - Chevaux mangent fourrage



Modélisation : classification relations

- Exemple : bibliothèque



Modélisation : classification relations

- Exemple : bibliothèque
  - Des boîtes des flèches : manque d'un formalisme précis
  - Dimensions du modèle
    - Statique
    - Dynamique
  - Vues
    - De l'utilisateur
    - Architecture logicielle
    - Déploiement
  - Granularité
    - Niveau de détail du modèle
      - Dans l'exemple : quel est l'intérêt de la spécialisation
        - » BD, poche, classique
- UML propose un cadre formel et ouvert

Plan

- Introduction
- Modélisation : classification relation
- UML : les diagrammes
- Cas d' utilisation
- Diagramme de Classes
- États transitions
- Séquences
- Conclusion

UML : Les diagrammes

- Une seule vue ne suffit pas
  - Dimension structurelle statique
    - Classe, objet
  - Dimension dynamique
    - Utilisation
    - États (automates)
    - Dimension des interactions
      - Communication et séquences

UML : Les diagrammes

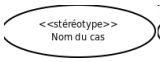
- Diagrammes structurels ou diagrammes statiques (UML Structure)
  - ***diagramme de classes*** (Class diagram)
  - diagramme d'objets (Object diagram)
  - diagramme de composants (Component diagram)
  - diagramme de déploiement (Deployment diagram)
  - diagramme de paquetages (Package diagram)
  - diagramme de structures composites (Composite structure diagram)
- Diagrammes comportementaux ou 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)

Plan

- Introduction
- Modélisation : classification relation
- UML : les diagrammes
- Cas d' utilisation
- Diagramme de Classes
- États transitions
- Séquences
- Conclusion

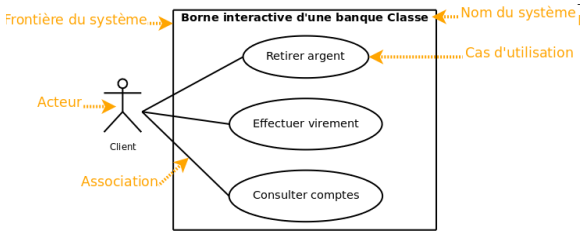
Cas d'utilisation – Use Case

- Expression des besoins
  - recueillir, analyser, organiser les besoins
- Éléments de diagramme
  - Acteur : personne externe, un processus ou une chose qui interagit avec un système
  - Cas d'utilisation : une unité cohérente représentant une fonctionnalité visible de l'extérieur



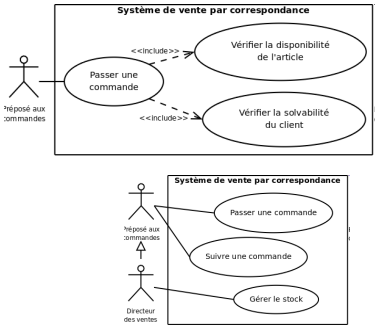
Cas d'utilisation – Use Case

- Exemple

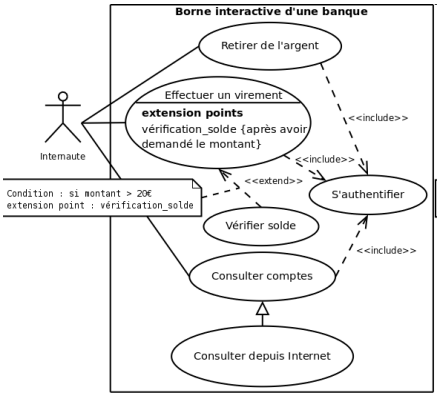


Cas d'utilisation – Use Case

- Relations entre cas
  - Inclusion
    - A inclut B lorsque A appelle B pour sa réalisation.
  - Extension
    - A étend B lorsque le cas d'utilisation A peut être appelé au cours de l'exécution du cas d'utilisation B. L'extension est optionnelle.
  - Généralisation / spécialisation
- Relations entre acteurs
  - Généralisation / spécialisation



Cas d'utilisation – Use Case



Plan

- Introduction
- Modélisation : classification relation
- UML : les diagrammes
- Cas d'utilisation
- Diagramme de Classes
- États transitions
- Séquences
- Conclusion

Diagramme de Classes

- Digramme le plus important
  - Structure interne qui fournit une représentation abstraite des classes (objets) du système qui réalisent les cas d'utilisation
- Éléments de digramme
  - Classe

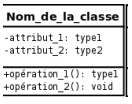


Diagramme de Classes

- Relations entre classes
  - Héritage / généralisation / spécialisation
    - La classe enfant possède toutes les caractéristiques des ses classes parents.
    - Une classe enfant peut redéfinir plusieurs méthodes de la classe parent
    - Toutes les associations de la classe parent s'appliquent aux classes enfants

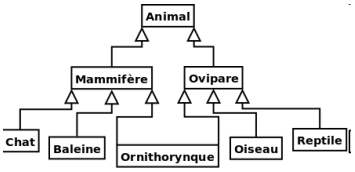


Diagramme de Classes

- Relations entre classes
  - Composition
    - A est composé de B
    - Si A disparaît alors B disparaît également
  - Agrégation

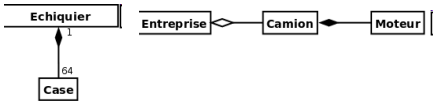
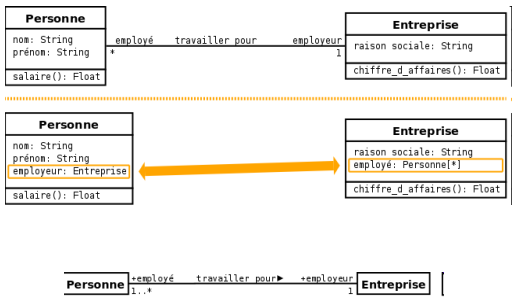


Diagramme de Classes

- Relations entre classes
  - Association



Plan

- Introduction
- Modélisation : classification relation
- UML : les diagrammes
- Cas d'utilisation
- Diagramme de Classes
- États transitions
- Séquences
- Conclusion

Diagramme d'états /transitions

- Rôle : spécifier le comportement logique interne d'un objet d'une classe sous forme d'un automate à états finis
- Éléments du diagramme
  - État

    - Initial
    - Final

  - Transition et évènement

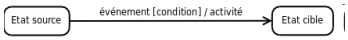
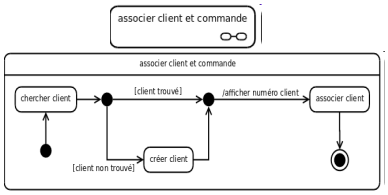


Diagramme d'états /transitions

- Plus de détails
  - État composite
    - État encapsulant un automate
  - Historique



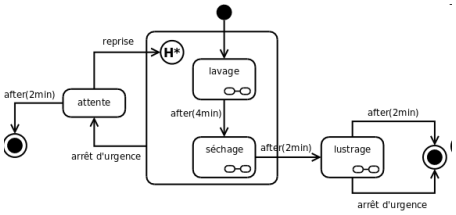


Diagramme d'états /transitions

- Plus de détails
  - Concurrence
    - Des états fonctionnant en parallèle

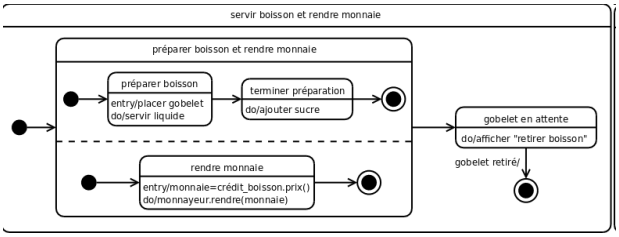
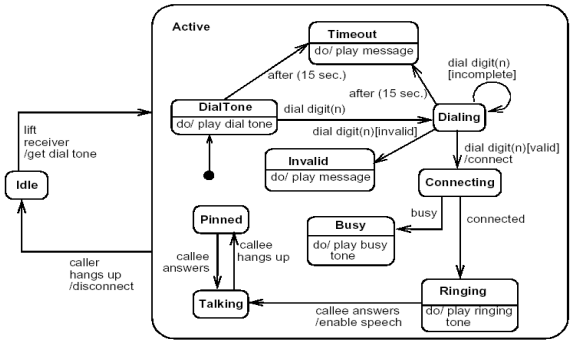


Diagramme d'états /transitions



Plan

- Introduction
- Modélisation : classification relation
- UML : les diagrammes
- Cas d'utilisation
- Diagramme de Classes
- États transitions
- Séquences
- Conclusion

Diagramme de séquences

- Rôle
  - Précise les messages échangés (appel de méthodes) entre les objets dans un ordre chronologiques
  - Décrit « comment » et par qui les cas d'utilisation sont réalisés
- Éléments
  - Objet (éventuellement acteur)
  - Message / signal

Diagramme de séquences

- Message
  - Synchrone
    - Méthode avec retour
  - Asynchrone (signal)
  - Création / destruction

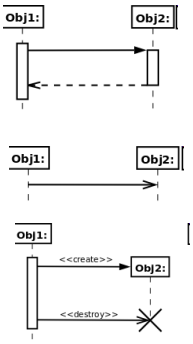


Diagramme de séquences

- Messages
  - A appelle B
  - A appelle B
    - B appelle A

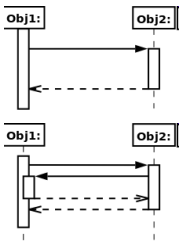


Diagramme de séquences

- Exemple : jeux démineur

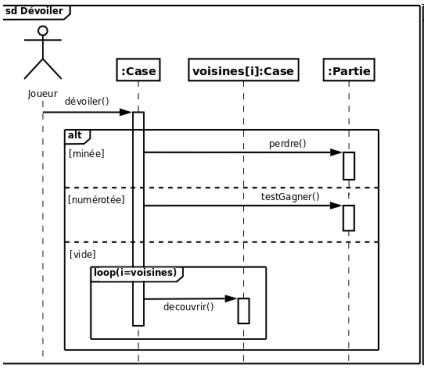
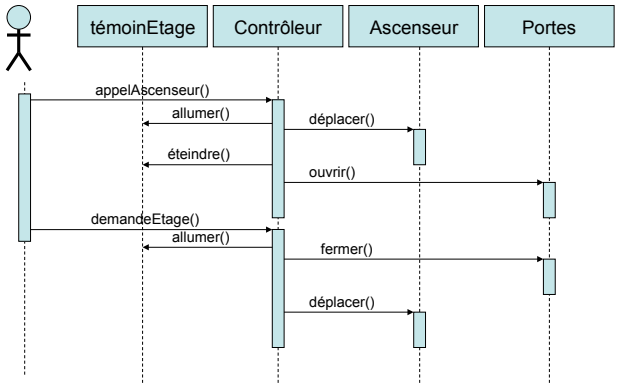


Diagramme de séquences



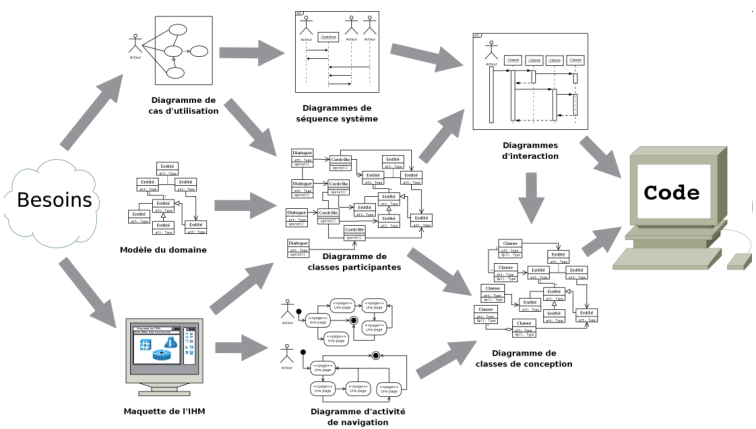
Plan

- Introduction
- Modélisation : classification relation
- UML : les diagrammes
- Cas d'utilisation
- Diagramme de Classes
- États transitions
- Séquences
- Conclusion

Conclusion

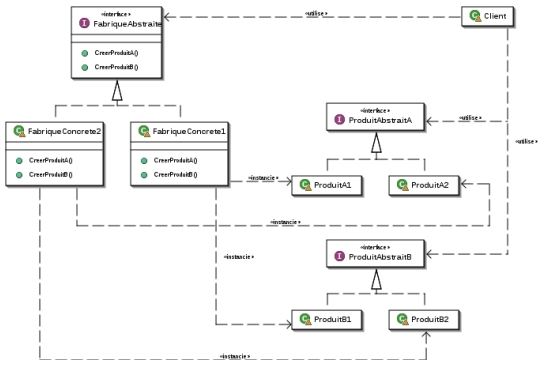
- UML est un outil généraliste
  - Informatique
  - Gestion / économie / finances
  - Création
    - SupInfoGame
  - Médecine
- UML ne fait pas tout
  - Nécessite une méthodologie

Conclusion



conclusion

- Forme type de conception : patron de conception



Conclusion

- Forme type de conception : patron de conception

```
/* * GUIFactory Example */
public abstract class GUIFactory {
        public static GUIFactory getFactory() {
                int sys = readFromConfigFile("OS_TYPE");
                if (sys == 0) { return (new WinFactory()); } else { return (new OSXFactory()); } }
        public abstract Button createButton();}

class WinFactory extends GUIFactory {  public Button createButton() { return (new WinButton()); } }
class OSXFactory extends GUIFactory { public Button createButton() { return (new OSXButton()); } }

public abstract class Button {
        private String caption;
        public abstract void paint();
        public String getCaption(){ return caption; }
        public void setCaption(String caption){ this.caption = caption; }}

class WinButton extends Button { public void paint() { System.out.println("I'm a WinButton: "+ getCaption()); } }
class OSXButton extends Button { public void paint() { System.out.println("I'm a OSXButton: "+ getCaption()); } }

public class Application { public static void main(String[] args) {
        GUIFactory aFactory = GUIFactory.getFactory();
        Button aButton = aFactory.createButton();
        aButton.setCaption("Play"); aButton.paint(); }
```

MERCI ☺