UML
Unified Modeling Language

M1 MIAGE - SIMA - 2006-2007
Yannick Prié
UFR Informatique - Université Claude Bernard Lyon 1

Objectifs de ce cours

- Présentation générale de UML
  - historique
  - principes généraux
- Présentation des différents types de diagrammes

UML en un transparent

- Unified Modelling Language
- Unification
  - de nombreux langages de modélisation graphique OO des années 1990,
  - de diagrammes et de principes de modélisation à succès
- Défini par l'OMG (Object Management Group)
- Définit un méta-modèle et des types de diagrammes

Plan du cours

- Introduction à UML
- Généralités sur la notation
- Diagrammes de classes, objets, packages
- (Diagrammes de) cas d'utilisation
- Diagrammes d'interaction
- Diagrammes d'activité
- Diagrammes de machines d'états
- Diagrammes de composants et de déploiement
- Autres diagrammes UML
- Autres diagrammes non UML
- Autres points liés à UML

Plan du cours

- **Introduction à UML**
- Généralités sur la notation
- Diagrammes de classes, objets, packages
- (Diagrammes de) cas d'utilisation
- Diagrammes d'interaction
- Diagrammes d'activité
- Diagrammes de machines d'état
- Diagrammes de composants et de déploiement
- Autres diagrammes UML
- Autres diagrammes non UML
- Autres points liés à UML

Un foisonnement de méthodes

- Fin 80 / début 90
  - orientation de plus en plus marquée vers l'objet
- Conséquence naturelle, mise en place de méthodes
  - OOD : Object Oriented Design (Booch, 1991)
  - HOOD : Hierarchical Object Oriented Design (Delatte et al., 1993)
  - OOA : Object Oriented Analysis (Schlaer, Mellor, 1992)
  - OOA/OOD : (Coad, Yourdon, 1991)
  - OMT : Object Modeling Technique (Rumbaugh, 1991)
  - OOSE : Object Oriented Software Engineering (Jacobson, 1992)
  - OOM : Object Oriented Merise (Bouzeghoub, Rochfeld, 1993)
  - Fusion (Coleman et al., 1994)
- Bilan
  - de nombreuses méthodes (>50)
  - ayant des avantages et des inconvénients
  - des concepts assez proches, des notations différentes

Vers une unification

- 1994
  - tentative de normalisation de l'OMG, sans effet
  - Rumbaugh (OMT) rejoint Booch (OOD) chez Rational Software
    - objectif : créer une *méthode* en commun (méthode unifiée)
- 1995
  - présentation de la version 0.8 de la *méthode*
  - arrivée de Jacobson (OOSE) chez Rational
- 1996
  - implication de l'OMG (sous pression des industriels pour favoriser l'interopérabilité des modèles)
  - *langage* unifié UML 0.9 (Unified Modeling Language),
- 1997
  - UML 1.0 sort chez Rational
  - UML 1.1 adopté par l'OMG comme standard officiel

Evolutions d'UML

- 1997-2003
  - adoption par les entreprises
  - UML 1.1 à UML1.5 : modifications/améliorations
- 2005
  - UML 2.0
  - quelques nouveaux diagrammes
  - changements importants au niveau du méta-modèle, pour permettre d'utiliser UML pour la programmation

Unified Modeling Language

- Combinaison de principes à succès
  - modélisation de données (E/A), de l'activité, objet, en composants…
- Objectif
  - visualiser / spécifier / construire / documenter les artefacts de la conception d'une application
- La norme elle-même
  - méta-modèle et familles de diagrammes
- Utilisation
  - pas de méthode préconisée
  - pas de spécification technologique

Objectifs d'UML

- *Montrer les limites d'un système et ses fonctions principales* (pour les utilisateurs) à l'aide des cas d'utilisation et des acteurs
- Illustrer les réalisations de CU à l'aide de diagrammes d'interaction
- *Modéliser la structure statique d'un système* à l'aide de diagrammes de classes, associations, contraintes
- *Modéliser la dynamique, le comportement des objets* à l'aide de diagrammes de machines d'états
- *Révéler l'implantation physique de l'architecture* avec des diagrammes de composants et de déploiement
- Possibilité d'étendre les fonctionnalités du langage avec des stéréotypes
- Un langage utilisable par l'homme et la machine : permettre la génération automatique de code, et la rétro-ingénierie

Modèles, vue et diagrammes UML

- Modèle
  - abstraction d'un système composée d'un ensemble d'éléments de modèle
  - ce qui est construit par et ce qui est perçu au travers des diagrammes (par le concepteur, le lecteur)
  - conforme au méta-modèle UML
- Vue
  - projection d'un modèle suivant une perspective qui omet les éléments non pertinents pour cette perspective. Elle se manifeste dans des diagrammes
  - ex. : vue statique, vue fonctionnelle…
- Diagramme
  - présentation graphique d'éléments de visualisation représentant des éléments de modèle (graphe)
  - ex. : diagramme de classes, de séquences…

(Larman 2005)

Exemples de diagrammes

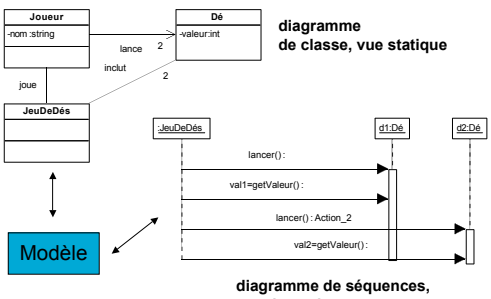
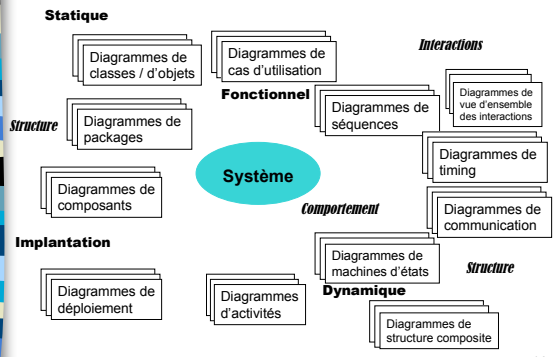


diagramme de classe, vue statique

diagramme de séquences, vue dynamique

Panorama des diagrammes



(Fowler/Mellor)

3 modes d'utilisation d'UML

- Esquisse
  - conception / communication
  - incomplétude
- Plan
  - exhaustivité
  - outils bidirectionnels
- Programmation
  - model Driven Architecture / UML exécutable
  - implantation automatique
  - réaliste ?

Focus sur les diagrammes

Focus sur le méta-modèle

Conception et UML

- Différentes façons de voir UML : différentes façons de penser
  - la conception
  - l'objectif et l'efficacité d'un processus de génie logiciel
- donc
  - essayer de comprendre le point de vue de l'auteur pour chaque publication / site sur UML
- UML n'est pas une méthode…
- … mais des principes de conception orientée objet sont sous-jacents
  - aux diagrammes
  - aux façons de les présenter
- donc
  - difficile de présenter uniquement les diagrammes
  - on parlera aussi de méthode, de bonnes pratiques

Généralités sur les méthodes OO

- Grandes caractéristiques
  - itératives (*vs* cascade)
    - analyse et conception tout au long du projet, pas seulement au début
  - centrées sur les cas d'utilisation
    - besoins réels
  - centrées sur l'architecture
- Découpage d'un projet en activités
  - besoins : comprendre dans quoi s'insère le système et ce qu'il doit faire
  - analyse : fonctionnement du système à hau niveau
  - conception : fonctionnement logiciel
  - réalisation : codage
  - tests, déploiement…

À propos de cette présentation

- Présentation de UML 2, quelques points de UML 1
  - faites attention à la syntaxe quand vous rencontrez un diagramme
- Présentation = synthèse de nombreuses lectures
  - mixe syntaxe et utilisation
  - synthèse personnelle des bonnes pratiques présentées
- Présentation d'UML non exhaustive
  - ce cours contient *beaucoup* de choses utiles
  - pour plus de précisions : livres de référence
  - pour la description exacte (syntaxe et sémantique) : http://www.omg.org/uml
- UML et le web
  - beaucoup de sites web parlent d'UML
  - on trouve du bon et du moins bon

UML et la règle

- Deux types de règles pour l'utilisation d'UML
  - normatives
    - comment il faut faire, comité d'experts : *norme*
  - descriptives
    - comment les gens font, usages, modes : *conventions* dans l'utilisation
    - peuvent être en contradiction avec la norme (surtout pour UML2)
- Règles
  - utiliser le sous-ensemble d'UML qui vous convient
  - droit de supprimer n'importe quel élément d'un diagramme
  - droit d'utiliser n'importe quel élément d'un diagramme dans un autre
    - ce qui compte pour les auteurs d'UML, c'est le méta-modèle, pas les diagrammes
  - liberté de dessiner ce que l'on veut
    - surtout en mode esquisse, sur papier ou au tableau

Plan du cours

- Introduction à UML
- **Généralités sur la notation**
- Diagrammes de classes, objets, packages
- (Diagrammes de) cas d'utilisation
- Diagrammes d'interaction
- Diagrammes d'activité
- Diagrammes de machines d'état
- Diagrammes de composants et de déploiement
- Autres diagrammes UML
- Autres diagrammes non UML
- Autres points liés à UML

Mots-clé

- Objectif
  - regrouper en familles des éléments similaires d'un modèle
  - pour ne pas multiplier les symboles différents dans les diagrammes
- Ornements textuels
  - associés à des éléments du modèle
  - certains mots-clé sont prédéfinis par UML
- Notation
  - « mot-clé »
  - ex. « abstract »

Valeurs étiquetées

- Objectif
  - attacher une information arbitraire à un élément de modélisation
- Paires (nom,valeur)
  - associées à des éléments de diagramme
- Notation : nom=valeur
  - ex. : auteur=YP, version=1.3

Stéréotypes

- Étendre le méta-modèle
  - définir des profils de valeurs étiquetées pour des éléments de modélisation
  - plus formels que les mots-clé
- Notation : « stéréotype »
  - ex. « gestion » avec des valeurs étiquetées associées si nécessaire
- Certains sont prédéfinis par UML
- Possibilité d'associer une icône
  - forme visuelle déterminée
  - ex. : pour « control »

Contraintes

- Relation sémantique quelconque
  - concernant un ou plusieurs éléments du modèle
  - définissant des propositions devant être maintenues à *Vrai* pour garantir la validité du système modélisé
- Notation : {contrainte}
  - contenu formel ou informel
  - à côté des éléments concernés
  - ex. {frozen}, {jamais détruit !}, {x - y < 10}
- Certaines sont prédéfinies
  - ex. xor, ordered
- D'autres créées par l'utilisateur
  - langue, pseudo-code, OCL…

Commentaires

- Commentaire
  - annotation quelconque associée à un élément du modèle
  - pas de sémantique pour le modèle
- Notation : note
  - rectangle avec coin replié, lien pointillé avec l'élément de visualisation concerné
  - cercle en bout de ligne : plus précis
- Il existe des mots-clé prédéfinis
  - ex. « besoin », « responsabilité »

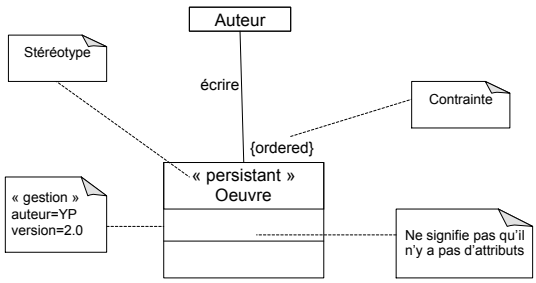
Dépendance

- Relation sémantique faible
  - relation d'utilisation unidirectionnelle entre deux éléments
  - relation sémantique non structurelle entre client et fournisseur
- Notation
  - flèche pointillée de l'élément source vers l'élément cible, éventuellement stéréotype

Diagrammes

- Diagrammes
  - éléments de dessin dont on dispose
    - formes nœuds et relation de graphe
    - formes conteneurs
    - texte
- Principe
  - la sémantique d'UML impose de conserver
    - graphe
    - contenant / contenu
    - proximité
  - liberté pour le reste (positions…)
  - n'importe quelle information peut être supprimée dans un diagramme
    - pas de déduction due à l'absence d'un élément

Exemple général



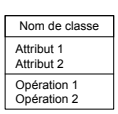
Plan du cours

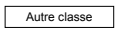
Diagrammes de classes : présentation générale

- Diagrammes fondamentaux
  - les plus connus, les plus utilisés
- Présentent la vue statique du système
  - représentation de la structure et des déclarations comportementales
  - classes, relations, contraintes, commentaires…
- Permettent de modéliser plusieurs niveaux
  - conceptuel (domaine, analyse)
  - implémentation (code)

Classes

- Descripteurs de jeux d'objets
  - structure / comportement / relations / sémantique communs
- Représentation
  - rectangle à trois compartiments
    - nom
    - attributs
    - opérations
  - plus ou moins de détails suivant les besoins
- Nom : singulier, majuscule (en général)
  - ex. : Fichier, Client, Compte, Chat

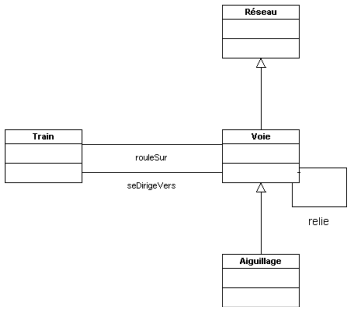




Relations entre classes/ liens entre objets

- Association
  - les instances des classes sont liées
  - possibilité de communication entre objets
  - relation forte : composition
- Généralisation/spécialisation
  - les instances de la sous-classe sont des instances de la super-classe (niveau conceptuel)
  - héritage (niveau implémentation)
- Dépendance
  - la modification d'une classe peut avoir des conséquences sur une autre
- Réalisation
  - une classe réalise une interface

Un exemple



Utilisation des diagrammes de classes

- Expression des besoins
  - modélisation du domaine
- Conception
  - spécification : gros grain
- Construction
  - implémentation : précis
  - rétro-ingénierie

Petit exercice

- Dessiner un diagramme de classe du domaine avec les classes suivantes
  - étudiant
  - enseignant
  - cours
  - salle de classe

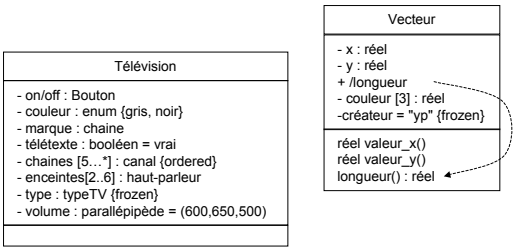
Attributs

Visibilité nom : type [multiplicité] = valeur_initiale {propriétés}

- public + privé - protégé # package ~
- Facultatif mais impératif pour l'implémentation
- Facultatif ex. couleurs : Saturation [3] points : Points [2..*]
- Facultatif
- Facultatif ex. {frozen} mise à jour interdite {obligatoire} valuation oblig.

- Remarques
  - /nom : attribut dérivé (calculé)
  - souligné : attribut statique (de classe)
  - {frozen} : disparu de UML2 ; à utiliser quand-même

Attributs : exemple



Opérations de classes

visibilité nom (liste de paramètres) : type-retour {propriétés}

argument ::= direction nom : type = valeur-défaut

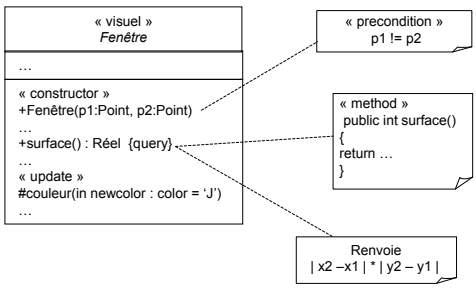
public **+**
privé **-**
protégé **#**
package **~**

in | out | inout

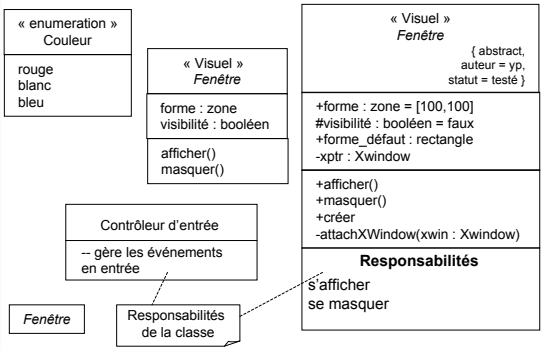
abstract
query
…

- Remarques
  - notation : *opération abstraite* / opération statique
  - opérations = comportement d'une classe, trouvées en examinant les diagrammes d'interaction
  - méthode = implémentation d'une opération dont elle spécifie l'algorithme ou la procédure associée
  - pré et post-conditions, description du contenu : commentaires + OCL

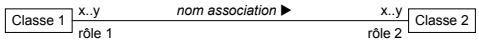
Opérations : exemple



Autres exemples de classes



Associations

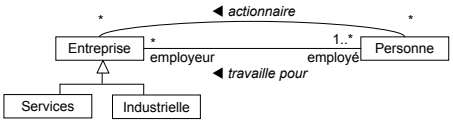


Nom : forme verbale, sens de lecture avec flèche

Rôles : forme nominale, identification extrémité association

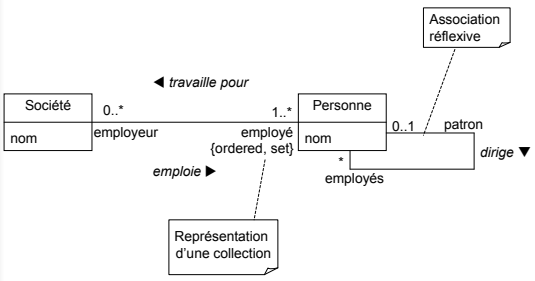
Multiplicité : 1, 0..1, 0..*, 1..*, n..m

Mots-clés : *set*, *ordered set* (uniques) ; *bag*, *list* (doublons)



Les associations ont une durée de vie, sont indépendantes les unes des autres, sont héritées, comme les attributs

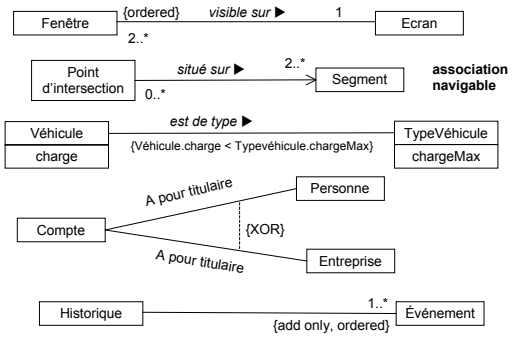
Associations : exemple



Associations : remarques

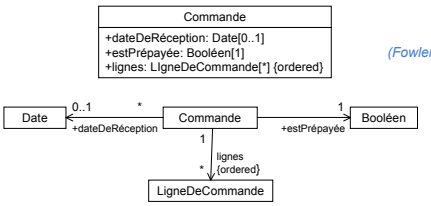
- Tout objet doit être accessible via un lien
  - ne peut recevoir de message sinon
  - liens plus ou moins permanents : voir "Visibilités"
- Multiplicité
  - nombre d'instances d'une classe en relation avec une instance d'une autre classe
  - pour chaque association
    - deux décisions à prendre : deux extrémités
- Directionnalité ⟷ ⟶
  - bidirectionnalité par défaut, evt explicitée
  - restriction de la navigation à une direction

Associations et contraintes



Propriétés
caractéristiques structurelles des classes

- Concept unique regroupant attributs et associations monodirectionnelles : équivalence des représentations
- Pour choisir
  - attribut (texte) pour les types de données
    - objets dont l'identité n'est pas importante
  - association pour insister sur les classes



(Fowler, 2004)

Agrégation et composition

- Associations asymétriques, fortes
- Agrégation
  - non nommée, structure d'arbre sous-jacente (pas de cycle), rôle prépondérant d'une extrémité

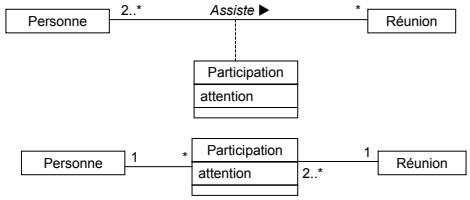
- Composition
  - non partage des éléments composants, création et destruction des composants avec le composite

Composition, agrégation et association

- Quelques questions à se poser
  - asymétrie et lien de subordination entre instances des deux classes (agrégation/composition) ou indépendance des objets (association) ?
  - propagation d'opérations ou d'attributs du tout vers les parties ? (agrégation/composition)
  - création et destruction des parties avec le tout ? (composition)
- Remarques importantes
  - dans le doute, toujours utiliser une association (moins contrainte)
  - pour certains auteurs importants, oublier l'agrégation
    - agrégation = placebo denué de sens

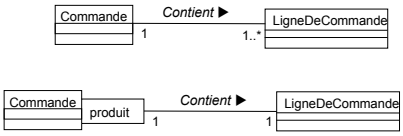
Classes d'association

- Pour ajouter attributs et opérations à des associations
- Quelques indices pour l'utilisation
  - un attribut est lié à une association
  - la durée de vie des instances de la CA dépend de l'association
  - association N..N entre deux classes + informations liées à l'association



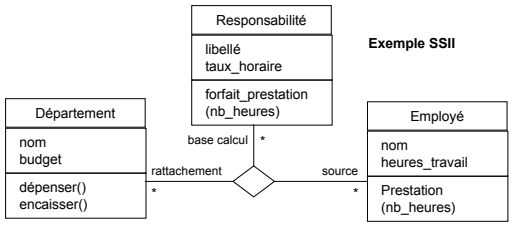
Associations qualifiées

- Equivalent UML des dictionnaires
- Sélection d'un sous-ensemble des objets qui participent à l'association à l'aide d'une clé.
  - cet attribut est propriété de l'association

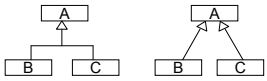


Association n-aire

- Groupe de liens entre au moins trois instances
- Instance de l'association = n-uplet des attributs des instances impliquées

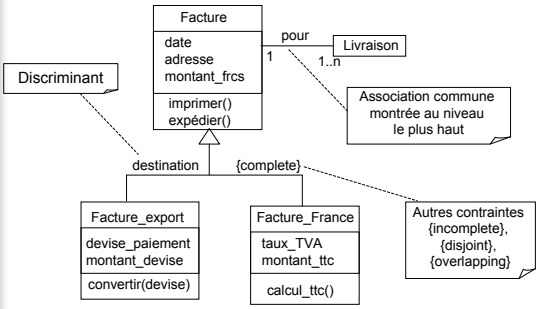


Généralisation spécialisation



- Deux interprétations
  - niveau conceptuel
    - organisation : un concept est plus général qu'un autre
  - implémentation
    - héritage des attributs et méthodes
- Pour une bonne classification conceptuelle
  - principe de substitution / conformité à la définition
    - toutes les propriétés de la classe parent doivent être valables pour les classes enfant
  - « A *est une sorte de* B » (mieux que « A *est un* B »)
    - toutes les instances de la sous-classe sont des instances de la super-classe (définition ensembliste)
- Spécialisation
  - relation inverse de la généralisation

Hiérarchie de classes



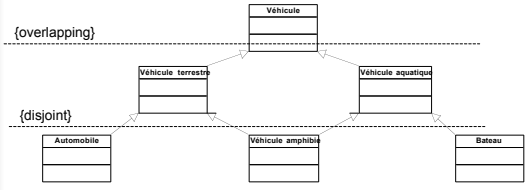
(Larman, 2005)

Conseils pour la classification conceptuelle

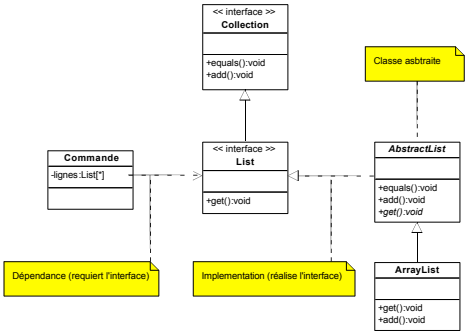
- Partitionner une classe en sous-classes
  - la sous-classe a des attributs et/ou des associations supplémentaires pertinents
  - par rapport à la superclasse ou à d'autres sous-classes, la sous-classe doit être gérée, manipulée, on doit agir sur elle ou elle doit réagir différemment, et cette distinction est pertinente
  - le concept de la sous-classe représente une entité animée (humain, animal, robot) qui a un comportement différent de celui de la superclasse, et cette distinction est pertinente
- Définir une super-classe
  - les sous-classes sont conformes aux principes de substitution et « sorte-de »
  - toutes les sous-classes ont au moins un même attribut et/ou une même association qui peut être extrait et factorisé dans la superclasse

Généralisation multiple

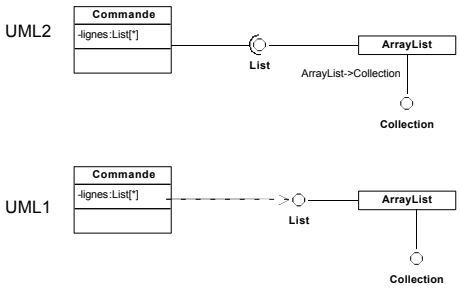
- Autorisée en UML
- Attention aux conflits : il faut les résoudre
- Possibilité d'utiliser aussi délégations ou interfaces



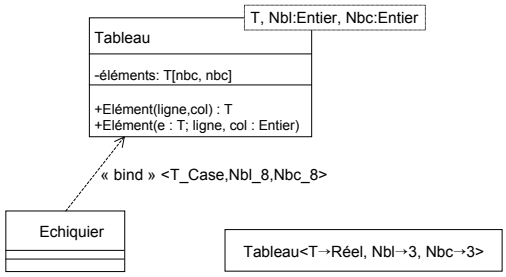
Interfaces et classes abstraites



Interface et utilisation : notation



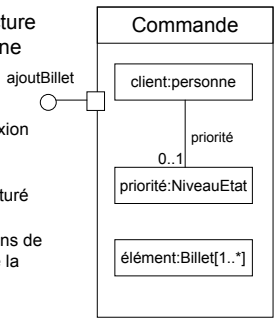
Classes paramétrables (templates)



Deux notations de la paramétrisation d'une classe paramétrable

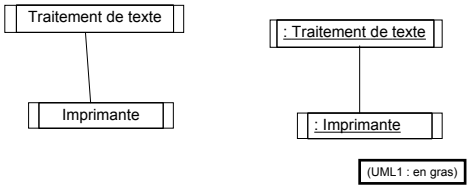
Classes structurées

- Description de la structure d'implémentation interne d'une classe
- Contient
  - ports : points de connexion avec l'environnement (evt. interne)
  - parties : fragment structuré de la classe
  - connecteurs : connexions de deux parties au sein de la classe



Classes actives

- Classe dont les instances sont des objets actifs
  - possèdent leur propre thread
  - dans un environnement multitâche

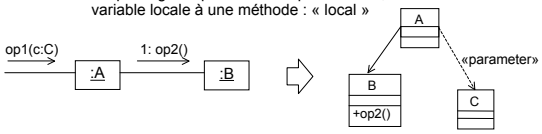


Diagrammes de classes et code objet

- A voir en TP et/ou à la fin du cours

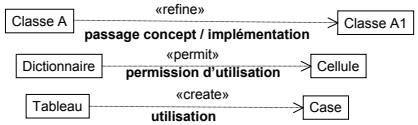
Liens et visibilité

- Possibilité d'envoyer un message d'un objet à l'autre
- Lien durable / visibilité attribut ou globale
  - permet l'envoi de message entre objets
  - se matérialise par une association navigable entre classes
- Lien temporaire / visibilité paramètre ou locale
  - résulte d'une utilisation temporaire d'un objet par un autre
  - se matérialise par une dépendance entre classes
    - ex. passage de paramètre : « parameter », variable locale à une méthode : « local »



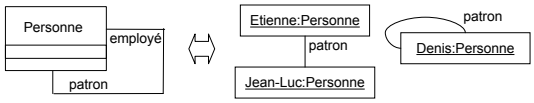
Relations de dépendance

- 3 grands types
  - abstraction : différents niveaux d'abstraction
    - ex. «refine», «trace», «derive»
  - permission d'utilisation (cf. friend en C++)
    - ex. «permit»
  - utilisation
    - ex. «use», «create», «call», «parameter»
- Conseil
  - utiliser une dépendance pour tout ce qui n'est pas spécifié

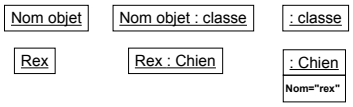


Diagrammes d'objets

- Pour représenter un instantané du système
  - les objets et leurs liens
  - objets = spécification d'instances
- Quand les utiliser ?
  - pour montrer un contexte
    - collaborations sans messages
  - quand une structure complexe est trop difficile à comprendre avec un diagramme de classe
    - ex. : récursivité, associations multiples, etc.



Objets



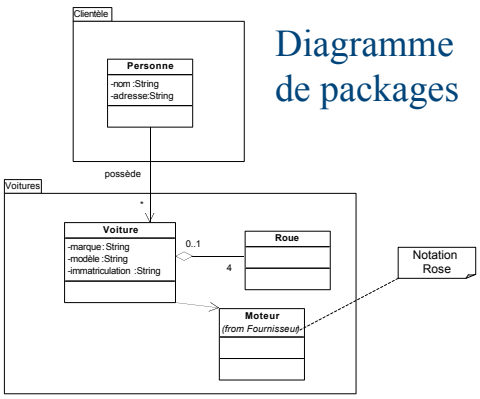
- Multi-objet (UML1)
  - modéliser un jeu
    - comme un objet unique avec des opérations sur le jeu
    - comme jeu d'objets individuels avec leurs opérations
  - utile pour les collections dans les diagrammes de communication (voir plus loin)
- UML2 : utiliser plutôt une classe structurée

Multi-objet UML1

Package

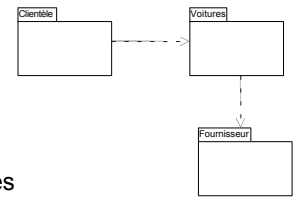
- Mécanisme général pour
  - organiser les éléments et les diagrammes du modèle (notamment les classes)
    - partitionner, hiérarchiser
    - clarifier
  - les nommer
    - un package définit un espace de nom
    - deux éléments ne peuvent avoir le même nom dans un package
- Un package
  - contient des éléments
    - y compris d'autres packages : hiérarchie de packages
  - peut en importer d'autres
  - peut posséder des interfaces

Diagramme de packages



Dépendances entre packages

- Découlent des dépendances entre éléments des packages
  - notamment les classes
- Les dépendances ne sont pas transitives
  - modifier Fournisseur n'oblige pas à modifier Clientèle



Utilisation des diagrammes de packages

- Organisation globale du modèle
  - hiérarchies de packages contenant diagrammes et éléments
- Organisation des classes en packages pour
  - contrôler la structure du système
    - comprendre et partager
    - obtenir une application plus évolutive et facile à maintenir
      - ne pas se faire déborder par les modifications
      - viser la généricité et la réutilisabilité des packages
  - avoir une vue claire des flux de dépendances entre packages
    - les minimiser

Packages et nommage

- Noms pleinement qualifiés
  - équivalent à chemin absolu
    - ex. package java::util, classe java::util::Date
- Stéréotypes de dépendance
  - « import » : les éléments passent dans l'espace de nommage
    - ex. classe Date depuis le package qui importe
  - « access » : sont accessibles
    - ex. classe java::util::Date depuis le package qui importe

Principes du découpage en packages

- Cohérence interne du package : relations étroites entre classes
  - fermeture commune
    - les classes changent pour des raisons similaires
  - réutilisation commune
    - les classes doivent être réutilisées ensemble
- Indépendance par rapport aux autres packages
- Un package d'analyse contient généralement moins de 10 classes

Bien gérer les dépendances

- Les minimiser pour maintenir un couplage faible
  - dépendances unidirectionnelles
    - *cf.* associations navigables
  - pas de cycles de dépendances
    - ou au moins pas de cycles inter-couches
  - stabilité des dépendances
    - plus il y a de dépendances entrantes, plus les interfaces de package doivent être stables

Packages : divers

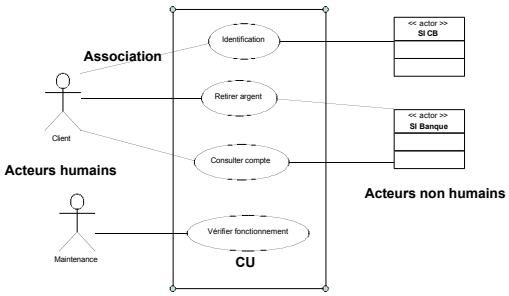
- Packages considérés comme
  - simples regroupements
  - sous-systèmes opérationnels
    - comportement + interfaces
- Package vu de l'extérieur
  - classe publique gérant le comportement externe (*cf.* pattern *Façade*)
  - interfaces
- Pour un package utilisé partout (très stable)
  - mot-clé « global »
- Utilité pratique d'un package Commun
  - regrouper les concepts largement partagés, ou épars
- Lien entre packages et couches (niveaux)
  - une couche est composée de packages

Plan

Cas d'utilisation

- Technique pour capturer les exigences fonctionnelles d'un système
  - déterminer ses limites
  - déterminer ce qu'il devra faire
    - mais pas comment il devra le faire
    - point de vue de l'utilisateur
- Pour cela
  - déterminer les rôles qui interagissent avec lui
    - acteurs
  - déterminer les grandes catégories d'utilisation
    - cas d'utilisation
  - décrire textuellement des interactions
    - scénarios

Diagramme de cas d'utilisation



Petit exercice à faire en classe

- Quels sont les acteurs et les cas d'utilisation d'un système d'information pour l'UFR informatique ?

Utilisation

- Passer du flou du cahier des charges à des fonctionnalités exprimées dans le langage du domaine
  - dialogue entre concepteurs et utilisateurs
- Pour l'expression complète des besoins, tout au long d'un processus de conception de système d'information
- Attention
  - ce ne sont pas les diagrammes de CU qui sont importants, mais les descriptions textuelles des scénarios

Acteur

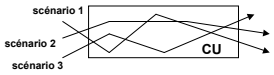
- Entité (humain ou machine) située hors du système
  - permettant d'en déterminer les limites
  - jouant un rôle par rapport à lui
  - déclenchant un stimulus initial entraînant une réaction du système
  - ou au contraire étant sollicité par le système au cours d'un scénario
- Un acteur est décrit précisément en quelques lignes
- 4 grandes catégories
  - acteurs principaux (fonctions principales du système)
  - acteurs secondaires (administration / maintenance)
  - matériel externe
  - autres systèmes

Client : personne qui se connecte au distributeur bancaire à l'aide de sa carte. Peut avoir ou non un compte dans la banque qui possède le distributeur.

Cas d'utilisation

- Ensemble de séquences d'action réalisées par le système, produisant un résultat observable pour un acteur particulier
  - ex. identification, retrait de liquide
- Un CU
  - définit un ensemble de scénarios d'exécution incluant les cas d'erreurs
  - est définit par une famille de scénarios impliquant le même acteur (déclencheur) avec le même objectif
- Un CU recense les informations échangées et les étapes dans la manière d'utiliser le système, les différentes points d'extension et cas d'erreur

Scénarios



- Séquence particulière de messages dans le CU pendant une interaction particulière
  - "chemin" dans le cas d'utilisation
- Tous les scénarios d'un CU sont issus du même acteur et ont le même objectif
- Description du CU
  - ensemble de scénarios couvrant le CU
  - documents avec flot d'événements
    - détaille ce qui se passe entre utilisateur et le système quand le CU est exécuté
      - flot nominal des événements (80 %)
      - flots d'événements alternatifs
      - flots d'exceptions (terminaison incorrecte)
  - serviront de base pour les jeux d'essais

Documentation d'un CU

- Fiche textuelle
  - champs de description : nom, préconditions…
  - lisible et informelle
    - français simple, phrases descriptives
    - pas trop long (personne ne lit 10 pages)
  - décrivant
    - un scénario nominal
      - suite d'étapes avec objectifs de l'acteur bien identifiés et menés à bien
    - des points d'extension et étapes d'extensions
    - des points d'échec
    - des liens vers d'autres scénarios s'il y a trop d'étapes

(Cockburn)

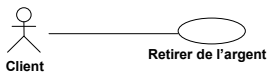
Niveaux pour les cas d'utilisation

- Plusieurs niveaux de description
  - Abrégé, informel, détaillé
- Plusieurs niveaux d'objectif
  - Cerf-volant
    - objectif stratégique (fonction SI dans organisation, on se rapproche des processus métier)
  - Surface de la mer
    - objectif utilisateur (fonction SI pour utilisateur)
  - Poisson
    - objectif informaticien (sous-fonction interne au système)
- Plusieurs portées de conception
  - organisation (boîte blanche ou noire)
  - système (boîte blanche ou noire)
  - composant

Description d'un CU

- Nom
- Contexte d'utilisation
- Portée
- Niveau
- Acteur principal
- Intervenants et intérêts
- Préconditions
- Garanties minimales
- Garanties en cas de succès
- Déclencheur
- Scénario nominal
  - étapes
- Extensions
  - étapes d'extension
- Variantes de technologie ou de données
- Informations connexes

Exemple scénarios pour CU



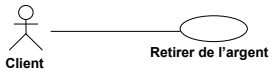
CU : Retirer de l'argent
Portée : système DAB
Niveau : objectif utilisateur
Acteur principal : Client
Intervenants et intérêts : Banque, Client
Préconditions : compte approvisionné
Garanties minimales : rien ne se passe
Garanties en cas de succès : de l'argent est retiré, le compte est débité de la même somme
…

Exemple scénarios pour CU

…
Scénario nominal :
1. Le Client introduit sa carte dans le lecteur.
2. Le DAB décrypte l'identifiant de la banque, le numéro de compte et le code secret de la carte, valide de la banque et le numéro de compte auprès du système principal.
3. Le client saisit son code secret. Le DAB valide par rapport au code secret crypté lu sur la carte.
4. Le client sélectionne retrait, et un montant multiple de 10 € (min 20 €)
5. Le DAB soumet au principal système de la banque le compte client et le montant demandé, et reçoit en retour une confirmation et le nouveau solde du compte
6. Le DAB délivre la carte, l'argent et un reçu montrant le nouveau solde
7. Le DAB consigne la transaction

…

Exemple scénarios pour CU



…

Extensions :

*a. Panne générale.

*a1. Le DAB annule la transaction, signale l'annulation, et rend la carte.

2a. Carte volée.

2a1. Le DAB confisque la carte volée

Inclusion autre scénario

4a. Plus de billets de 10 €

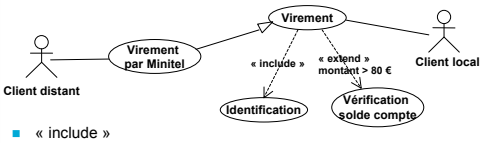
4a1. Le DAB arrondit la somme demandée à un multiple de 20 €.

4a2. Le Client valide la nouvelle somme demandée.

5a. Solde insuffisant.

5a1. Le DAB signale que la somme demandée est trop élevée et rend la carte.

Relations entre CU



- « include »
  - la réalisation d'un CU nécessite la réalisation d'un autre, sans condition, à un point d'extension (le seul important)
- « extend »
  - entre deux instances de CU : le comportement de CU1 peut être complété par le comportement de CU2 (option avec condition et point d'extension)
  - conseil : ne pas utiliser, ou seulement si on ne peut troucher à CU1
- « generalize »
  - héritage. (conseil : ne pas utiliser)
- Bref
  - le diagramme de CU est une bonne table des matières, pas plus

Compléter les CU

- Avec tout ce qui permet de mieux expliquer
  - modèle du domaine
  - diagrammes de séquence système,
  - diagramme d'activité, de machines d'états

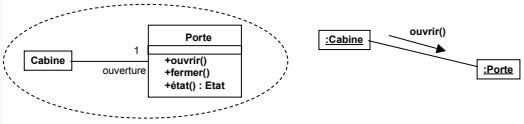
Quelques conseils

- Pas plus de 20 CU
- Pas de définition fonctionnelle en utilisant les relations de CU
- Retour sur les CU dans un cours dédié, un TD dédié

Plan du cours

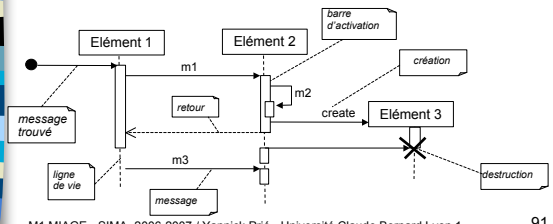
Collaborations et interactions

- Collaboration
  - ensemble de rôles joués par des classes, contexte d'interaction
- Interaction
  - communication entre instances des éléments d'une collaboration
  - ensemble partiellement ordonné de messages
  - plusieurs interactions possibles pour une même collaboration
- Éléments d'une interaction
  - participants (UML1 : objets, UML2 : souvent objets)
  - liens (supports de messages)
  - messages (déclenchant des opérations)
  - rôles joués par les extrémités de liens



Diagrammes de séquences

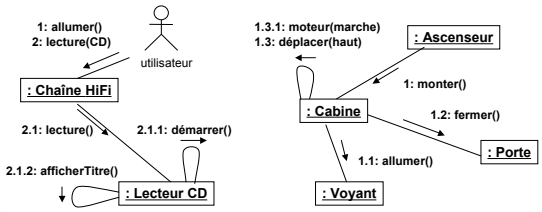
- Interactions entre éléments dans une séquence *temporelle*
  - aspect chronologique ne rendant pas compte explicitement du contexte
  - permet de bien montrer qui fait quoi dans une interaction
- Description de scénarios typiques et des exceptions



Diagrammes de communication

(UML1 : diagrammes de collaboration)

- Diagramme d'objets rendant compte de la dynamique
  - structure spatiale permet la collaboration d'objets
  - dimension temporelle : ordre des messages
    - numérotation pointée



Petit exercice à faire en classe

- Dessiner un diagramme de communication impliquant le passage de la balle entre deux tortues d'équipes différentes.

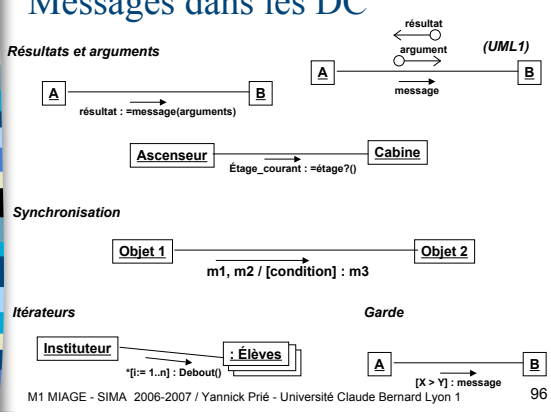
Utilisation

- Etudier/spécifier le comportement
  - du système dans sa globalité au sein d'un cas d'utilisation
    - se concentrer sur les événements du système considéré comme boîte noire
      - diagramme de séquence système (exemple plus loin)
  - de plusieurs objets au sein d'un cas d'utilisation
    - réalisations de CU comme des interactions dans une société d'objets
      - Conseil : dessiner diagrammes de classes et d'interaction en même temps
- Illustrer/étudier un fonctionnement
  - diagramme qui traverse les couches : de l'IHM aux données
  - rétro-ingénierie

Messages

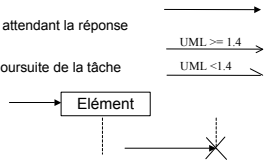
- Matérialisation d'une communication avec transmission d'information entre
  - émetteur (source)
  - récepteur (destination)
- Un message déclenche
  - une opération,
  - l'émission d'un signal
  - la création/destruction d'un objet
- Deux types principaux
  - appel de procédure ou flot de contrôle emboîté (retour implicite)
    - déplacer()
  - flot de contrôle asynchrone
    - démarrer()
  - autres : à plat, dérobant (réception si attente), minuté (message actif pendant Dt)

Messages dans les DC



Messages dans les diagrammes de séquences

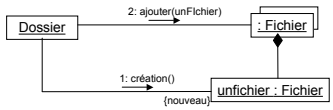
- Notation résultat = message(arguments)
- Echange de messages
  - flèches d'appel standard
    - blocage de l'émetteur en attendant la réponse
  - flèche d'appel asynchrone
    - pas d'attente du retour, poursuite de la tâche
  - Retour
  - Message de création
  - Message de destruction



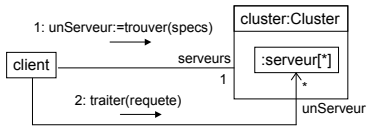
- Lancement de l'interaction venant de l'extérieur
  - 1er message = « message trouvé »

Gestion de collections

- Créer et ajouter (UML1 : multi-objet)



- Récupérer et utiliser (UML2 : structure composite)

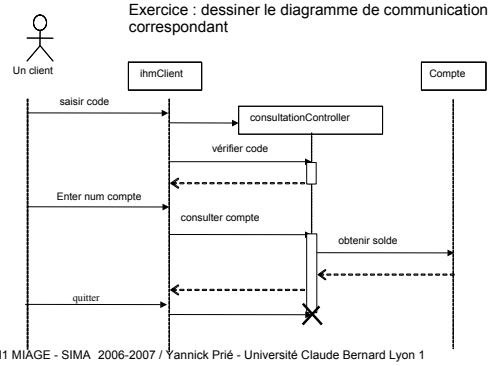


Raffinements DS/UML2

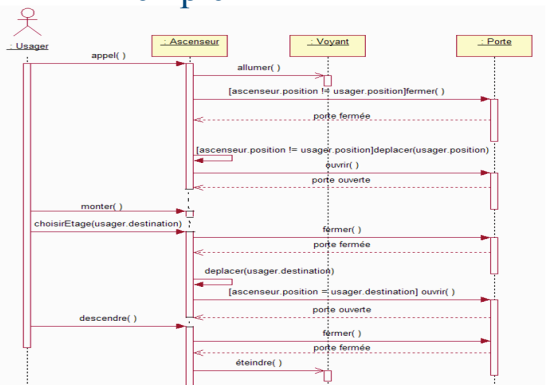
- Diverses possibilités de participants
  - interfaces : spécifier quelle interface participe à l'interaction
  - classes : pour appeler une méthode de classe
- Représentation polymorphisme / classe abstraite

Equivalence entre diagrammes

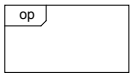
Exercice : dessiner le diagramme de communication correspondant



Exemple

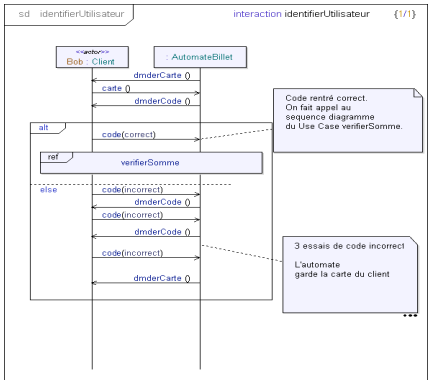


Cadre d'interaction

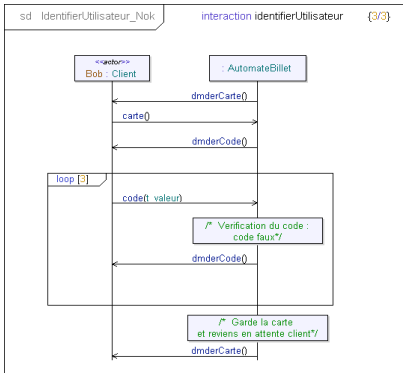


- Cadre nommé par un opérateur qui entoure un fragment critique du DS
  - alt
    - fragment alternatif, conditions dans les gardes
  - loop
    - fragment à répéter tant que la condition de garde est vrai
    - notion de boîte d'action avec itérateur
  - opt
    - fragment optionnel exécuté si la garde est vraie
  - par
    - fragments qui s'exécutent en parallèle
  - region
    - region critique dans laquelle un seul thread doit s'exécuter
  - ref
    - passage à un autre diagramme de séquence
- Attention
  - ne pas représenter des algorithmes : trop compliqué

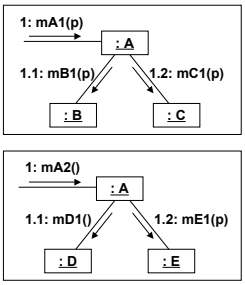
alt



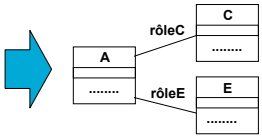
loop



Déduire structure et responsabilité des diagrammes d'interaction



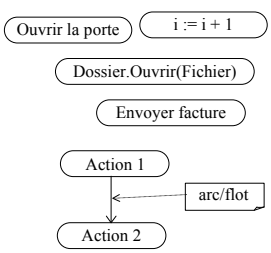
- Liens
  - ➜ associations
- Messages
  - ➜ opérations



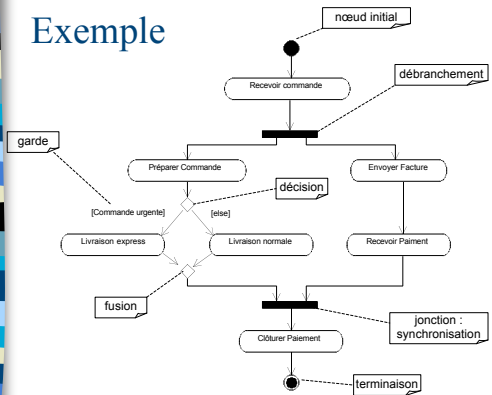
Plan du cours

Diagrammes d'activité

- Diagramme d'activité
  - présenter les activités séquentielles d'un processus
  - activité = suite d'actions
- Action
  - travail à réaliser
  - nœud du graphe
- Transition
  - contrainte d'enchaînement
  - relation du graphe
- Raffinements
  - débranchements / jointures
  - décisions / fusions
  - entrée / terminaison
  - ressources utilisées (objets)



Exemple



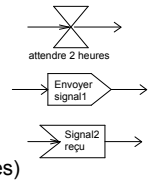
Petit exercice à faire en classe

- Modéliser les activités autour d'un enseignement de l'UFR informatique.

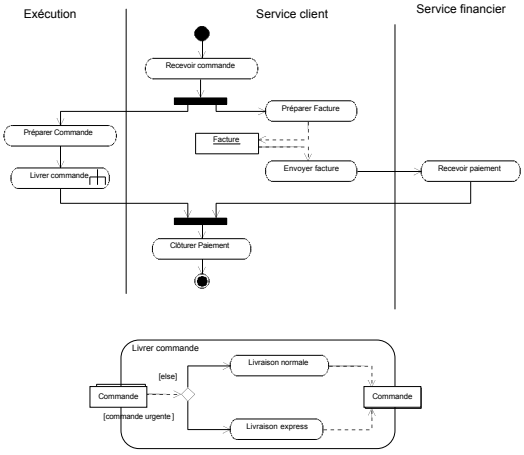
Utilisation pour modéliser

- Les processus métier de l'organisation
  - qui fait quoi, où
  - les enchaînement d'activité (workflow)
- Les flots de données
  - DFD (Data Flow Diagram) en UML
- La logique procédurale
  - algorithmes complexes, parallèles
  - organisation séquentielle globale des activités de plusieurs objets
    - vs. diag. machines d'états : un objet

Diagrammes avancés

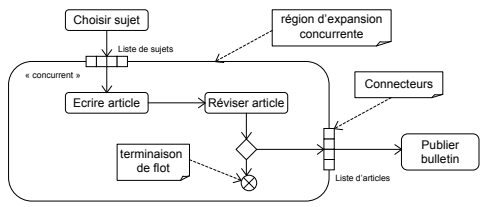


- Actions liées à des signaux
  - délai
  - envoi / réception
- Utilisation d'objets
  - en entrée ou sortie d'action
- Partitions (UML1 : *swimlanes*, travées)
  - montrer les responsabilités au sein du mécanisme ou d'une organisation
- Décomposition des actions
  - appeler une sous-activité (un autre diagramme d'activité) dans une action



Connecteurs, régions d'expansion, terminaison de flots

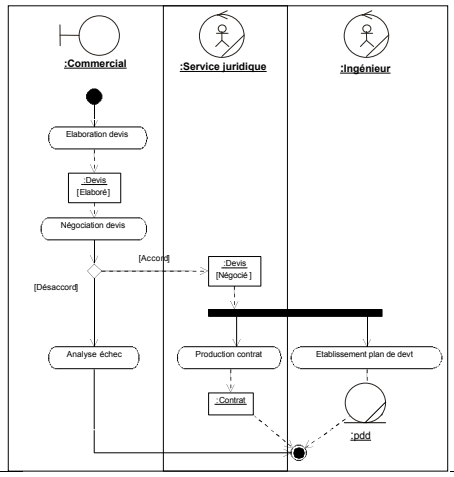
- Connecteurs
  - *cf.* objets paramètres entrée/sortie actions
- Régions d'expansion
  - actions qui se passent pour plusieurs éléments de même type (itératif ou concurrent)



Modélisation de processus métier

- Modéliser le fonctionnement de l'organisation
- Objets responsables (stéréotypes)
  - Case worker
    - interaction avec l'ext. de l'entreprise
  - Internal worker
  - Entity
    - objet passif

- Partitions / couloirs d'activité

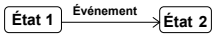


Plan du cours

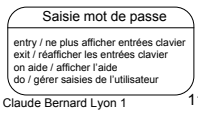
- Introduction à UML
- Généralités sur la notation
- Diagrammes de classes, objets, packages
- (Diagrammes de) cas d'utilisation
- Diagrammes d'interaction
- Diagrammes d'activité
- **Diagrammes de machines d'état**
- Diagrammes de composants et de déploiement
- Autres diagrammes UML
- Autres diagrammes non UML
- Autres points liés à UML

Diagrammes de machines d'états

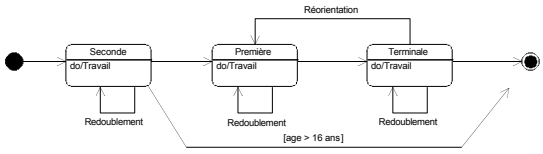
- Abstraction des comportements possibles pour une classe
  - automate à états finis décrivant les chemins possibles dans le cycle de vie d'un objet
- Etat d'un objet
  - situation nommée d'un objet qui répond à certaines conditions (durée/stabilité)



- Transition entre états
  - réponse de l'objet dans un certain état à l'occurrence d'un événement
    - passage d'un état à un autre sur événement + condition respectée,
    - action à exécuter
- Dans un état
  - activité : continue (sonnerie), tâche de fond (pagination), attente, suite d'actions…



Exemple



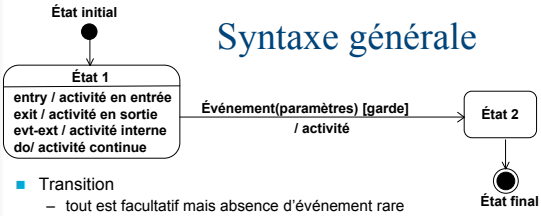
Petit exercice à faire en classe

- Tracer un diagramme de machines d'états pour un objet « TP-Etudiant ».

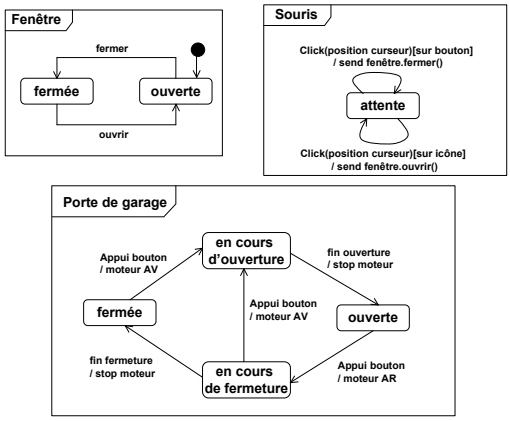
Utilisation

- Pour se concentrer sur le fonctionnement d'une classe
  - décrire / fixer le comportement concret de la vie d'une objet
    - lié à un ou plusieurs scénarios
- Pour les classes complexes
  - objets réactifs complexes (objets métier…)
  - protocole et séquences légales (sessions…)
  - en général pas plus de 10% des classes d'une application
    - plus en télécommunication / moins en informatique de gestion
- Larman
  - navigation dans un site web, IHM
    - enchaînement de pages/fenêtres

Syntaxe générale

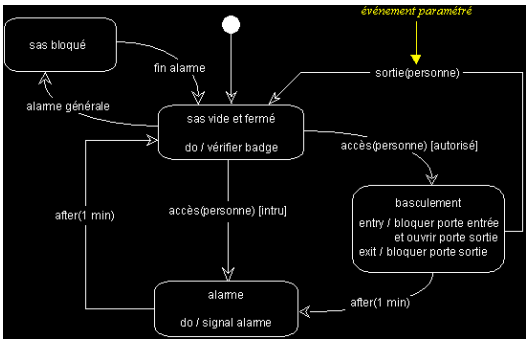


- Transition
  - tout est facultatif mais absence d'événement rare
  - événements
    - résultants de messages entre objets
    - internes : when(maximum atteint)
    - temporels : after(3 jours)
  - activité classique : envoyer un message à une cible
    - send cible.message(arguments)
- État
  - activités internes (ordinaires) : instantanées
    - autotransition sur événement extérieur, instantané
    - deux activités spéciales : sur entrée et sortie
  - activités continues : peuvent être interrompues
    - do / activité



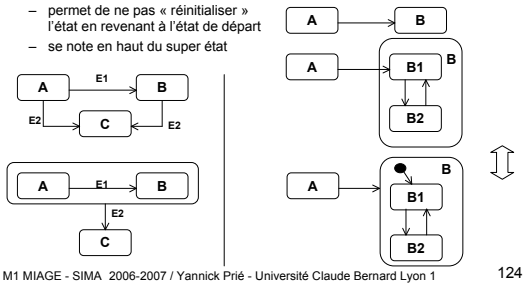
http://uml.free.fr/cours/p20.html

Autre exemple

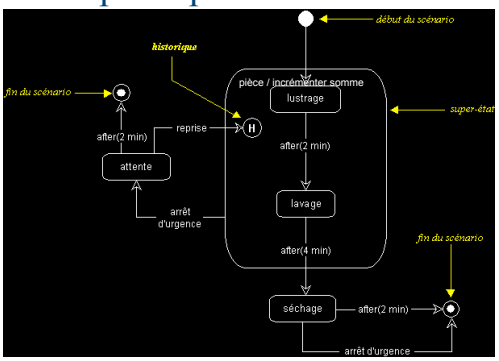


Super-états / états composites

- Pour factoriser un comportement
  - transitions déclenchées par le même événement, conduisant au même état
- Transition interne
  - couple événement / activité sans effet sur l'état courant
  - permet de ne pas « réinitialiser » l'état en revenant à l'état de départ
  - se note en haut du super état

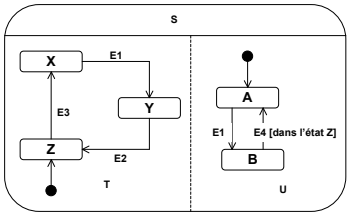


Exemple super-état



États concurrents

- Pour décomposer des états complexes
- Exercice : trouver le diagramme d'état « à plat » équivalent



Implémentation

- Utilisation de `case … switch`
  - déconseillé
- Utilisation du pattern état
  - arborescence de classes états
  - délégation de la gestion de l'état
- Tables d'états
  - représentation tabulaire du diagramme
    - état source, cible, événement, garde, procédure à exécuter
  - permet de « paramétrer » le comportement de la classe

Plan du cours

- Introduction à UML
- Généralités sur la notation
- Diagrammes de classes, objets, packages
- (Diagrammes de) cas d'utilisation
- Diagrammes d'interaction
- Diagrammes d'activité
- Diagrammes de machines d'état
- **Diagrammes de composants et de déploiement**
- Autres diagrammes UML
- Autres diagrammes non UML
- Autres points liés à UML

Diagrammes de composants

- Objectif
  - représenter l'organisation et les dépendances entre composants logiciels
  - description des composants et de leurs relations dans le système en construction
- Composant
  - partie physique et remplaçable d'un système qui se conforme à et fournit la réalisation d'interfaces
  - doit être compris comme un élément qu'on peut acheter, associer à d'autres composants (*cf. HiFi)*
  - division en composants
    - décision technique *et* commerciale (Fowler)
- Remarque
  - UML1 : composant = n'importe quel élément, y compris fichiers.
  - UML2 : utiliser les *artefacts* pour représenter des structures physiques (jar, dll…)

(Fowler 04)

Diagramme de composants

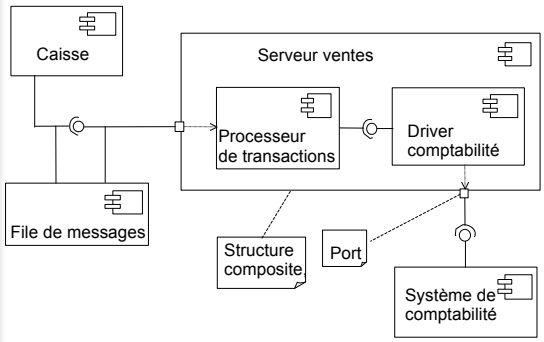


Diagramme de déploiement

- Disposition physique des différents matériels qui entrent dans la composition d'un système, ainsi que disposition des programmes exécutables sur ces matériels.
  - visualiser la distribution des composants dans l'entreprise
  - unités = nœuds
    - équipements = matériel
    - environnement d'exécution = logiciel
  - un noeud contient des artefact : classes, ...
- Relations entre éléments : supports de communication

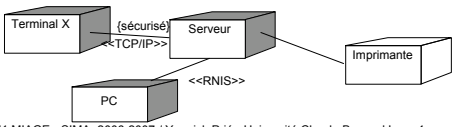
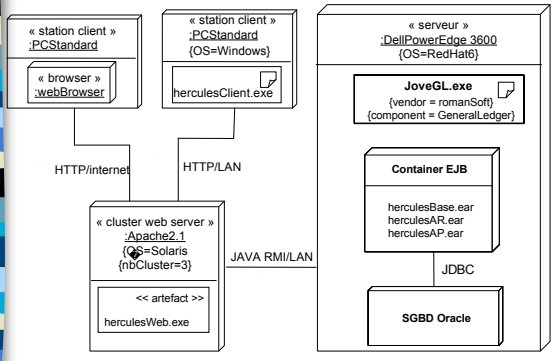


Diagramme de déploiement



Plan du cours

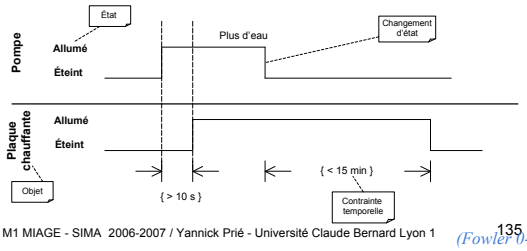
- Introduction à UML
- Généralités sur la notation
- Diagrammes de classes, objets, packages
- (Diagrammes de) cas d'utilisation
- Diagrammes d'interaction
- Diagrammes d'activité
- Diagrammes de machines d'état
- Diagrammes de composants et de déploiement
- **Autres diagrammes UML**
- Autres diagrammes non UML
- Autres points liés à UML

Vue d'ensemble des interactions

- Mixte diagramme activité / diagrammes de séquences
  - les actions sont remplacées par des diagrammes de séquence
- Trop tôt pour juger de l'utilisation / utilité effective

Diagramme de timing

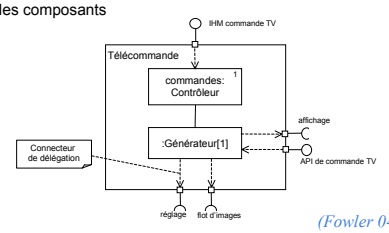
- Interactions avec focus sur les changements d'états d'objets et les contraintes temporelles associées
  - ligne de vie horizontale
- Utilisé surtout dans les applications temps réel



(Fowler 04)

Structures composites (classeurs structurés)

- Décomposer structurellement une classe
  - parties, connecteurs
- Montrer les réalisations / utilisations d'interfaces
  - ports, interfaces
- Adaptés pour les composants



(Fowler 04)

Diagrammes de collaborations

- Non officiels dans UML2, se rapprochent des diagrammes de structure composite
- Permettent de présenter les éléments impliqués dans une collaboration, et le rôle qu'ils y jouent
  - fixer les éléments et les rôles pour les diagrammes d'interaction
- En théorie utilisés pour représenter des *patterns*

Plan du cours

- Introduction à UML
- Généralités sur la notation
- Diagrammes de classes, objets, packages
- (Diagrammes de) cas d'utilisation
- Diagrammes d'interaction
- Diagrammes d'activité
- Diagrammes de machines d'état
- Diagrammes de composants et de déploiement
- Autres diagrammes UML
- **Autres diagrammes non UML**
- Autres points liés à UML

Diagrammes de contexte (Roques, 2004)

- Diagramme de contexte statique
  - diagramme de classe
    - une classe système
    - tous les acteurs autour
- Diagramme de contexte dynamique
  - diagramme de communication qui résume les messages entre système et acteurs (pas de numérotation)
- Diagramme de contexte statique étendu
  - diagramme de contexte statique avec
    - Attributs et opérations de hauut niveau pour le système et les acteurs non humains
- Remarque
  - « diagrammes de classes avec messages »
    - diagramme de classe avec résumé des messages entre classes

Diagramme de flux d'écrans informel (Fowler, 2004)

- Un rectangle par écran
- Des flèches pour la navigation
  - éventuellement un nom signifiant le lien

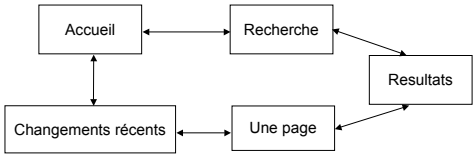


Table de décision (Fowler)

- Pour représenter des conditions logiques complexes
- Deux parties
  - conditions
  - conséquences

Client privilégié	X	X	O	O	N	N
Commande prioritaire	O	N	O	N	O	N
International	O	O	N	N	N	N
Prix	150	100	70	50	80	60
Alerte	oui	oui	oui			

Cartes CRC

- Classes - Responsabilités - Collaborateurs
  - à la base inventé pour l'enseignement
- Jouer des scénarios avec des cartes
  - 5-6 participants
- Une carte
  - nom de classe
  - tableau à deux colonnes
    - responsabilité de la classe : quelque chose d'un objet doit faire
    - collaborateurs : classes avec lesquelles il faut collaborer pour assurer la responsabilité

Classe	
Responsabilite1	Coll1,2

- Jeu
  - déterminer des classes de départ avec responsabilités évidentes
  - jouer les scénarios, ajouter les responsabilités, créer de nouvelles classes, *etc.*
- Voir par exemple
  - http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/crc_b/

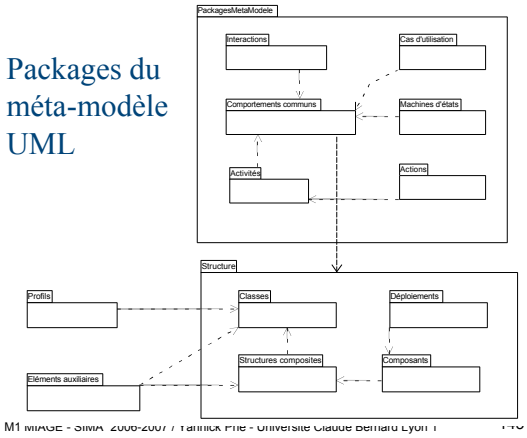
Plan du cours

- Introduction à UML
- Généralités sur la notation
- Diagrammes de classes, objets, packages
- (Diagrammes de) cas d'utilisation
- Diagrammes d'interaction
- Diagrammes d'activité
- Diagrammes de machines d'état
- Diagrammes de composants et de déploiement
- Autres diagrammes UML
- Autres diagrammes non UML
- **Autres points liés à UML**

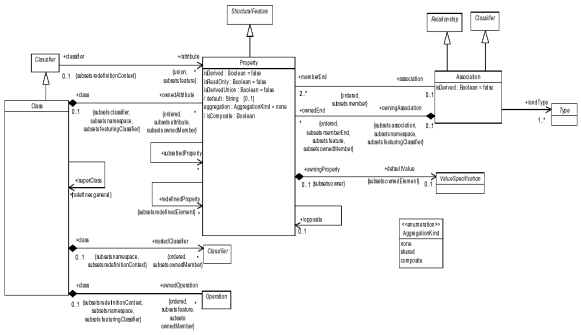
Méta-modèle

- L'ensemble de UML est décrit en UML
- Méta-modèle UML
  - description formelle de tout ce qu'il est possible de construire et de la sémantique associée
  - nécessaire pour les fabricants d'outils
- Essentiellement
  - diagrammes de classes avec contraintes et description de la signification dynamique des éléments
- Remarque : MOF (Meta Objet Facility)
  - Méta-méta-modèle permettant de décrire UML
  - mais aussi CWM
    - Common Warehouse Metamodel : structure de BD

Packages du méta-modèle UML



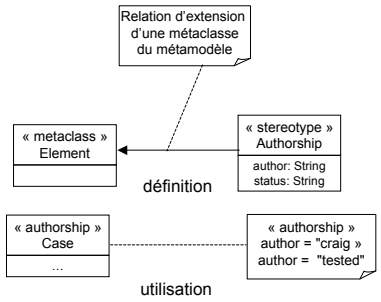
Extrait du méta-modèle



D'après OMG UML2 Superstructure, Figure 30

(Larman, 2005)

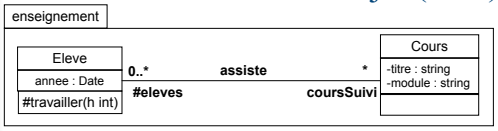
Extension d'UML : stéréotypes



Modèle UML / programme OO

- Modèle
  - classes
    - attributs, associations
    - opérations
    - spécialisation
  - packages
  - interactions
    - séquences de messages entre objets
- Possibilité de traduire tout ou partie du modèle dans un langage
  - objet ; Java, C++, C#, *etc.*
  - relationnel : classes, attributs

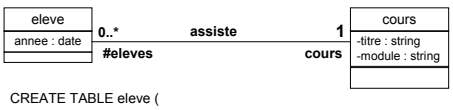
Traduction en objet (Java)




```
package enseignement ;
public class Eleve
{
    private Date annee ;
    private Court coursSuivi[] ;
    public Eleve() ;
    public Date getAnnee()
    {
        return annee ;
    }
    public void setAnnee(Date uneAnnee)
    {
        annee = uneAnnee ;
    }
    ...
```

```
    ...
    public associerCoursSuivi(unCours :
    Cours)
    {  /* ecrire ici */
    }
    public nbCoursSuivi
     {
        return coursSuivi.length ;
    }
    protected travailler(h :  int)
    {
        /* ecrire ici */
    }
    /* etc. */
}
```

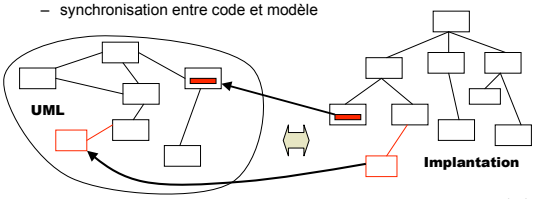

Traduction en relationnel




```
CREATE TABLE eleve (
    eleve_id NUMBER (5) ,
    annee DATE,
    PRIMARY KEY (eleve_id)
) ;
CREATE TABLE cours (
    eleve_id NUMBER (5)  REFERENCES eleve(eleve_id) ,
    cours_id NUMBER (5) ,
    titre CHAR (128) ,
    module CHAR(48) ,
    PRIMARY KEY (cours_id)
) ;
```

Ingénierie UML

- Pro-ingénierie
  - générer le code à partir du modèle
  - outils : paramétrage (héritage, associations…)
- Rétro-ingénierie
  - générer le modèle à partir de l'implantation
  - seuls les outils automatiques peuvent le faire
- Ingénierie bidirectionnelle (roundtrip engineering)
  - synchronisation entre code et modèle



Outils UML

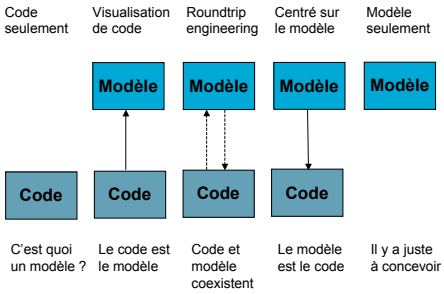
- Outils de dessins améliorés
  - intègrent les diagrammes comme simples formes
- Outils UML
  - gestion des diagrammes et du modèle
    - Vérification de cohérence
  - génération de code
    - squelettes de classes / contenus des méthodes (peu)
  - rétro-ingénierie
    - diagrammes de classes
    - diagrammes de séquences (peu)
  - de plus en plus intégrés / en compléments d'autres outils
    - IDE, gestion de projet, du risque, des besoins, de la qualité, des tests, du workflow, *etc.*

XMI

- XML Metadata Interchange
- Pour échanger des modèles UML entre outils
- Utilisation d'une syntaxe XML
  - eXtensible Markup Language
- Remarque
  - génération de documentation HTML
    - transformation XSL
  - diagrammes
    - transformation en SVG (Standard vector Graphics)

D'après http://www-128.ibm.com/developerworks/rational/library/3100.html

Modèles et code

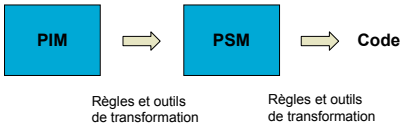


MDA : Model Driven Architecture

- Architecture pilotée par les modèles
  - mis en place et supporté par l'OMG
    - http://www.omg.org/mda/
  - UML comme langage de programmation
  - passer d'un développement centré sur le code à un développement centré sur les modèles
  - MDD (Model Driven Development)
    - terme plus général, non OMG
- Pour permettre, de la façon la plus intégrée possible
  - productivité
  - portabilité
  - interopérabilité
  - maintenance
  - documentation

MDA

- Deux types de modèles
  - PIM (Platform Independant Model)
    - en UML
  - PSM (platform specific model)
    - pas obligatoirement en UML



MDA : conclusion

- Pour certains
  - l'avenir de l'informatique
  - des outils existent
  - des « leaders » utilisent et tentent de promouvoir
- D'autres sont moins convaincus
  - entre
    - « ça ne marchera jamais »
    - « je demande à voir »
- Acceptation
  - possibilité de programmer mieux et plus vite (moins cher)
  - rapport apprentissage / gain
    - dans les entreprises
    - dans les écoles
- A suivre…

Contrainte

- Condition ou restriction sémantique associée à un ou plusieurs éléments de modèle exprimée
  - en langue
  - dans un langage formel
- Assertion qui doit être vraie
  - entre des appels d'opérations (qui changent le système)
  - à des moments précis par rapport aux appels d'opérations

OCL Object Constraint Language

- Standardisé par l'OMG
- Permet d'exprimer des contraintes de façon formelle
- Expression
  - d'invariants au sein d'un classe ou d'un type : bon fonctionnement des instances
  - contraintes au sein d'une opération : bon fonctionnement de l'opération
  - pré- et post- conditions d'opérations : avant et après l'exécution
    - *cf.* programmation par contrats (Meyer)
  - gardes : sur la modification de l'état d'un objet
  - expressions de navigation : chemins
- Utilisation
  - génération de code
  - MDA

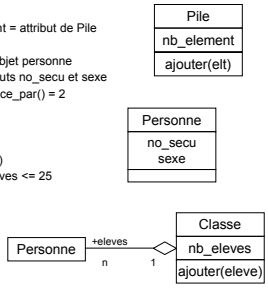
OCL : exemples

context nom_élément [**inv**|**pre**|**post**] : expression de la contrainte

```
context Pile inv :
    self.nb_elements >= 0   -- nb_element = attribut de Pile

context Personne inv    -- intégrité de l'objet personne
                                / attributs no_secu et sexe
    if sexe = "F"  then no_secu.commence_par() = 2
    else no_secu.commence_par() = 1
    endif

context Classe::ajouter(un_eleve : eleve)
    pre classe_non_surchargée : nb_eleves <= 25
    post : eleves→exists(un_eleve)
```



Conclusions sur UML

- Propriétés d'UML
  - unification de concepts de modélisation
  - puissance d'expression
    - nombreux formalismes (issus de méthodes existantes)
  - compromis formalisation / niveau d'abstraction / indépendance aux langages / sémantique fixée / extensibilité
- Langage universel
  - pour de multiples domaines
  - pour diverses activités de la conception
  - dans différents modes
    - esquisse, plan, programme

Conclusions sur UML

- Standard international : adopté un peu partout
  - les diagrammes sont simples, faciles à lire et à communiquer
  - beaucoup de variantes locales
  - outils puissants
    - dessin
    - pro et rétro ingénierie
    - MDA
- UML n'est qu'un langage
  - encapsule tout ou partie de la sémantique de description
  - ne dit pas comment construire les modèles
- Il faut utiliser des méthodes
  - démarches de conception et d'utilisation des diagrammes et des modèles