Développement de logiciel avec UML

Survol et exemples

Pascal ANDRE

MIAGE
Université de Nantes

Master Miage M1

Introduction

UML : un langage de spécification multi-formalisme

UML : précision avec OCL

UML : une méthode de développement

UML : un processus unifié

UML : outils et vérification

Perspectives

Plan

Introduction

UML : un langage de spécification multi-formalisme

UML : précision avec OCL

UML : une méthode de développement

UML : un processus unifié

UML : outils et vérification

Perspectives

Introduction : le contexte

Développement à objets :

- méthodes d'analyse classiques $\Longrightarrow$ orientation objet
- programmation à objets (Smalltalk, C++, Eiffel, CLOS, Java, C#...)

Introduction : le contexte

Développement à objets :

- méthodes d'analyse classiques $\Longrightarrow$ orientation objet
- conception à objets = clé du développement à objet
- programmation à objets (Smalltalk, C++, Eiffel, CLOS, Java, C#...)

Introduction : le contexte

Développement à objets :

- méthodes d'analyse classiques $\Longrightarrow$ orientation objet
  $\rightsquigarrow$ business process
- conception à objets = clé du développement à objet
  $\rightsquigarrow$ encore en mutation
- programmation à objets (Smalltalk, C++, Eiffel, CLOS, Java, C#...)
  $\rightsquigarrow$ classes, components, web

Introduction : le contexte

Développement à objets :

- méthodes d'analyse classiques $\Longrightarrow$ orientation objet
  $\rightsquigarrow$ business process
- conception à objets = clé du développement à objet
  $\rightsquigarrow$ encore en mutation
- programmation à objets (Smalltalk, C++, Eiffel, CLOS, Java, C#...)
  $\rightsquigarrow$ classes, components, web

∇ concepts et utilisation $\rightsquigarrow$ chapitre 6 [AV01b]

Introduction : le constat

Constat (fin des années 1980)

- évolution rapide des technologies (Web)
- complexité croissante des besoins et des applications
- technologies anciennes limitées pour les nouvelles applications
  $\Longrightarrow$ migration technologique
- retour d'expérience sur la technologie à objets
- nombre pléthorique de méthodes (convergence des concepts ?)
- cacophonie pour les DSI

Introduction : le constat

Constat (fin des années 1980)

- évolution rapide des technologies (Web)
- complexité croissante des besoins et des applications
- technologies anciennes limitées pour les nouvelles applications
  $\Longrightarrow$ migration technologique
- retour d'expérience sur la technologie à objets
- nombre pléthorique de méthodes (convergence des concepts ?)
- cacophonie pour les DSI

UML et UP une réponse à un besoin urgent dans le contexte du développement à objets.

Introduction : vers un standard

Objectifs = Unifier

- langage commun : normalisation
- outils standards, modèles interopérables
- couverture GL et objet
- gestion de projet à objets

Introduction : vers un standard

Objectifs = Unifier

- langage commun : normalisation
- outils standards, modèles interopérables
- couverture GL et objet
- gestion de projet à objets

Deux approches dans la normalisation :

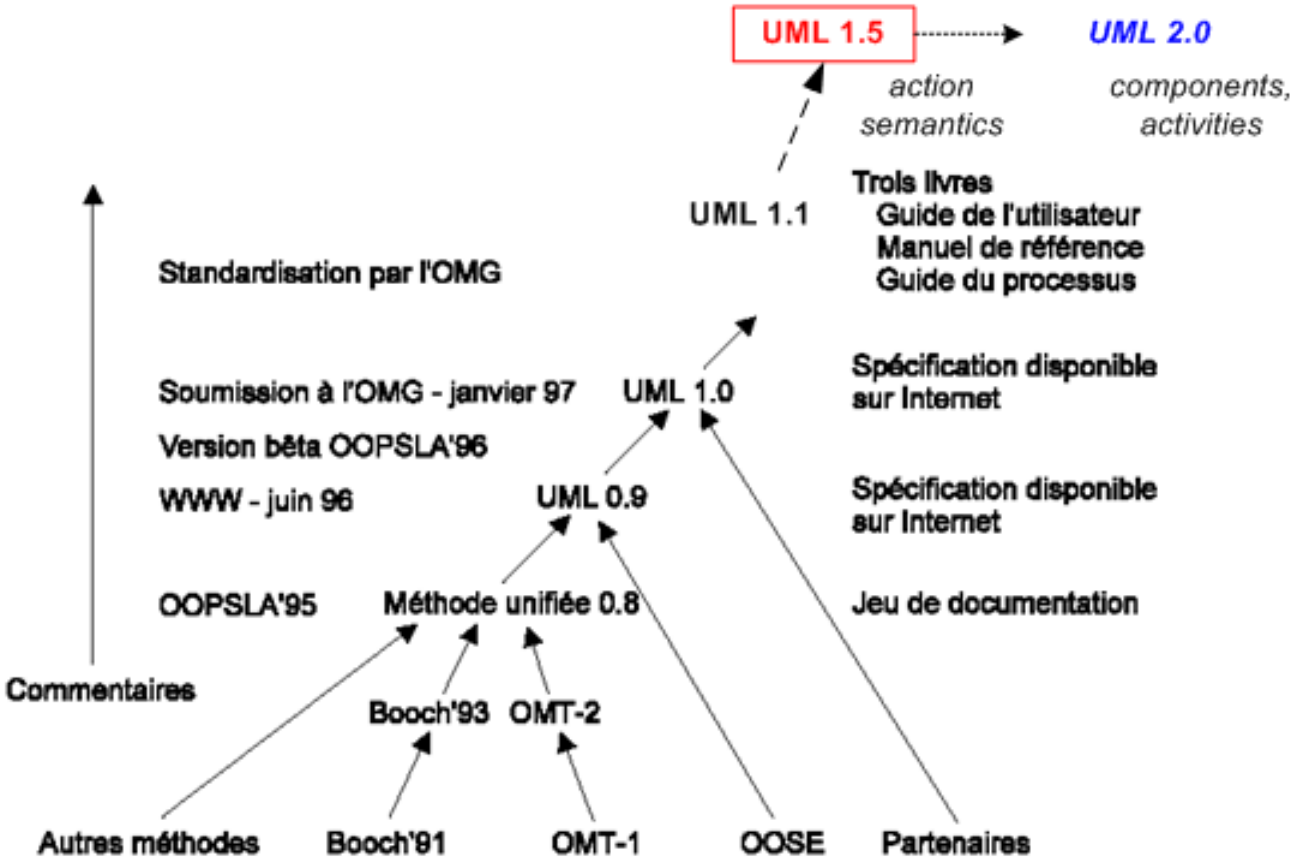
- plus petit dénominateur commun : CORBA/IDL
- rassembleur : UML

Introduction : état des lieux - Notation

Langage = Unified Modeling Language

- convergence, stabilisation : versions 0.8 à 2.0
- acceptation : outils et méthodes compatibles UML
- syntaxe et règles : méta-modèle et MOF
- sémantique informelle
- évolution de la notation : une base et des personnalisations (profils)

Introduction : état des lieux - Notation



Introduction : état des lieux - Notation

Principales influences =

- Booch Catégories et sous-systèmes
- Embley Classes singletons et objets composites
- Fusion Description des opérations, numérotation des messages
- Gamma, et al.Frameworks, patterns, et notes
- Harel Automates (Statecharts)
- Jacobson Cas d'utilisation (use cases)
- Meyer Pré- et post-conditions
- Odell Classification dynamique, éclairage sur les événements
- OMT Associations
- Shlaer-MellorCycle de vie des objets
- Wirfs-Brock Responsabilités (CRC)

Plus récemment

- Composants (CBD)
- Architectures de logiciels (ADL)

Introduction : état des lieux - Processus

Processus = X Unified Process

- convergence, stabilisation :
  - principes (itératif, incrémental, architecture, UC)
  - uniquement des solutions propriétaires (RUP, Y...)
  - pratiques convergentes du développement ?
- fortement lié à l'outils
- une normalisation ? le méta-modèle SPEM

Introduction : domaine d'application

Applicable à tout développement logiciel (à objets)

- Systèmes d'information, SIG...
- Systèmes temps réels, embarqués...
- Interfaces, simulateurs, calcul
- Applications diverses

Couverture complète du cycle de développement.

- Analyse des besoins
- ...
- Intégration et tests

Bibliographie sommaire

[Gro03] Object Management Group.
The OMG Unified Modeling Language Specification, version 1.5.
Technical report, Object Management Group, available at http://www.omg.org/cgi-bin/doc?formal/03-03-01, June 2003.

[MG00] Pierre-Alain Muller and Nathalie Gaertner.
Modélisation objet avec UML.
Eyrolles, 2000.
ISBN 2-212-09122-2, 2e édition.

[AV01] Pascal André and Alain Vailly.
Spécification des logiciels, Deux exemples de pratiques récentes : Z et UML, volume 2 of Collection Technosup.
Editions Ellipses, 2001.
ISBN 2-7298-0774-8.

[AV03] Pascal André and Alain Vailly.
Exercices corrigés en UML ; Passeport pour une maîtrise de la notation., volume 5 of Collection Technosup.
Editions Ellipses, 2003.
ISBN 2-7298-1725-5.

Plan

Introduction

UML : un langage de spécification multi-formalisme

UML : précision avec OCL

UML : une méthode de développement

UML : un processus unifié

UML : outils et vérification

Perspectives

UML : langage de spécification

- Notation complète et extensible
- Structuration des concepts
- Diagrammes
- Modèles et vues
- Multi-formalisme
- Classification

UML : une notation complète et extensible

- Complète
  UML inclut un grand nombre de concepts autour de

UML : une notation complète et extensible

- Complète
  UML inclut un grand nombre de concepts autour de
  - l'objet : objets, classes, opérations, attributs, relations, envois de message, etc.

UML : une notation complète et extensible

- Complète
  UML inclut un grand nombre de concepts autour de
  - l'objet : objets, classes, opérations, attributs, relations, envois de message, etc.
  - l'analyse des besoins : acteurs, cas d'utilisation,

UML : une notation complète et extensible

- Complète
  UML inclut un grand nombre de concepts autour de
  - l'objet : objets, classes, opérations, attributs, relations, envois de message, etc.
  - l'analyse des besoins : acteurs, cas d'utilisation,
  - la conception du logiciel : composants, modules, processus,

UML : une notation complète et extensible

- Complète
  UML inclut un grand nombre de concepts autour de
  - l'objet : objets, classes, opérations, attributs, relations, envois de message, etc.
  - l'analyse des besoins : acteurs, cas d'utilisation,
  - la conception du logiciel : composants, modules, processus,
  - l'implantation : nœuds, liaisons, déploiement.

UML : une notation complète et extensible

- Complète
  UML inclut un grand nombre de concepts autour de
  - l'objet : objets, classes, opérations, attributs, relations, envois de message, etc.
  - l'analyse des besoins : acteurs, cas d'utilisation,
  - la conception du logiciel : composants, modules, processus,
  - l'implantation : nœuds, liaisons, déploiement.

  $\Longrightarrow$ décrire des modèles couvrant l'ensemble du cycle de développement.

UML : une notation complète et extensible

- Complète
  UML inclut un grand nombre de concepts autour de
  - l'objet : objets, classes, opérations, attributs, relations, envois de message, etc.
  - l'analyse des besoins : acteurs, cas d'utilisation,
  - la conception du logiciel : composants, modules, processus,
  - l'implantation : nœuds, liaisons, déploiement.

  $\Longrightarrow$ décrire des modèles couvrant l'ensemble du cycle de développement.
- Extensible

UML : une notation complète et extensible

- Complète
  UML inclut un grand nombre de concepts autour de
  - l'objet : objets, classes, opérations, attributs, relations, envois de message, etc.
  - l'analyse des besoins : acteurs, cas d'utilisation,
  - la conception du logiciel : composants, modules, processus,
  - l'implantation : nœuds, liaisons, déploiement.

  $\Longrightarrow$ décrire des modèles couvrant l'ensemble du cycle de développement.
- Extensible
  UML autorise l'enrichissement ou la personnalisation de la notation au moyen des stéréotypes.

UML : structuration des concepts

- Paquetages : point de vue utilisateur
- Diagrammes : point de vue langage $\Longrightarrow$ la seule normalisée
- Modèles : point de vue méthode
- Vues : point de vue méthode

La notation UML : diagrammes 1/3

UML 1.x propose neuf types de combinaisons cohérentes et complémentaires : les **diagrammes**.

- Les diagrammes de cas d'utilisation (UC - *Use Case*) décrivent les acteurs et l'utilisation du système.
- Les diagrammes de classes représentent les classes et les relations statiques entre ces classes : classe, attribut, opération, visibilité, interface, association, agrégation, héritage, dépendance...
- Les diagrammes d'objets (pas toujours considéré comme un diagramme) décrivent des objets et des liens. Les objets peuvent être actifs et définir leur flot de contrôle. Sur ces liens (réels ou virtuels) circulent des messages. Les envois de messages sont synchrones ou asynchrones, avec ou sans résultats.

La notation UML : diagrammes 2/3

- Les diagrammes d'objets se retrouvent sous deux formes dans UML :
  - Les diagrammes de séquence, qui donnent une vision temporelle des interactions en objets en mettant l'accent sur l'ordonnancement des échanges entre objets.
  - Les diagrammes de collaboration, qui donnent une vision spatiale des interactions en mettant l'accent sur les liaisons entre objets.

La notation UML : diagrammes 3/3

- Les diagrammes états-transitions modélisent le comportement des objets au cours du temps.
- Les diagrammes d'activités décrivent le flot de contrôle interne aux opérations. A grande échelle, ils représentent aussi les échanges entre objets.
- Les diagrammes de composants mettent en évidence les composants d'implémentation et leurs relations.
- Les diagrammes de déploiement définissent la structure matérielle et la distribution des objets et des composants.

En plus : stéréotypes, paquetages, notes, contraintes.

La notation UML2

Une notation encore plus complète : 13 diagrammes

- Objets et de Paquetages deviennent des diagrammes à part entière
- Diagramme de collaboration devient (une fois simplifié) Diagramme de communication
  La collaboration devient un élément des structures composites.
- Les diagrammes de structures composites placent la hiérarchie de composition au premier plan avec une nette orientation composants et architecture de logiciels (ADL).
- ...

La notation UML2 (suite)

- ...
- Les diagrammes d'interaction sont un mélange d'activités et de séquence.
- Les diagrammes de temps (*timing*) permettent la description d'évolution temporelle usuelle en génie électrique.

Par ailleurs, les diagrammes d'activités sont fortement enrichis pour inclure les DFD.

Aperçu de la notation UML

- Principaux concepts pour chaque diagramme
- Exemples
- Pas de considération métamodèle
  - concepts communs (stéréotypes, relations, notes, paquetages, sous-systèmes...)
  - factorisations "sémantiques"

  car ils évoluent en permanence.

  Par exemple les stéréotypes sont fortement associés aux profils en UML 2.0 (mots-clés), les paquetages deviennent un diagramme en UML 2.0.
- Principalement les notations UML 1.x
- Ne pas oublier les actions et OCL.

Aperçu de la notation UML

1. Modèles d'approche
2. Modèles de structure
3. Modèles de la dynamique
4. Modèles des traitements fonctionnels

Modèles d'approche

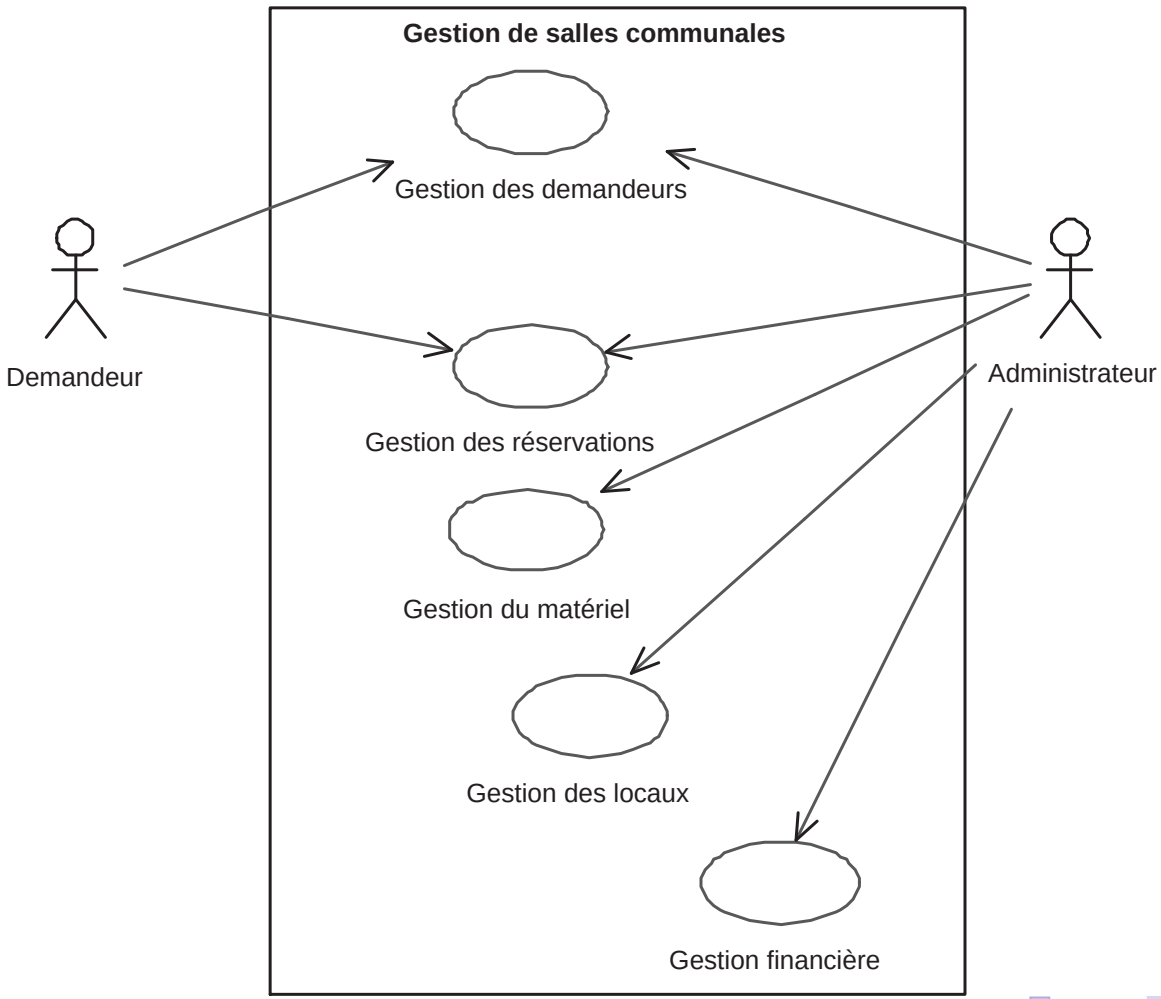
1. cas d'utilisation
2. scénarios
3. accessoirement : activités

Modèles d'approche : cas d'utilisation 1/8

Définitions [Gro03] :

- *A* **use case** *is a kind of classifier representing a coherent unit of functionality provided by a system, a subsystem, or a class as manifested by sequences of messages exchanged among the system (subsystem, class) and one or more outside interactors (called actors) together with actions performed by the system (subsystem, class).*[Gro03]
- *An* **actor** *defines a coherent set of roles that users of an entity can play when interacting with the entity. An actor may be considered to play a separate role with regard to each use case with which it communicates.*
- *An* **extension** *point is a reference to one location within a use case at which action sequences from other use cases may be inserted. Each extension point has a unique name within a use case, and a description of the location within the behavior of the use case.*

Modèles d'approche : cas d'utilisation 2/8



Modèles d'approche : cas d'utilisation 3/8

Relations :

- **Association** - *The participation of an actor in a use case; that is, instances of the actor and instances of the use case communicate with each other. This is the only relationship between actors and use cases.*
- **Extend** - *An extend relationship from use case A to use case B indicates that an instance of use case B may be augmented (subject to specific conditions specified in the extension) by the behavior specified by A. The behavior is inserted at the location defined by the extension point in B, which is referenced by the extend relationship.*
- **Generalization** - *A generalization from use case C to use case D indicates that C is a specialization of D.*
- **Include** - *An include relationship from use case E to use case F indicates that an instance of the use case E will also contain the behavior as specified by F. The behavior is included at the location which defined in E.*

Modèles d'approche : cas d'utilisation 4/8

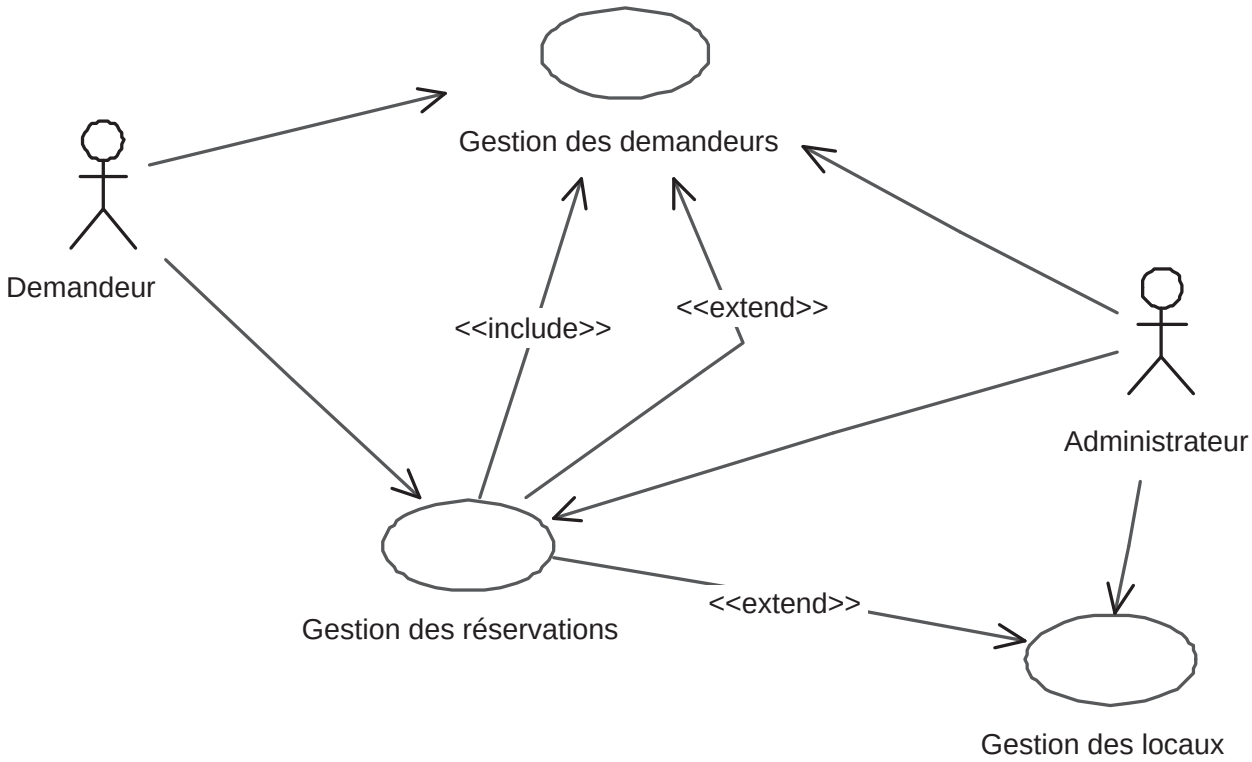


Figure 1 : *Cas d'utilisation, version à relations - Salles*

Modèles d'approche : cas d'utilisation 5/8

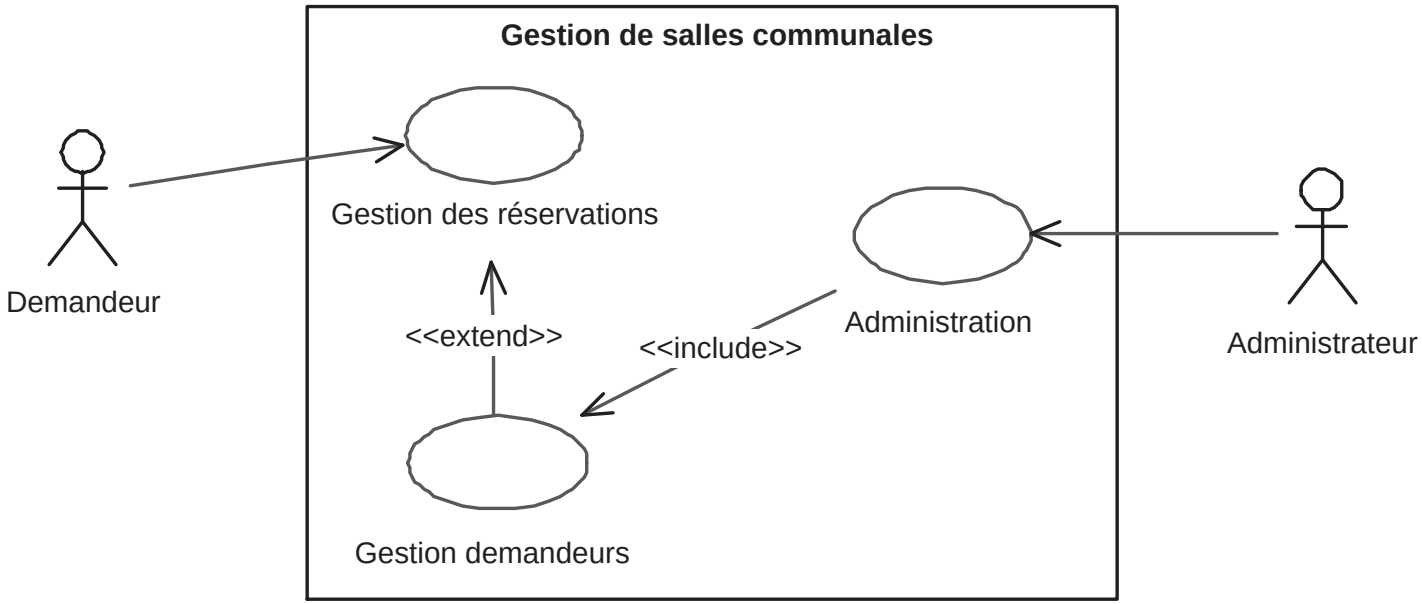


Figure 2 : *Cas d'utilisation, version à contexte - Salles*

Modèles d'approche : cas d'utilisation 6/8

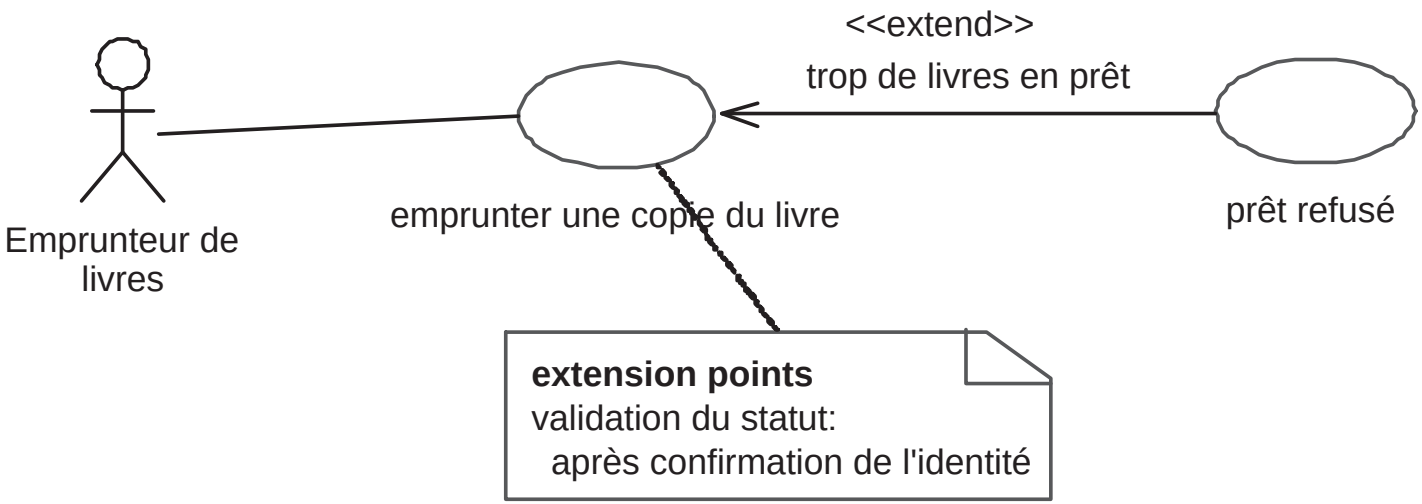


Figure 3 : *Cas d'utilisation, extensions*

Modèles d'approche : cas d'utilisation 7/8

Points clés du diagramme des cas d'utilisation

- Abstrait
- Granularité : entre découpage fonctionnel et modulaire
- Lisibilité
- Description textuelle

Modèles d'approche : cas d'utilisation 8/8

Cas d'utilisation : `Gestion des reservations`
acteurs primaires : `Demandeur`
invariant : `Unicite de reservation` Une salle n'est pas réservée pour deux demandeurs différents au même moment.
description La gestion des reservations comprend la réservation des salles, la consultation des réservations, l'annulation des réservations.
cas : `Réservation` Les éléments de la réservations sont saisis et recherchés dans la base en fonction de critères donnés : salle, demandeur, matériel, durée, manifestation, date. A tout moment, il est possible de consulter le planning des réservations en cours. Si tous les éléments sont corrects et qu'il n'y a pas de conflit de réservation, le montant est calculé et la réservations confirmée. Le numéro de la réservation est fourni par le système au demandeur.

Modèles d'approche : cas d'utilisation 8/8 (suite)

Cas d'utilisation : `Gestion des reservations (suite)`	
cas :	`Consultation des reservations`
La consultation prend plusieurs formes : recherche d'une réservation par son numéro, par demandeur, par date ou par salle, consultation du planning des réservations.	
cas :	`Annulation d'une reservation`
Après recherche de la réservation, le demandeur confirme sa suppression.	
exceptions	
cas :	`Reservation avec un demandeur inexistant`
précondition :	Le demandeur n'est pas inscrit.
résultat :	Il y a création du demandeur (voir UC Gestion des demandeurs) avant d'établir la réservation.

Modèles d'approche

1. cas d'utilisation
2. scénarios
3. accessoirement : activités

Modèles d'approche : scénarios 1/3

diagramme de séquence simplifié

- voir diagramme de séquence
- objets (acteurs + système)
- envoi de message (paramètres...)
- commentaires

Modèles d'approche : scénarios 2/3

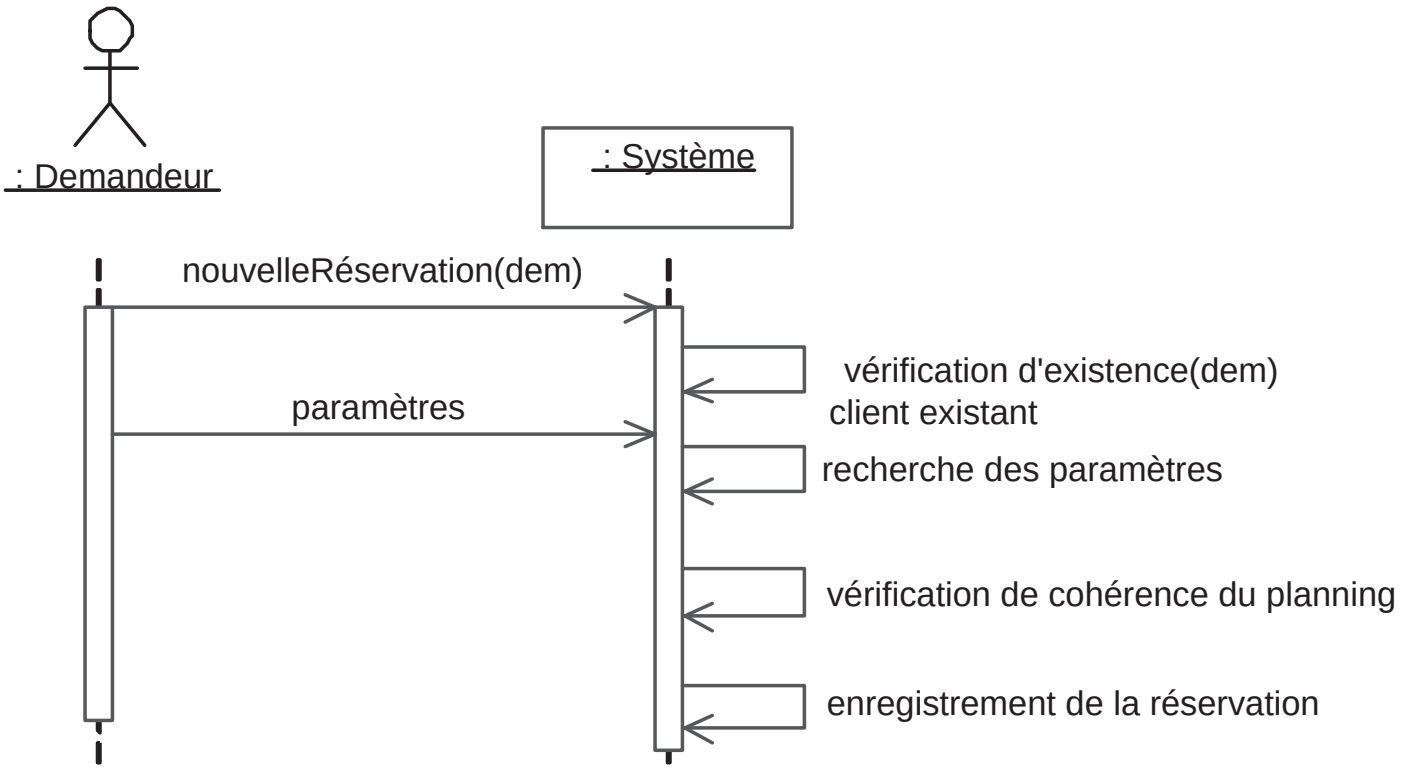
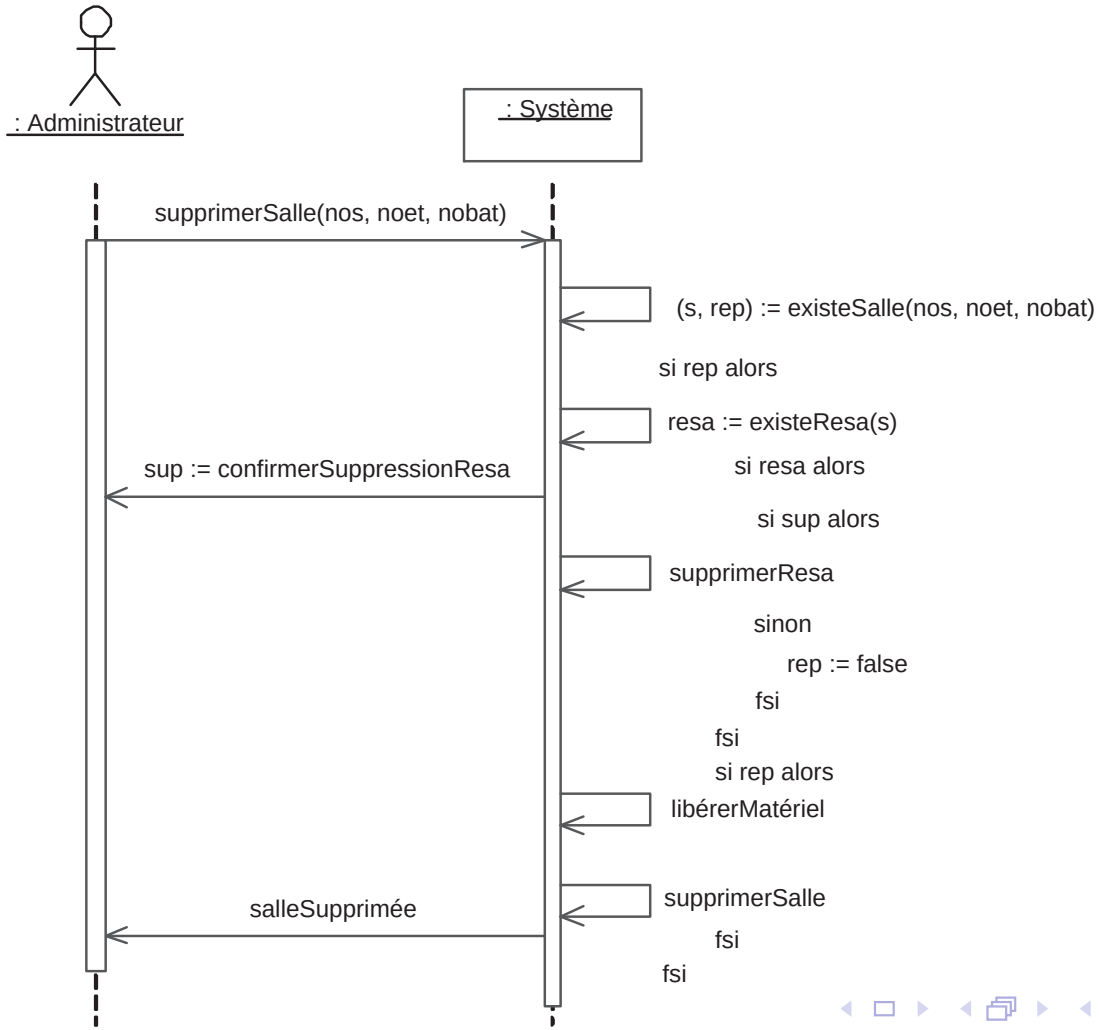


Figure 4 : *Scénario normal de l'ajout d'une réservation - Salles*

Modèles d'approche : scénarios 3/3



Modèles d'approche

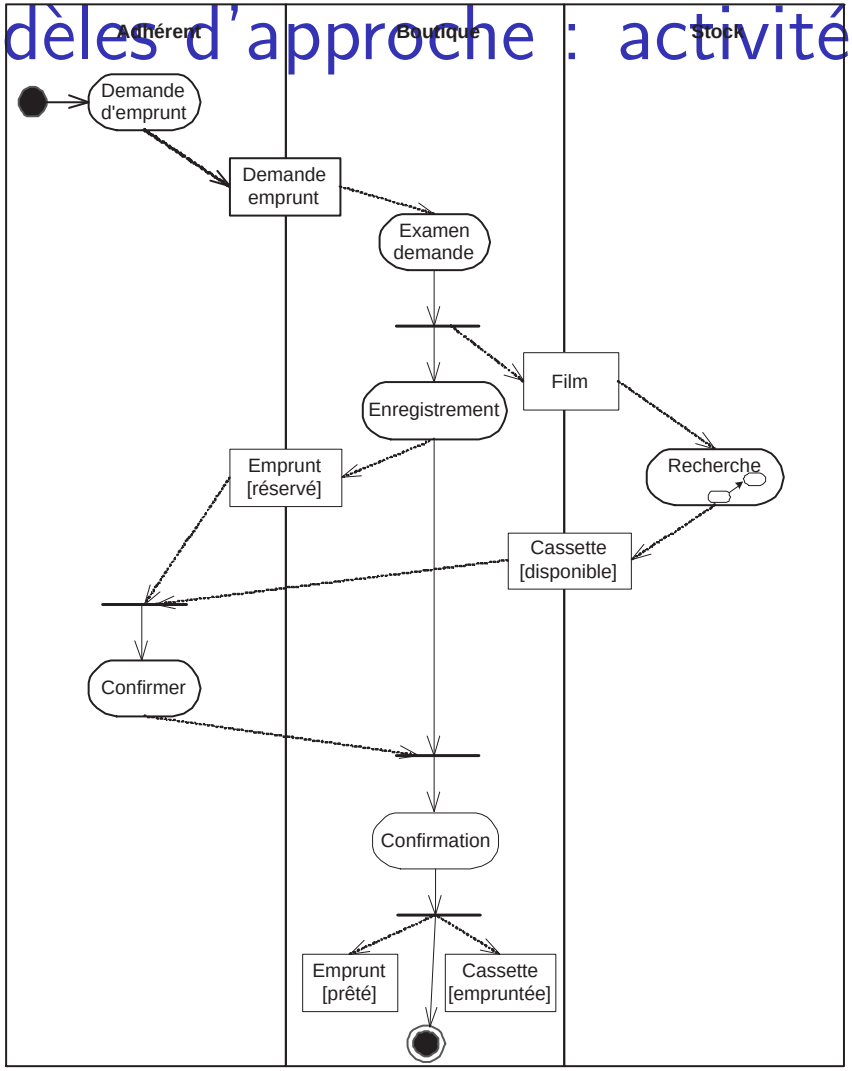
1. cas d'utilisation
2. scénarios
3. accessoirement : activités

Modèles d'approche : activité étendue 1/2

diagramme d'activité étendu

- voir diagramme d'activité
- activités, actions
- structures de contrôle
- envoi/réception de message ou signaux (paramètres...)
- flots et synchronisations
- couloirs (noms, objets, rôles...)

Modèles d'approche : activité étendue 2/2



Aperçu de la notation UML

1. Modèles d'approche
2. Modèles de structure
3. Modèles de la dynamique
4. Modèles des traitements fonctionnels

Modèles de structure

1. collaborations
2. classes
3. composants
4. déploiement

Modèles de structure : collaborations 1/6

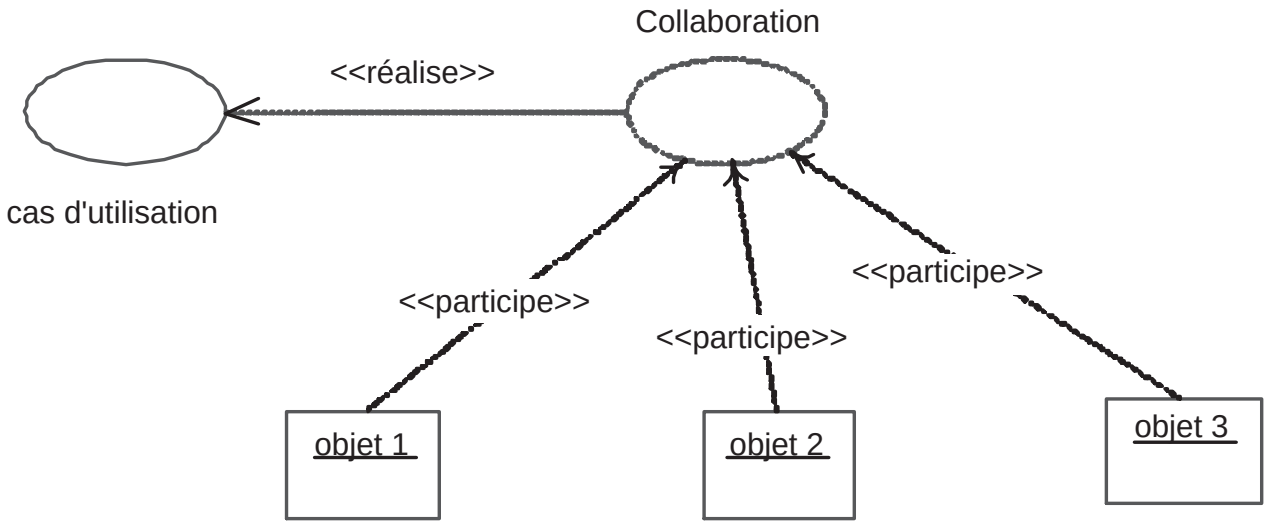
diagramme d'objet avec messages

- objets et liens, multi-objets, objet actif, instance
- envoi de message (numérotation hiérarchique, paramètres...)
- structures de contrôle, synchronisation

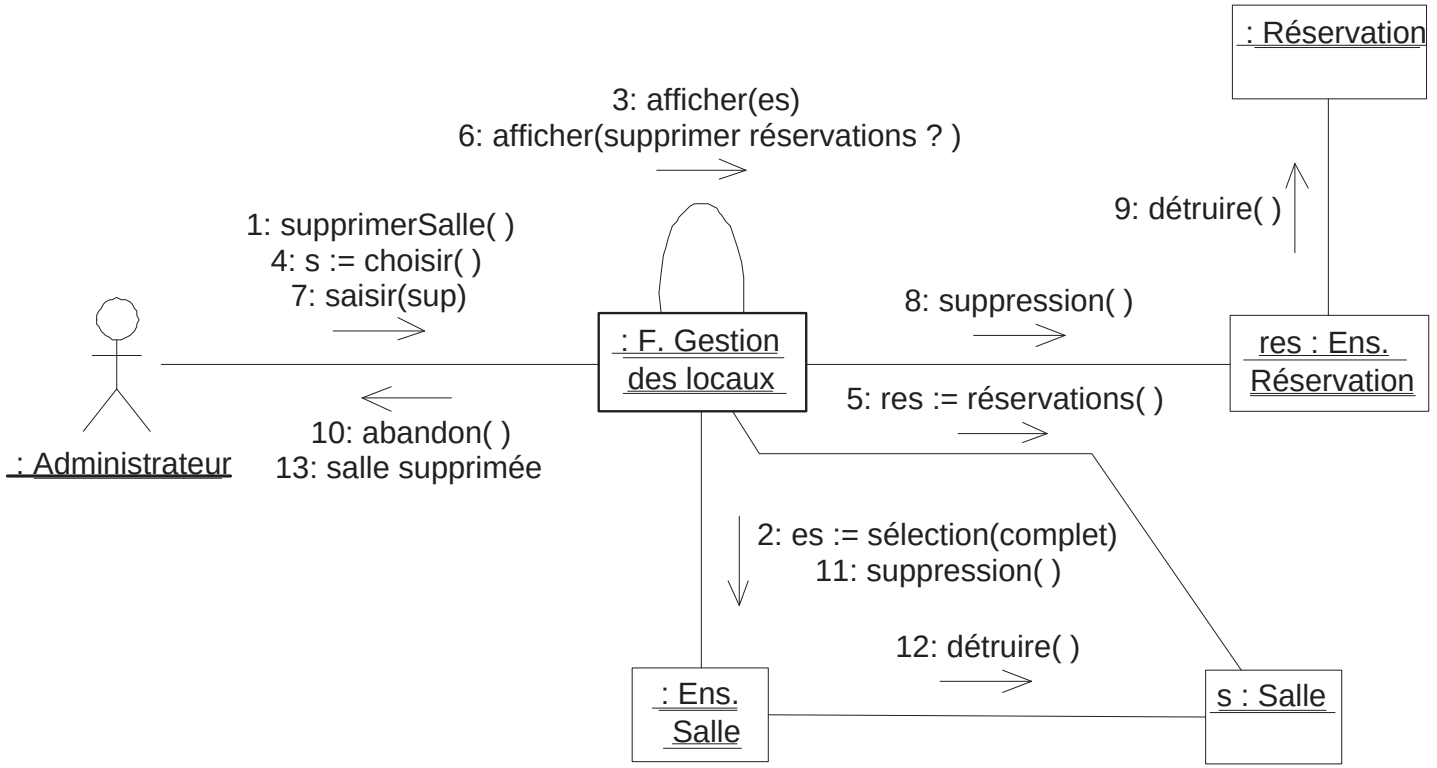
UML 2.0 $\Longrightarrow$ diagramme de communication tandis que la collaboration fait partie des structures composites

UML 2.0 $\Longrightarrow$ diagramme de communication : pas de gardes et d'itération

Modèles de structure : collaborations 2/6



Modèles de structure : collaborations 3/6



Modèles de structure : collaborations 4/6

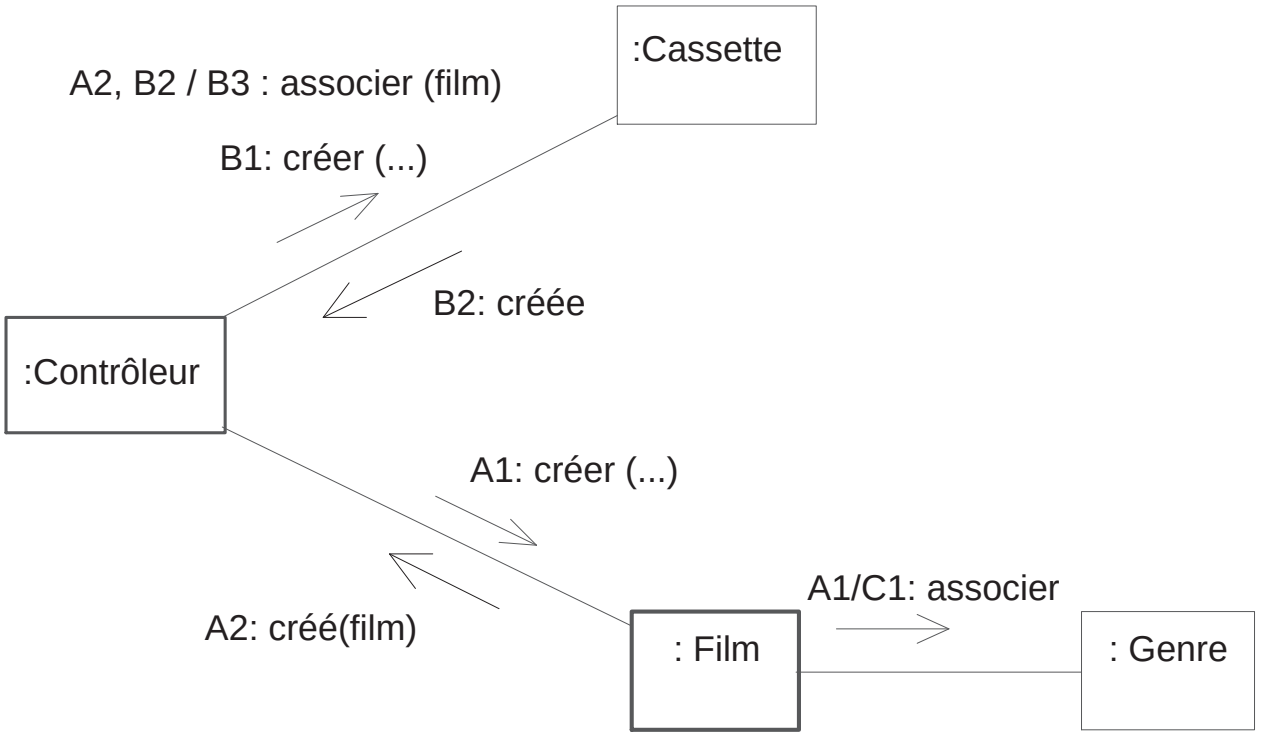
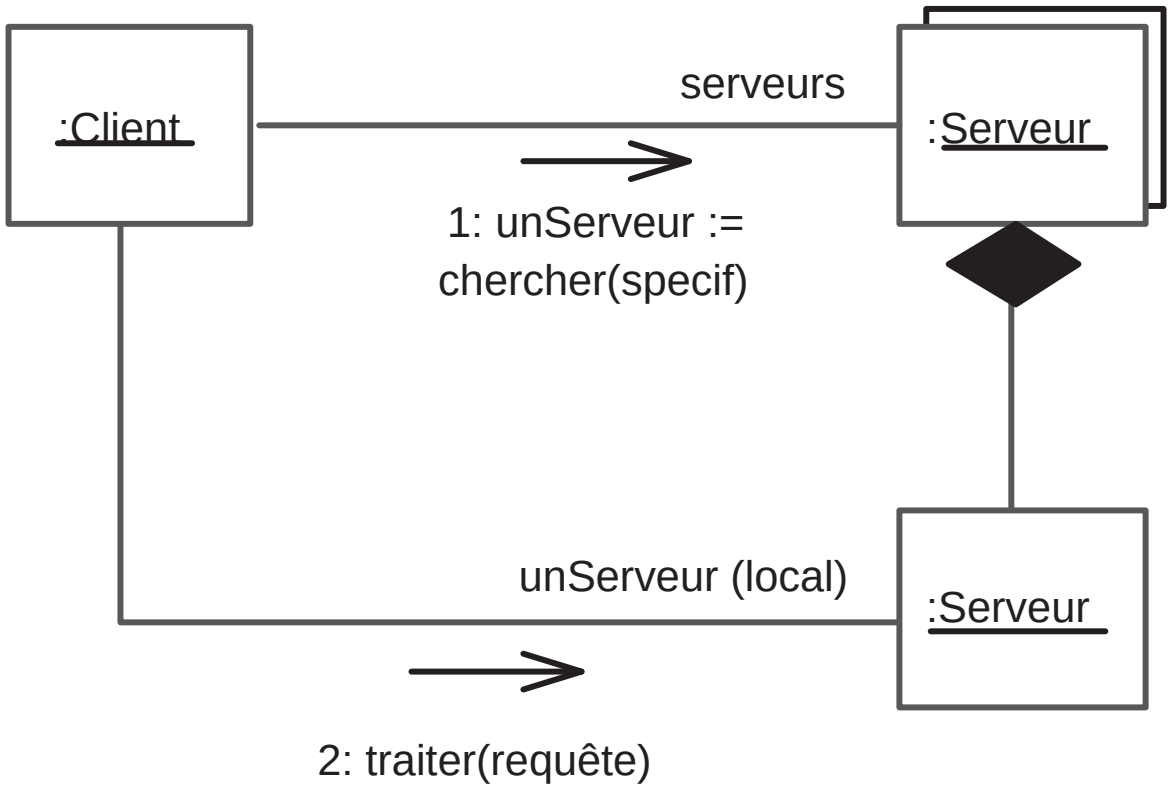


Figure 6 : *Diagramme de collaboration, synchronisation - Club vidéo*

Modèles de structure : collaborations 5/6



Modèles de structure

1. collaborations
2. classes
3. composants
4. déploiement

Modèles de structure : classes 1/9

diagramme de classes = la référence en UML ...
et dans le MOF (!)

- E-A-P : classes, propriétés (attributs, opérations)
- associations, agrégation, composition,
- propriétés dérivées, classes-associations, qualification
- généralisation/spécialisation, généricité
- classes abstraites, interfaces, héritage multiple

UML 2.0 $\Longrightarrow$ classes orientée composant (ports, interfaces requises, offertes...) + structures composites

UML 2.0 $\Longrightarrow$ composant hérite de classes

Modèles de structure : classes 2/9

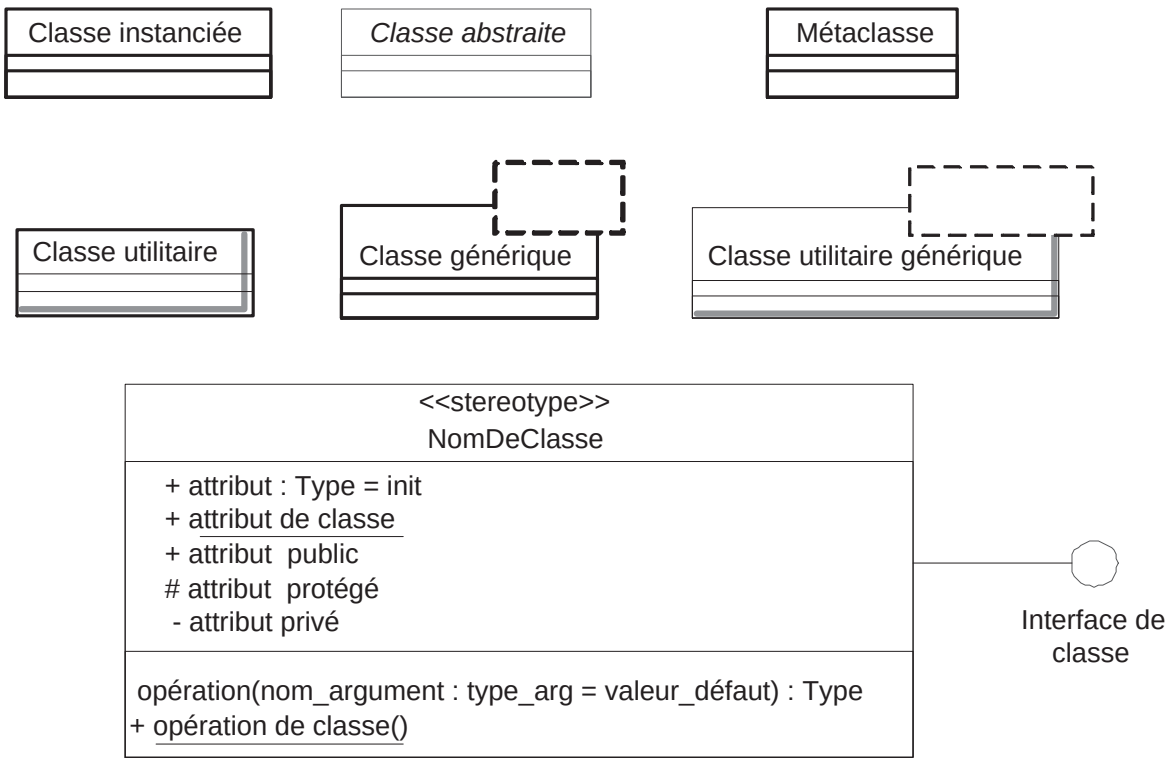


Figure 7 : *Représentations de classes*

Modèles de structure : classes 3/9

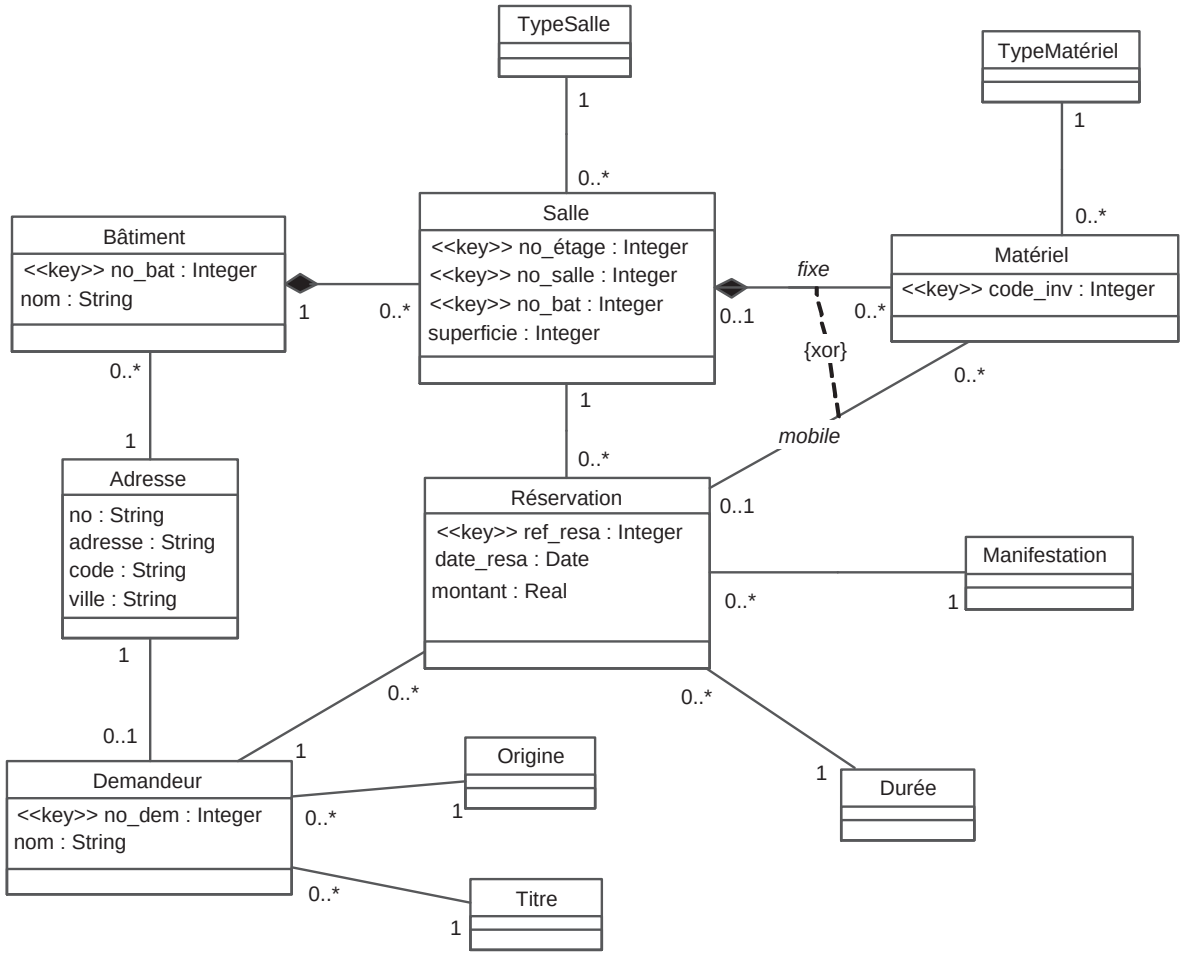


Figure 8 : *Diagramme de classes du domaine "Salles"*

Modèles de structure : classes 4/9

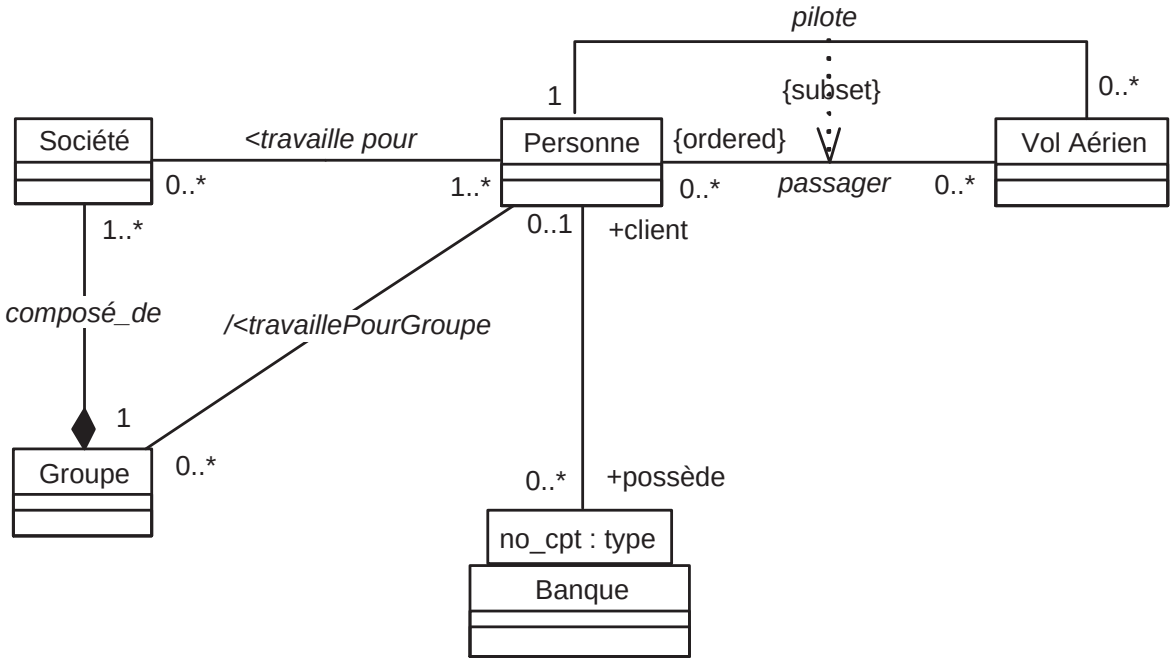


Figure 9 : *Diagramme des classes, association qualifiée*

Modèles de structure : classes 5/9

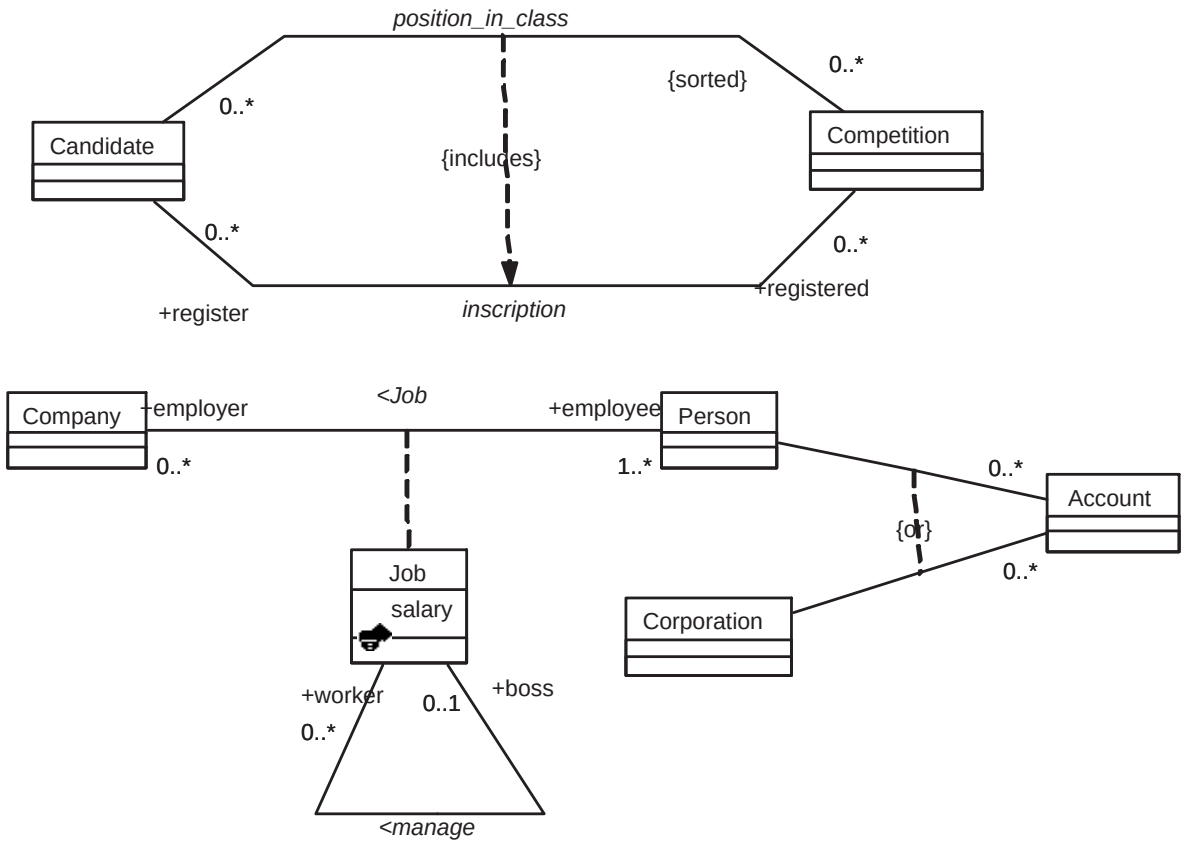


Figure 10 : *Diagramme des classes, contraintes*

Modèles de structure : classes 6/9

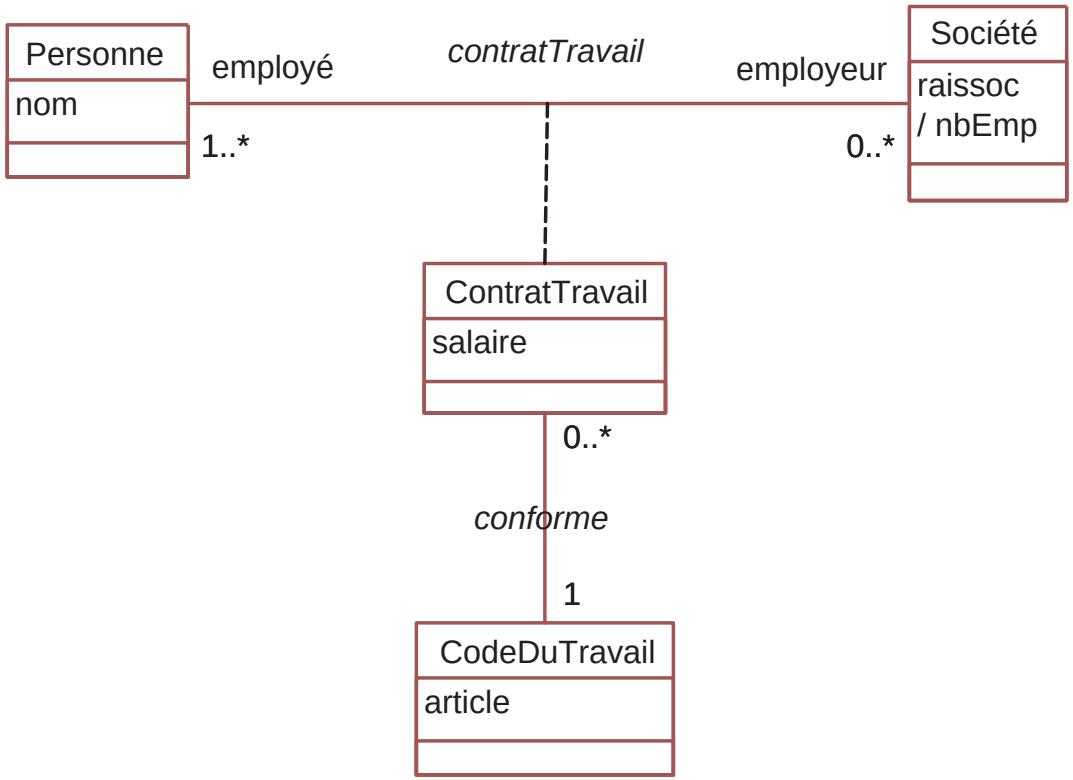


Figure 11 : *Diagramme des classes, classe-association*

Modèles de structure : classes 7/9

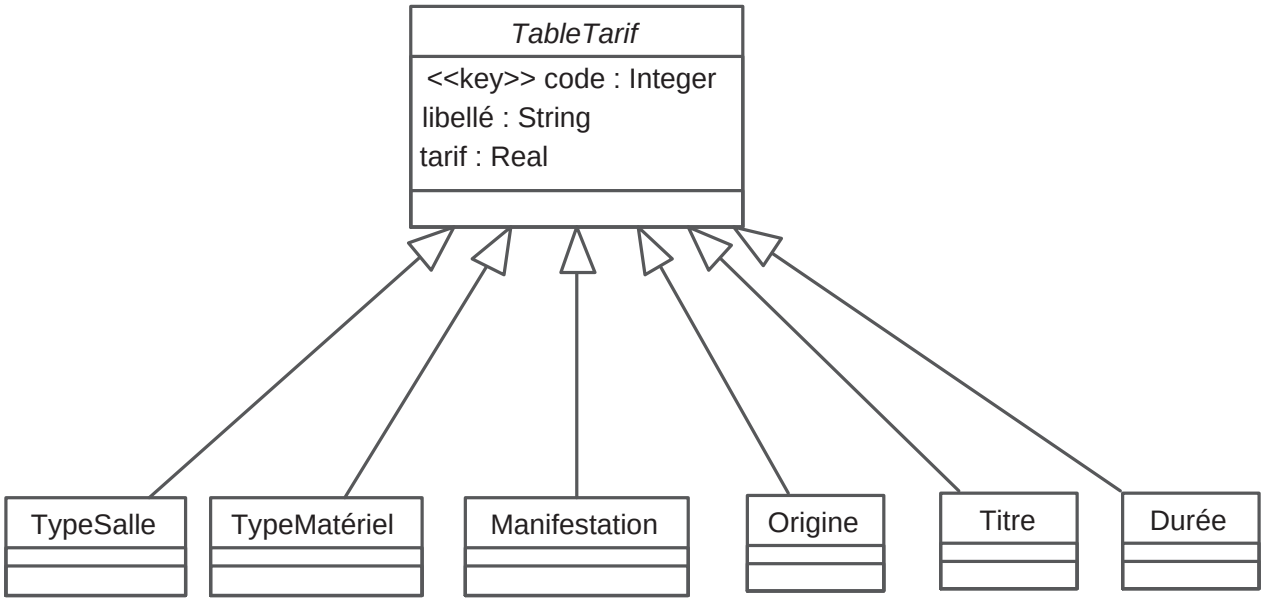


Figure 12 : *Diagramme de classes, abstraction des tables de tarifs - Salles*

Modèles de structure : classes 8/9

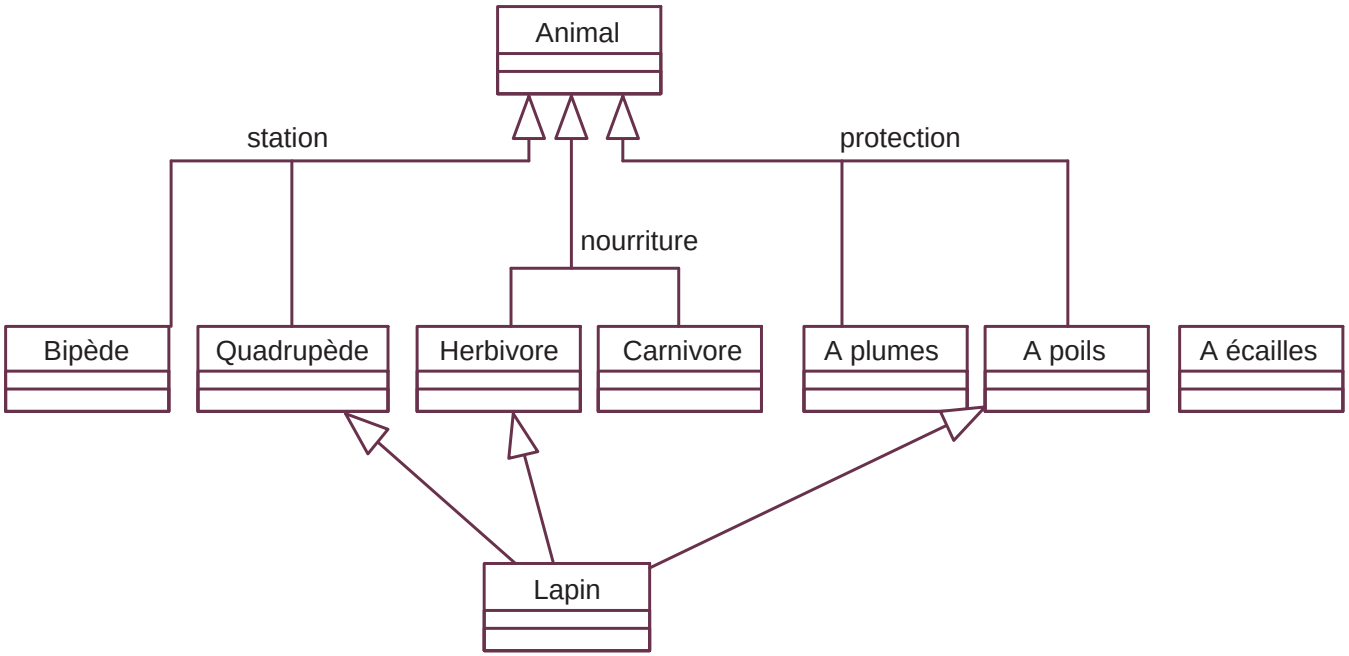


Figure 13 : *Diagramme des classes, héritage et discriminant*

Modèles de structure : classes 9/9

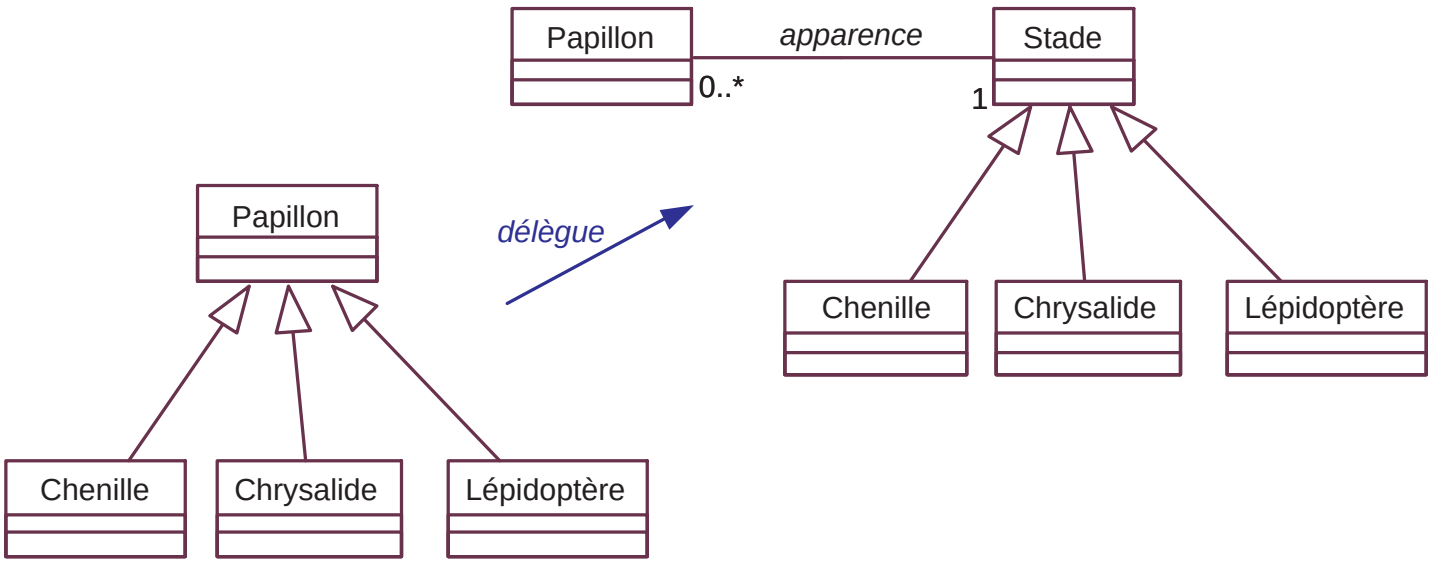


Figure 14 : *Diagramme des classes, héritage et délégation*

Modèles de structure

1. collaborations
2. classes
3. composants
4. déploiement

Modèles de structure : composants 1/3

diagramme de composants = vague en UML 1.x

- artifacts de programmation (très vaste)
- architecture du logiciel
- UML 1.x
  - composants, processus, applications, bibliothèques
  - dépendances
  - interfaces, couches

UML 2.0 $\Longrightarrow$ composant de programmation (ports, interfaces requises, offertes...) + structures composites

Modèles de structure : composants 2/3

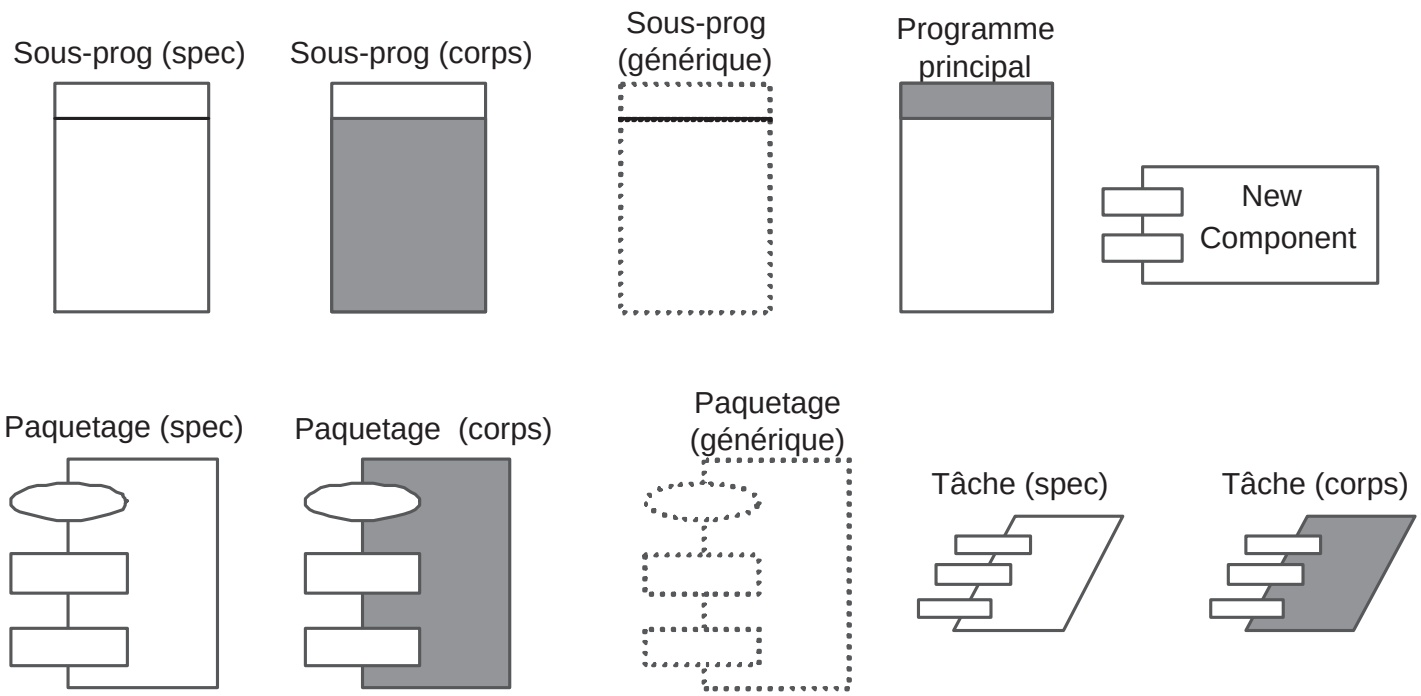


Figure 15 : *Diagramme de composants notation étendue*

Modèles de structure : composants 3/3

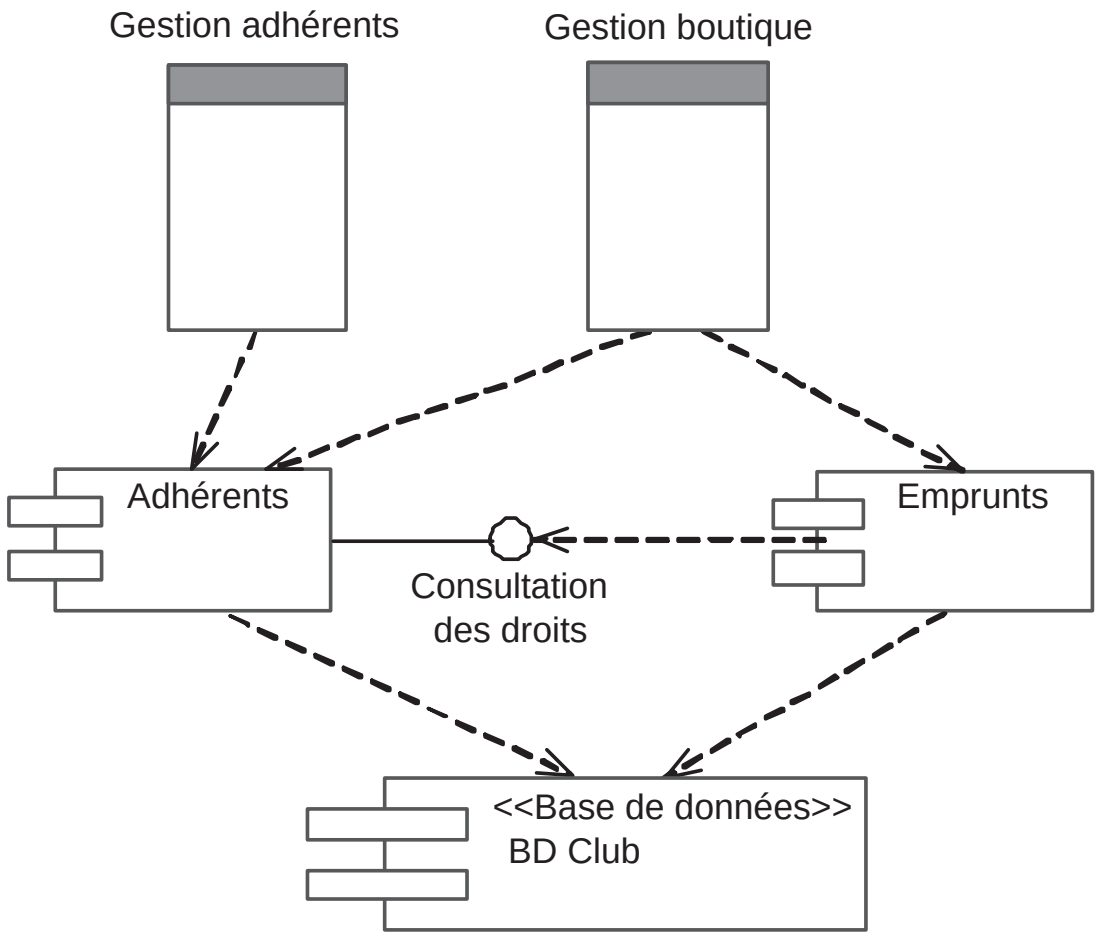


Figure 16 : *Diagramme de composants partiel - Club vidéo*

Modèles de structure

1. collaborations
2. classes
3. composants
4. déploiement

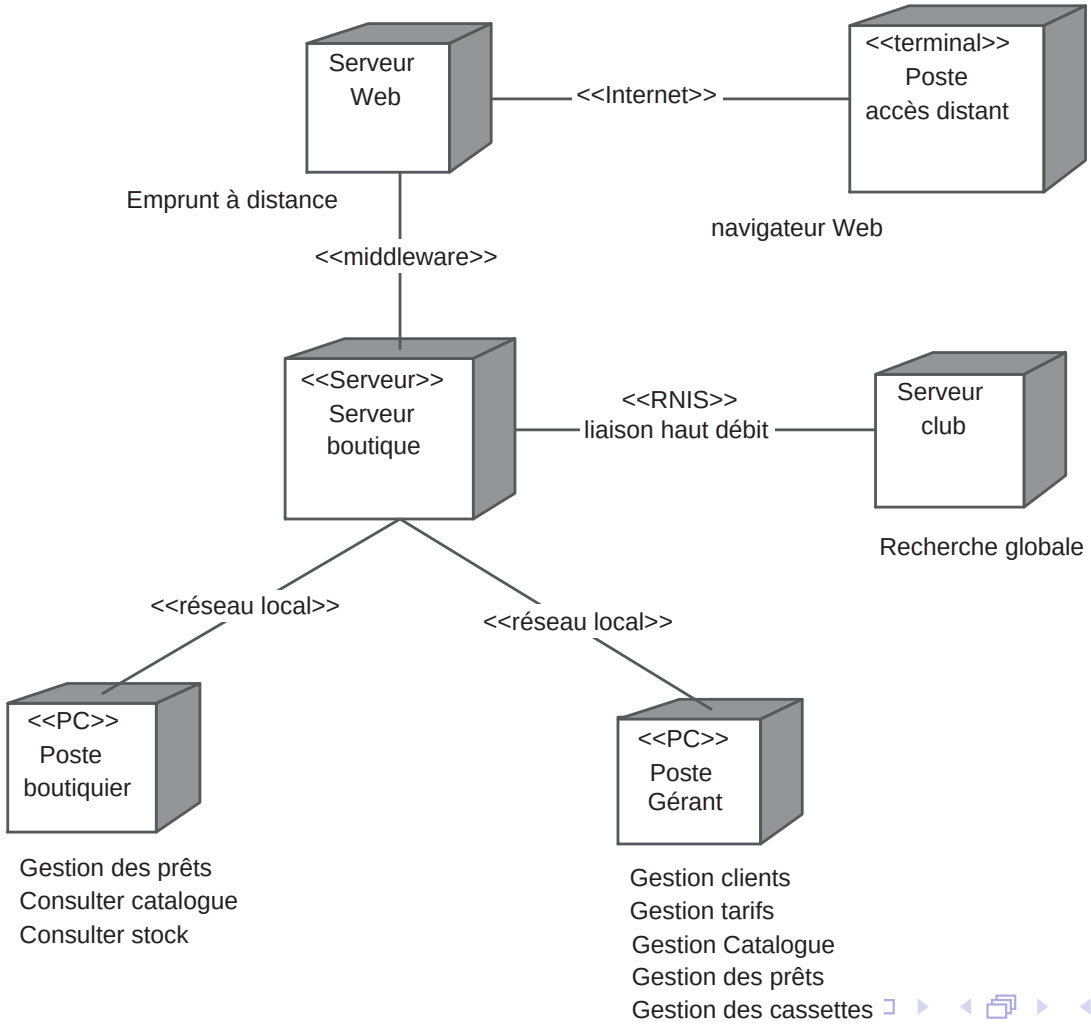
Modèles de structure : déploiement 1/3

diagramme de déploiement =

- architecture physique
- répartition des composants sur les nœuds physiques (processeurs)
- UML 1.x
  - répartition des composants sur les nœuds physiques
  - liaisons $\Longrightarrow$ réseaux

UML 2.0 $\Longrightarrow$ artefacts et non plus composants sur les nœuds

Modèles de structure : déploiement 2/2



Aperçu de la notation UML

1. Modèles d'approche
2. Modèles de structure
3. Modèles de la dynamique
4. Modèles des traitements fonctionnels

Modèles de la dynamique

1. séquences
2. états-transitions
3. activités

Modèles de la dynamique : séquences 1/5

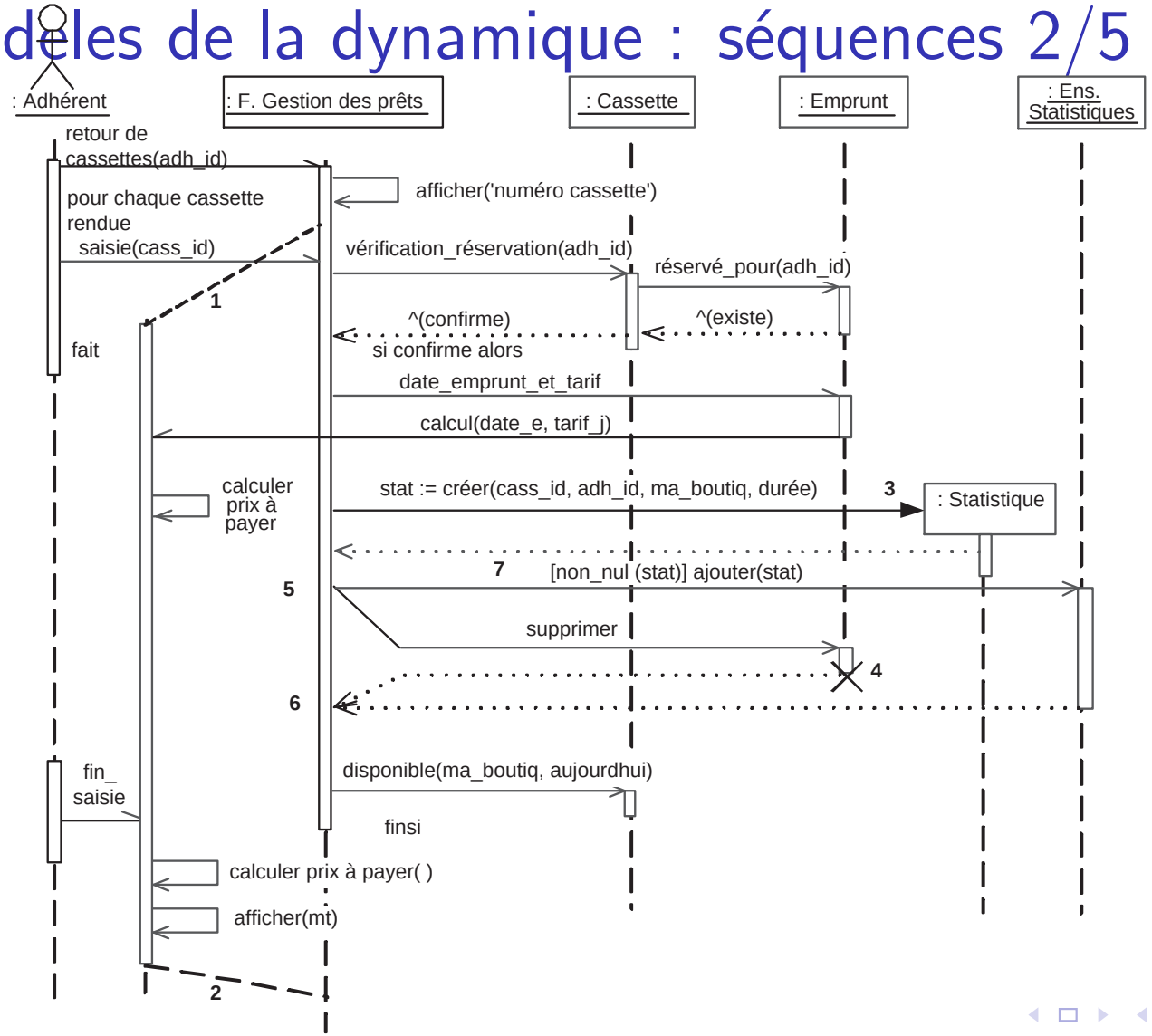
diagramme de séquences = évolution temporelle des échanges dans une collaboration (MSC)

- vue temporelle des diagrammes de collaboration
- objets, envoi de message (numérotation implicite, paramètres...), signaux...
- retour, envoi asynchrone, création/suppression d'objets
- structures de contrôle, flots parallèles, synchronisation, contraintes temporelles...

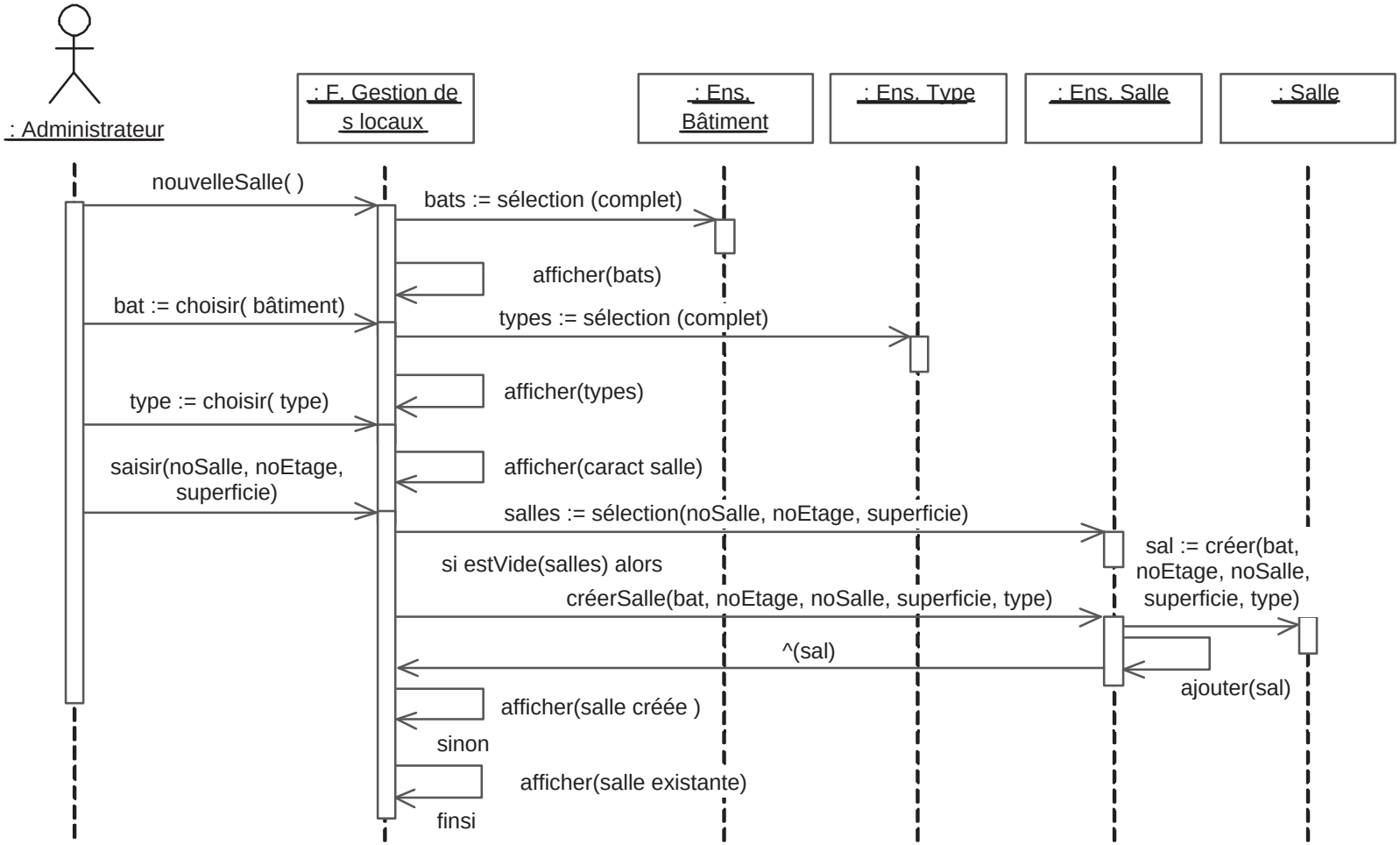
UML 2.0 $\Longrightarrow$ ajout des "*frames*" (structures de contrôle et références), sources indéterminées, *timing*

UML 2.0 $\Longrightarrow$ pas de gardes et d'itération, sync/async

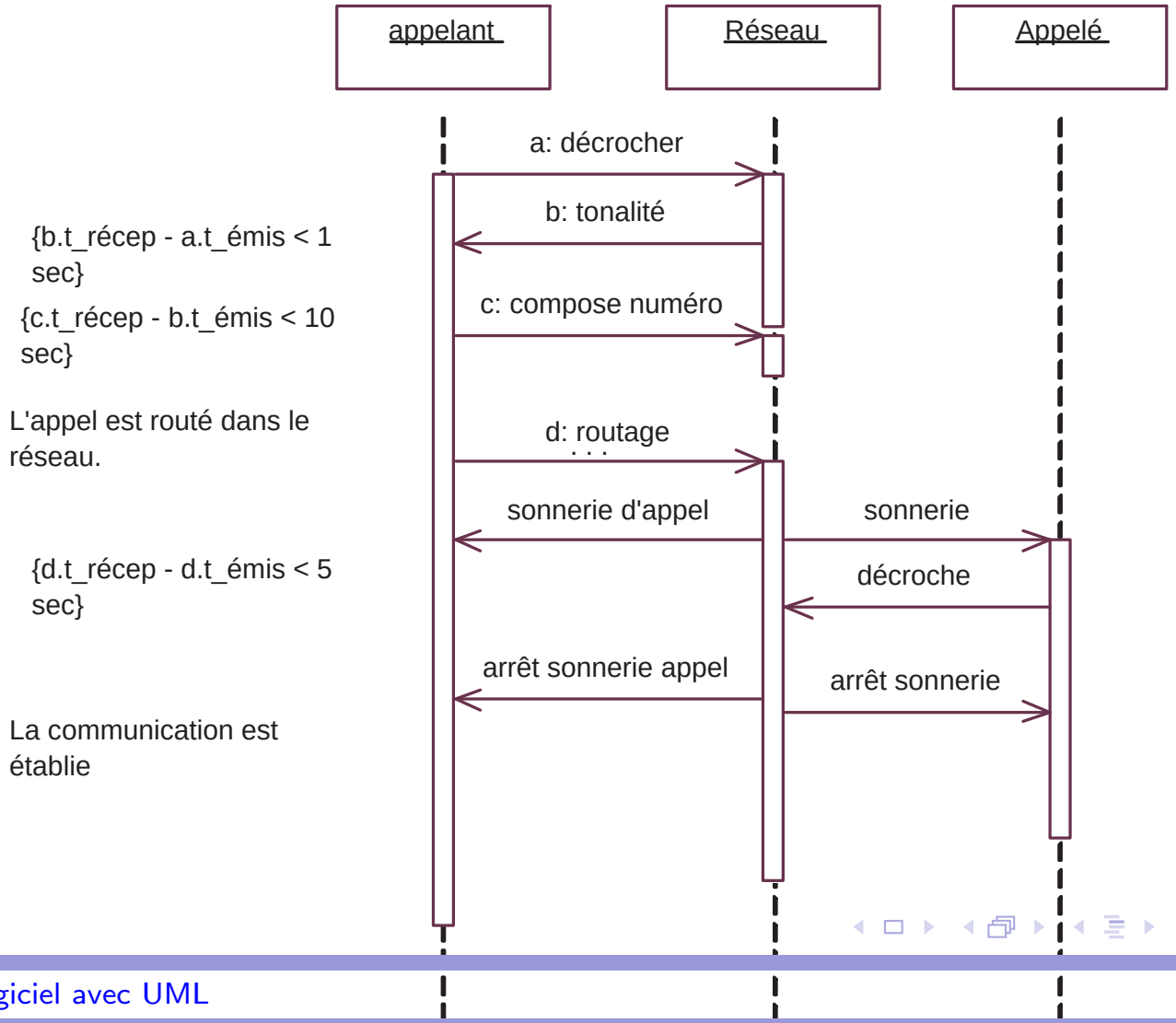
Modèles de la dynamique : séquences 2/5



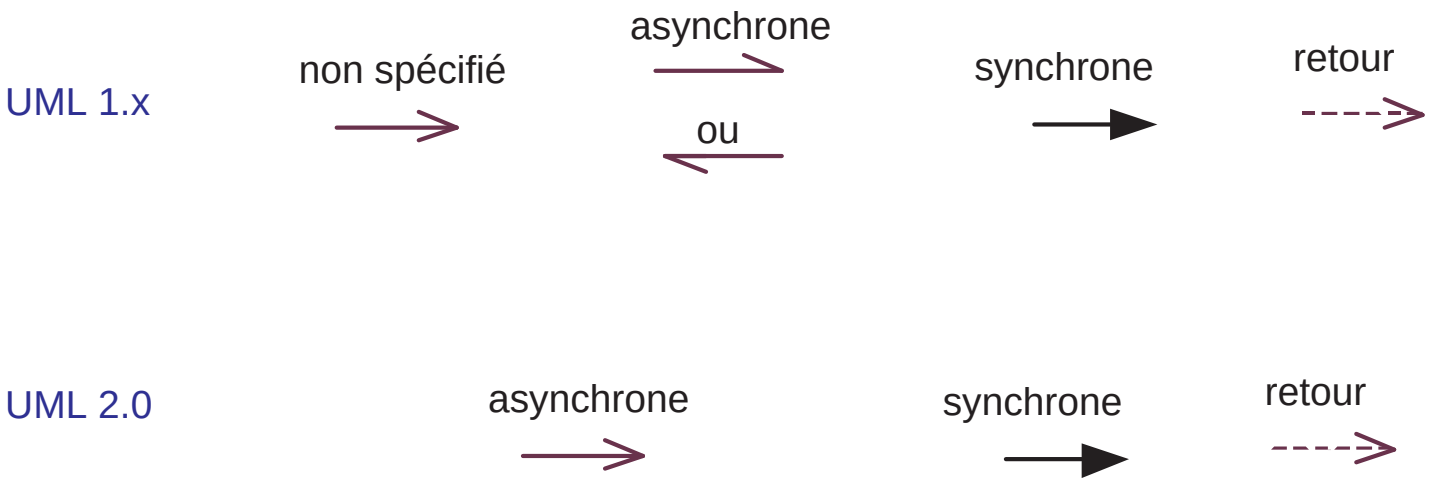
Modèles de la dynamique : séquences 3/5



Modèles de la dynamique : séquences 4/5



Modèles de la dynamique : séquences 5/5



Modèles de la dynamique

1. séquences
2. états-transitions
3. activités

Modèles de la dynamique : E-T 1/5

diagramme états-transitions = comportement dynamique des objets (*statecharts*)

- automates hiérarchiques
- états, transitions, événements, gardes
- actions, activités, opérations
- hiérarchisation (superstate, composite)
- pseudo-états (H*, synchro, jonctions, souches...)

UML 2.0 $\Longrightarrow$ machines à états *behavior statemachine*, *protocol statemachine*

UML 2.0 $\Longrightarrow$ héritage, points d'E/S et abandon, activités et activité-do

Modèles de la dynamique : E-T 2/5

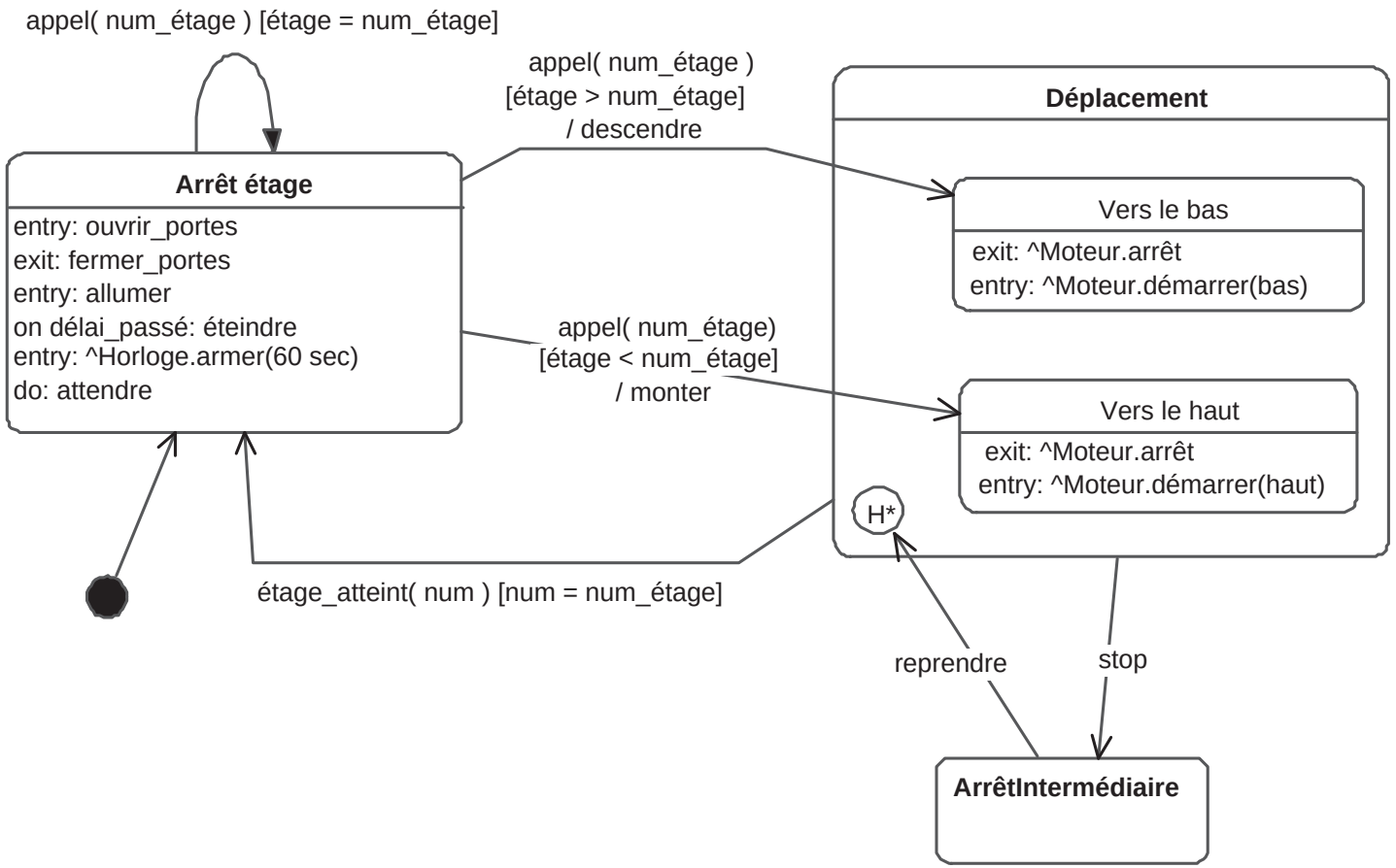


Figure 17 : *Diagramme états-transitions, cabine - Ascenseur*

Modèles de la dynamique : E-T 3/5

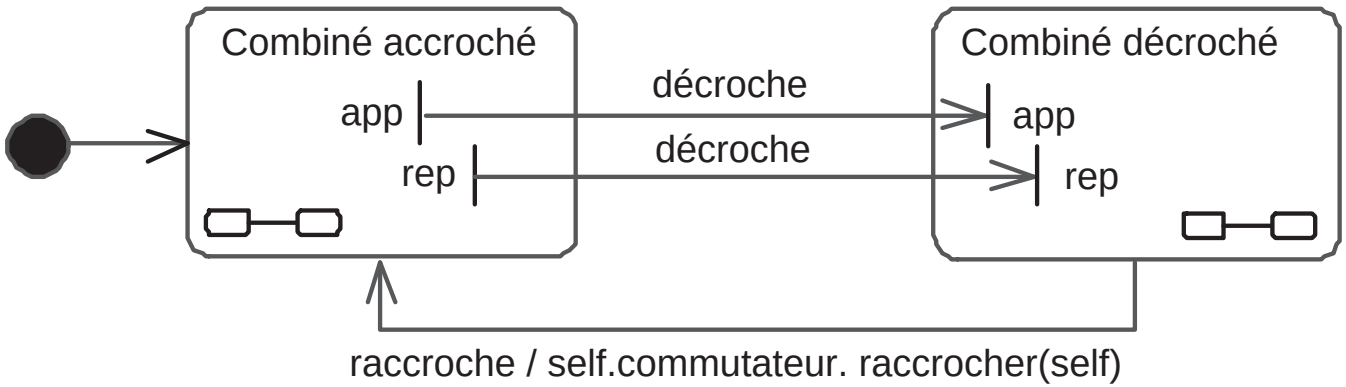
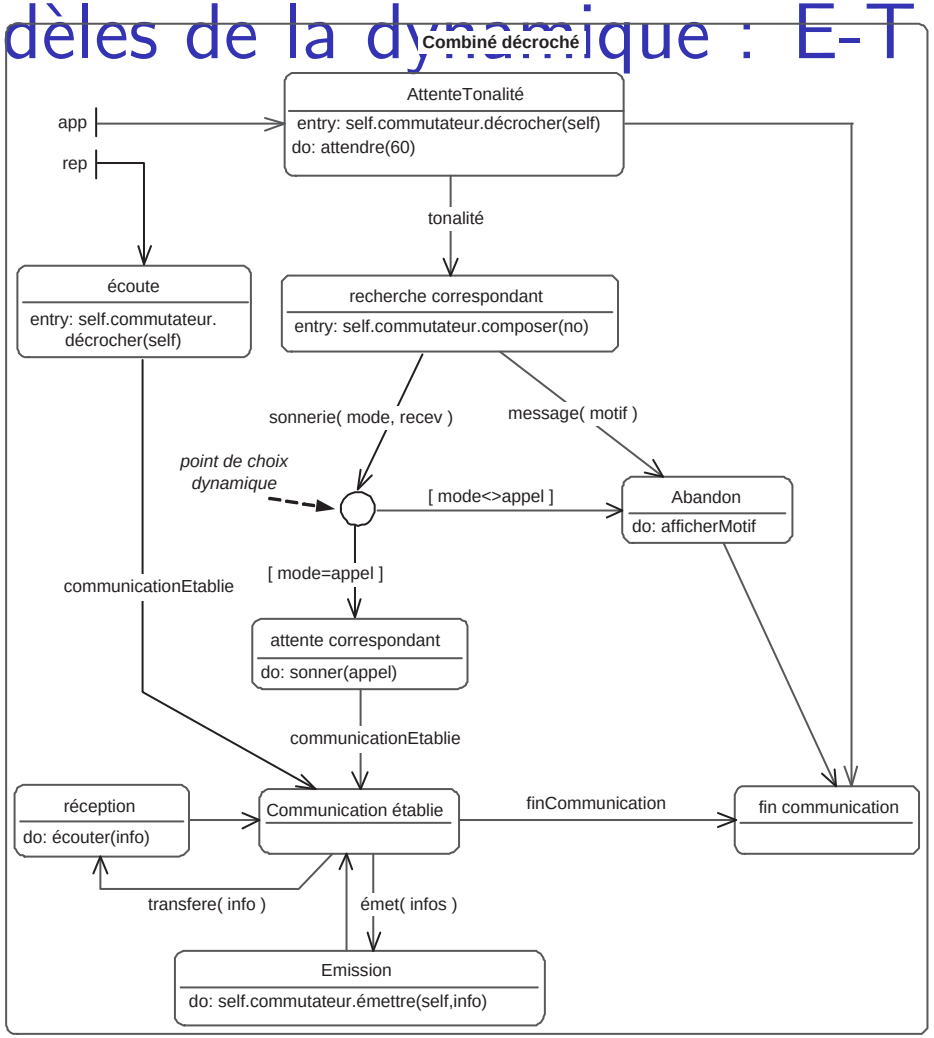
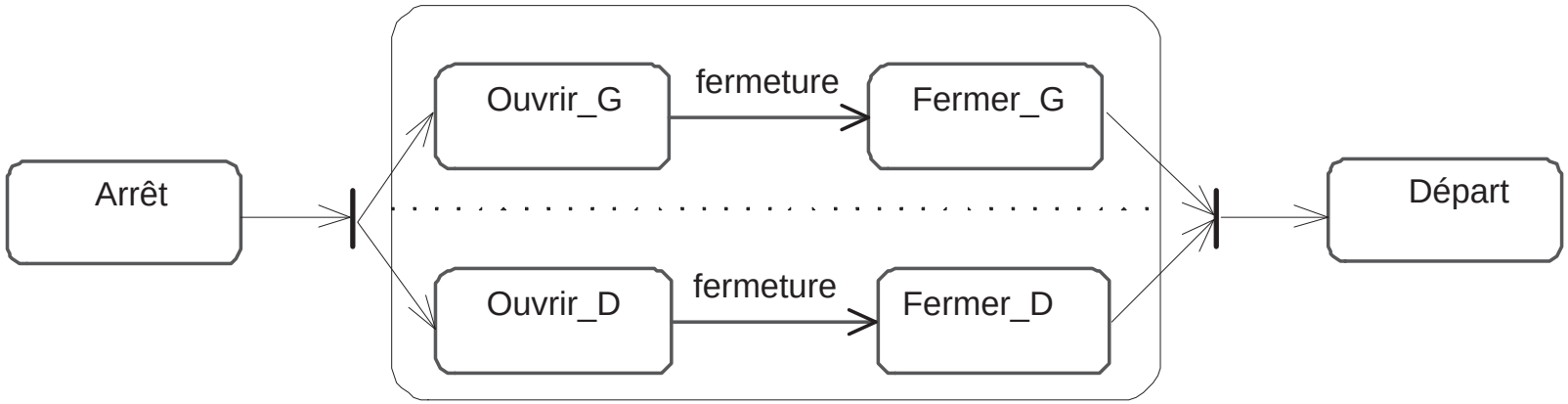


Figure 18 : *Diagramme états-transitions d'un objet* `PosteAbonné`

Modèles de la dynamique : E-T 4/5



Modèles de la dynamique : E-T 5/5



Modèles de la dynamique

1. séquences
2. états-transitions
3. activités

Modèles de la dynamique : activités 1/4

diagramme d'activité

- activités, actions $\rightsquigarrow$ états-transitions
- structures de contrôle
- envoi/réception de message ou signaux (paramètres...)
- flots et synchronisations
- extension : couloirs (noms, objets, rôles...)

++ voir sémantique des actions *Action Semantics* dans UML 1.5

Modèles de la dynamique : activités 2/4

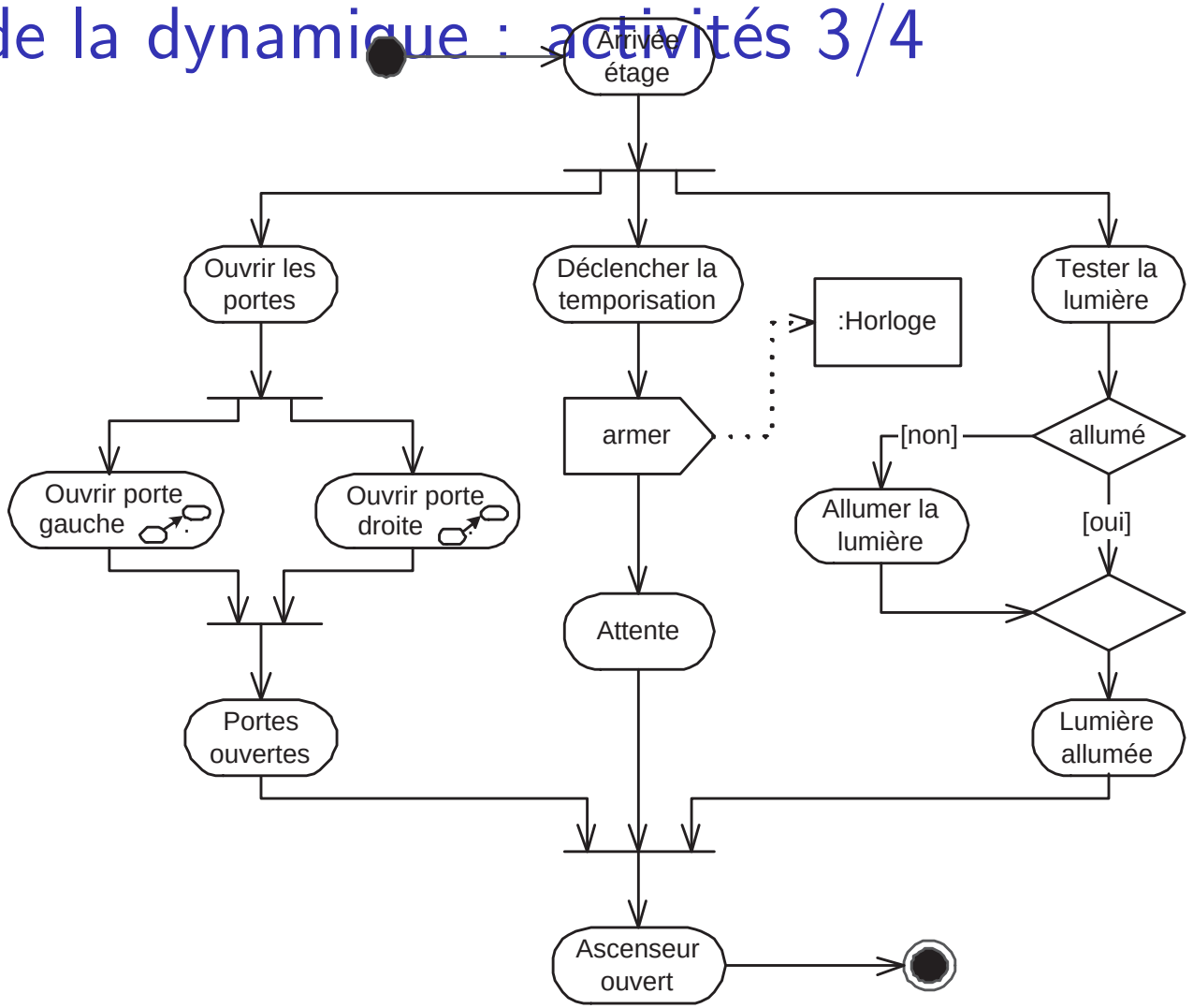
diagramme d'activité (suite)

UML 2.0 $\Longrightarrow$ enrichi des DFD $\rightsquigarrow$ *business process*

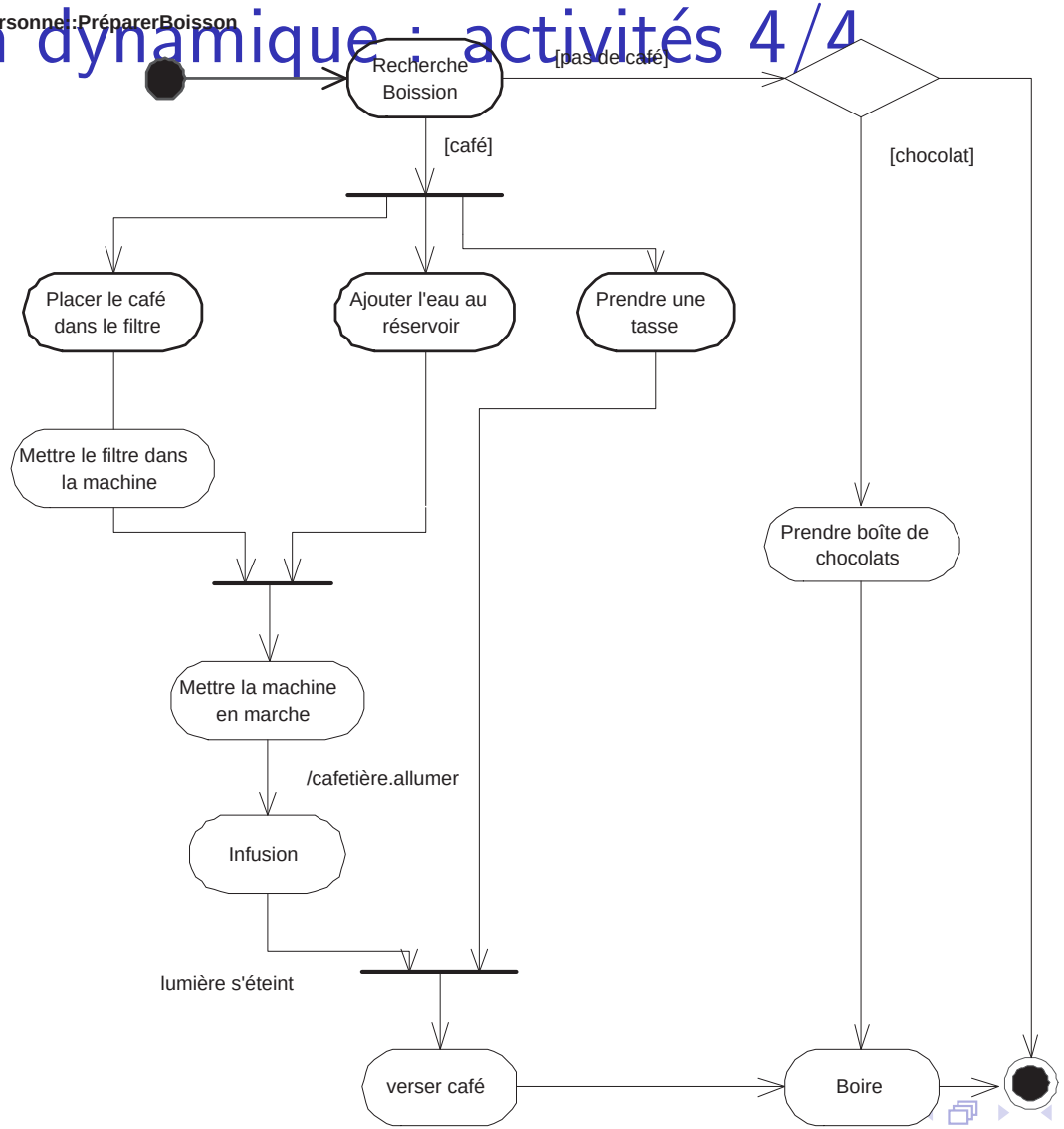
- couloirs bidimensionnels (groupage de postes)
- conditions et jetons (MCT, Petri, Grafcet)
- signaux temporels
- débranchements, jonctions, décisions, fusions, terminaisons de flots
- arcs et flots
- connecteurs, régions d'expansion

UML 1.x	UML 2.0
action	activité
activité	activité-do
branchement	décision

Modèles de la dynamique : activités 3/4



Modèles de la dynamique : activités 4/4



Modèles de la dynamique UML 2.0

1. séquences
2. états-transitions
3. activités
4. interactions (séquences + activités) ??
5. *timing*

Aperçu de la notation UML

1. Modèles d'approche
2. Modèles de structure
3. Modèles de la dynamique
4. Modèles des traitements fonctionnels

Modèles des traitements fonctionnels

1. déclaratif : OCL
2. opérationnel : activités + actions
   - diagrammes d'activités (et états-transitions)
   - opérations
   - diagrammes d'activités étendus
3. Action Semantics $\Longrightarrow$ *développer...*
4. Activités UML2 : DFD + DA
   - conditions et jetons (MCT, Petri, Grafcet)
   - signaux temporels
   - débranchements, jonctions, décisions, fusions, terminaisons de flots
   - arcs et flots
   - connecteurs, régions d'expansion

La notation UML : modèles et vues

Les diagrammes décrivent des aspects complémentaires mais non disjoints du système.
Un modèle est un ensemble de diagrammes et de documents.

- Modèle des besoins, d'analyse, de conception, de réalisation, d'implantation, de déploiement, de test...
- Modèle statique, modèle dynamique, modèle fonctionnel
- Modèle de type, modèle d'instances...
- Méta-modèle
- etc.

Les vues sont des aspects des modèles. Les modèles ne dépendent pas de la notation mais de la méthode de développement.

La notation UML : multi-formalisme

Les relations entre diagrammes sont de plusieurs types :

- multi-aspect : classes, états-transitions, activités
- type/instance : classes et objets, UC et scénarios
- abstraction : classes et composant, classes d'analyse et classe d'implantation
- complémentaires : classes et UC
- chevauchement : séquences et collaborations, activités et états-transitions

Les concepts et notations sont variés : l'objet assure la cohésion sémantique.

La notation UML : classification

Les concepts (et diagrammes) peuvent être classés selon plusieurs axes :

- activité de développement : besoins, analyse, conception, réalisation
- abstraction : méta-type/type/instance : définitions, exemplaires
- aspect du système : statique, dynamique, fonctionnel
- degré de formalisme : cas d'utilisation, classe, états-transition...

Chaque axe reprend une préoccupation des acteurs du développement (méthodologiste, analyste, utilisateur, développeur...).

La notation UML : en résumé

UML est un langage complet mais complexe.

Contrairement à UML 1.x, dans UML 2.0 la sémantique est plus modulaire mais le recouvrement des diagrammes interdit toute sémantique commune.

UML : Esperanto ou Babel ?

Avec un langage

- complexe
- à géométrie variable (sémantique)
- éléments combinables à souhait

UML : Esperanto ou Babel ?

Avec un langage

- complexe
- à géométrie variable (sémantique)
- éléments combinables à souhait

Peut-on écrire des spécifications de qualité ?

- cohérentes
- complètes
- lisibles et exploitables
- etc.

UML : complexité d'usage

- L'interprétation varie avec le contexte :
  exemple des diagrammes d'activités
- Le contenu varie avec le niveau d'abstraction :
  exemple des classes en analyse et en implantation,
- Dans un modèle, on trouve des préoccupations différentes :
  utilisation, classes, objets, composants
- L'usage varie avec la méthode de développement :
  méthode itérative $\Longrightarrow$ degré de précision.

On fait abstraction des besoins non fonctionnels.

UML : Esperanto ou Babel ?

Chacun a sa sémantique d'UML en fonction de son expérience, des langages et environnements de développement utilisés, des applications développées, des besoins requis.

UML : Esperanto ou Babel ?

Chacun a sa sémantique d'UML en fonction de son expérience, des langages et environnements de développement utilisés, des applications développées, des besoins requis.

UML : Esperanto ou Babel ?

Chacun a sa sémantique d'UML en fonction de son expérience, des langages et environnements de développement utilisés, des applications développées, des besoins requis.
C'est encore plus vrai avec UML2

Problème :
de nombreuses modélisations erronées, incohérentes, incomplètes...

UML : Esperanto ou Babel ?

Chacun a sa sémantique d'UML en fonction de son expérience, des langages et environnements de développement utilisés, des applications développées, des besoins requis.
C'est encore plus vrai avec UML2

Problème :
de nombreuses modélisations erronées, incohérentes, incomplètes...

Solutions :

- Gérer la complexité
- Proposer un compilateur (ou un interpréteur)
- Proposer un correcteur
- Autres solutions...

UML : gérer la complexité

1. Grouper les diagrammes par activité :
   - modèles d'approche (UC, scénarios, activités)
   - modèles logiques (classes, E-T, activités, séquences, collaborations)
   - modèles d'implantation (composants, déploiement, classes, collaborations)

UML : gérer la complexité

1. Grouper les diagrammes par activité :
   - modèles d'approche (UC, scénarios, ~~activités~~)
   - modèles logiques (classes, E-T, activités, séquences, collaborations)
   - modèles d'implantation (composants, déploiement, ~~classes, collaborations~~)
2. Limiter l'usage des diagrammes par activité.

UML : gérer la complexité

1. Grouper les diagrammes par activité :
   - modèles d'approche (UC, scénarios, ~~activités~~)
   - modèles logiques (classes, E-T, activités, séquences, collaborations)
   - modèles d'implantation (composants, déploiement, ~~classes, collaborations~~)
2. Limiter l'usage des diagrammes par activité.
3. Grouper les diagrammes par type Instanciation :
   scénario $\longrightarrow$ UC / séquence, collab. $\longrightarrow$ Classe, E-T, activités

UML : gérer la complexité

1. Grouper les diagrammes par activité :
   - modèles d'approche (UC, scénarios, ~~activités~~)
   - modèles logiques (classes, E-T, activités, séquences, collaborations)
   - modèles d'implantation (composants, déploiement, ~~classes, collaborations~~)
2. Limiter l'usage des diagrammes par activité.
3. Grouper les diagrammes par type Instanciation :
   scénario $\longrightarrow$ UC / séquence, collab. $\longrightarrow$ Classe, E-T, activités
4. Associer les diagrammes complémentaires :
   E-T $\leftrightarrow$ activité, E-T $\leftrightarrow$ classe, opération $\leftrightarrow$ activité, composant $\leftrightarrow$ déploiement

UML : proposer un compilateur

- Aspects syntaxiques : le méta-modèle
  - problème de classification des concepts
  - pas de grammaire complète
  - mais un jeu de règle de vérifications (complet ? cohérent ?)

UML : proposer un compilateur

- Aspects syntaxiques : le méta-modèle
  - problème de classification des concepts
  - pas de grammaire complète
  - mais un jeu de règle de vérifications (complet ? cohérent ?)
- Aspects sémantiques : langage naturel

$\Longrightarrow$ pas satisfaisant

  - des travaux en cours
  - multi-formalisme

UML : proposer un compilateur

- Aspects syntaxiques : le méta-modèle
  - problème de classification des concepts
  - pas de grammaire complète
  - mais un jeu de règle de vérifications (complet ? cohérent ?)
- Aspects sémantiques : langage naturel

$\Longrightarrow$ pas satisfaisant

  - des travaux en cours
  - multi-formalisme
- Executable UML
  - traduction complète (sémantique opérationnelle)
  - génération de code par le compilateur
  - extraction pour les spécifications formelles

UML : proposer un correcteur

- Objectifs limités : vérifier des propriétés
  - de systèmes : redondances, non-blocage...
  - de modèles : cohérence, complétude...
  - de processus : équivalences, traçabilité...

UML : proposer un correcteur

- Objectifs limités : vérifier des propriétés
  - de systèmes : redondances, non-blocage...
  - de modèles : cohérence, complétude...
  - de processus : équivalences, traçabilité...
- Evolutif
  - le correcteur s'adapte au compilateur
  - le correcteur s'intègre dans différents outils
  - la base de règles est incrémentale, paramétrable

UML : proposer un correcteur

- Objectifs limités : vérifier des propriétés
  - de systèmes : redondances, non-blocage...
  - de modèles : cohérence, complétude...
  - de processus : équivalences, traçabilité...
- Evolutif
  - le correcteur s'adapte au compilateur
  - le correcteur s'intègre dans différents outils
  - la base de règles est incrémentale, paramétrable
- Rigoureux
  - formaliser les règles (OCL, spec. formelles)
  - vérifier la base de règles (cohérence...)

UML : proposer un correcteur

- Objectifs limités : vérifier des propriétés
  - de systèmes : redondances, non-blocage...
  - de modèles : cohérence, complétude...
  - de processus : équivalences, traçabilité...
- Evolutif
  - le correcteur s'adapte au compilateur
  - le correcteur s'intègre dans différents outils
  - la base de règles est incrémentale, paramétrable
- Rigoureux
  - formaliser les règles (OCL, spec. formelles)
  - vérifier la base de règles (cohérence...)
- Automatisable, génération de test

Plan

Introduction

UML : un langage de spécification multi-formalisme

UML : précision avec OCL

UML : une méthode de développement

UML : un processus unifié

UML : outils et vérification

Perspectives

Object Constraint Language

Eléments clés

- Langage à objets déclaratifs (relativement) formel
- Inspiré de Syntropy (et donc de Z)
- Typage
- Navigation
- Assertions et contraintes
- Méta Object Protocol

Object Constraint Language

Détails dans le chapitre 9 [AV01]

- Types de base et MOP $\Longrightarrow$ section 2

Object Constraint Language

Détails dans le chapitre 9 [AV01]

- Types de base et MOP $\Longrightarrow$ section 2
- Navigation $\Longrightarrow$ section 3.2

Object Constraint Language

Détails dans le chapitre 9 [AV01]

- Types de base et MOP $\Longrightarrow$ section 2
- Navigation $\Longrightarrow$ section 3.2
- Assertions $\Longrightarrow$ section 3.1, 3.3

OCL : Types 1/7

- `OclAny`
- Types de base

Type	Valeurs	Opérations
`Boolean`	`true`, `false`	`and`, `or`, `xor`, `not`, `implies`, `if-then-else`
`Integer`	3, -15	`*`, `+`, `-`, `/`, `abs`, `div`, `mod`, `max`, `min`, `<`, `>`, `<=`, `>=`
`Real`	2.212, -1.777	`*`, `+`, `-`, `/`, `abs`, `floor`, `round`, `max`, `min`, `<`, `>`, `<=`, `>=`
`String`	`'abc'`	`size`, `concat`, `toUpper`, `toLower`, `substring`

- Types énumérés `enum {v1, v2, v3, v4} #v2`.
- Collections
- Types UML (classes, associations, état...)
- Types OCL-MOP (réflexion, typage)

OCL : Types 2/7 : Collection

Opération	Commentaire
`size`	nombre d'éléments de la collection
`isEmpty / notEmpty`	collection vide / non vide
`includes(OclAny)`	test d'appartenance d'un objet
`excludes(OclAny)`	test de non appartenance d'un objet
`count(OclAny)`	nombre d'occurrence de l'objet
`includesAll(Coll(T))`	inclusion de la collection paramètre
`excludesAll(Coll(T))`	intersection vide avec la collection paramètre
`sum`	addition des éléments de la collection
`exists/forall(OclExpr)`	au moins un / tout élément vérifie l'expression
`isUnique(OclExpr)`	l'expression est évaluée différemment pour chaque elt
`sortedBy(OclExpr)`	rend la séquence des éléments triée par l'expression
`iterate(OclExpr)`	évalue une expression accumulée sur chaque élément
`select/reject(OclExpr)`	rend les éléments vérifiant (ou pas) l'expression
`collect(OclExpr)`	rend la collection des évaluations

OCL : Types 3/7 : Set

Opération	Commentaire
`union(Set(T))`	union de deux ensembles
`union(Bag(T))`	union avec un multi-ensemble (rend un objet Bag)
`intersection(Set(T))`	intersection de deux ensembles
`intersection(Bag(T))`	intersection avec un multi-ensemble (rend un objet Bag)
`-(Set(T))`	différence de deux ensembles
`including(T)`	ajout de l'élément à l'ensemble
`excluding(T)`	retrait de l'élément à l'ensemble
`symmetricDifference (Set(T))`	différence entre l'union et l'intersection de deux ensembles
`asSequence`	conversion d'ensemble en séquence d'ordre quelconque
`asBag`	conversion de l'ensemble en multi-ensemble

OCL : Types 4/7 : Bag

Opération	Commentaire
`union(Bag(T))`	union de deux multi-ensembles
`union(Set(T))`	union avec un ensemble (rend un objet Bag)
`intersection(Bag(T))`	intersection de deux multi-ensembles
`intersection(Set(T))`	intersection avec un ensemble (rend un objet Bag)
`including(T)`	ajout de l'élément au multi-ensemble
`excluding(T)`	retrait de l'élément du multi-ensemble
`asSet`	conversion en ensemble
`asSequence`	conversion en séquence d'ordre quelconque

OCL : Types 5/7 : Sequence

Opération	Commentaire
`union(Sequence(T))`	concaténation de deux séquences
`append(T)`	ajout de l'élément à la fin de la séquence
`prepend(T)`	ajout de l'élément en tête de la séquence
`subSequence(low,up)`	séquence d'indices entre low et up
`at(i)`	séquence à l'indice i (`1<= i <=size`)
`first`	premier élément de la séquence
`last`	dernier élément de la séquence
`including(T)`	ajout de l'élément en fin de séquence
`excluding(T)`	retrait de l'élément de la séquence
`asSet`	conversion de la séquence en ensemble
`asBag`	conversion de la séquence en multi-ensemble

OCL : Système de Types 6/7

- Types UML
  - Classes, Etat, ...
  - Propriétés (attribut, opération, rôles)
  - Associations (qualification, classes, collections...)
- Type `OclAny`
  - =, <>
  - `OclAsType` : transtypage (accès propriété)
  - `OclIsTypeOf` : test de supertype direct
  - `OclIsKindOf` : test de supertype
  - `OclIsNew` : objet créé (dans postcondition)
  - `OclIsInState` : test d'état `OclState`
- `OclType` : tout type OCL
  - `name`, `attributes`, `operations`, `associationEnds`
  - `supertypes`, `allSupertypes`
  - `allInstances`
- `OclExpression` : expressions d'OCL

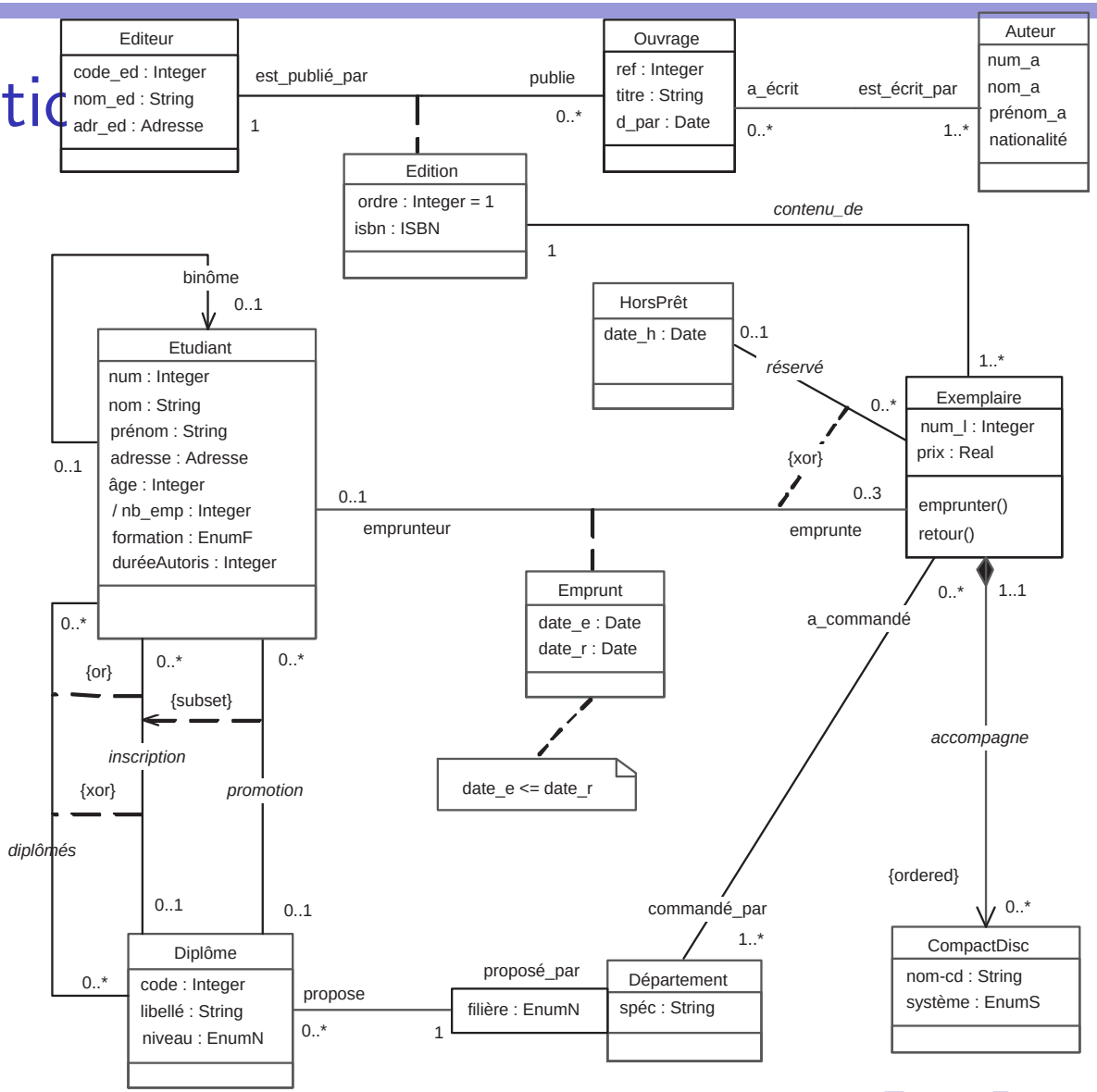
OCL : Types 7/7 : Conformité

Type	Conforme à
tous	OclAny
Set(T)	Collection(T)
Bag(T)	Collection(T)
Sequence(T)	Collection(T)
Integer	Real

OCL : Navigation

- Propriétés (attribut, opération, rôles *associationEnd*)
- Rôles par défaut
- Notation pointée
- Raccourcis et transitivité
- Cardinalités $\Longrightarrow$ collections
- Exemples

OCL : Navigatio



OCL : Pratique 1/5

- Expression OCL
  - rattachée à un élément de modélisation quelconque
  - gardes
  - contraintes
  - propriété dérivée...
- Assertion
  - Invariant de classe
  - Pré-post condition
  - Invariant de système
- Déclaration locale (`let ... in`)

Contexte - la variable `self`

OCL : Pratique 2/5 : Invariant de classe

```
context Etudiant
  inv binôme:
      self.binôme <> self      -- pas de monôme, implicite par agrégation
  inv âge:
      self.âge ≥ 14            -- les étudiants ont au moins 14 ans
      formation = #continue implies âge ≥ 25
          -- les étudiants de formation continue ont plus de 25 ans
  inv durées:
          -- l'attribut duréeAutoris donne le nombre maximum jour
          -- de prêts pour cet étudiant (crédit maximum)
      if  self.assiste_à→isEmpty then
          duréeAutoris = 0
      else  if  self.assiste_à.niveau = #DESS then
                duréeAutoris = 30
            else
                duréeAutoris = 20
            endif
      endif
```

OCL : Pratique 3/5 : Opération

```
context Exemplaire::emprunter(date_e : Date ; date_r : Date ;
                              étud : Etudiant) : Boolean
  pre: self.emprunt→isEmpty and   -- disponible
       date_e ≤ date_r and        -- cohérence des dates
       étud.emprunt→size < 3 and  -- n'a pas 3 emprunts en cours
       date_r ≤ (date_e + étud.duréeAutoris)
                  -- l'étudiant est autorisé pour cette durée
  post:  -- let introduit une variable quantifiée existentiellement
       let emp : Emprunt in
          emp.date_e = date_e and
          emp.date_r = date_r and
          self.emprunt = emp and
          étud.emprunt = étud.emprunt@pre→including(emp) and
          emp.étudiant = étud and
          emp.exemplaire = self and
          result = true
```

{query} / {abstract} / {concurrency = sequential
guarded, concurrent}

OCL : Pratique 4/5 : dérivé / Association

Propriété dérivée

context Etudiant inv nb_emprunts:
(nb_emp = self.emprunt→size) and (nb_emp ≤ 3)
-- *le nombre d'emprunts est le cardinal de l'ensemble des emprunts*

Association

context Exemplaire
(self .emprunt→isEmpty) or (self . réservé →isEmpty)
-- *exclusion du prêt d'ouvrages réservés*
context Etudiant
not (self . inscrit_en →isEmpty and self .diplômé_de→isEmpty)
-- *contrainte de totalité (a or b) ¡=¿ not(not a and not b)*

-- *contexte global*
diplômés. allInstances →excludesAll (inscription . allInstances)
-- *xor : les étudiants inscrits à un diplôme n'en sont pas diplômés*
inscription . allInstances → includesAll (promotion. allInstances)
-- *subset : les étudiants de la promotion d'un diplôme y sont inscrits*

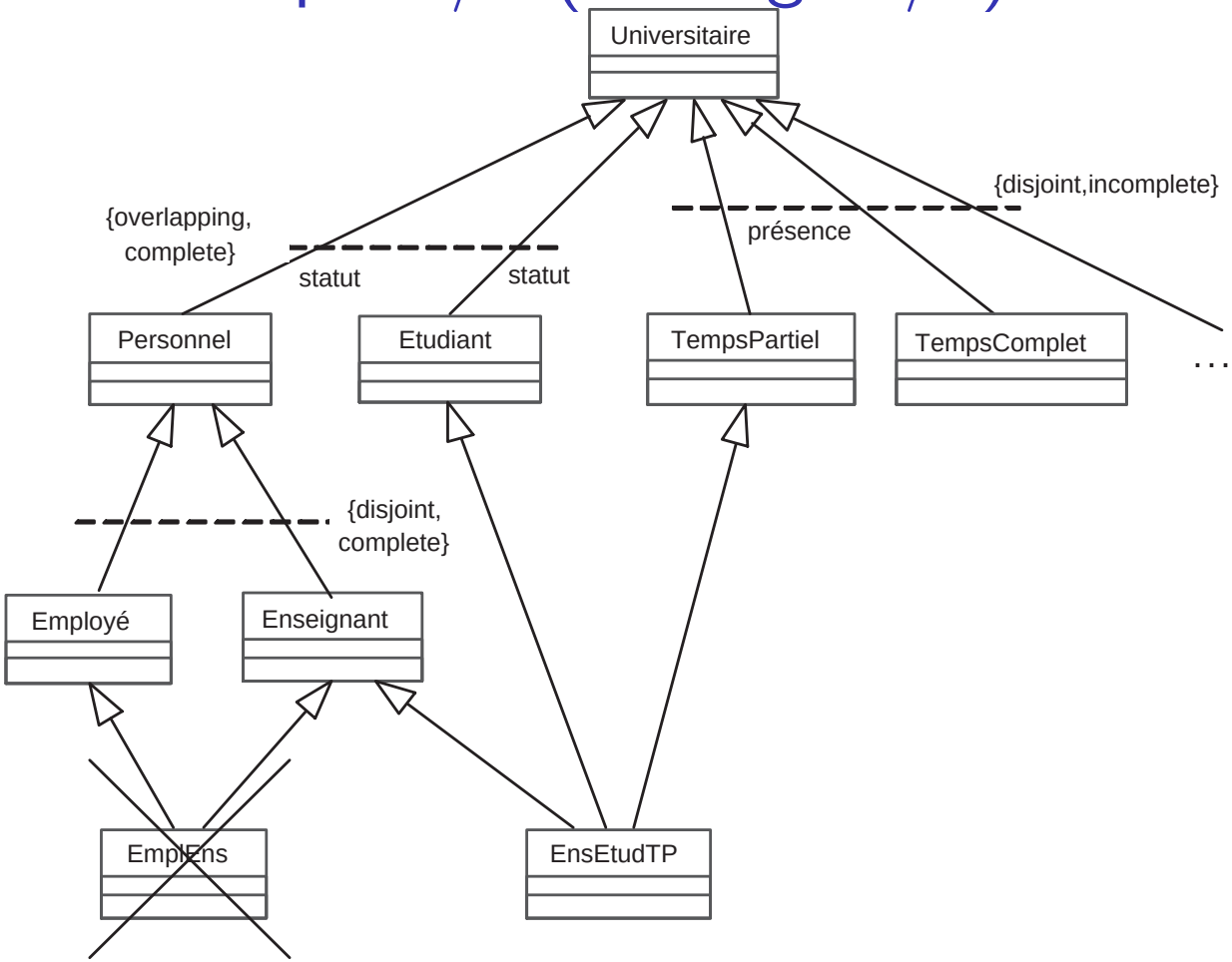
OCL : Pratique 5/5 (héritage 1/2)

Prédéfini

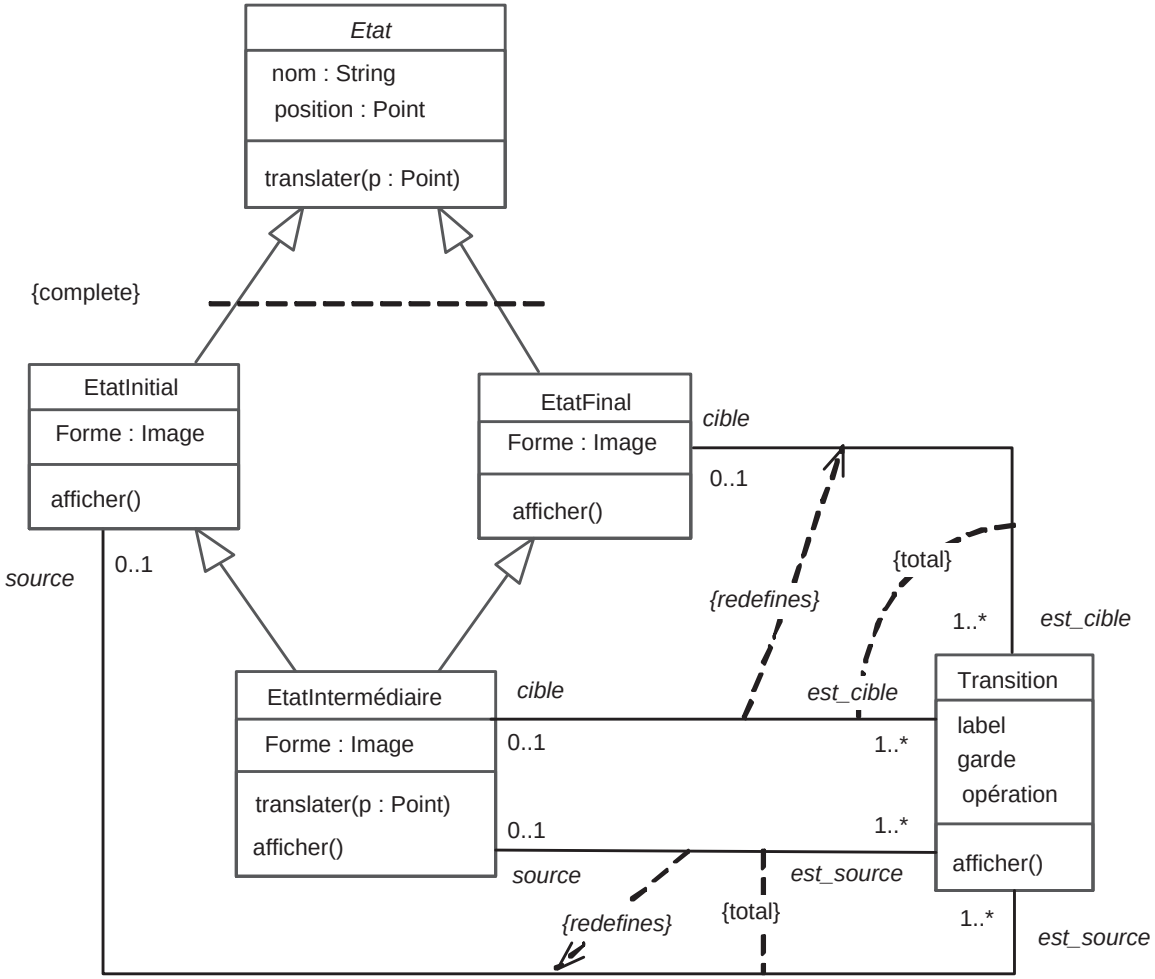
- exclusion
  - `overlapping` : autorise l'héritage multiple
  - `disjoint` : interdit l'héritage multiple
- totalité
  - `complete` : il n'y a pas d'autres sous-classes.
  - `incomplete` : d'autres sous-classes n'ont pas été définies.
- un discriminant définit une vue partielle sur l'héritage.

```
Employé.allInstances → forAll (
        i | not i.oclIsTypeOf(TempsComplet))
```

OCL : Pratique 5/5 (héritage 2/2)



OCL : Redéfinitions



Plan

Introduction

UML : un langage de spécification multi-formalisme

UML : précision avec OCL

UML : une méthode de développement

UML : un processus unifié

UML : outils et vérification

Perspectives

UML : méthode

Version simplifiée du processus : 4 activités dans le développement
Présentation de la notation utilisée dans les activités.

1. Analyse des besoins : cas d'utilisation et scénarios
2. Analyse : diagrammes d'objets et de classes, états-transitions
3. Conception : classes, composants et déploiement
4. Implantation : composants et déploiement

UML : méthode

1. Analyse des besoins : cas d'utilisation et scénarios
2. Analyse : diagrammes d'objets et de classes, états-transitions
3. Conception : classes, composants et déploiement
4. Implantation : composants et déploiement

Analyse des besoins : aperçu

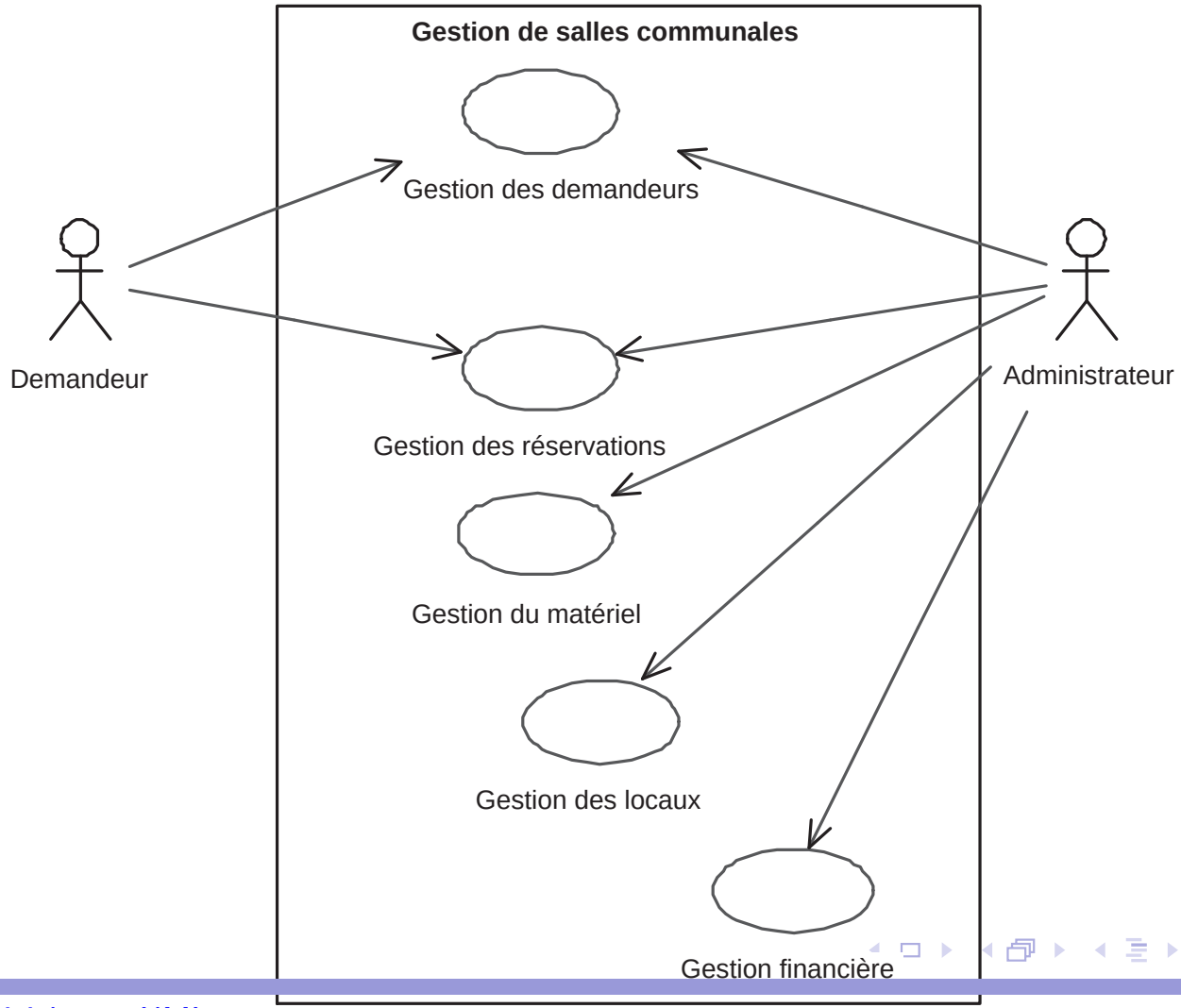
Requirements

- comprendre le contexte du système
  - modèle du domaine
  - modèle du métier
- définir les besoins
  - fonctionnels $\Longrightarrow$ Cas d'utilisation, scénarios
  - non fonctionnels
    contraintes matérielles, d'interface, de performance... sécurité, disponibilité, accessibilité, qualité...

Analyse des besoins : modèles

- diagrammes de cas d'utilisation
  - acteurs
  - cas d'utilisation
  - relations
- par cas d'utilisation
  - descriptions textuelles
  - illustration : scénarios
    - objets : acteurs, système
    - interactions : séquences

Analyse des besoins : cas d'utilisation 1/5



Analyse des besoins : cas d'utilisation 2/5

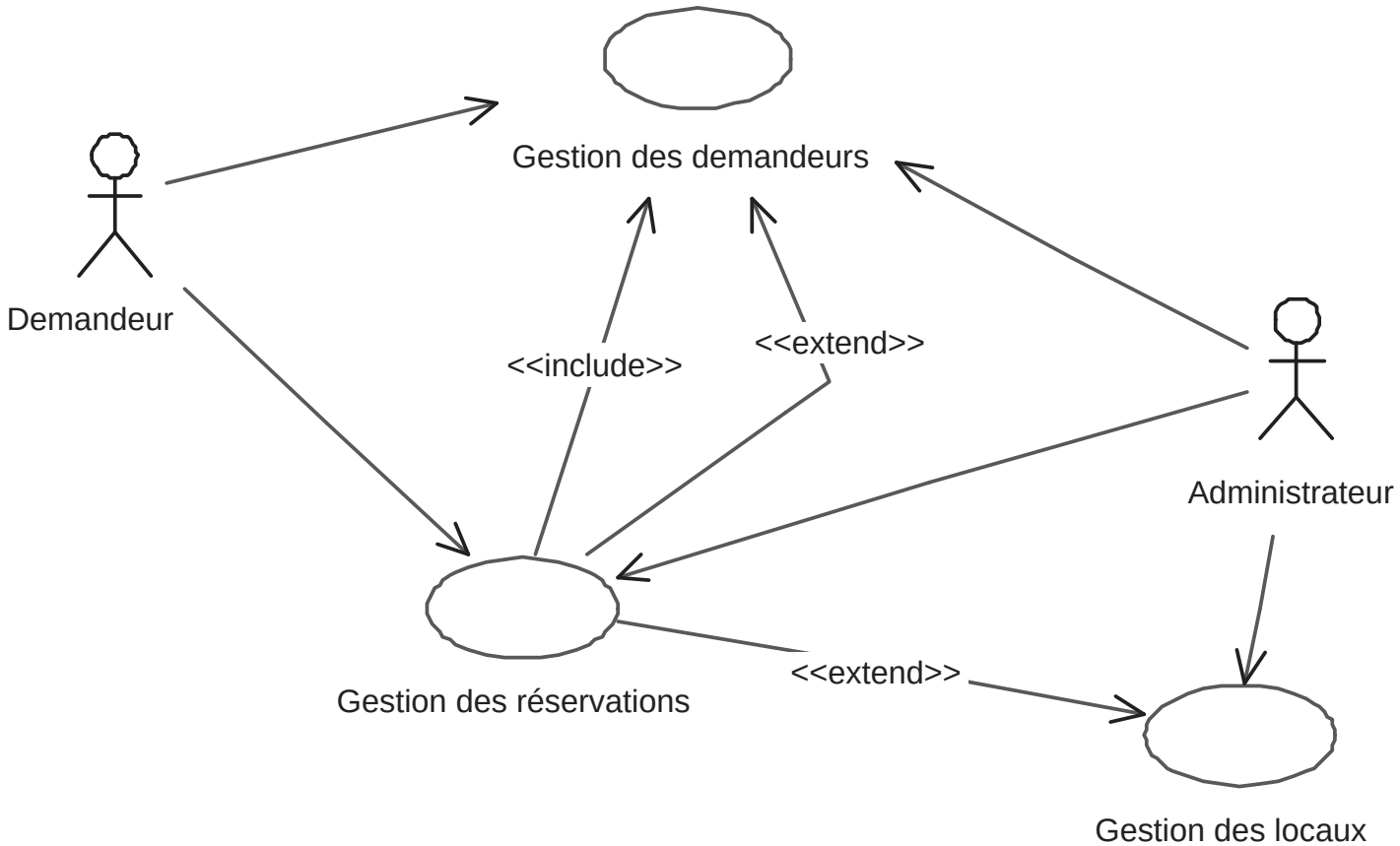


Figure 20 : *Cas d'utilisation, version préliminaire - Salles*

Analyse des besoins : cas d'utilisation 3/5

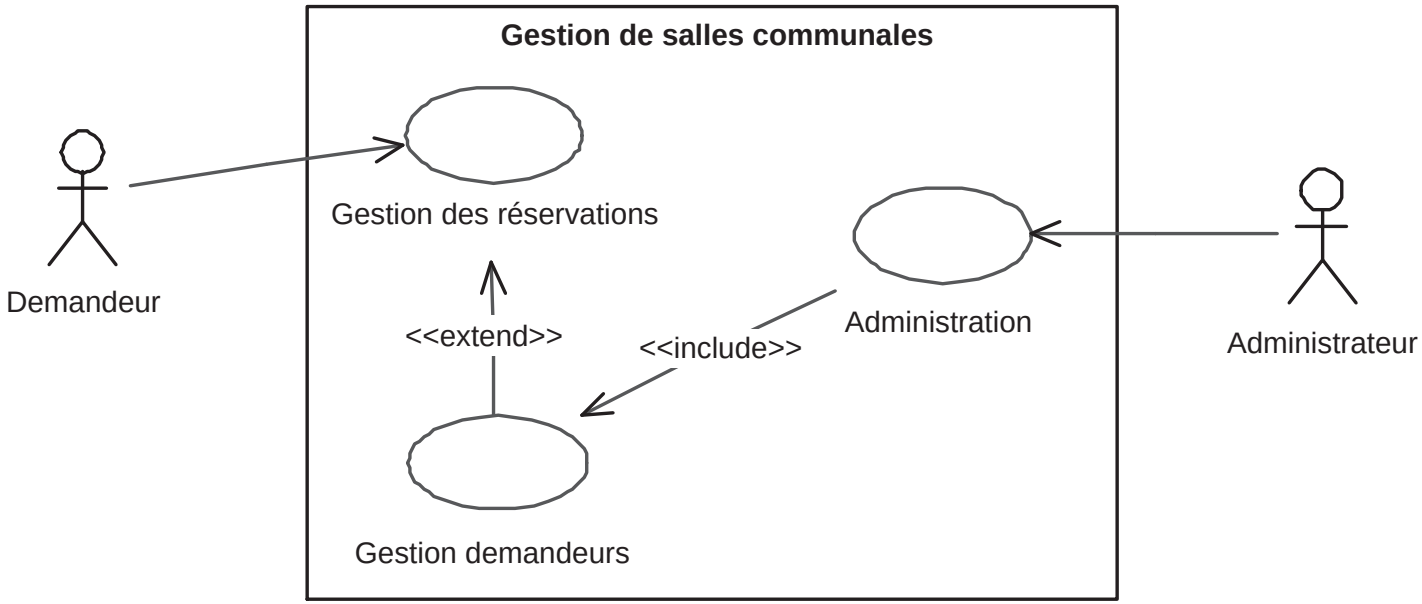


Figure 21 : *Cas d'utilisation, version préliminaire - Salles*

Analyse des besoins : cas d'utilisation 4/5

Points clés du diagramme des cas d'utilisation

- Abstrait
- Granularité : entre découpage fonctionnel et modulaire
- Lisibilité
- Description textuelle

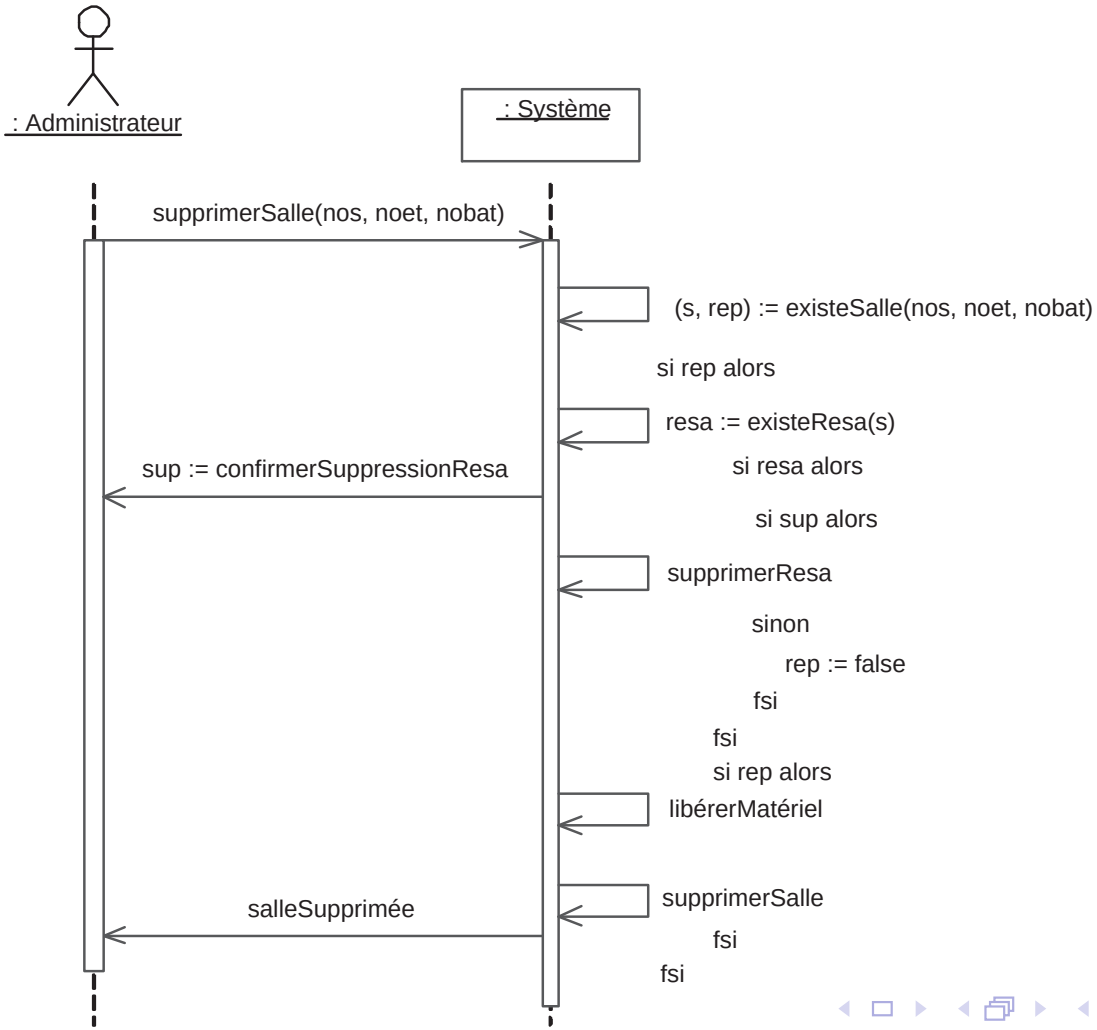
Analyse des besoins : cas d'utilisation 5/5

Cas d'utilisation : `Gestion des reservations`
acteurs primaires : `Demandeur`
invariant : `Unicite de reservation` Une salle n'est pas réservée pour deux demandeurs différents au même moment.
description La gestion des reservations comprend la réservation des salles, la consultation des réservations, l'annulation des réservations.
cas : `Réservation` Les éléments de la réservations sont saisis et recherchés dans la base en fonction de critères donnés : salle, demandeur, matériel, durée, manifestation, date. A tout moment, il est possible de consulter le planning des réservations en cours. Si tous les éléments sont corrects et qu'il n'y a pas de conflit de réservation, le montant est calculé et la réservations confirmée. Le numéro de la réservation est fourni par le système au demandeur.

Analyse des besoins : scénarios 1/2

- objectif : illustrer les cas d'utilisation (représentativité)
  - un par cas normal
  - un par exception
- notation : diagramme de séquence simplifié
  - objets (acteurs + système)
  - envoi de message (paramètres...)
- potentiellement un diagramme d'activité (*business model*)

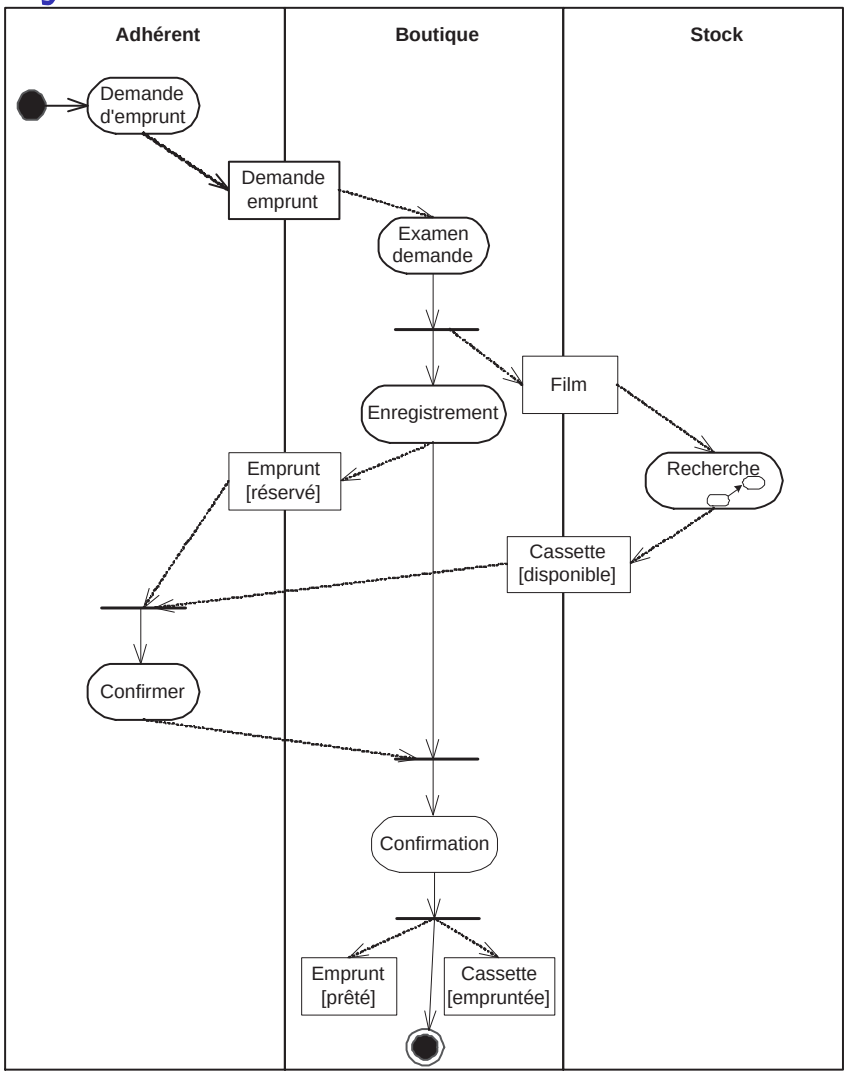
Analyse des besoins : scénarios 2/2



Analyse des besoins : activités 1/2

- objectif : décomposer des tâches complexes (*business process*)
  - en sous-tâches
  - par secteur
- notation : diagramme d'activités étendues par des couloirs

Analyse des besoins : activité 2/2



Analyse des besoins : bilan

- description des besoins
  - besoins fonctionnels
  - besoins non fonctionnels
  - en option :
    - modèle du domaine
    - modèle du métier
    - glossaires, IHM, prototype...
- description validée par l'utilisateur
- support pour les tests

= point de départ de l'analyse

UML : méthode

1. Analyse des besoins : cas d'utilisation et scénarios
2. Analyse : diagrammes d'objets et de classes, états-transitions
3. Conception : classes, composants et déploiement
4. Implantation : composants et déploiement

Analyse : aperçu

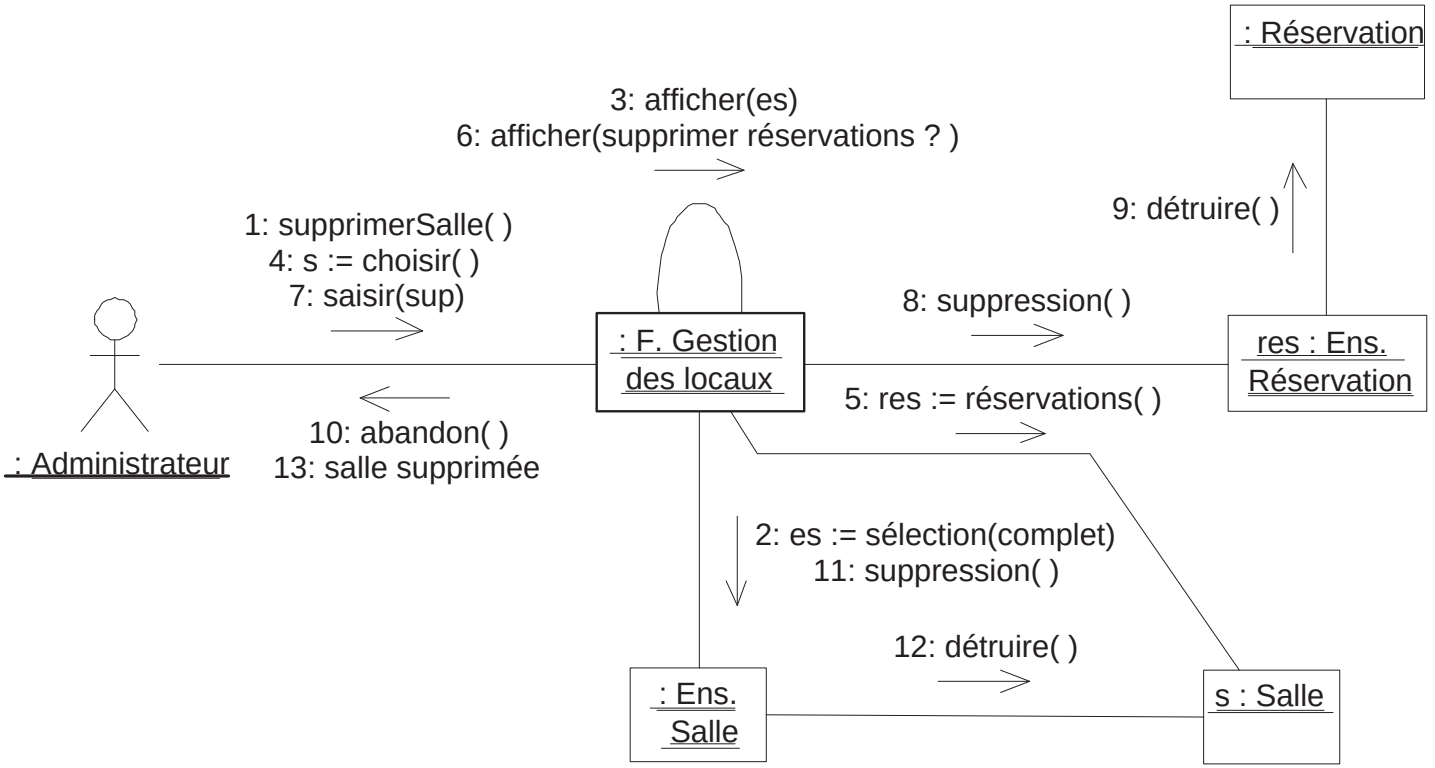
Analysis

- décrire le système indépendemment de son implantation
  - affiner l'analyse des besoins
  - décrire la prise en compte des besoins par le système
  - décrire l'architecture du système
- modélisation à objets
- structuration en sous-systèmes

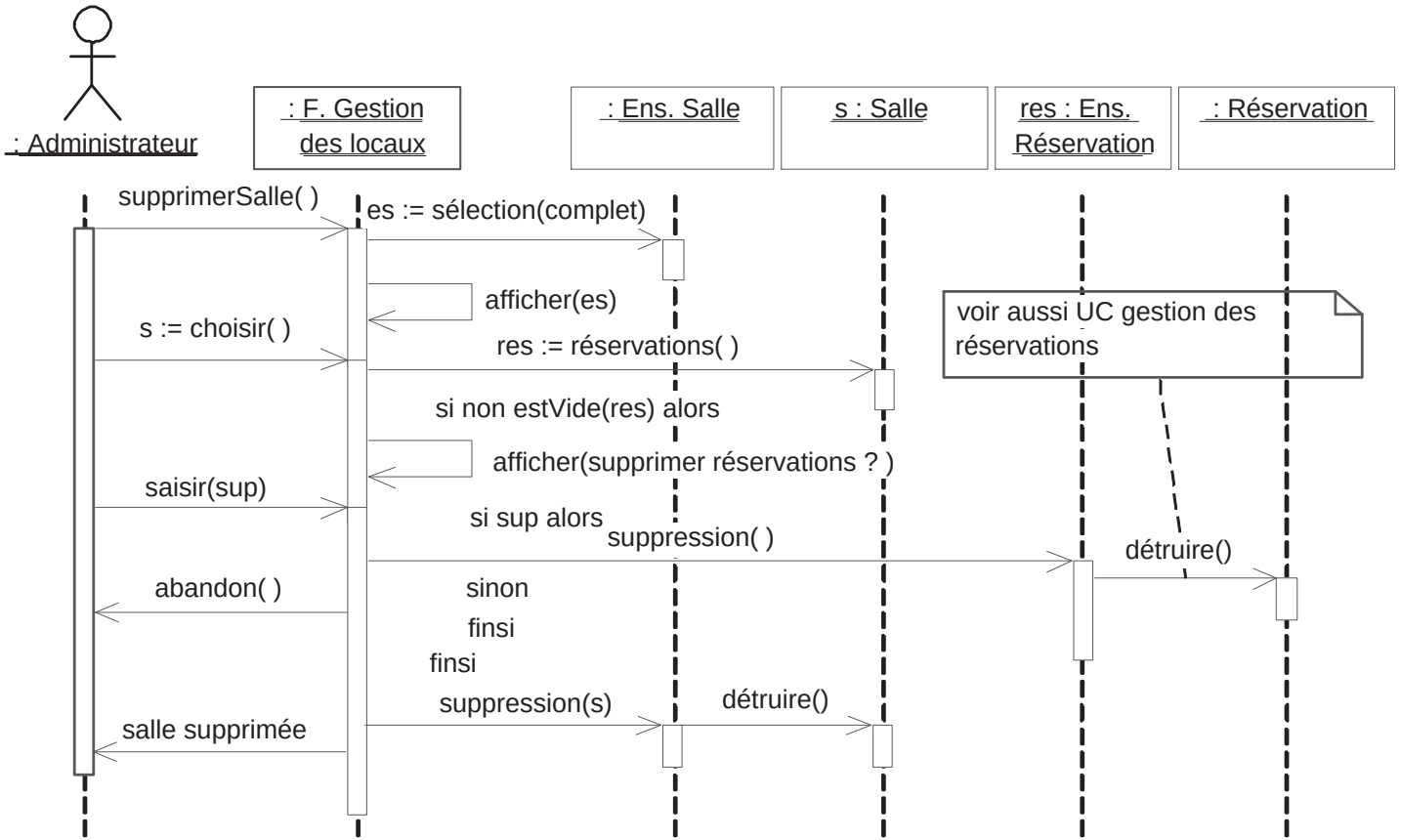
Analyse : modèles

- diagrammes d'objets
  - acteurs, objets
  - séquences
  - collaborations
- diagrammes de classes
  - classes
  - relations
  - enrichissements
- diagrammes états-transitions et diagrammes d'activités

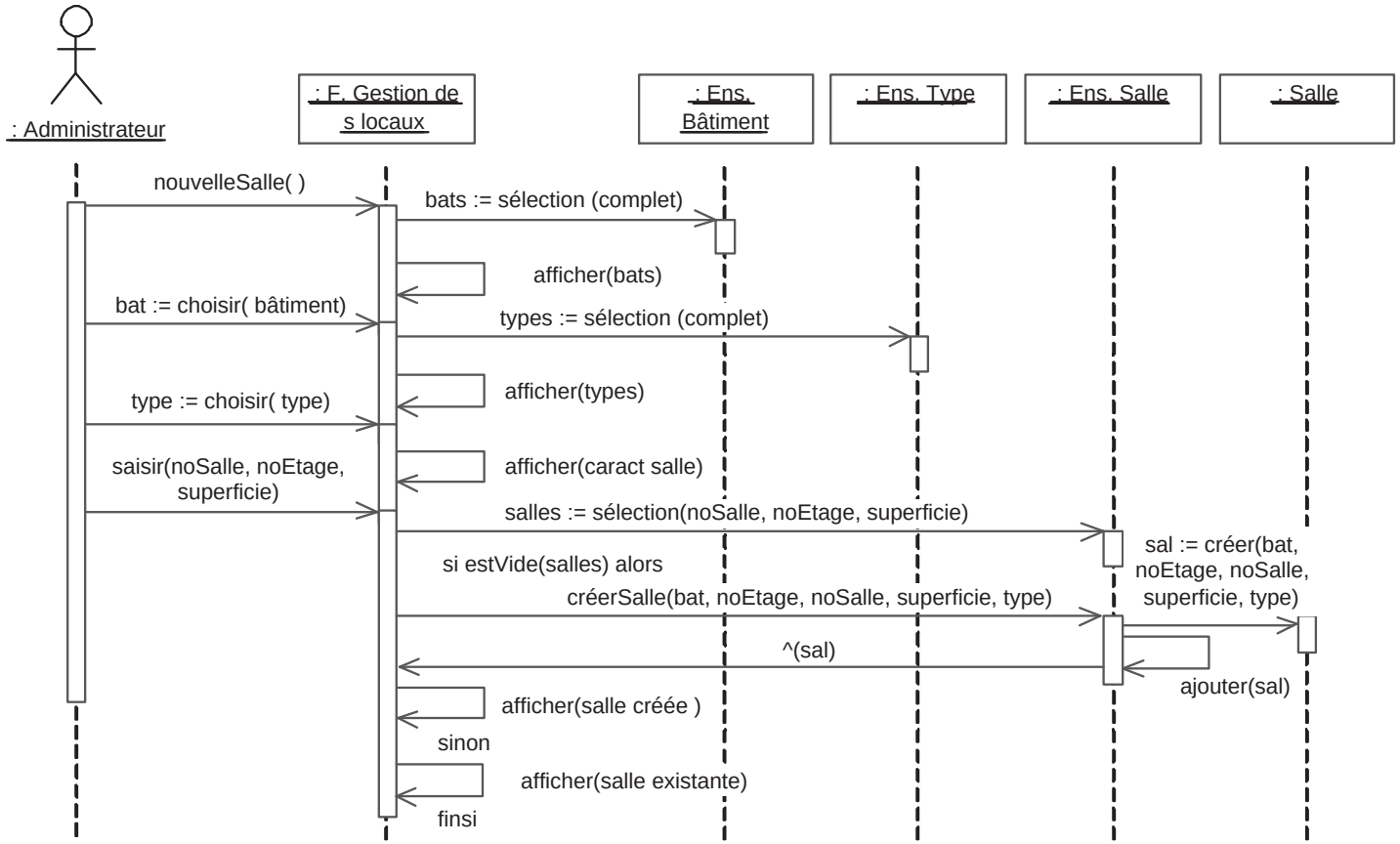
Analyse : notations (collaboration)



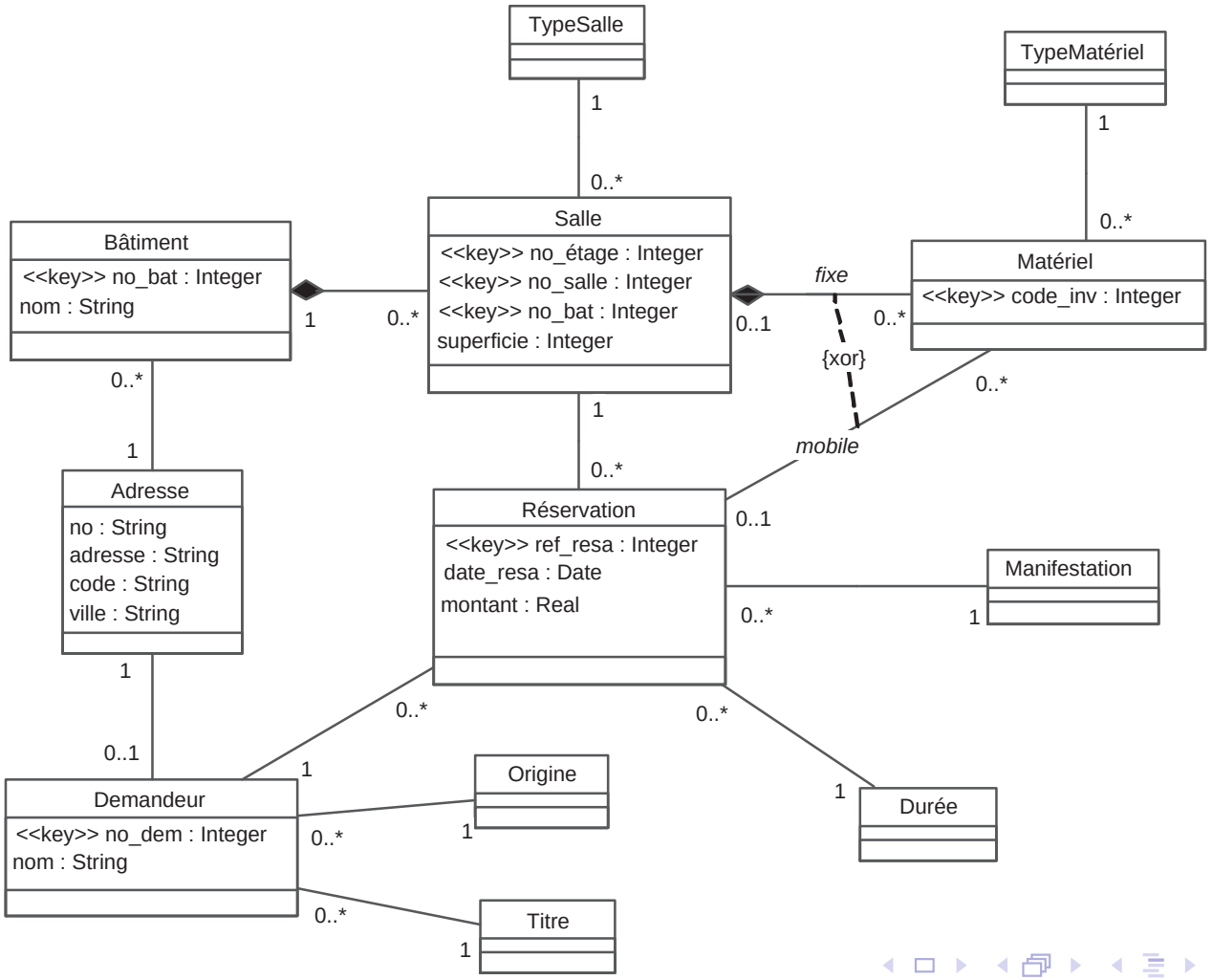
Analyse : notations (séquence)



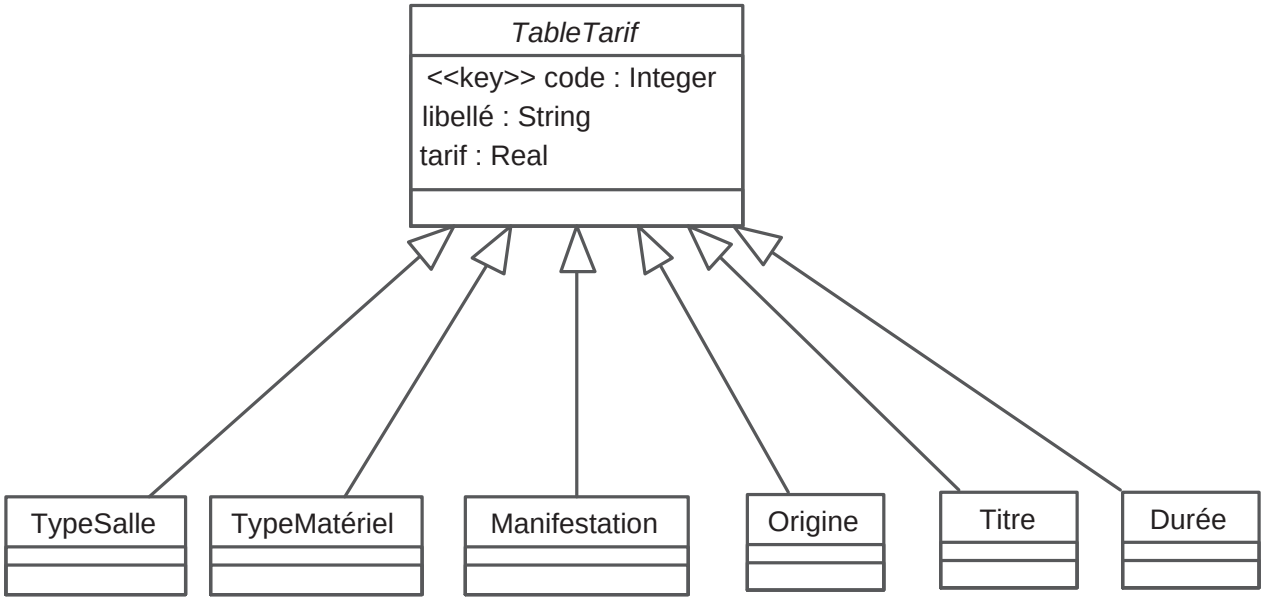
Analyse : notations (séquence)



Analyse : notations (classes)



Analyse : notations (classes)



Analyse : notations (classes)

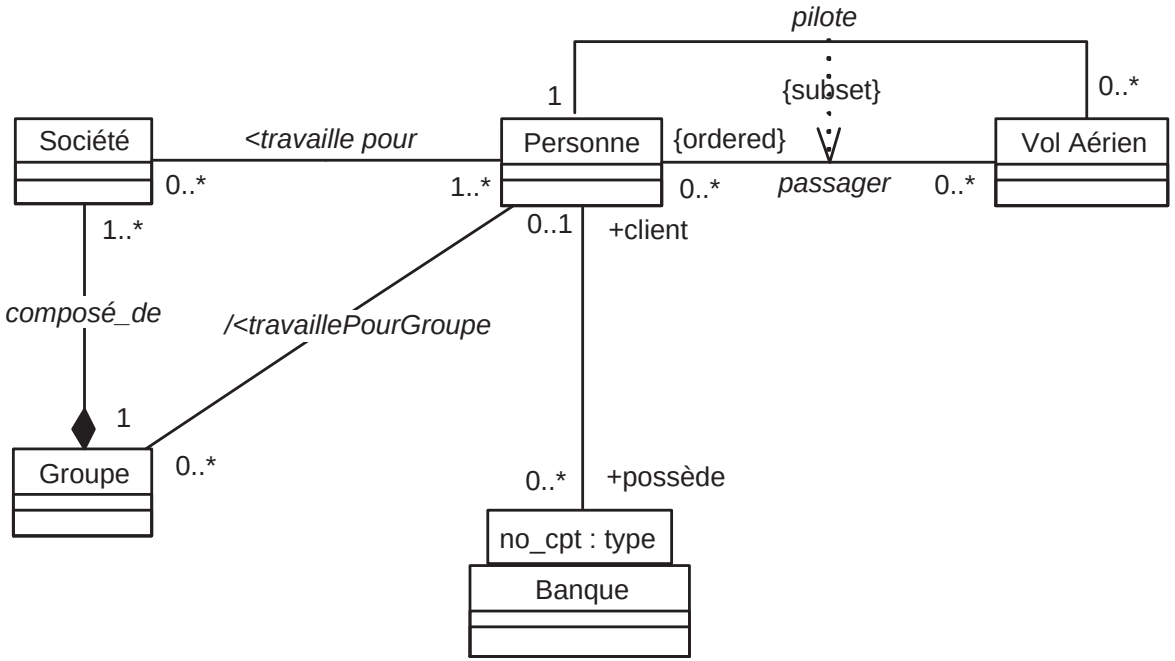
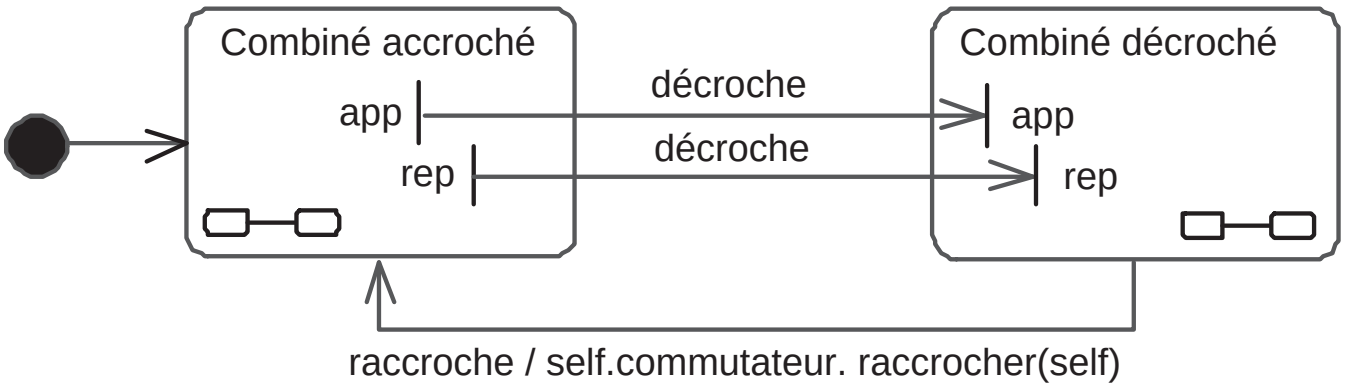


Figure 22 : *Diagramme des classes, association qualifiée*

Analyse : notations (opération OCL)

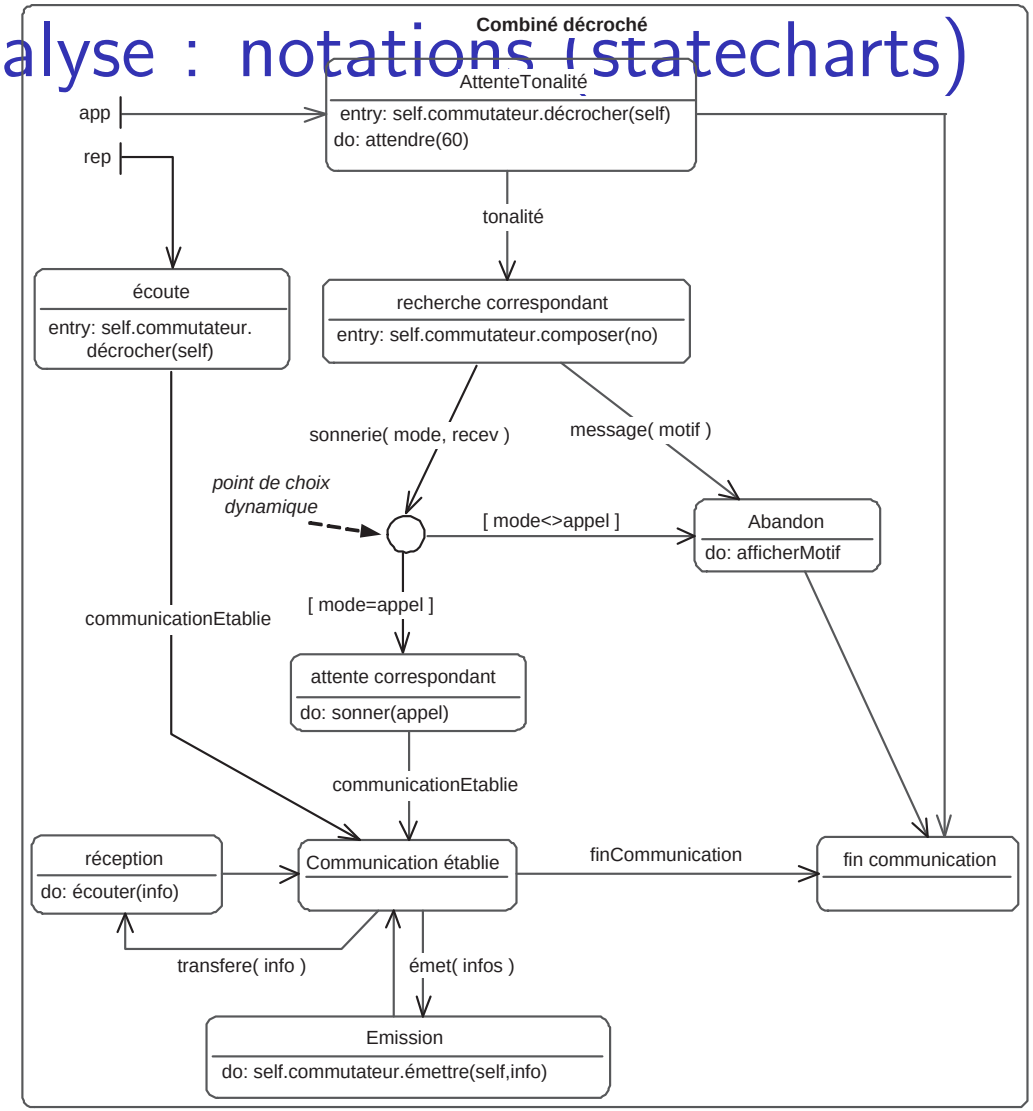
```
context Salle :: créerSalle (bat, noEtage, noSalle,
                  superficie , type) : Salle
pre:
    -- le bâtiment et la salle existent
    Bâtiment. allInstances →includes(bat) and
    Type. allInstances →includes(type)
post:
    -- soit sal l'objet créé
    let sal : Salle in
        Salle . allInstances@pre →excludes( sal ) and
        sal .no_étage = noEtage and sal. no_salle = noSalle and
        sal .no_bat = bat.no_bat and sal . superficie = superficie and
        sal . typeSalle = type and sal .bâtiment = bat and
        -- ajout explicite dans l'ensemble des instances
        Salle . allInstances = Salle. allInstances@pre →including ( sal )
        result = sal
```

Analyse : notations (statecharts)

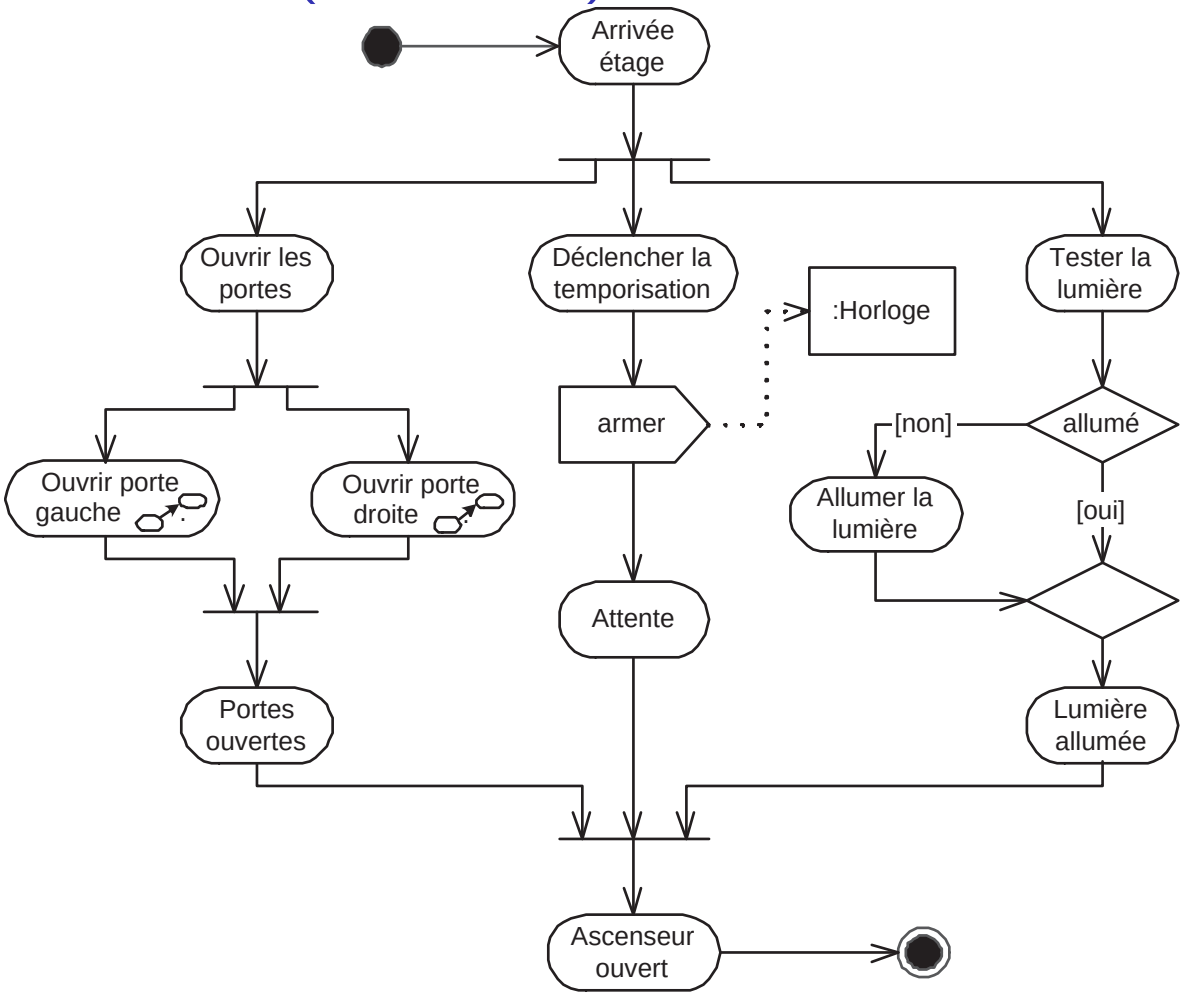


PosteAbonné

Analyse : notations (statecharts)



Analyse : notations (activités)



Analyse : processus

- Point de départ : analyse des besoins
- Architecture : structuration du système
  - Objets Métiers/Interface/Contrôle/Utilitaires : on groupe les classes par nature.
  - Héritage/Association/Instanciation : on groupe les classes par type de relation.
  - Organisation logique e.g. Achat/Finance/Approvisionnement/Statistiques.
  - Répartition géographique ou d'application (architecture C/S n-tier).

vue en couches

UML : méthode

1. Analyse des besoins : cas d'utilisation et scénarios
2. Analyse : diagrammes d'objets et de classes, états-transitions
3. Conception : classes, composants et déploiement
4. Implantation : composants et déploiement

Conception : aperçu

Design

- décrire le système dans le contexte de son implantation
  - affiner l'analyse
  - décrire la prise en compte des aspects logiciels : persistence, concurrence, sécurité...
  - décrire l'architecture logicielle et matérielle du système
- modélisation à objets ou composants
- structuration en sous-systèmes, en couches

Conception : modèles

- diagrammes de composants
  - composants, processus, applications, bibliothèques
  - dépendances
  - interfaces, couches
- diagrammes de déploiement
  - nœuds et répartition
  - liaisons et protocoles
- diagrammes de classes
- diagrammes Etats-transitions et Activités

Conception : notations (composants)

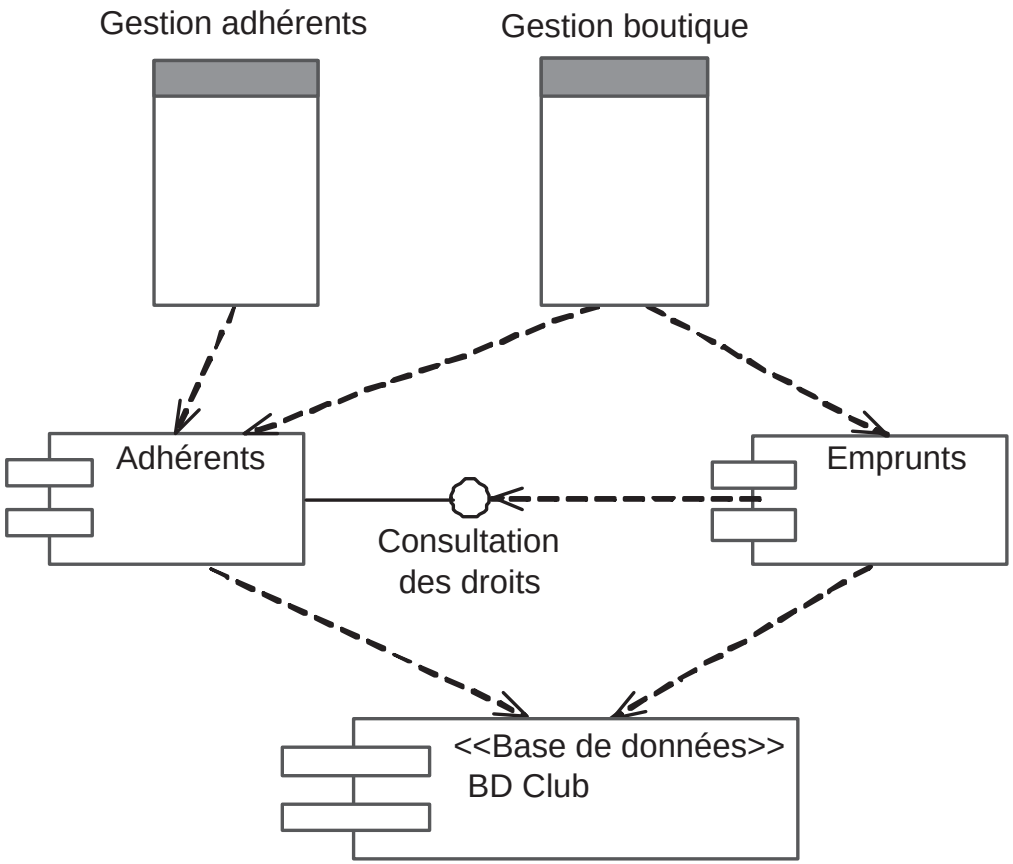
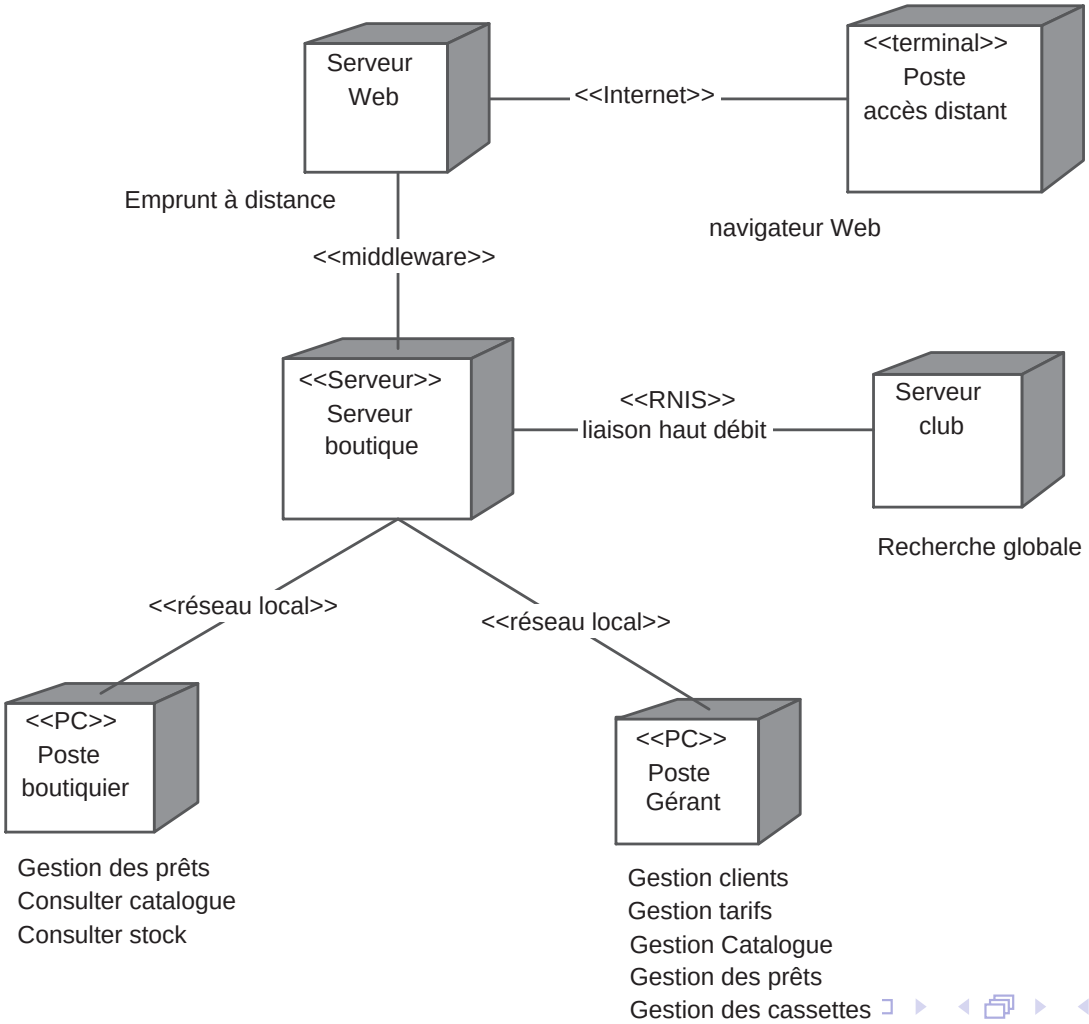


Figure 23 : *Diagramme de composants partiel - Club vidéo*

Conception : notations (déploiement)



UML : méthode

1. Introduction
2. Analyse des besoins : cas d'utilisation et scénarios
3. Analyse : diagrammes d'objets et de classes, états-transitions
4. Conception : classes, composants et déploiement
5. Implantation : composants et déploiement

Implantation : aperçu

Implementation

- coder la conception
  - implanter les algorithmes
  - implanter les couches logicielles
  - implanter les aspects systèmes, BD, sécurité...
- modélisation à objets ou composants
- déploiement

Implantation : modèles

- diagrammes de composants
  - composants, processus, applications, bibliothèques
  - dépendances
  - interfaces, couches
- diagrammes de déploiement
  - nœuds et répartition
  - liaisons et protocoles
- diagrammes de classes ?
- diagrammes états-transitions et diagrammes d'activités ?

Implantation : notations

- voir conception
- fichiers, bibliothèques, pages web, composants...
- documentation de programmation

Implantation : processus

$\Longrightarrow$ lié aux techniques de programmation, aux support technique (*frameworks*) et à l'environnement de développement...

Plan

Introduction

UML : un langage de spécification multi-formalisme

UML : précision avec OCL

UML : une méthode de développement

UML : un processus unifié

UML : outils et vérification

Perspectives

Processus unifié : généralités

Retour sur le développement du logiciel (tome 1, p. 9)

Méthode

- Philosophie = objet
- Formalisme = UML
- Démarche = processus unifié (?) $\Longleftarrow$
- Outils = à suivre $\Downarrow$

UML/processus : généralités

qui fait quoi et comment

UML/processus : généralités

qui fait quoi et comment

Quatre approches :

- Méthodes classiques
  - de l'analyse aux tests d'intégration
  - cycle linéaire, en cascade, en V
  - restriction ou pas des diagrammes à chaque niveau
  - exemple simple : [AV01b]

UML/processus : généralités

qui fait quoi et comment

Quatre approches :

- Méthodes classiques
- Processus unifié (RUP, 2TUP)
  - élabore le modèle final par enrichissement progressifs du modèle d'analyse,
  - basée sur une notation unique (UML),
  - support d'un processus itératif et incrémental, centré sur l'architecture et les cas d'utilisation,
  - concepteurs d'UML

UML/processus : généralités

qui fait quoi et comment

Quatre approches :

- Méthodes classiques
- Processus unifié (RUP, 2TUP)
- MDA - *Model Driven Approach*
  - élabore le modèle final par transformations successives de modèles
  - les modèles indépendants des plates-formes (PIM) sont transformés des modèles dépendants des plates-formes (PSM)
  - proposé par l'OMG

UML/processus : généralités

qui fait quoi et comment

Quatre approches :

- Méthodes classiques
- Processus unifié (RUP, 2TUP)
- MDA - *Model Driven Approach*
- méthodes “agiles” (Scrum, XP, Lean, Puma...)
  - validation rapide : *donne la part belle aux programmeurs et aux clients, PDD*
  - principes de bonne pratique de la programmation à objets (TDD, pair prog, ...)
  - souple, évolutif, cycles courts (sprints Scrum), kanbans
  - adapté aux petites applications et structures (réactifs)

UML/processus : généralités

qui fait quoi et comment

Quatre approches :

- Méthodes classiques
- Processus unifié (RUP, 2TUP)
- MDA - *Model Driven Approach*
- méthodes “agiles” (Scrum, XP, Lean, Puma...)

D’autres sociétés proposent d’autres méthodes : OPEN, Objecteering/Softeam, Rhapsody/I-Logix, Catalysis/ICON Computing, Together/Borland, etc.

Processus unifié : aperçu

Pas de processus unifié

- Rational Unified Process
- Two Track Unified Process (2TUP)
- Scrum, Puma (Lean, XP)
- etc

Processus unifié : aperçu

Pas de processus unifié

- Rational Unified Process
- Two Track Unified Process (2TUP)
- Scrum, Puma (Lean, XP)
- etc

Mais des principes communs...

UML/processus : Unified Process

- Itératif
- Incrémental
- Architecture
- Cas d'utilisation

Préoccupations du développement et de la gestion de projet

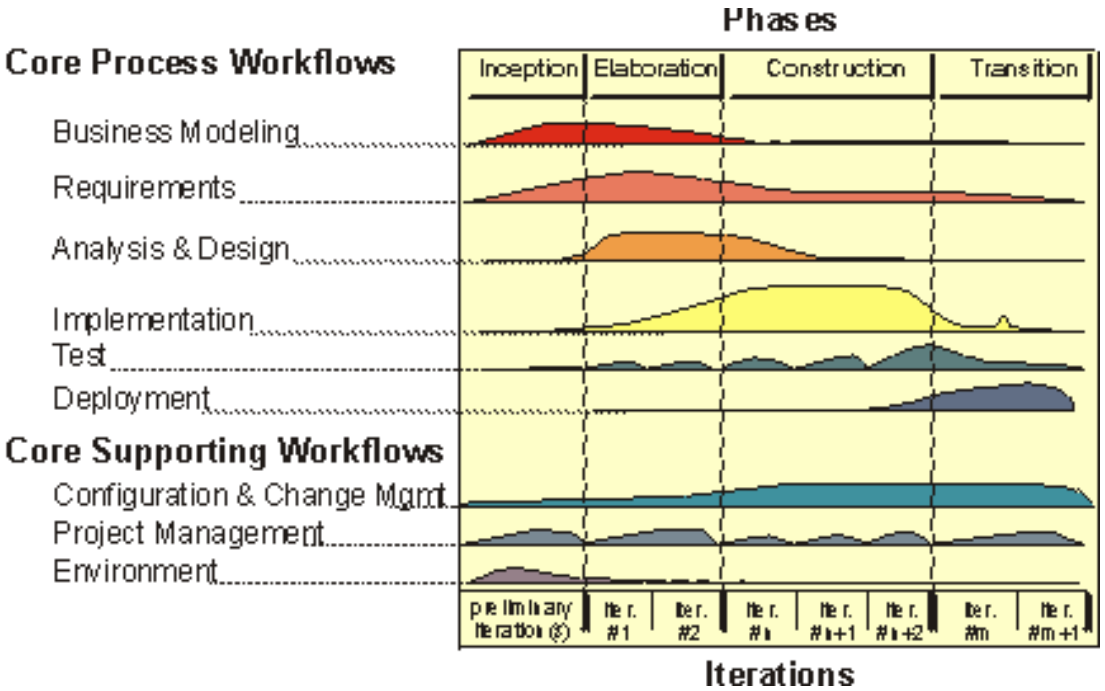
RUP : architecture 1/2

Deux axes

- Activités
  - développement (analyse des besoins $\Longrightarrow$ test)
    $\Longrightarrow$ développement du logiciel (tome 1, p. 13)
    un modèle produit par activité (cf les domaines)
  - support
    - Gestion de configuration & versions
    - Gestion de projet (organisation, risques, planification)
    - Environnement (support et méthode)
- Itérations
  - Grain fin : itération
  - Gros grain : phases

RUP : architecture 2/2

Coordination des deux axes



Effort de développement

Entrelacement des activités de développement et de support dans chaque itération.

Processus unifié : itérations et phases

- chaque itération produit une version du système : un jalon mineur

Processus unifié : itérations et phases

- chaque itération produit une version du système : un jalon mineur
- les phases définissent les grandes étapes du développement : les jalons majeurs, qui contrôlent ainsi le nombre d'itérations

Processus unifié : itérations et phases

- chaque itération produit une version du système : un jalon mineur
- les phases définissent les grandes étapes du développement : les jalons majeurs, qui contrôlent ainsi le nombre d'itérations
  - préparation (*inception*) :
    Etablir la faisabilité et le contexte du projet
    Résultat : *Lifecycle objectives*

Processus unifié : itérations et phases

- chaque itération produit une version du système : un jalon mineur
- les phases définissent les grandes étapes du développement : les jalons majeurs, qui contrôlent ainsi le nombre d'itérations
  - préparation (*inception*)
  - élaboration :
    Etablir l'architecture et planification contrôlée du projet
    Résultat : *Lifecycle architecture*

Processus unifié : itérations et phases

- chaque itération produit une version du système : un jalon mineur
- les phases définissent les grandes étapes du développement : les jalons majeurs, qui contrôlent ainsi le nombre d'itérations
  - préparation (*inception*)
  - élaboration
  - construction :
    Construire un système testable
    Résultat : *Initial Operational Capability*

Processus unifié : itérations et phases

- chaque itération produit une version du système : un jalon mineur
- les phases définissent les grandes étapes du développement : les jalons majeurs, qui contrôlent ainsi le nombre d'itérations
  - préparation (*inception*)
  - élaboration
  - construction
  - transition :
    Mettre le système en production pour l'utilisateur
    Résultat : *Product release*

Processus unifié : itérations et phases

- chaque itération produit une version du système : un jalon mineur
- les phases définissent les grandes étapes du développement : les jalons majeurs, qui contrôlent ainsi le nombre d'itérations
  - préparation (*inception*)
  - élaboration
  - construction
  - transition
- Tests et qualité sont pris en compte par le processus

RUP : synthèse des activités

- source inconnue UPp7.ps
- source inconnue rup-slc.pdf
- références [RJB99, Roy98]

Processus unifié : métamodélisation

Software Process Engineering Metamodel (SPEM)

- une architecture en 4 niveaux M3-M2-M1-M0 :
  - *MOF*,
  - *Process Metamodel* (UPM, UML),
  - *Process Model* (RUP, 2TUP, OPEN...),
  - *Performing process* (opérationnel sur un projet)

Processus unifié : métamodélisation

Software Process Engineering Metamodel (SPEM)

- une architecture en 4 niveaux M3-M2-M1-M0 :
  - *MOF*,
  - *Process Metamodel* (UPM, UML),
  - *Process Model* (RUP, 2TUP, OPEN...),
  - *Performing process* (opérationnel sur un projet)
- quatre éléments de base pour la modélisation :
  - les participants ou rôles (*workers*), le `qui`,
  - les tâches (*activities*), le `comment`,
  - les concepts et productions (*artifacts*), le `quoi`,
  - les activités (*workflows*)), le `quand`.

Processus unifié : métamodélisation

Software Process Engineering Metamodel (SPEM)

- une architecture en 4 niveaux M3-M2-M1-M0 :
  - *MOF*,
  - *Process Metamodel* (UPM, UML),
  - *Process Model* (RUP, 2TUP, OPEN...),
  - *Performing process* (opérationnel sur un projet)
- quatre éléments de base pour la modélisation :
  - les participants ou rôles (*workers*), le `qui`,
  - les tâches (*activities*), le `comment`,
  - les concepts et productions (*artifacts*), le `quoi`,
  - les activités (*workflows*)), le `quand`.

$\Longrightarrow$ personnaliser son processus

Plan

Introduction

UML : un langage de spécification multi-formalisme

UML : précision avec OCL

UML : une méthode de développement

UML : un processus unifié

UML : outils et vérification

Perspectives

Outils : généralités

Jungle des AGL

- Couplage fort méthode et outil.
- Offre florissante
- Offre fluctuante dans le temps (rachats...)
- Notations pas toujours standard
- Extensions de notation
- Souvent couplé avec un environnement de développement

Outils : fonctions attendues 1/2

- Modélisation visuelle : édition de diagrammes et de modèles.
- Génération de documentation (XML, XMI...), métriques pour l'évaluation.
- Vérification, animation, test des modèles avec ou pas OCL.
- Génération de code pour un ou plusieurs environnements (programmes, interfaces, bases de données, etc).
- Certains AGL sont dédiés à des plates-formes et d'autre pas. Les AGL se distinguent aussi par l'environnement système (Windows, Java, Unix).

Outils : fonctions attendues 2/2

- Rétro-conception de code Java, C++, etc. Cette fonction est à la base de la certification de cohérence entre les modèles d'analyse et le code généré (Model Driven Approach). Cela est évidemment plus facile si l'AGL est couplé à une plate-forme.
- Intégration de patrons.
- Extension temps réel.
- Référentiel commun et gestion de configurations.
- Gestion de projet (ressources, planification, communication, etc.), automatisation, personnalisation du processus de développement.

UML/outils : le marché

L'offre logicielle autour d'UML est florissante, près d'une centaine d'outils sont référencés, du simple éditeur de schémas à l'environnement complet de développement en passant par les générateurs de code ou la rétro-ingénierie. Les prix varient de 0 à 10000 euros. Cette offre est très fluctuante de par les fusions et rachats d'entreprise.

- http://www.jeckle.de/umltools.html
- http://www.objectsbydesign.com/tools/umltools_byCompany.html
- http://en.wikipedia.org/wiki/List_of_UML_tools
- http://www.cetus-links.org/oo_uml.html
- synthèse : document de TP
- présentation de E. Dieul dieul.pdf

UML : la vérification

- Principes
- Mise en œuvre

UML : principes de la vérification

- Découpage en domaines de vérification :
  - externe
  - logique
  - physique

UML : principes de la vérification

- Découpage en domaines de vérification :
  - externe
  - logique
  - physique
- Etapes :
  - propriété
  - règle
  - contrôle

UML : mise en œuvre de la vérification

- Trois niveaux :
  1. inter-domaine : propriétés du processus
  2. intra-domaine : propriétés des modèles (cohérence, conformité)
  3. diagramme : plutôt propriétés du système, aussi propriété des modèles

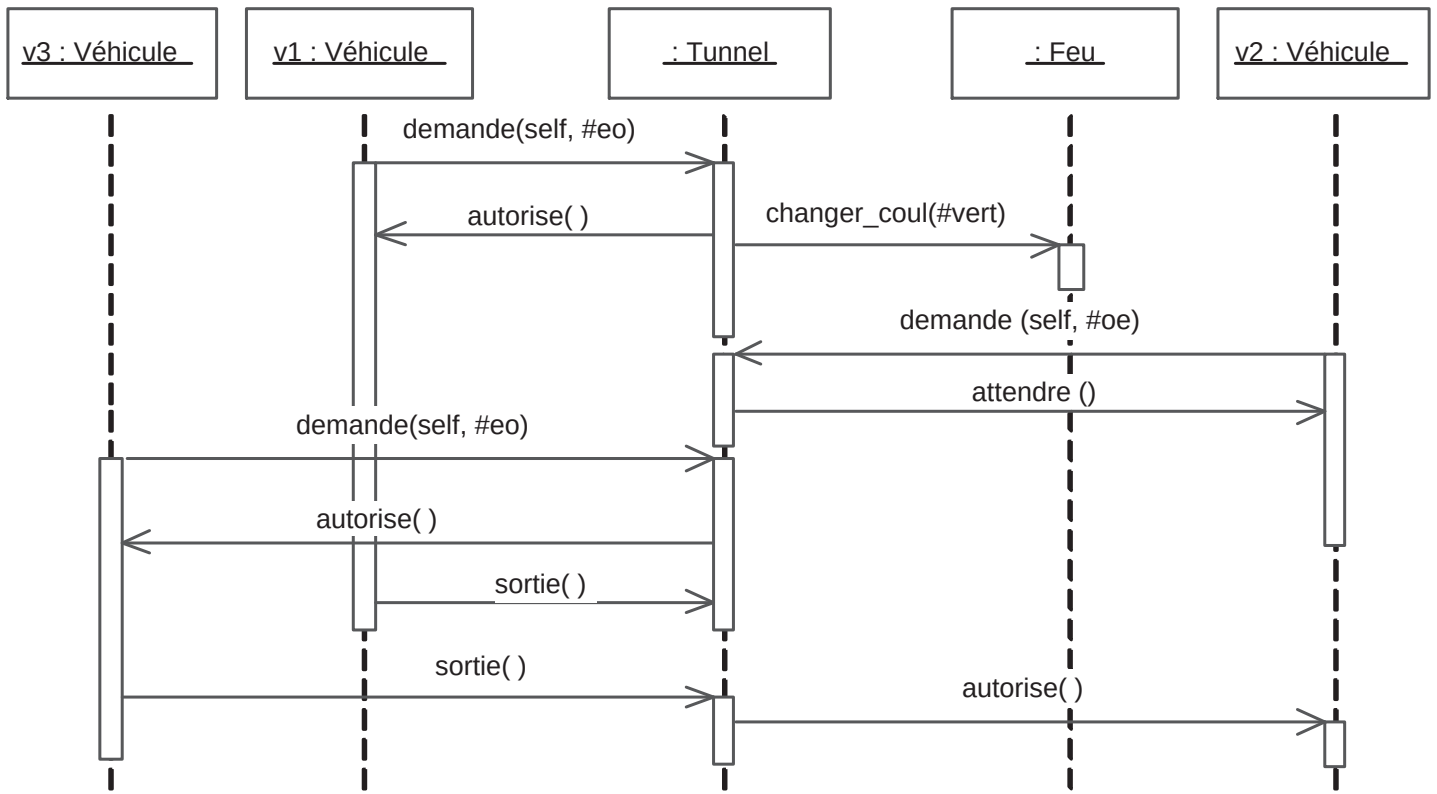
UML : mise en œuvre de la vérification

- Trois niveaux :
  1. inter-domaine : propriétés du processus
  2. intra-domaine : propriétés des modèles (cohérence, conformité)
  3. diagramme : plutôt propriétés du système, aussi propriété des modèles
- Progression :
  1. traçabilité, mais le reste ??
  2. cible principale de la vérification
  3. implanter les règles de théories éprouvées et les compléter

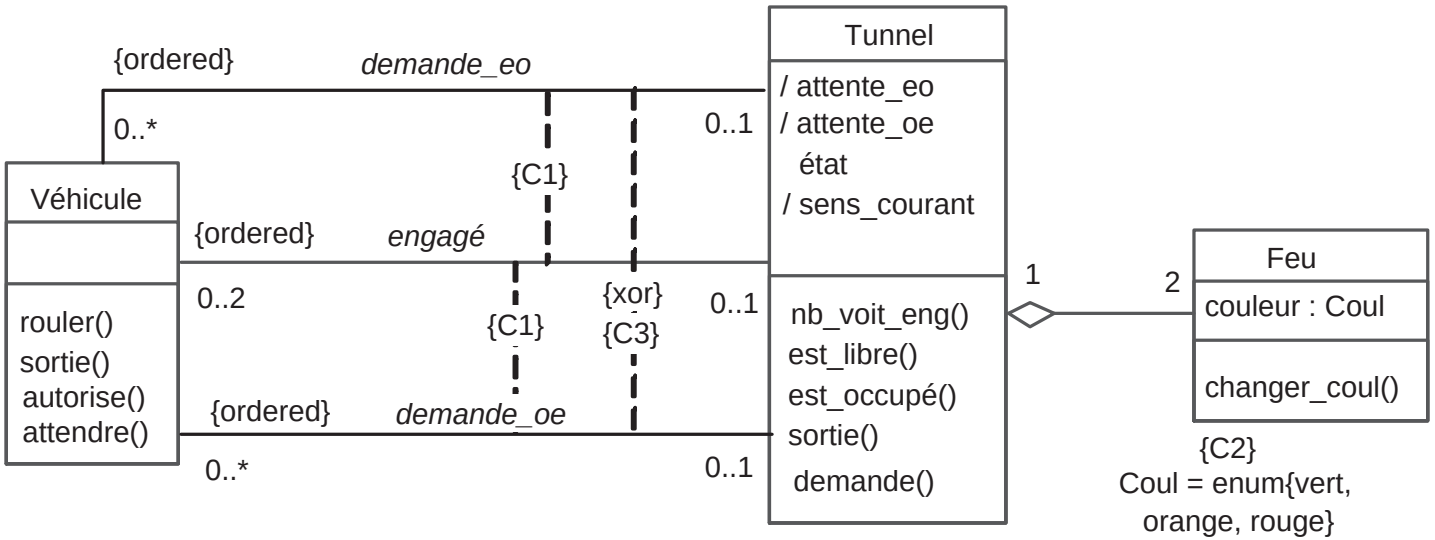
UML : exemple de vérification 1

Le tunnel présente un tronçon mono-voie d'une longueur suffisante pour que deux véhicules puissent s'engager. Les véhicules circulent dans le sens Est-Ouest (EO) ou Ouest-Est (OE). Chaque véhicule avance jusqu'au tunnel et attend une autorisation de passage. Une fois engagé, il progresse et libère le tronçon en signalant sa sortie au tunnel. L'ordre d'arrivée au tunnel n'est pas forcément conservé comme ordre de sortie mais l'ordre des véhicules à l'entrée du tunnel, pour une direction donnée est conservé.
([AV02])

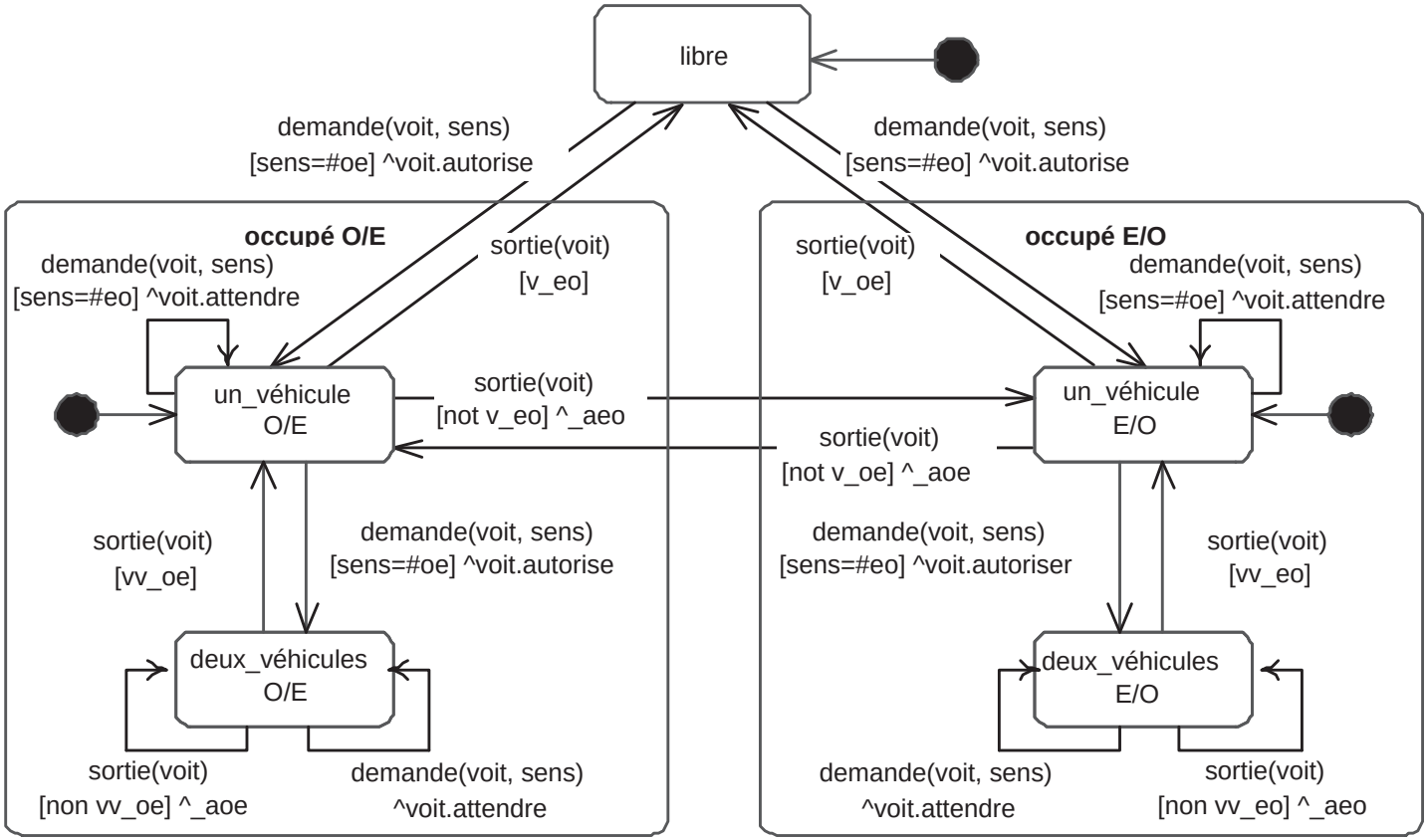
UML : exemple de vérification 2



UML : exemple de vérification 3



UML : exemple de vérification 4



Plan

Introduction

UML : un langage de spécification multi-formalisme

UML : précision avec OCL

UML : une méthode de développement

UML : un processus unifié

UML : outils et vérification

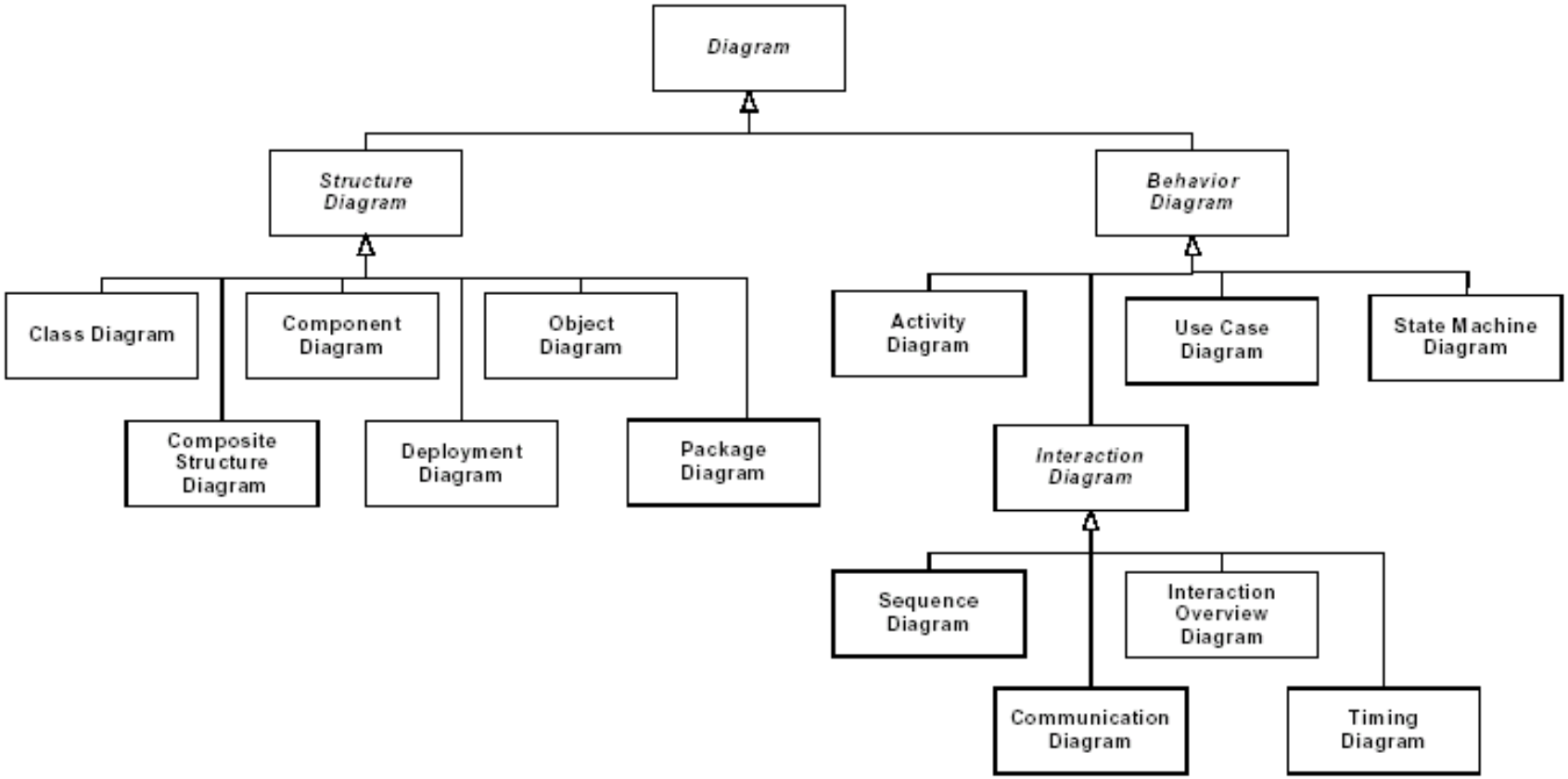
Perspectives

Les nouveautés d'UML 2.0

[Fow04]

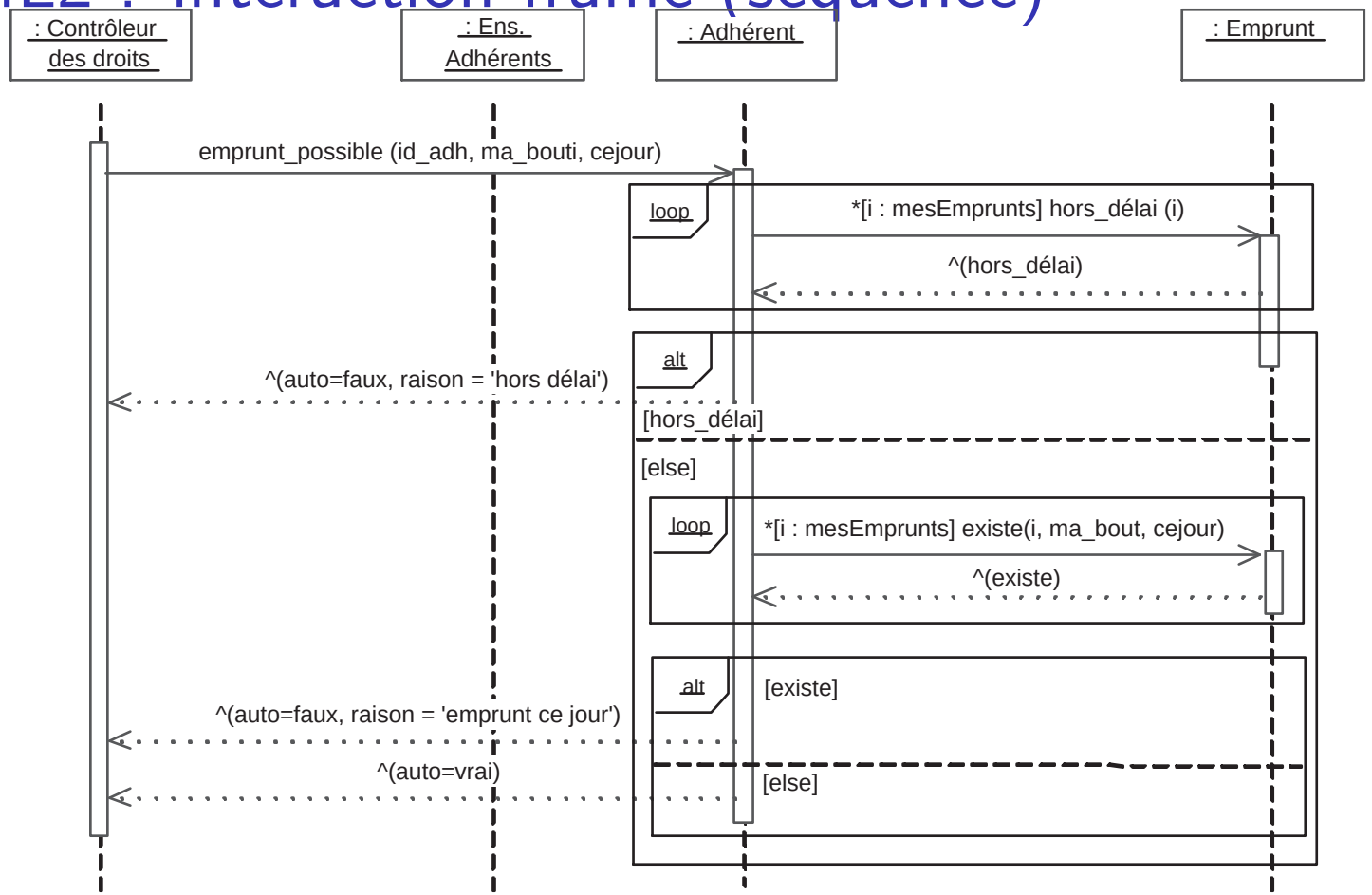
- UC, Classes, Composants, Déploiement
- Séquences, Machines à états
- Objets, Paquetages
- Composites
- Activités
- Nouveaux
  - timing
  - interactions

UML 2 : diagrammes



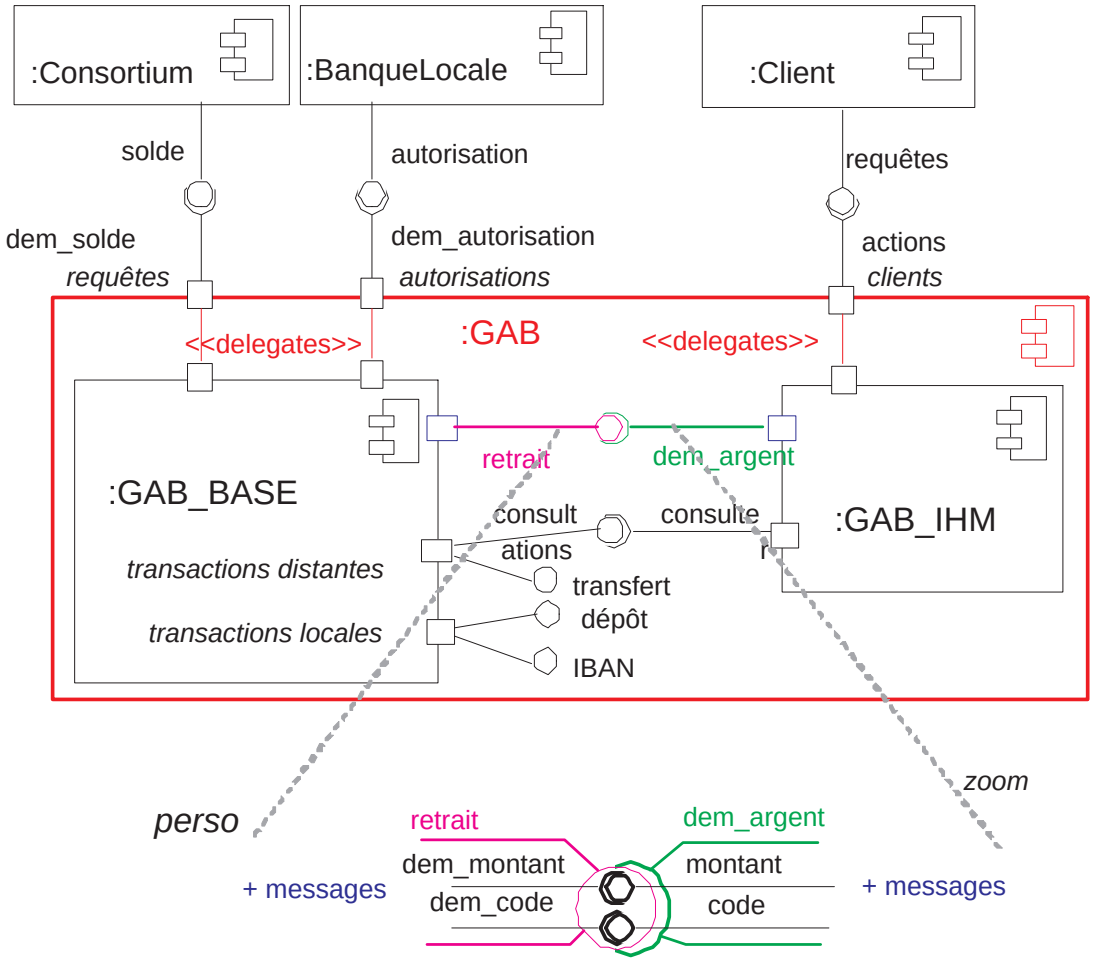
source Wikipedia

UML2 : interaction frame (séquence)



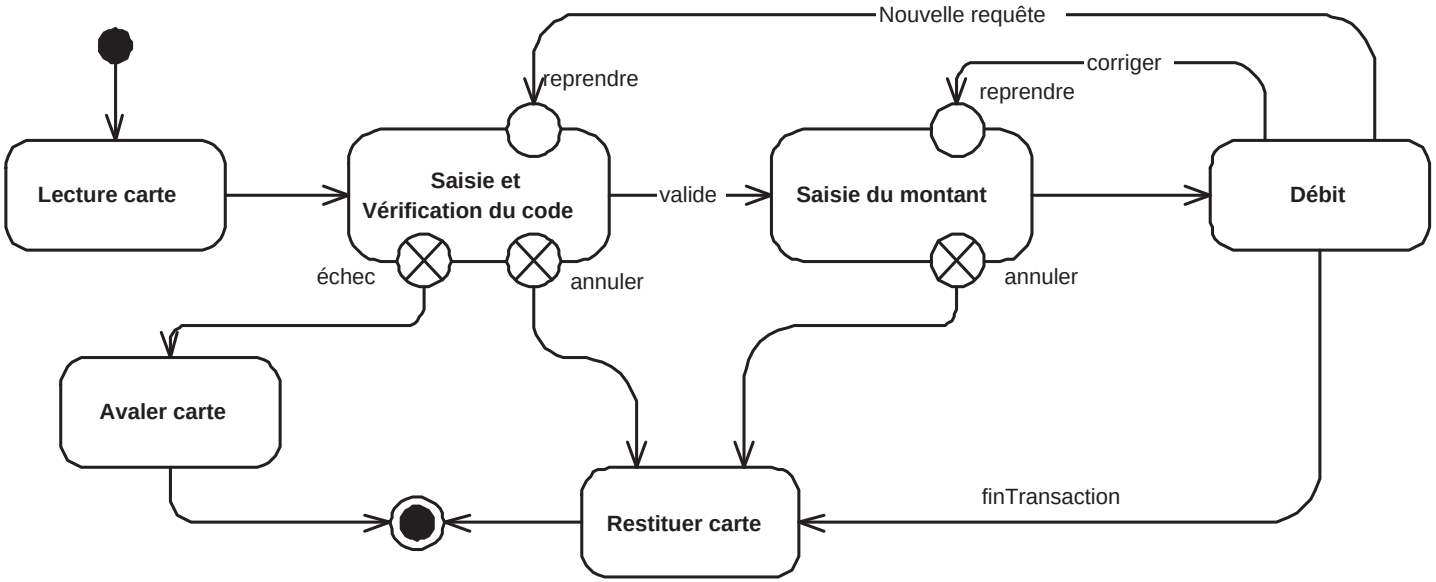
externalisation possible (partage) `loop/opt/alt/sd` nom/`ref` nom

UML2 : ports et services (composants)



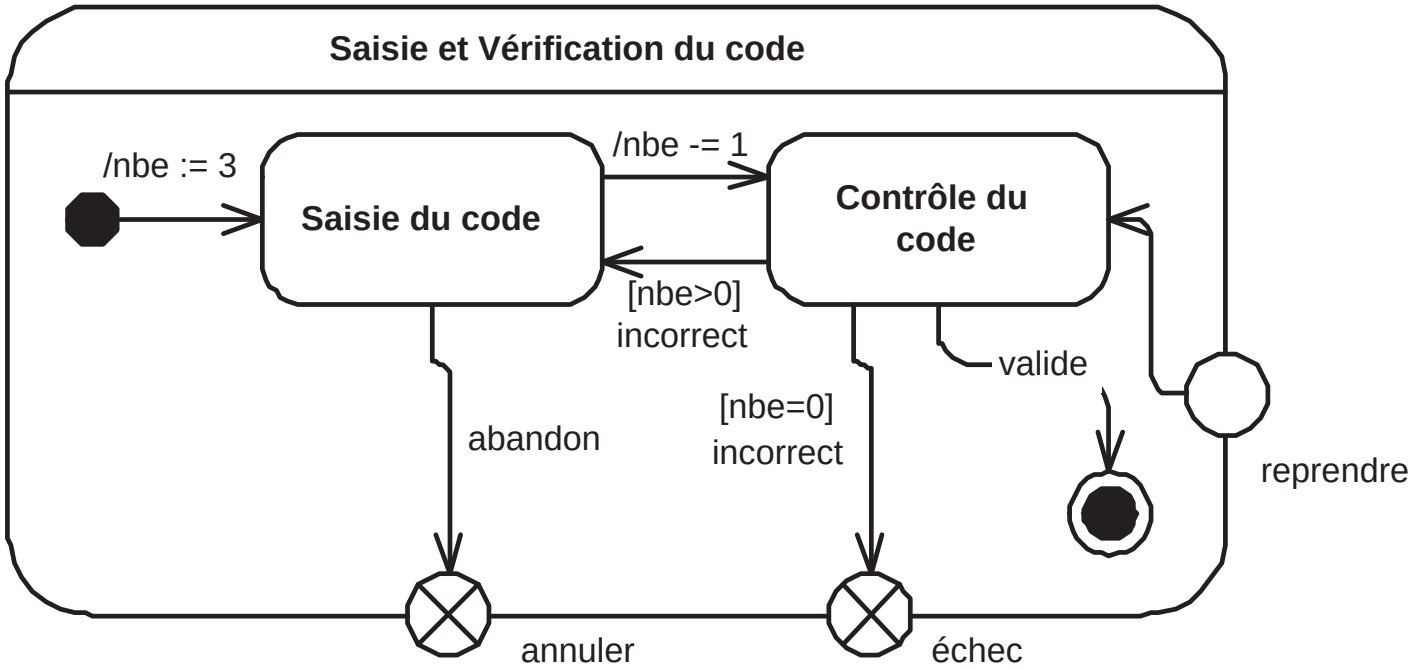
extensions des classes

UML2 : état composite (automates) 1/2



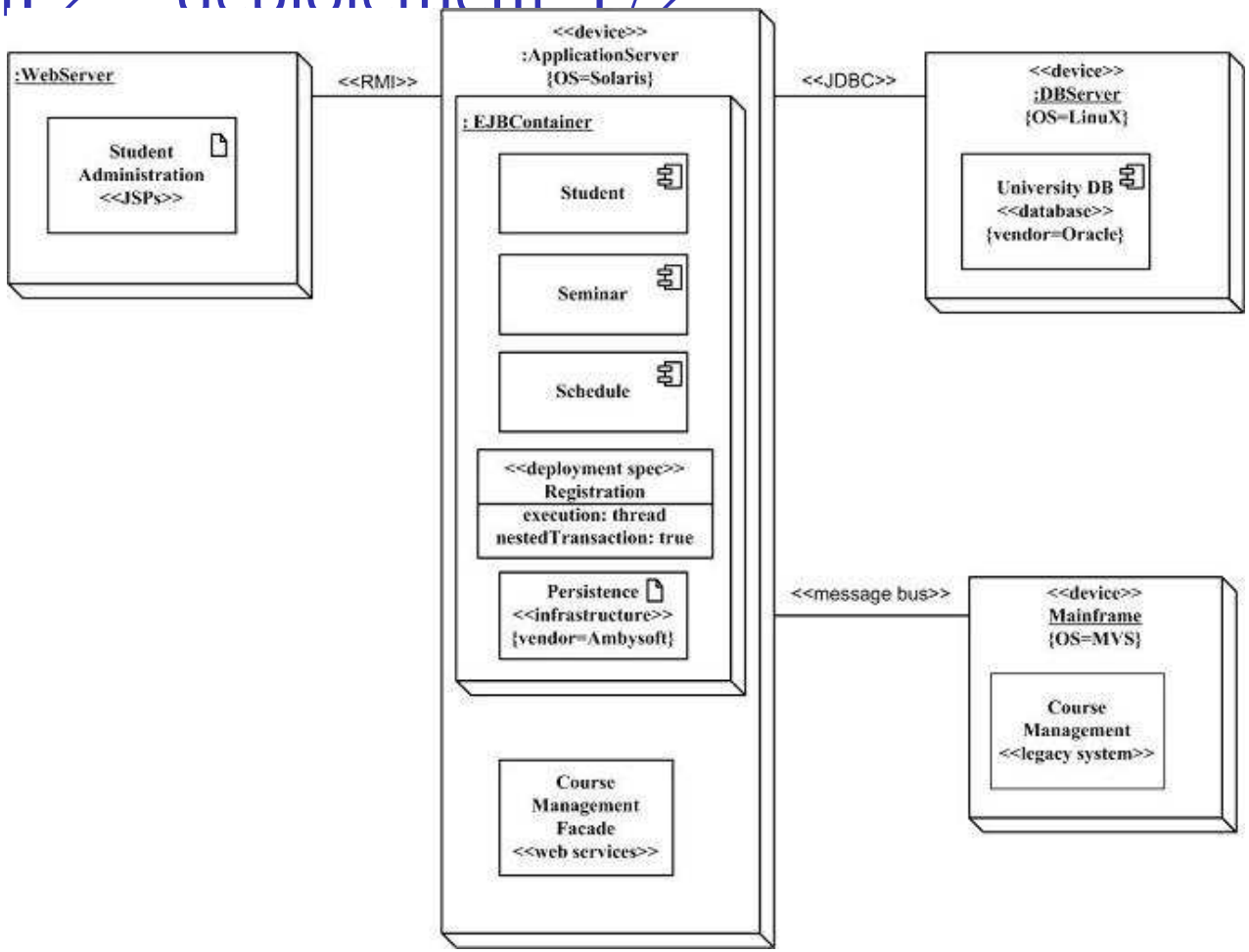
ports spécifiques (entrée, sortie)

UML2 : état composite (automates) 2/2



vue interne

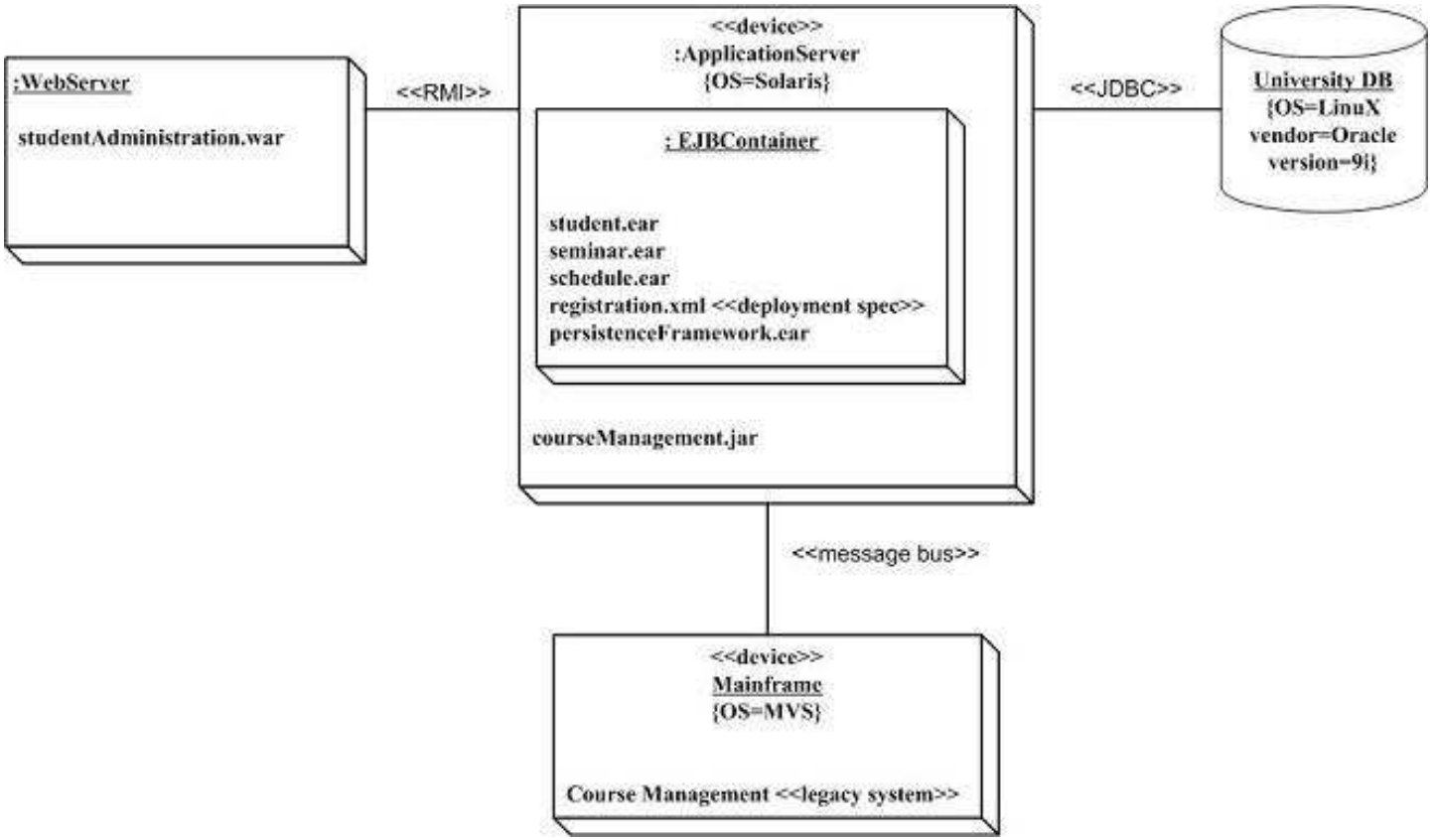
UML 2 : déploiement 1/2



source S. Ambler

www.agilemodeling.com/artifacts/deploymentDiagram.htm

UML2 : déploiement 2/2



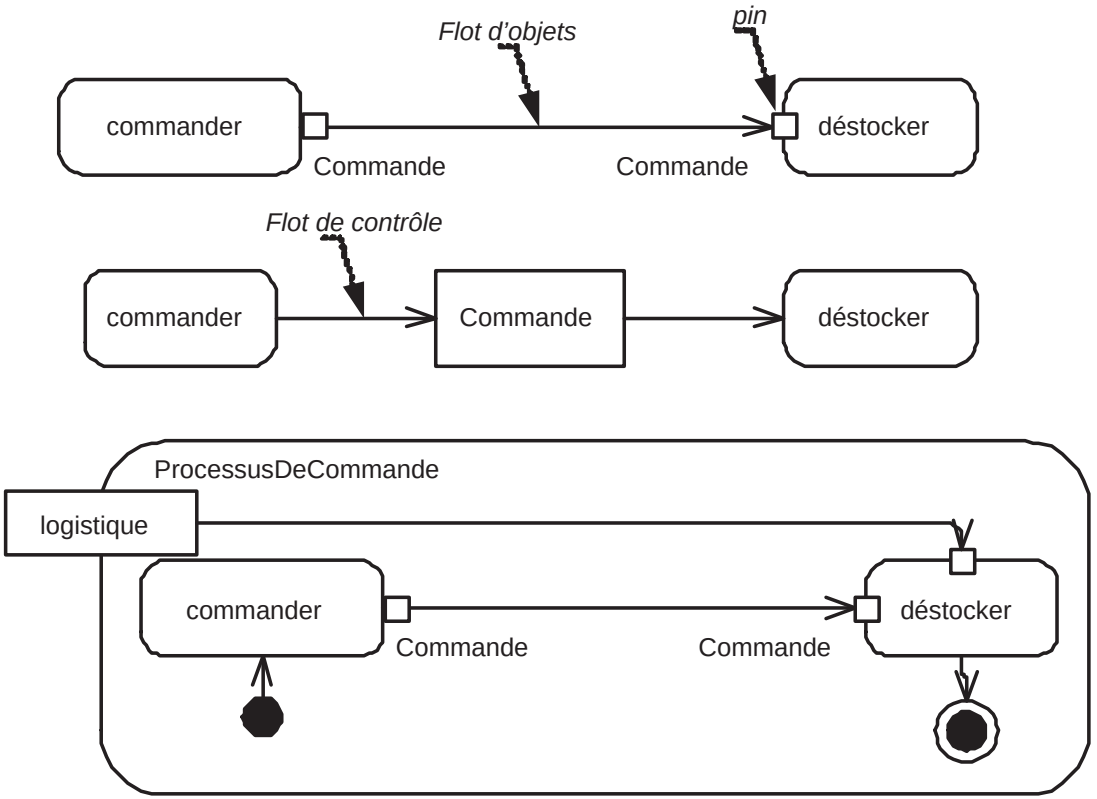
source S. Ambler www.agilemodeling.com/artifacts/deploymentDiagram.htm

UML2 : activités

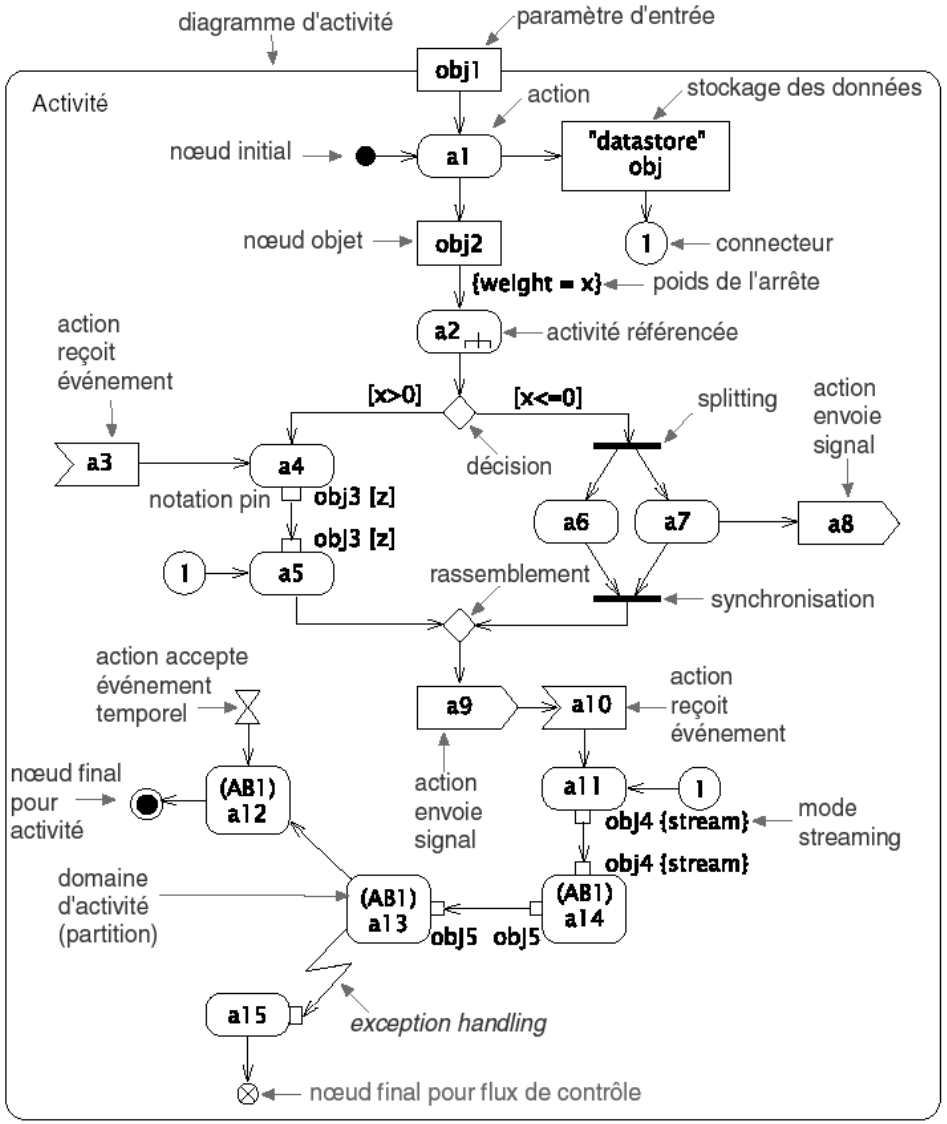
- diagramme qui a le plus évolué
- mixe actions, activités, flots de données (workflow), organigrammes, Petri...
- toutes sortes d'interprétations autour des traitements notamment processus métiers
- Action = traitement, les actions de base sont préféfinies
- Activités = enchaînement d'actions (hiérarchique)

Une série d'articles dans JOT
http://www.jot.fm/jot/issues/issue_2003_07/
http://www.conradbock.org/#UML2
Voir aussi
http://www.agilemodeling.com/style/activityDiagram.htm

UML2 : activités 1/2



UML2 : activi



source H. Balzert

UML 2 Compact Eyrolles

Perspectives

- UML 2.0, OCL 2.0 : converge ou diverge ?
- MDA, SPEM et autres standards ?
- Concrètrement :
  - fixer son besoin méthodologique
  - choix d'une ou plusieurs méthodes
  - choix d'outils

  $\Longrightarrow$ converger vers une méthode applicable et supportée par un outil.
- Intégration de le gestion de projet dans le développement à objet


Pascal André and Alain Vailly.
Spécification des logiciels ; Deux exemples de pratiques récentes : Z
volume 2 of Collection Technosup.
Editions Ellipses, 2001.
ISBN 2-7298-0774-8.


Martin Fowler.
UML 2.0.
Campus Press Reference. Pearson Education France, 2004.
ISBN 2-7440-1713-2.


Object Management Group.
The OMG Unified Modeling Language Specification, version 1.5.
Technical report, Object Management Group, available at http://www.omg.org/cgi-bin/doc?formal/03-03-01, June 2003.


James Rumbaugh, Ivar Jacobson, and Grady Booch.
The Unified Software Development Process.
Object-Oriented Series. Addison-Wesley, 1999.
ISBN 0-201-57169-2.


Walker Royce.
Software Project Management, A Unified Framework.
Object-Oriented Series. Addison-Wesley, 1998.
ISBN 0-201-30958-0.