CSC4102 : Conception préliminaire

Denis Conan, avec Chantal Taconet et Christian Bac

Janvier 2025

Sommaire

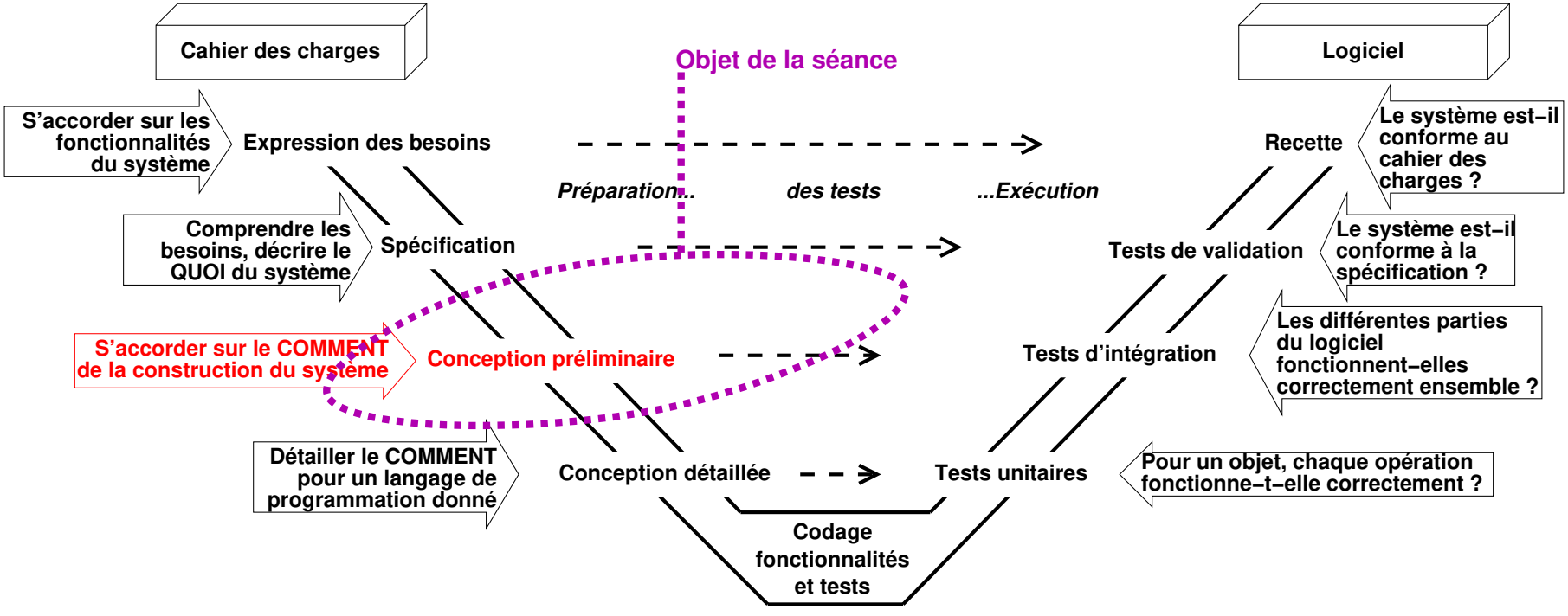
1. Motivations et objectifs de la séance
2. Aspects statiques, diagramme de classes
3. Aspects statiques, diagramme d'objets
4. Aspects dynamiques, diagramme de séquence
5. Mise en pratique en TP (2h) + HP (3h)

1 Motivations et objectifs de la séance

1.1 Contexte – Où en sommes-nous ?
1.2 Motivation et objectif : Concevoir avec UML
1.3 Contenu — Quels sont les objectifs de la séance ?

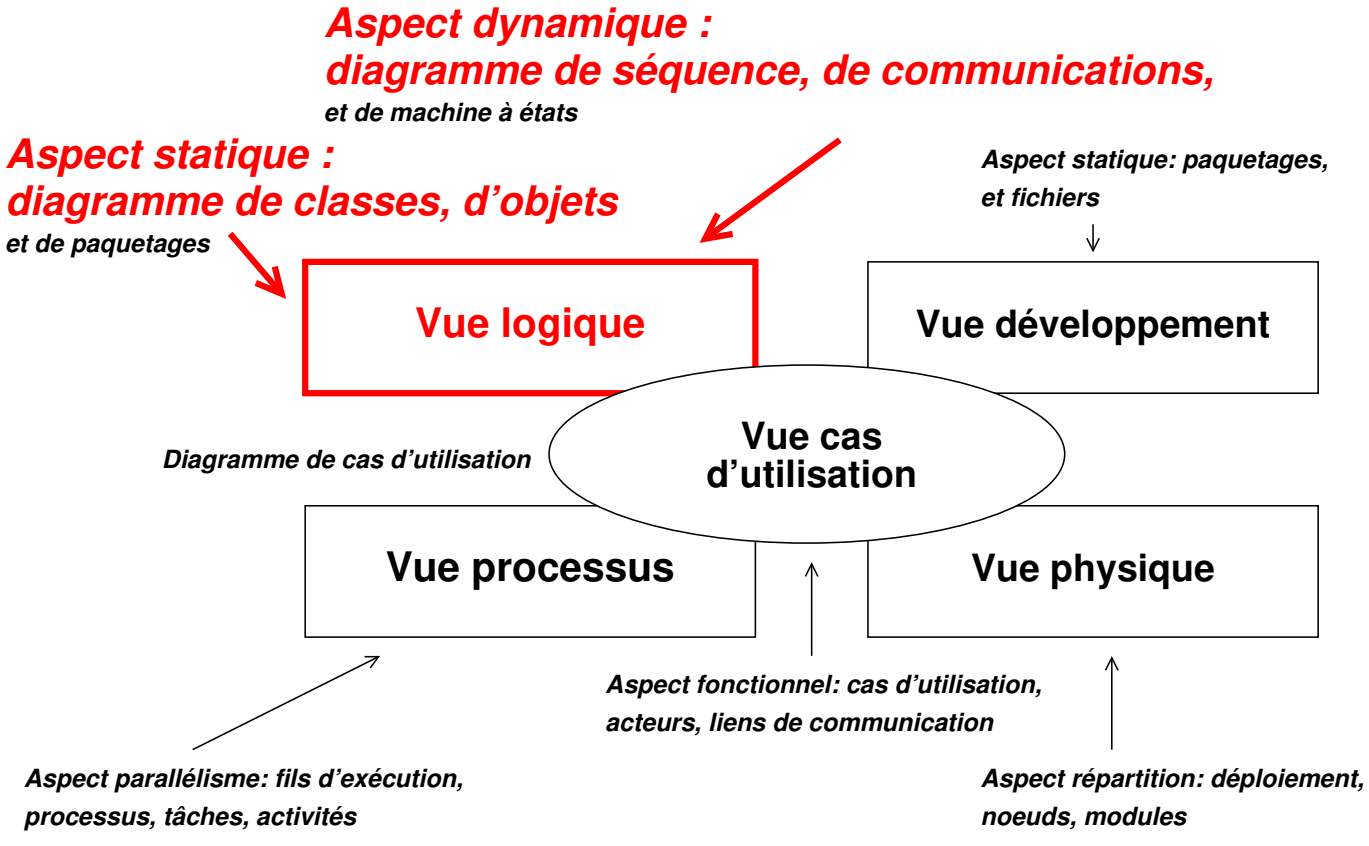
1.1 Contexte – Où en sommes-nous ?

- Premières décisions de conception, début du COMMENT
  - Diagramme de classes avec une première idée des attributs et des opérations
  - Diagrammes de séquence pour la réalisation des premiers cas d'utilisation avec le diagramme des classes



1.1.1 Différentes vues d'un système informatique

- Dans le modèle avec « 4+1 vues » [Kruchten, 1995]
  - La conception préliminaire commence la construction de la vue logique



1.2 Motivation et objectif : Concevoir avec UML

Martin Fowler, « *Is Design Dead?* » [Fowler, 2004]

- À propos des méthodes agiles comme *eXtreme Programming*[1]
  - *To its fans it is often seen as a rejection of design techniques (such as UML)*
  - *To its detractors this is a return to "code and fix" development*
- *The primary value of a diagram is communication*
- *A common use of diagrams is to explore a design before you start coding it*
- *UML is for documentation [...] when one group hands over to another*
  UML is useful here, providing the diagrams are selective

1. http://www.extremeprogramming.org/

1.3 Contenu — Quels sont les objectifs de la séance ?

- Conception préliminaire
  - Modélisation (de l'intérieur) du système, le « COMMENT »
    - Les aspects statiques dans un diagramme de classes
      - Avec des configurations particulières dans des diagrammes d'objets
    - Les aspects dynamiques dans des diagrammes d'interaction
      - Organisation temporelle (ligne de vie) : diagrammes de séquence
      - Organisation spatiale (graphe) : diagrammes de communications
        - Cf. « Pour aller plus loin » de la séance
- Dans ce cours, nous ignorons les tests d'intégration
  - Ce sont les plus difficiles à préparer, à programmer et à exécuter

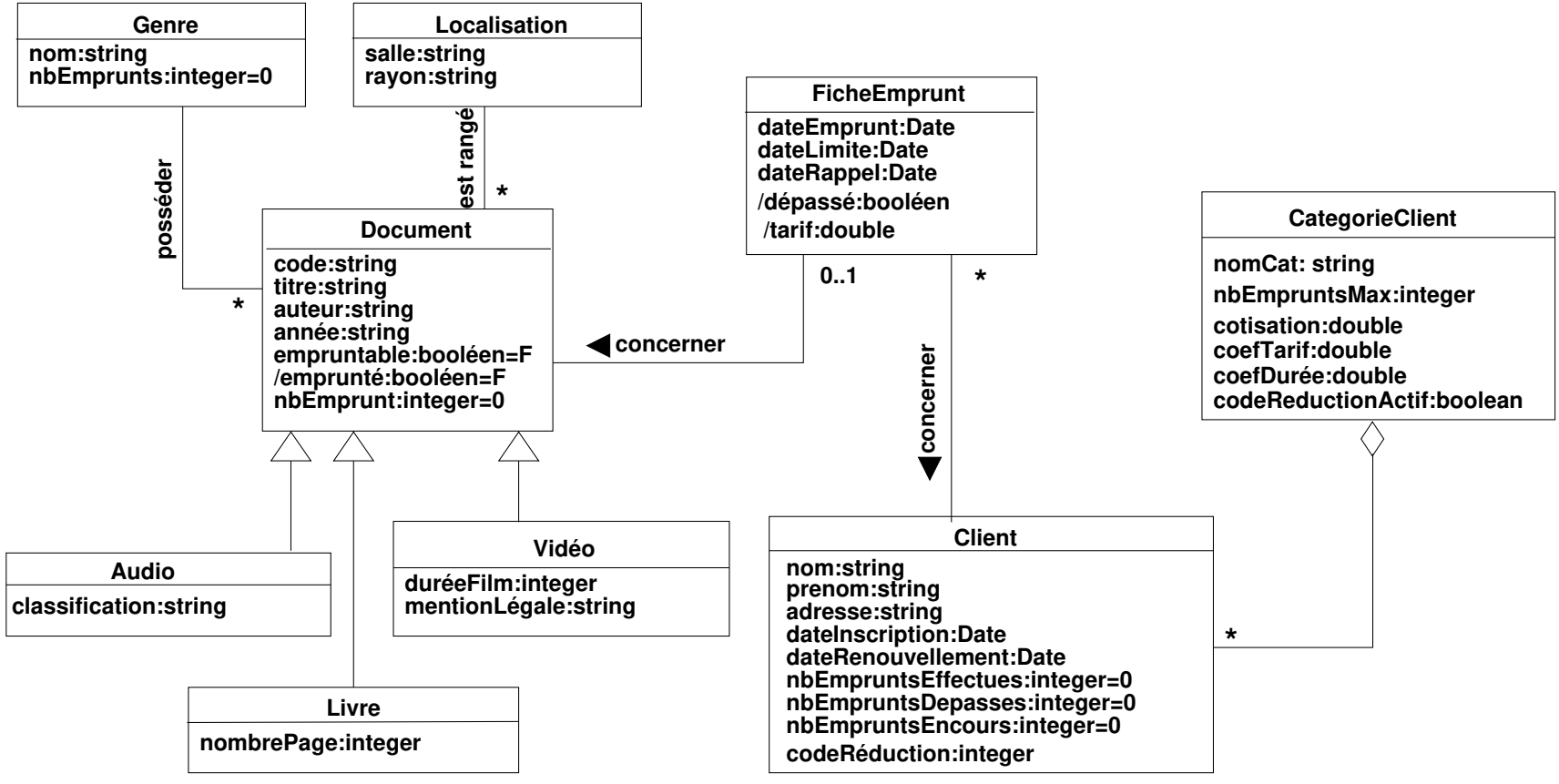
2 Aspects statiques, diagramme de classes

2.1 Diagramme avec le rappel des prérequis
2.2 Attributs et opérations de classe
2.3 Classe d'association
2.4 Type énuméré (énumération)
2.5 Classe abstraite
2.6 Généralisation spécialisation et redéfinition
2.7 Diagramme avec les nouveaux concepts
2.8 Éléments de méthodologie

. Dans cette première étude du diagramme de classes, sont laissés pour la conception détaillée et la programmation les éléments de modélisation suivants : navigabilité, composition, interface, et classe paramétrée

2.1 Diagramme avec le rappel des prérequis

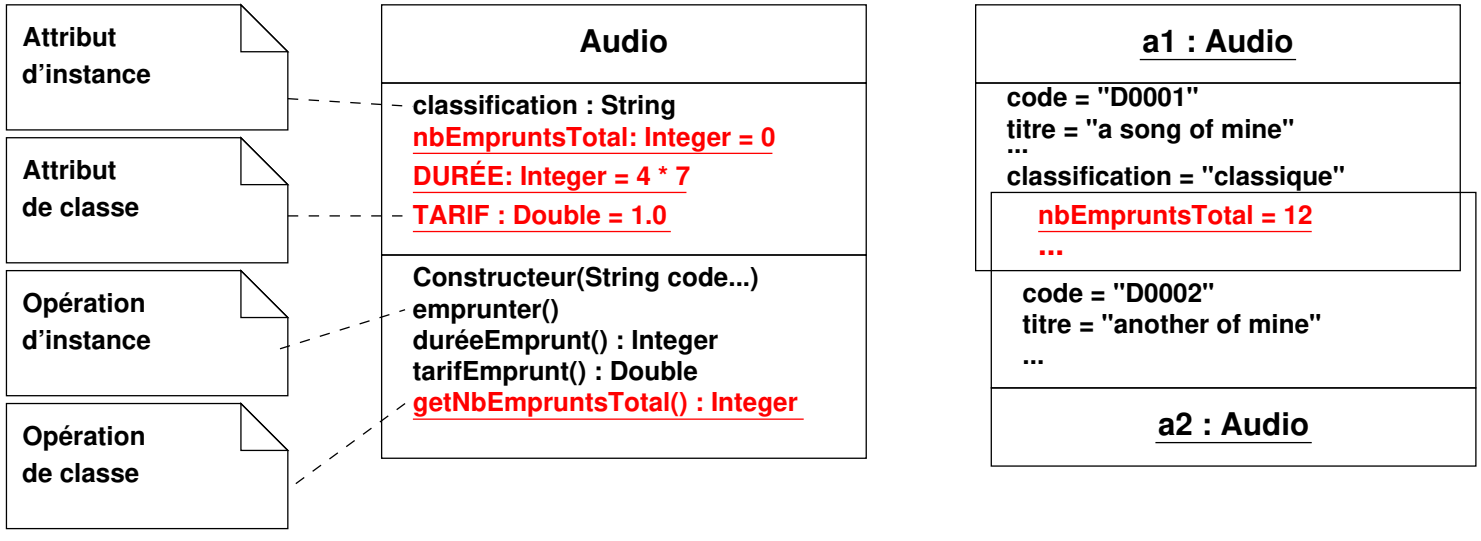
Classe, attribut (dérivé), opération, association bidirectionnelle, sens de lecture d'une association, multiplicité[2], agrégation, généralisation spécialisation



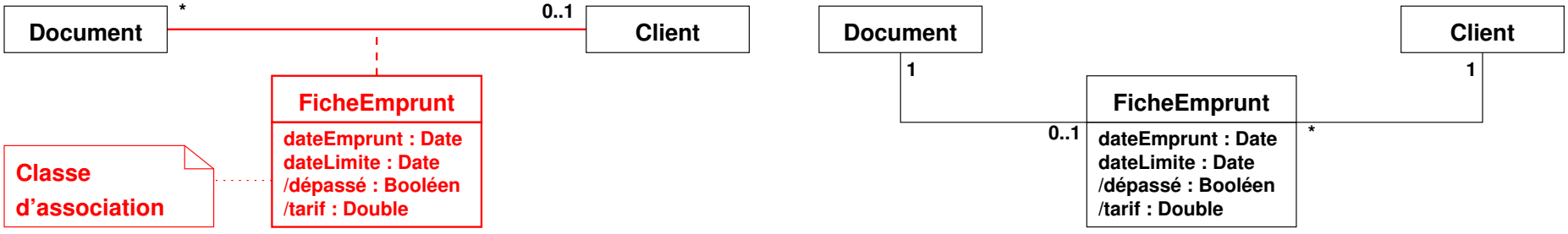
2. Attention aux confusions avec les multiplicités des modèles entité—association : (1) de « 0—*n* » à « 0..* » ou « * » en UML, et (2) placement du côté du complément d'objet en UML

2.2 Attributs et opérations de classe

- Attribut d'instance : chaque objet possède sa propre valeur
- Attribut de classe : valeur partagée par toutes les instances de la classe
- Opération d'instance : agit sur un objet, p.ex. `a1.emprunter()`
- Opération de classe : appliquée à la classe, p.ex. `Audio.getNbEmpruntsTotal()`



2.3 Classe d'association

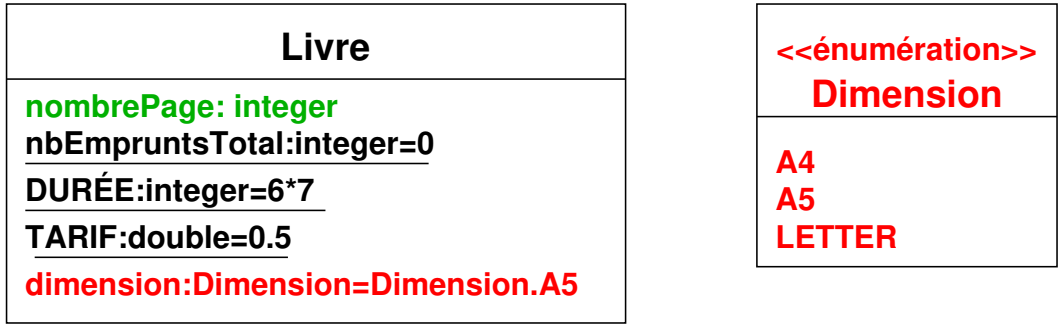


- Pour adjoindre des attributs à une association entre deux classes
  - Concept n'existant pas en langage de programmation
    $\Longrightarrow$ Traduction en deux associations lors de la conception détaillée
    - Lors de la traduction, attention aux multiplicités :
      - « 1 — 1 » devient « 1 — 1 » et « 1 — 1 »
      - « * — * » devient « 1 — * » et « * — 1 »
      - « 1 — * » devient « 1 — * » et « 1 — 1 »
      - « * — 1 » devient « 1 — 1 » et « * — 1 »

2.4 Type énuméré (énumération)

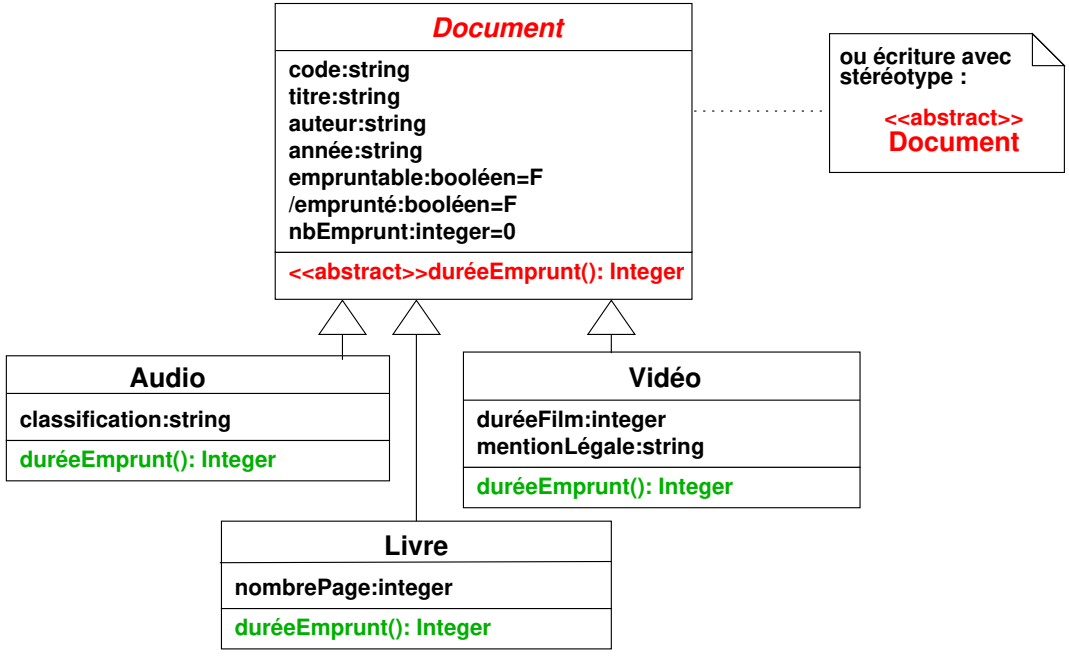
- Type énuméré, aussi appelé énumération = type de données dont les instances sont énumérées dans le modèle
  - Les instances sont appelées des « énumérateurs »
  - Les noms de ces instances sont des constantes

$\Longrightarrow$ Ensemble d'objets constants appelés énumérateurs



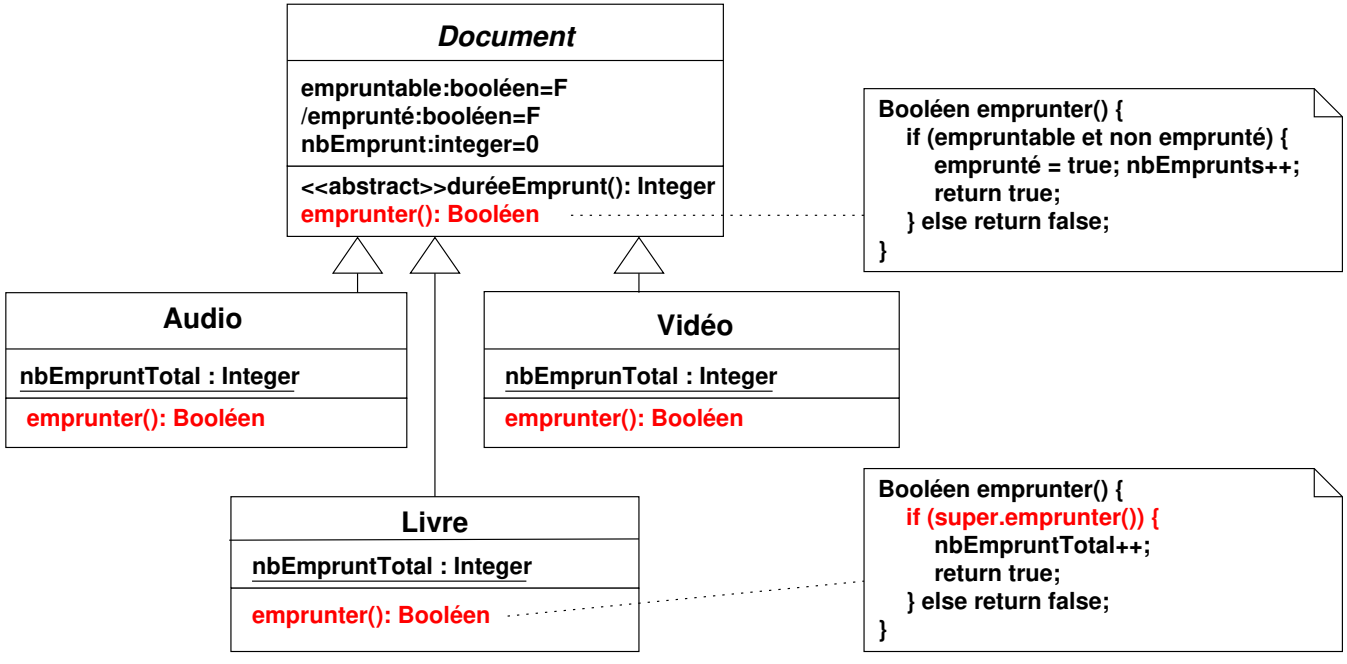
2.5 Classe abstraite

- Deux raisons non exclusives pour rendre une classe abstraite
  - Classe dont on ne veut pas d'instance
    - P.ex., on ne souhaite pas avoir de `Document` sans connaître son « sous-type »
  - Classe non complète car au moins une opération est abstraite
    - P.ex., `duréeEmprunt()` est déclarée, mais ne peut pas avoir de définition dans `Document` car dépend d'informations disponibles dans les classes enfants



2.6 Généralisation spécialisation et redéfinition

- Une opération de la classe parente peut être redéfinie dans la classe enfant
- Redéfinir (en anglais, *Override*) = remplacer
  - Possible d'appeler l'opération de la classe parente
    - P.ex. *super.emprunter()* dans l'exemple
    - Pour appeler un constructeur de la classe parente, *super(args)*



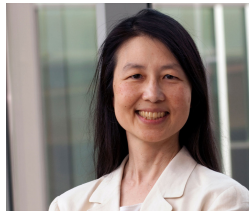
2.6.1 Sous-typage, transtypage et redéfinition

- Rappel : t est du type $T \equiv t \in T$; S est un sous-type de $T \equiv S \subseteq T$
- Transtypage vers le haut (*upcast*) $\Longrightarrow$ Subsomption [Abadi and Cardelli, 1996]
  - Avec s et t de type S et T respectivement, on peut écrire « `t := s` »
  - Après « `t := s` », T est appelé le type formel de t et S le type actuel de t
- Liaison dynamique ou tardive = appeler l'opération du type actuel
  - Si S redéfinit l'opération *op* préalablement définie dans T
    Après « `t := s` », « `t.op()` » utilise *op* redéfinie dans S
- Ne pas confondre redéfinition (*overriding*) et surcharge (*overloading*)[3]
  - Surcharge = dans la même classe, différentes opérations avec le même nom et des arguments différents (différence dans le nombre et/ou le type des arguments fournis)

3. C'est le risque de confusion entre la surcharge et la redéfinition qui nous incite à parler de surcharge dans cette diapositive.

2.6.2 Principe de substitution de Liskov&Wing *

Barbara Liskov, ACM A.M. Turing Award, 2008, [Liskov and Wing, 1994]

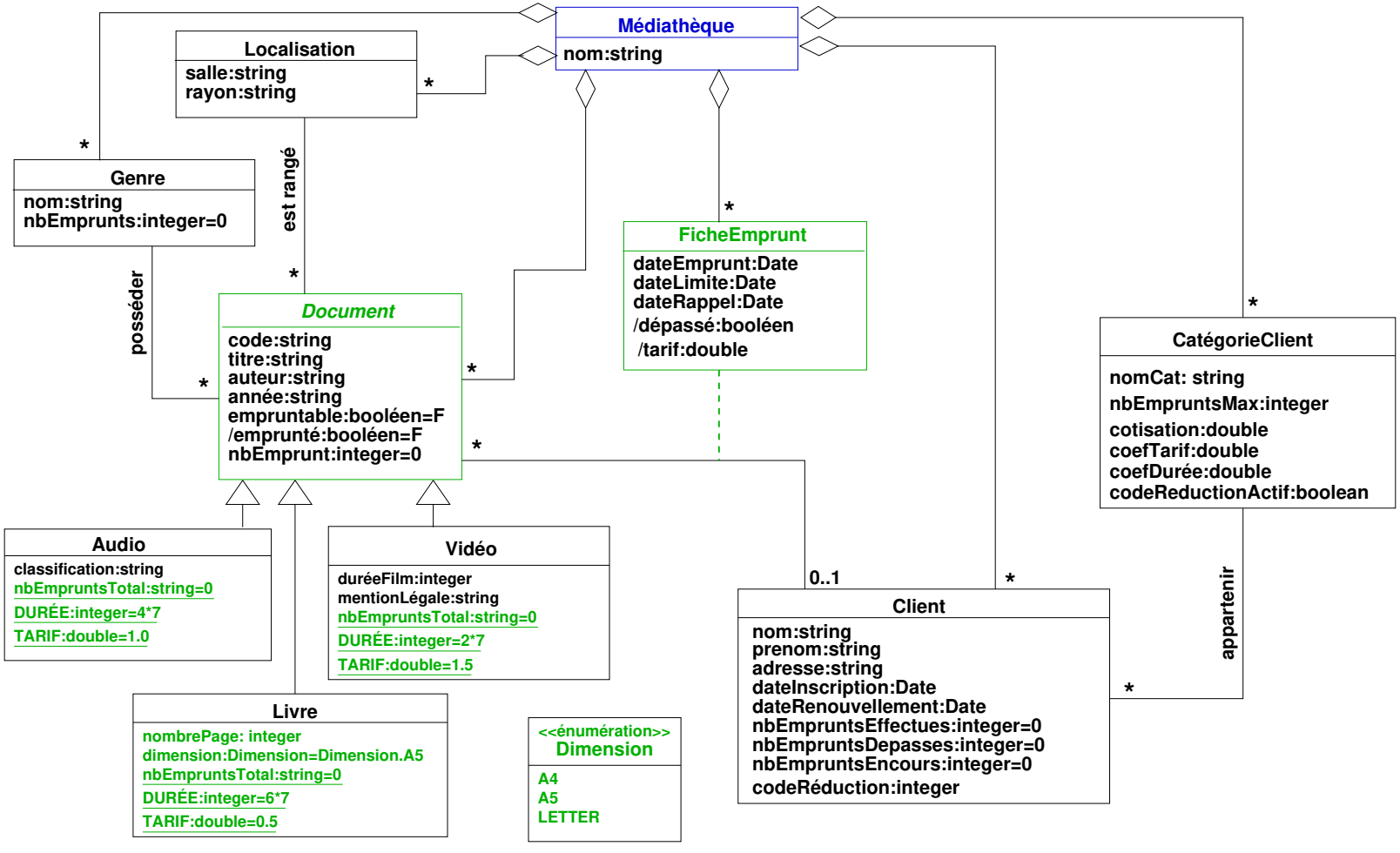


- La sémantique d'un objet est définie par :
  - Les préconditions et les postconditions des opérations
  - Les invariants des instances
    - Les propriétés d'un objet qui sont vérifiées pendant toute la vie de l'objet
- Lorsque cela est possible, chercher les bonnes propriétés suivantes :
  - Soient $op_T : A_T \rightarrow R_T$ et $op_S : A_S \rightarrow R_S$ avec $A_T = A_S \wedge R_T = R_S$[4]
  - Précondition de *op* de S = précondition de *op* de T qui peut être affaiblie[5]
    c.-à-d., précondition de *op* de $T \implies$ précondition de *op* de S
  - Postcondition de *op* de S = postcondition de *op* de T qui peut être renforcée[6]
    c.-à-d., postcondition de *op* de $T \impliedby$ postcondition de *op* de S
  - Invariant de S = invariant de T qui peut être renforcée
    c.-à-d., invariant de $T \impliedby$ invariant de S

4. Si différents alors « pour aller plus loin » : contravariance, covariance, invariance
5. L'opération redéfinie impose des contraintes moins fortes à l'appelant.
6. L'opération redéfinie rend un service avec plus de « qualité ».

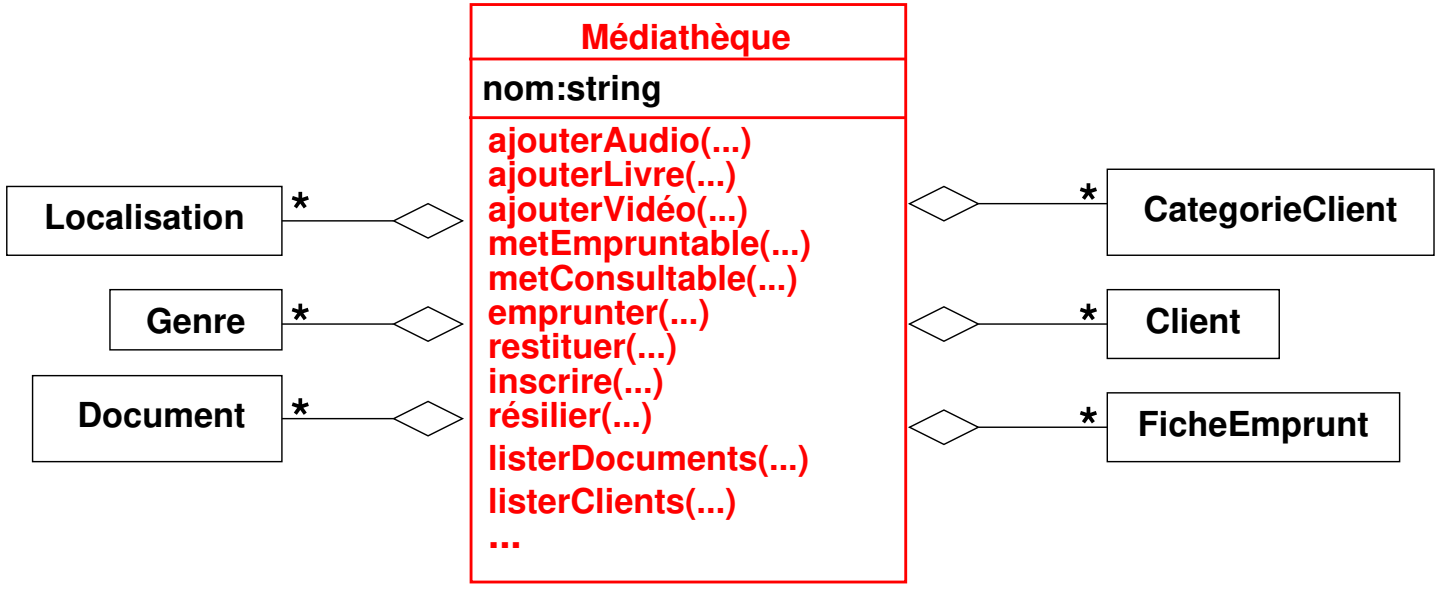
2.7 Diagramme avec les nouveaux concepts

Attribut/opération de classe, classe abstraite, classe d'association, énumération... façade



2.7.1 Premier patron de conception : « Façade »*

- Façade = point d'entrée du système
  - Les cas d'utilisation deviennent les opérations de la façade
    - L'algorithme de l'opération d'un cas d'utilisation est un parcours du graphe d'objets à partir de la façade
- Nous mettrons toujours une façade dans nos études de cas

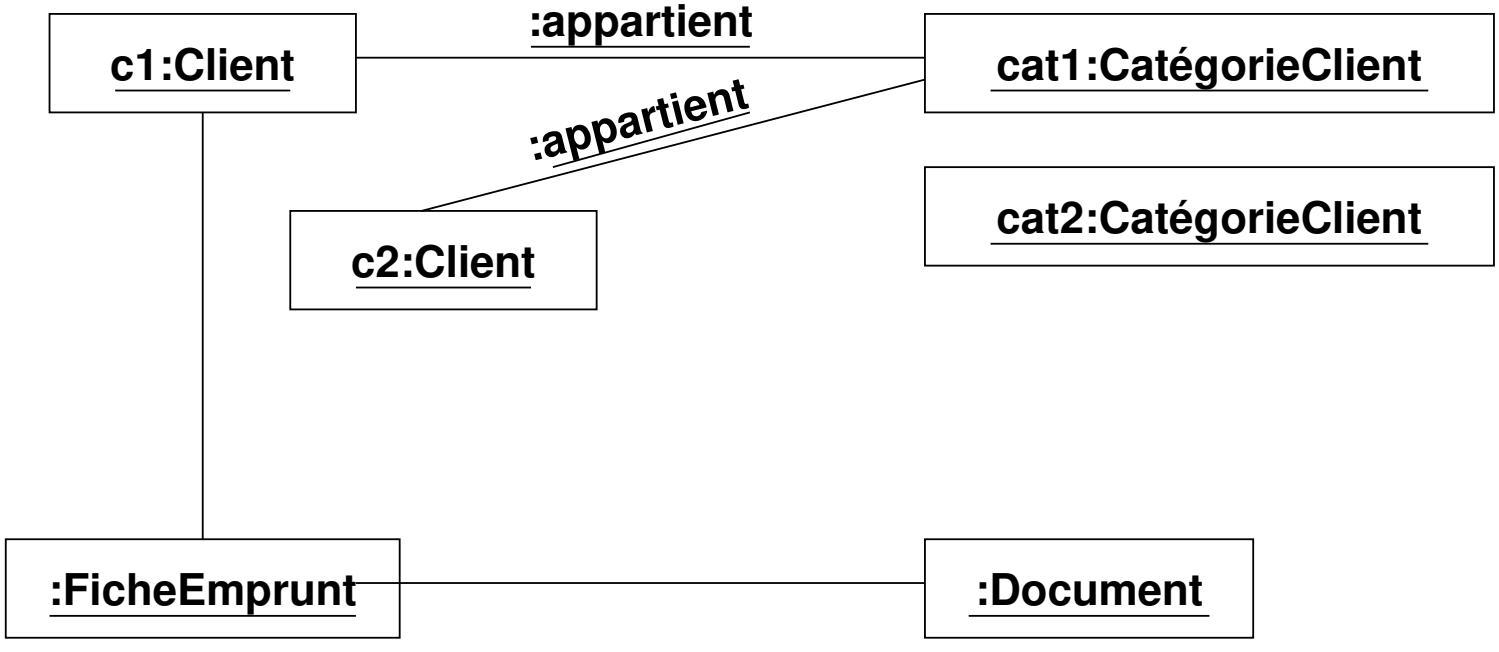


2.8 Éléments de méthodologie

- Analysez le texte pour rechercher les noms qui sont potentiellement des classes métier et les verbes qui les relient qui sont potentiellement des associations
  - Portez une attention particulière à :
    - « est une sorte de » (généralisation spécialisation)
    - « contient » (agrégation)
      - La composition (losange noir) sera étudiée à la prochaine séance
  - Indiquez les multiplicités des associations
- Construisez le diagramme de classes à partir de ces noms et de ces verbes
- Affinez le diagramme en éliminant les associations redondantes, en simplifiant le schéma tant qu'il respecte le cahier des charges
- Vérifiez que tous les concepts métier sont présents
- Vérifiez que les cas d'utilisation sont réalisables par parcours du graphe
- La liste des attributs/opérations sera complétée ultérieurement

3 Aspects statiques, diagramme d'objets

- Illustration des liens entre objets dans une configuration donnée
  - Permet d'exercer par exemple la multiplicité des associations
- Instance de classe = un nom (ou anonyme) + le nom de la classe + souligné
- Instance d'association = un nom (ou anonyme) + le nom de l'association + souligné

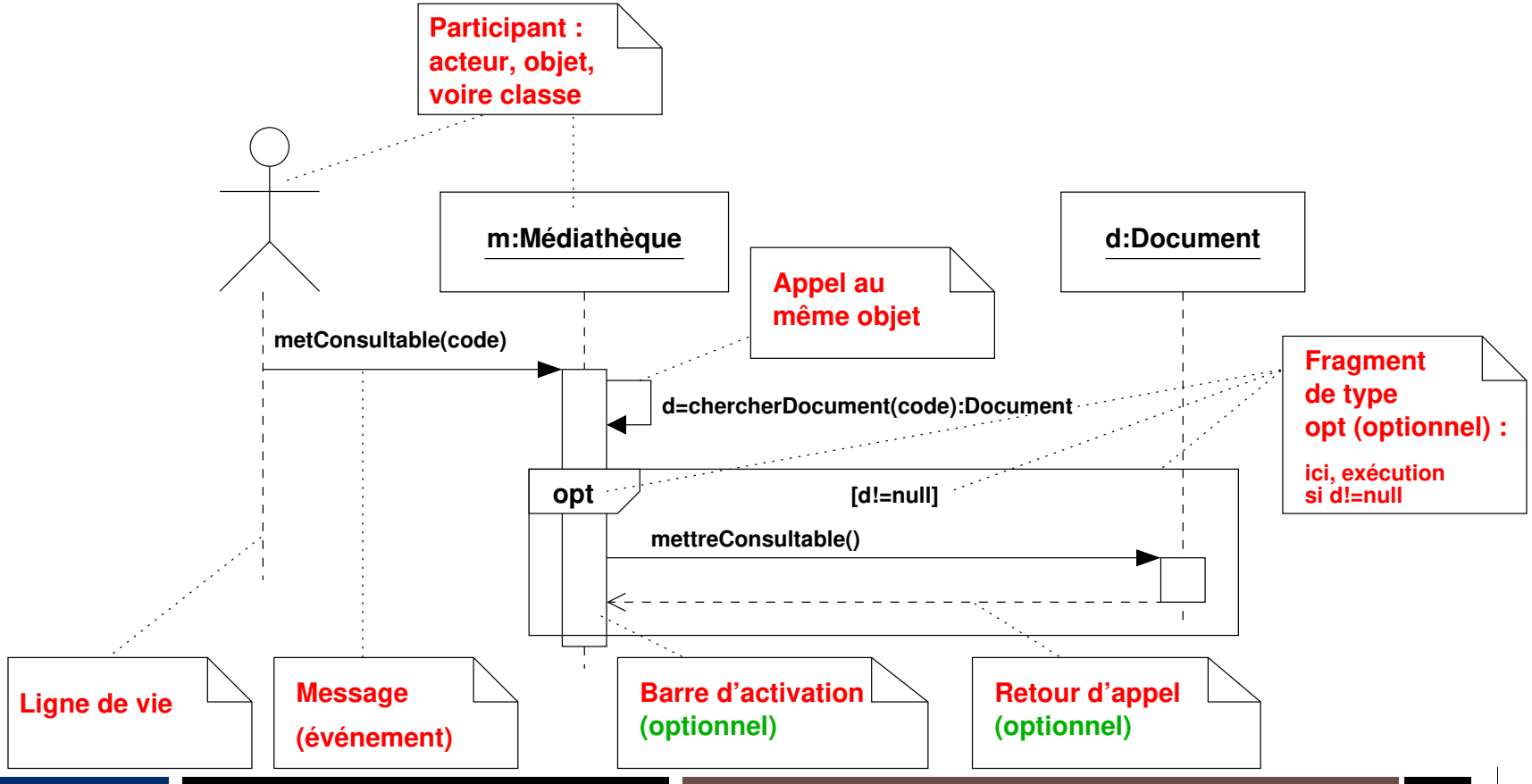


4 Aspects dynamiques, diagramme de séquence

4.1 Participant, message, barre d'activation, fragment
4.2 Syntaxe et types de messages
4.3 Création d'objet, et fragments `loop` et `ref`
4.4 Suppression d'objet
4.5 Fragments `alt` et `break`
4.6 Éléments de méthodologie
4.7 Exemple de description textuelle d'une séquence avant sa modélisation

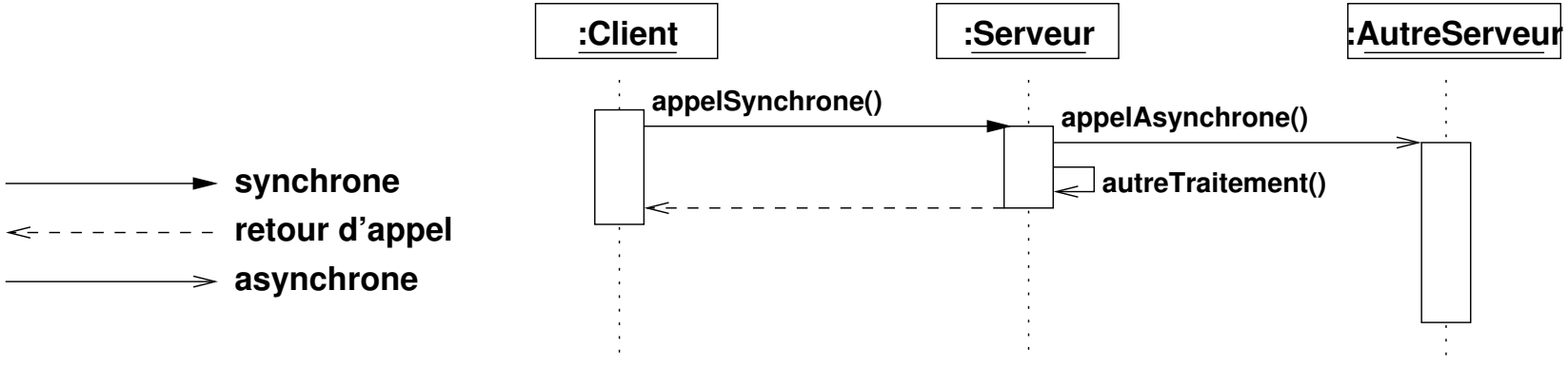
4.1 Participant, message, barre d'activation, fragment

- Diag. de séquence capture l'aspect temporel des échanges entre participants
- Pour nous : diagramme de séquence = comportement (normal, c.-à-d. sans erreur) d'un cas d'utilisation



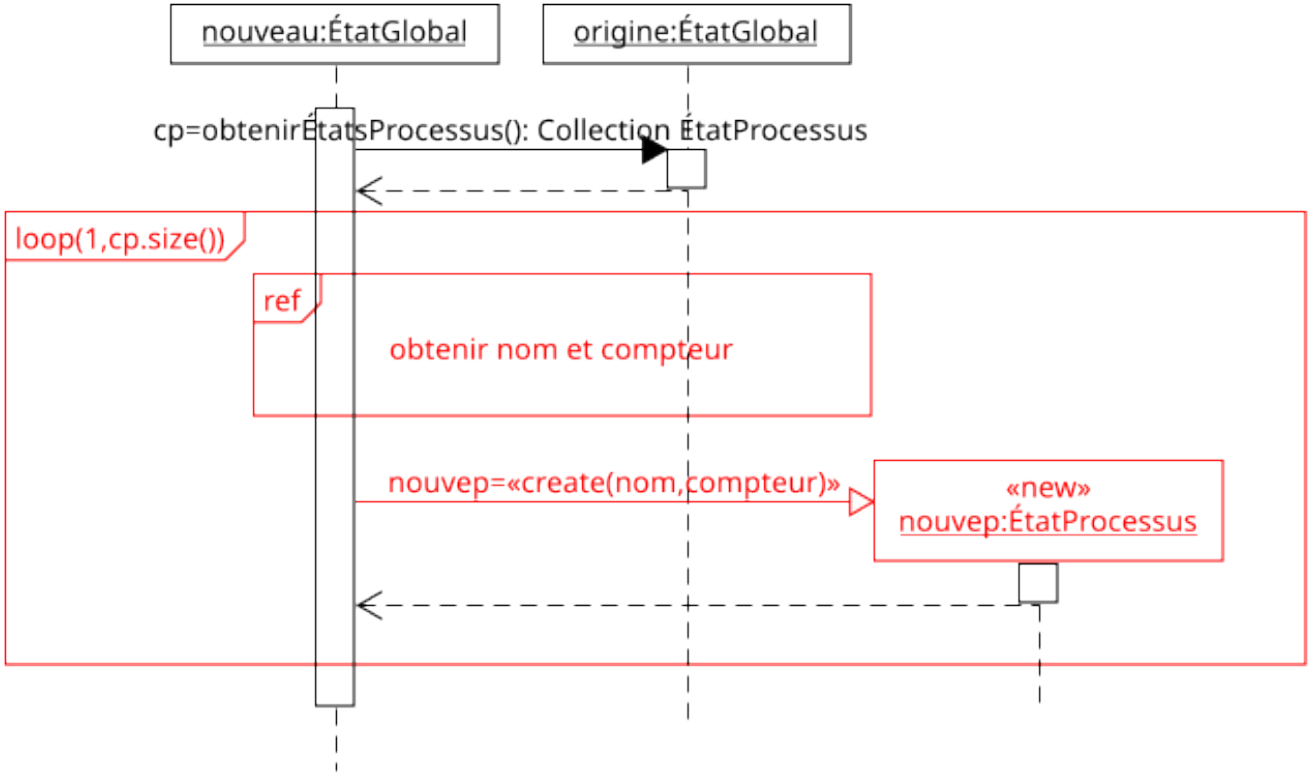
4.2 Syntaxe et types de messages

- Syntaxe complète
  - `variable = nomMessage(arguments) : type_retour`
    - `variable` peut être un attribut de l'objet appelant ou une variable locale
    - `nomMessage` est une opération de l'objet appelé (destinataire du message)
- Appel synchrone[7] : l'expéditeur est bloqué pendant le traitement
  - La représentation visuelle du retour d'appel est optionnelle (car implicite)
- Appel asynchrone : l'appelant continue son exécution pendant le traitement du message par l'appelé

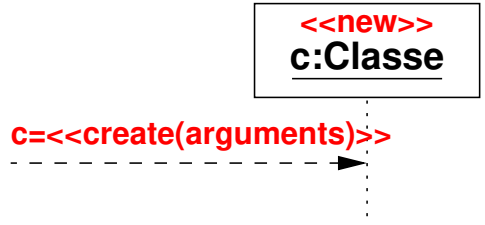


7. Dans le module, nous n'utilisons que des appels synchrones

4.3 Création d'objet, et fragments `loop` et `ref`

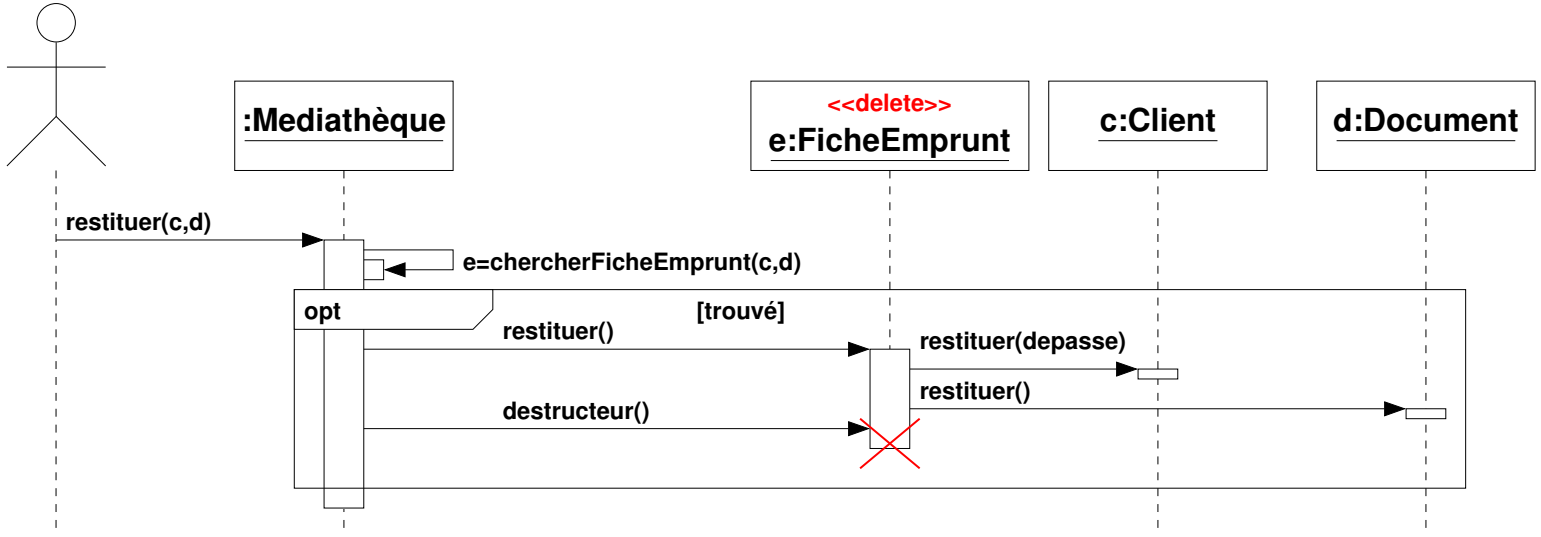


- Autre façon de modéliser la création d'un objet :

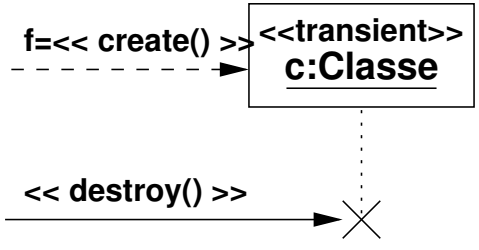


. Dans les outils, `seq` pour parcours en séquentiel et `par` pour parcours en parallèle

4.4 Suppression d'objet

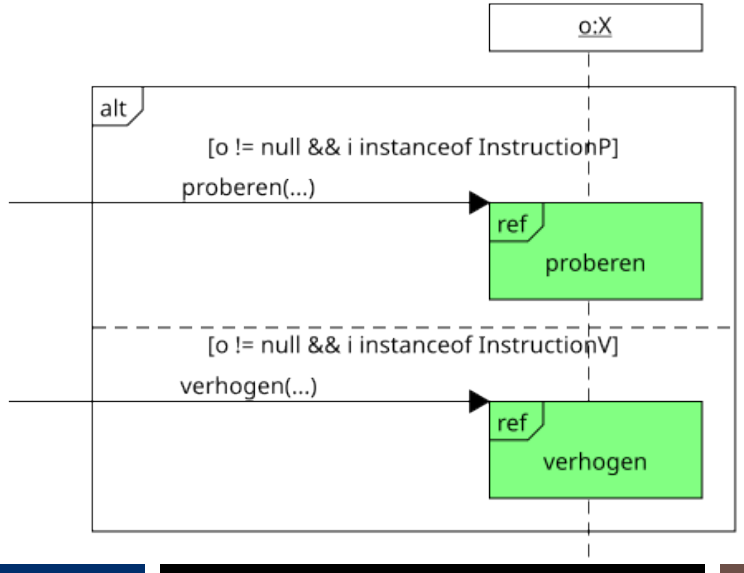


- Un objet peut aussi être éphémère
  c.-à-d., être créé puis détruit dans la même séquence

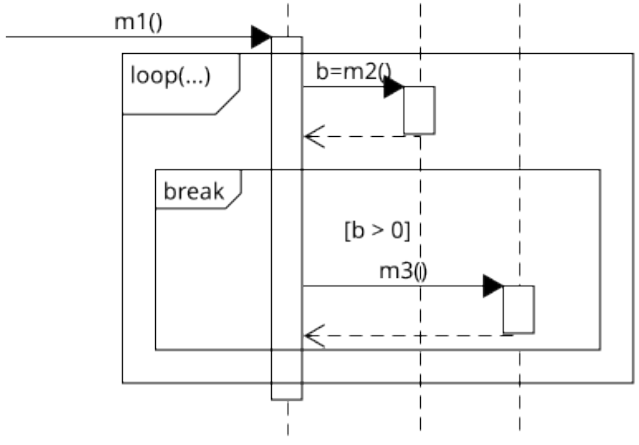


4.5 Fragments `alt` et `break`

- `alt` = `Si (...) { ... } Sinon Si (...) { ... } FinSi`
- Dans le module, nous ne modélisons pas les cas d'erreur
  - C.-à-d., condition d'un fragment opt fausse $\Longrightarrow$ message d'erreur
  - Autrement dit, nous n'utilisons pas de fragment `alt` pour la gestion des erreurs



- `break` = dans un fragment `loop`, interruption de l'itération et sortie de la boucle
- Dans le module, nous n'utilisons pas de fragment `break` pour la gestion des erreurs



4.6 Éléments de méthodologie

- Choisir les cas d'utilisation « importants »
- Modéliser le ou les cas normaux, c.-à-d. sans erreur :
  - Vérifier la cohérence entre les arguments de l'opération de la façade et les variables des préconditions et des postconditions
  - Écrire textuellement toutes les étapes de la séquence avant de débuter la modélisation (Cf. diapositive qui suit)
    1. Vérification de la précondition
    2. Effets du cas d'utilisation sur le système
  - Dans nos premiers diagrammes, débuter la séquence par un appel de l'acteur sur la façade
  - Vérifier que les messages sont envoyés via des associations qui existent
    - Cf. conception détaillée (séance 4) dans laquelle certaines associations ne seront plus navigables dans les deux sens

4.7 Exemple de description textuelle d'une séquence avant sa modélisation

- Plus qu'un conseil : rappelez-vous la précondition du cas d'utilisation !
- Par exemple, le cas d'utilisation « emprunter » de la médiathèque

1. Vérifier que l'emprunt est possible, c'est-à-dire vérifier[8]
   - 1.1 Si le client peut emprunter
     avec le nombre d'emprunts maximum dépendant de la catégorie du client
   - 1.2 Si le document est empruntable
   - 1.3 Si le document n'est pas déjà emprunté
2. Créer une fiche d'emprunt
   - 2.1 Déterminer la date de retour du prêt qui est indiquée au client
   - 2.2 Mettre à jour l'objet client
   - 2.3 Mettre à jour l'objet document
     - Qui met à jour l'objet genre
   - 2.4 Calculer et afficher le coût de l'emprunt
     selon le type de document et selon la catégorie du client

8. Version simplifiée, car nous ne vérifions pas si les arguments sont bien formés

5 Mise en pratique en TP (2h) + HP (3h)

- À partir du cahier des charges et de la spécification
  - Liste de toutes les classes + quelques attributs et opérations
  - Diagramme de classes (pas besoin d'y mettre les opérations)
  - Diagrammes de séquence
    - Commencez par les cas d'utilisation dont vous avez les préconditions et postconditions
- Rendu de la séance en HP : diagrammes PlantUML + `readme.md`
  - Vérifiez bien que les images `svg` et `png` sont à jour !
- Compléments « Pour aller plus loin » sur la conception préliminaire

Références I

Abadi, M. and Cardelli, L. (1996).
A Theory of Objects.
Springer-Verlag.

Fowler, M. (2004).
Is Design Dead ?
http://www.martinfowler.com/articles/designDead.html.

Kruchten, P. (1995).
The 4+1 View Model of Architecture.
IEEE Software, 12(6) :42–50.

Liskov, B. and Wing, J. (1994).
A Behavioral Notion of Subtyping.
ACM Transactions on Programming Languages and Systems, 16(6) :1811–1841.

Petre, M. (2013).
UML in practice.
In *Proc. 35th ACM International Conference on Software Engineering*, pages 722–731, San Francisco, USA.

Qui utilise UML ?

Marian Petre, « *UML in practice* », *Empirical study, over 2 years* [Petre, 2013]

- *[Four main] patterns of use of UML :*
  1. *No UML (35/50)*: another in-house formalism
     E.g., [for] the "whole" system (software + hardware)
     The notation doesn't really matter
  2. *Retrofit (1/50)*: *customer requirement*, *don't really use UML, but retrofit UML*
  3. *Automated code generation (3/50) : UML is not used in design, but is used to capture the design when it stabilizes, in order to generate code automatically*
  4. *Selective (11/50) : [UML is used in a personal way] after which it is discarded*
- *"Not by me, but by someone else"*
- *Overall conclusion stated by the author and inline with the interviewees :*
  - *UML is effective in software engineering education, because of what it captures, and where it directs attention*