UML
Unified Modeling Language
(Langage de Modélisation Unifié)

Lamine BOUGUEROUA
Lamine.bougueroua@esigetel.fr

Plan

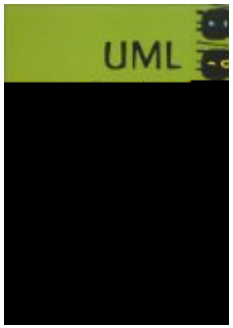
- Références
- Introduction générale
- Historique
- Introduction UML
- Modélisation
- Diagrammes
  - Diagramme de cas d'utilisation
  - Diagramme de classes
  -
  - Diagrammes d'interaction
  -
  -
- Démarche de modélisation avec UML

Références

- Documentations : (Livres Esigetel)

INFTOO38

- Titre : "UML 2 : Synthèse de cours & exercices corrigés".
  Auteur: Benoit Charroux
  ISBN: 978-2-7440-7124-9 (PEARSON EDUCATION)
  Langue: Français



NFTOO34

- Titre : "UML Principes de modélisation"
  Auteurs: Fannader Rémy, Leroux Hervé
  ISBN: 978-2-10-004650-8 (DUNOD)
  Langue: Français

- Titre : "Introduction à UML "
  Auteurs: Alhir Sinan Si,
  ISBN: 978-2-84177-279-7 (O'Reilly)
  Langue: Français

INF32

Introduction : GL

- Evolution :
  - Les ordinateurs devenaient plus puissants
  - Coût du matériel en diminution
  - La construction des logiciels plus complexe

- Conséquences : → La crise du logiciel

  La fabrication des logiciel trop chère

  Dépassement du budget

  Ergonomie discutable

  Les performances faibles

  Applications non évolutives

- le génie logiciel

Introduction : GL

Définition :

Les étapes :

- Analyse de besoin
- La spécification : fonctionnelle, technique
  - Conception
  - Le développement
  - Tests : unitaire, intégration, validation
- Recette : validation

- Maintenance et évolution : exploitation

➔

Introduction : GL

- **Problème de cycle en V:**

➢ La phase de test intervient tardivement dans le projet

✓ la détection des erreurs arrive trop tard

- **Cycle itératif** :

-

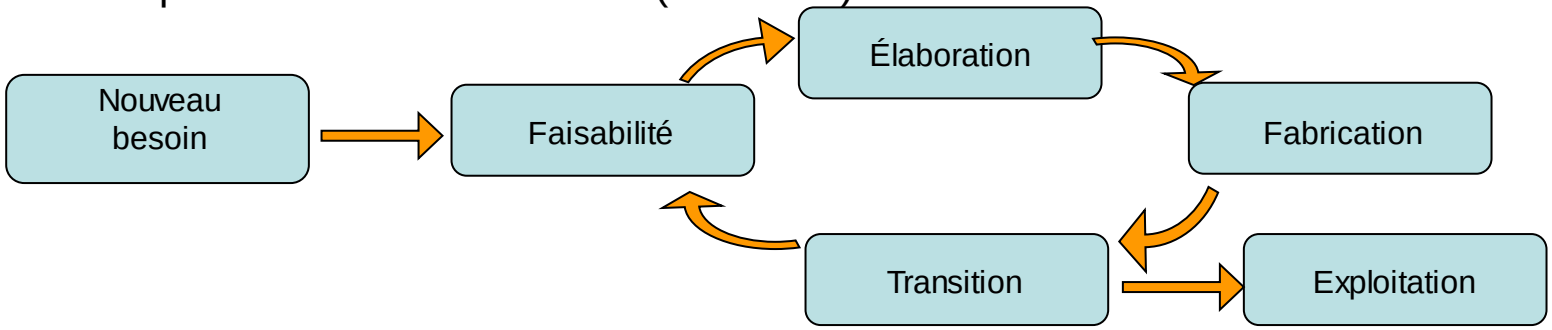
- Découpage du système en sous-systèmes qui facilite le travail en équipe

➢ Accepter un nouveau besoin (faisabilité)

➢ Faire la conception (élaboration)

➢ Passer à la réalisation (fabrication)

➢ Préparer la livraison au client (transition)



Introduction : POO

- La POO est un paradigme de programmation informatique élaboré par *Alan Kay* dans les années 1970

- Les notions sont apparues ultérieurement, de la nécessité de structurer encore plus les programmes informatiques, au fur et à mesure que leur complexité augmentait

- Les langages à objets permettent en effet de décomposer un problème en un certain nombre d'entités indépendantes les unes des autres, qui représentent chacune un intervenant dans le système à gérer

- Ces entités, que l'on appelle communément les objets, sont capables de se gérer elles-mêmes et de communiquer avec les autres pour leur offrir un certain nombre de services

Introduction : choix du modèle

- On n'utilise pas une technologie objet parce que c'est la mode, mais plutôt parce que c'est approprié
- Le choix de la technique et des outils se place donc dans le cadre d'une réflexion plus globale, parce qu'il impose la méthodologie utilisée pour résoudre le problème
- On ne fait pas de conception objet a posteriori, il faut qu'elle s'intègre même dans le cycle de développement
- L'approche systémique (années 80)
  - Modélisation des données et modélisation des traitements pour la gestion des projets internes aux entreprises
  - Merise (méthode française)
  - Axial, SSADM, SDM/S

Historique

- *La modélisation objet :* différents langages et méthodes ont été mis au point

OMT : *Object Modeling Technique*
- *James Rumbaugh*
- Classe, instance, objet, attribut, méthode
- *Modèles objet et dynamique*

- *Grady Booch (1993)*
- Classe, objet, attribut, méthode.
- *Diag*

OOSE : *Object Oriented Software Engineering*
- *Ivar Jacobson (1992)*
- 5 modèles : besoin, analyse, conception, impl., test
- Objet, contrôle, interface
- *Diag*

Coad
Odell
CASE

UM : *Unified Method*
- version 0.8 - 1995

UML : *Unified Modeling Language*
- version 0.9 - 1996

Rational Software
- version 1.0 - 1997

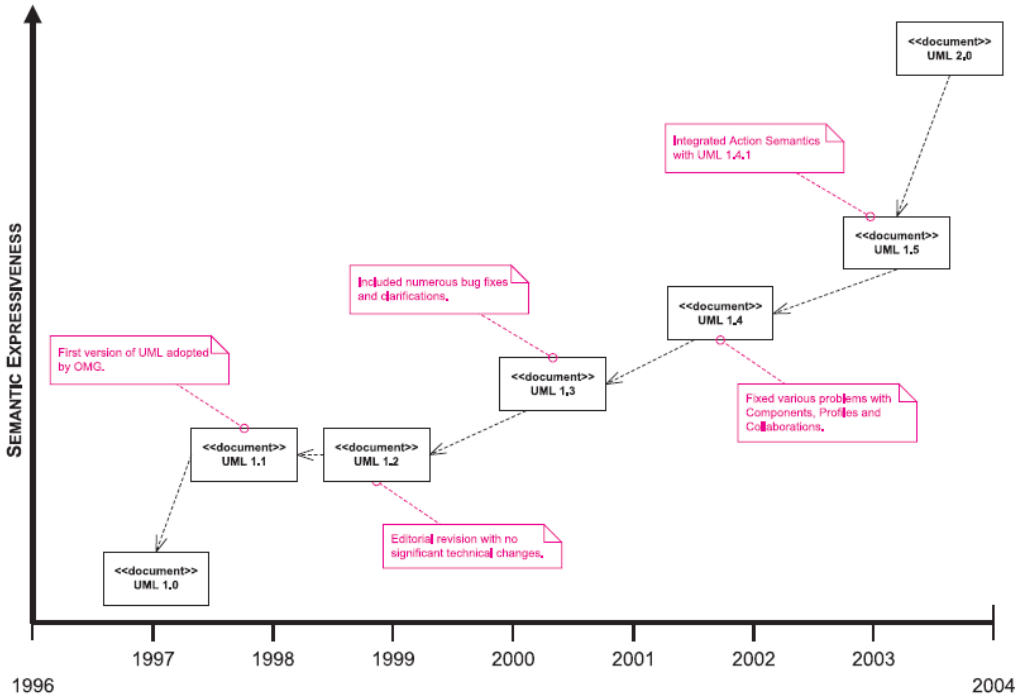
Standardisé en 1997 par l'Object Management Group
UML version 1.1

Historique - Versions

- Versions

UML 1.x

- 1.1 (1997)
- 1.2 (1999)
- 1.3 (Mars 2000) : 9 diagrammes
- 1.5 (Mars 2003)



UML 2.x

- 2.0 (2005)
- 2.2 (2009)
- 2.3 (Mars 2010) : 13 diagrammes
- 2.4 beta (Mars 2011)
- 2.4.1 (Aout 2011)

UML - Introduction

- Pourquoi UML ?

- La conception formelle est de plus en plus importante en fonction de la taille et la complexité du projet et en fonction du nombre de personnes impliquées

-
  -
  - Estimation des temps et des ressources utilisées et planification
  - Communication
  - Documentation du projet

UML - Introduction

- UML : Langage de Modélisation Unifié

- Langage utilisé pour la modélisation des traitements
- Résultat de la fusion de plusieurs langages de modélisation : Booch (Grady Booch), OMT (James Rumbaugh) et OOSE(Ivar Jacobson) en 1995
- Normalisation à partir de Novembre 1997 (OMG: Object Management Group)
-
- Un standard incontournable
- UML apporte une amélioration de la qualité des logiciels

UML - Introduction

«The *Unified Modeling Language* is a **notation**; that is a set of diagrams and diagram elements that may arranged to describe the design of a software system »

is not a process, nor is it a method comprising a notation and

- UML : Caractéristiques

- UML n'est pas une méthode ou un processus
- Indépendance par rapport aux langages de programmations
- Notation graphique

  Limite les ambigüités

-

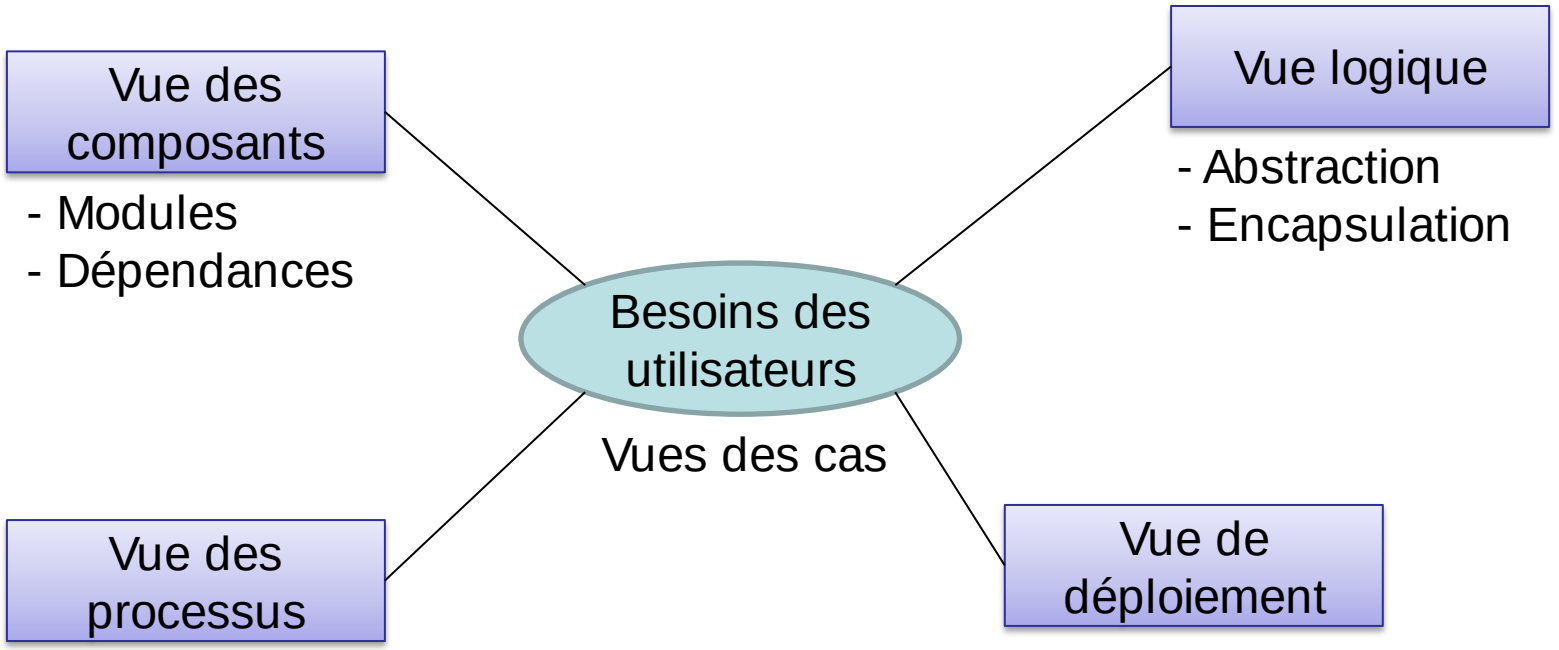
UML - Introduction

- UML : est un langage de représentation des modèles
  - Un modèle peut être définit et visualisé en se basant sur des diagrammes
  - aspect bien précis du modèle

- Comment modéliser?
  - Étape par étape : démarche itérative et incrémentale

Modélisation

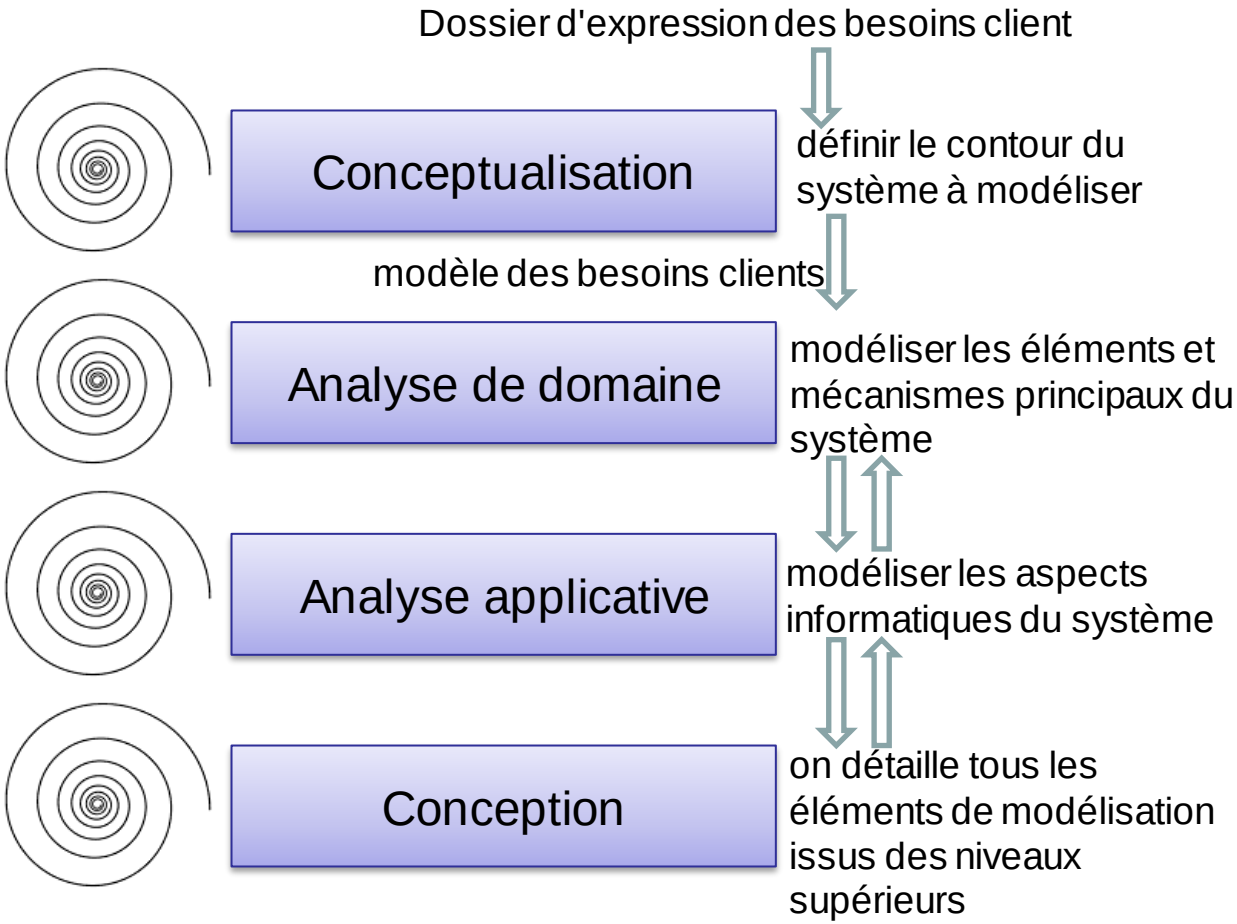
- Modélisation ?



Modélisation

- Modélisation ? démarche itérative et incrémentale

A chaque itération :
- on clarifie, affine et valide les besoins des utilisateurs
- on veille à la prise en compte des besoins des utilisateurs
- on vérifie que les besoins des utilisateurs sont satisfaits



Modélisation

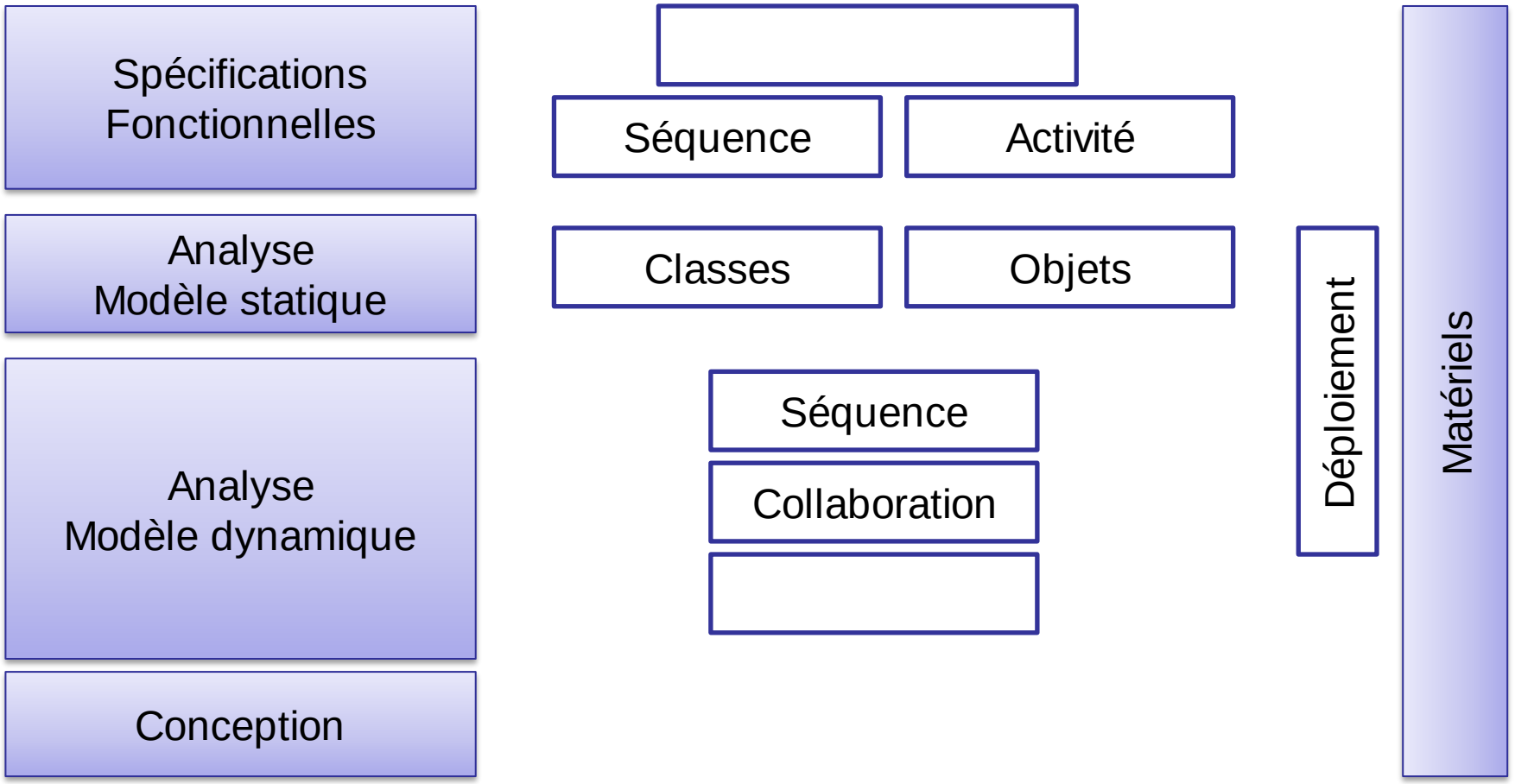
- Modélisation, comment?

- En utilisant plusieurs types de diagrammes : UML permet de définir et de visualiser un modèle, à l'aide de diagrammes
- Un diagramme UML est une représentation graphique, qui s'intéresse à un aspect précis du modèle : c'est une perspective du modèle, pas le modèle
- Combinés, les différents types de diagrammes UML offrent une vue complète des aspects statiques et dynamiques d'un système

Vues statiques du système :
(diagrammes structurels)
- diagrammes de cas d'utilisation
-

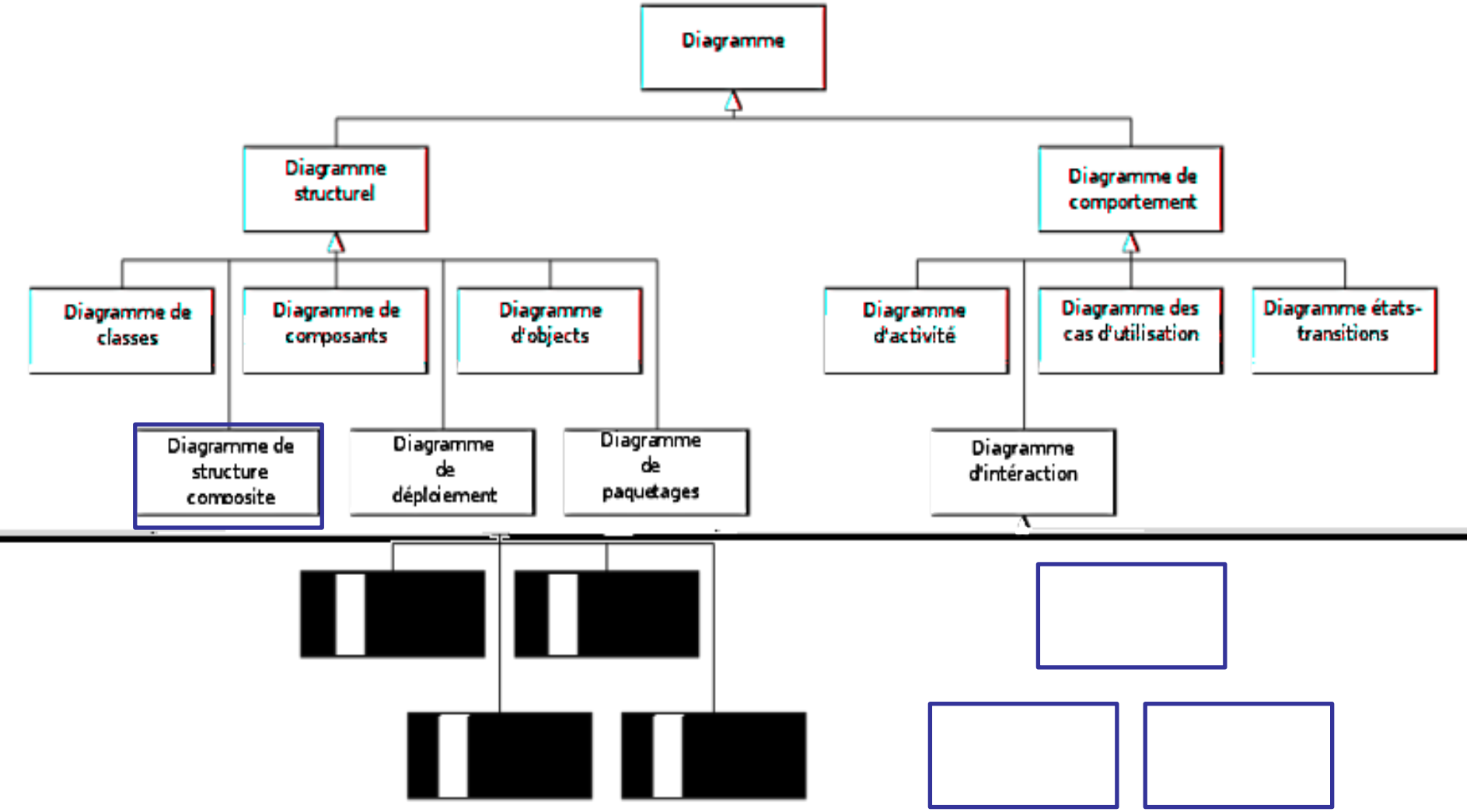
Diagrammes

- Diagrammes : 9 diagrammes



Diagrammes

- Diagrammes (UML 2.3) : 13 diagrammes

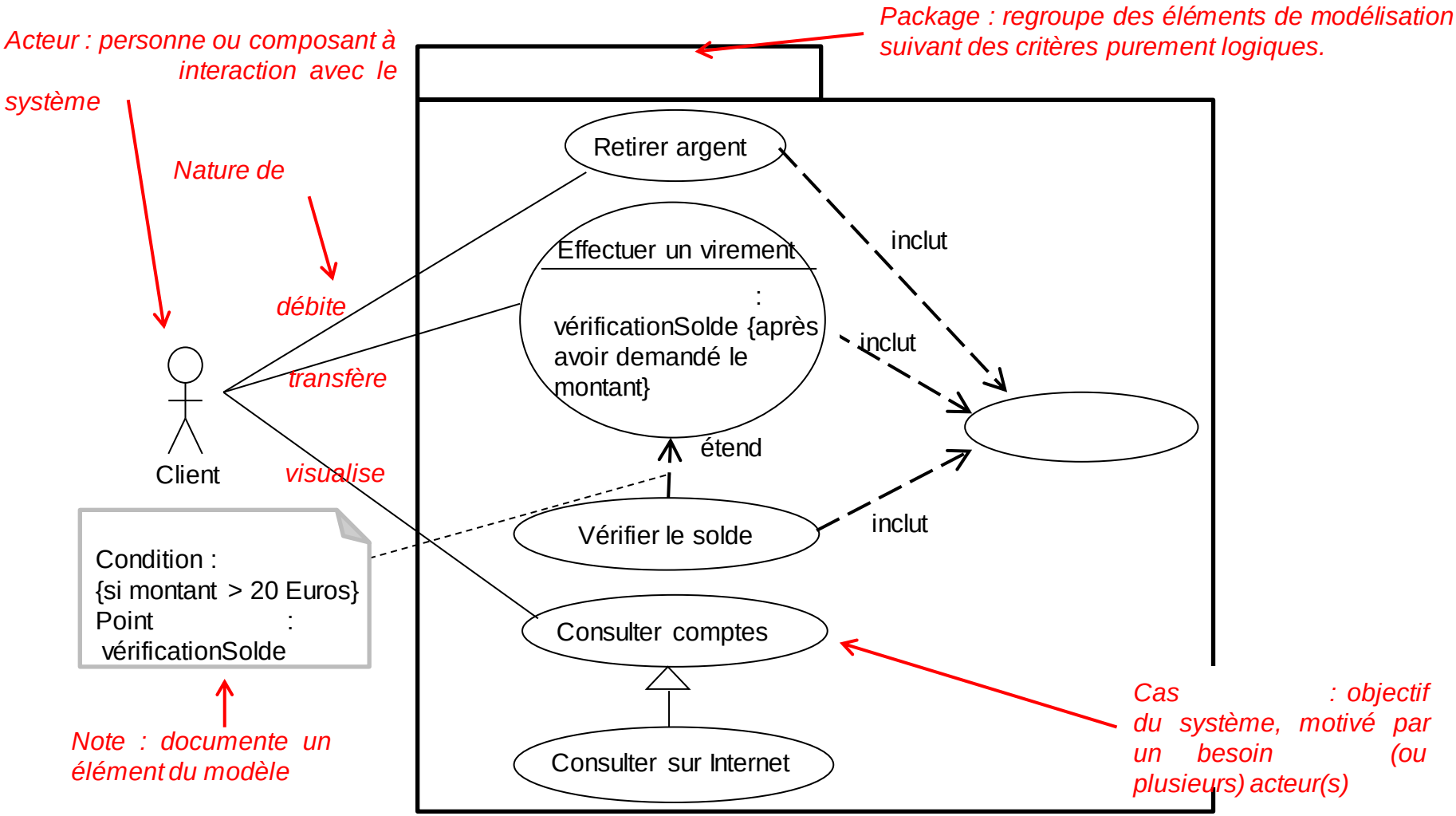


- Montre des acteurs qui interagissent avec les grandes fonctions système.
- une vision fonctionnelle et externe système

Objectifs:

formaliser un besoin => repérer les

délimiter un système.


Acteur : personne ou composant à interaction avec le système
Package : regroupe des éléments de modélisation suivant des critères purement logiques.
Nature de
débite
transfère
visualise
Client
Retirer argent
Effectuer un virement
:
vérificationSolde {après avoir demandé le montant}
inclut
inclut
étend
Vérifier le solde
inclut
Consulter comptes
Consulter sur Internet
Condition :
{si montant > 20 Euros}
Point :
vérificationSolde
Note : documente un élément du modèle
Cas : objectif du système, motivé par un besoin (ou plusieurs) acteur(s)

- Comment?

➢ Les use cases peuvent être décrits sous la forme de flots de différentes façons :

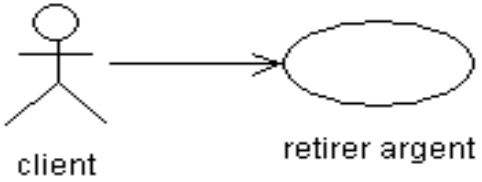
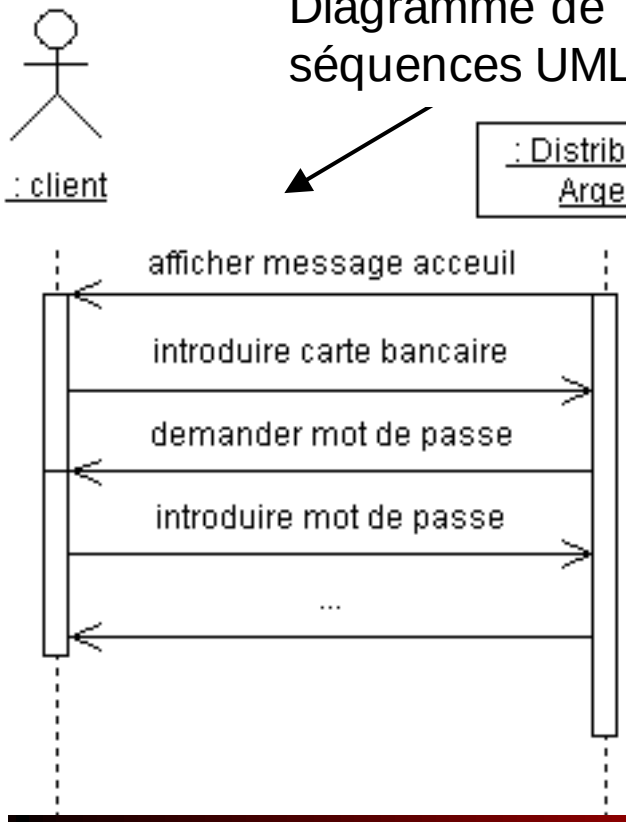


Diagramme de séquences UML

Forme textuelle

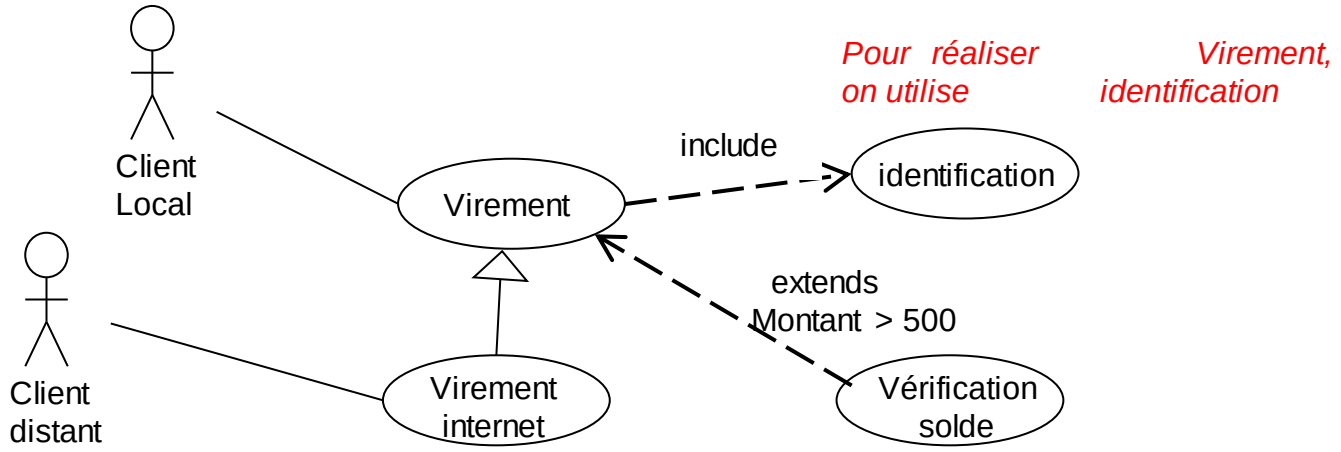


Le distributeur affiche un message

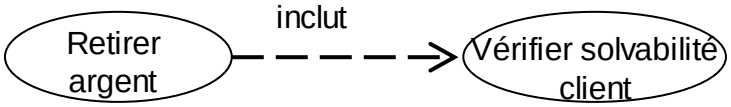
bancaire

le client introduit sa carte bancaire
le distributeur demande le mot de passe de la carte ;
...

- Les trois types de relations sont :
  - Généralisation : quand le cas enfant est une spécialisation du cas parent
  - Inclusion (« include » / « uses » / « inclut ») : quand le cas source comprend le cas destination (permet de décomposer des comportements partageables entre plusieurs cas d'utilisation différents)
  - Extension (« extends » / « etend » / « extention ») : quand le cas source ajoute son comportement au cas destination (l'extension peut-être soumise à condition)

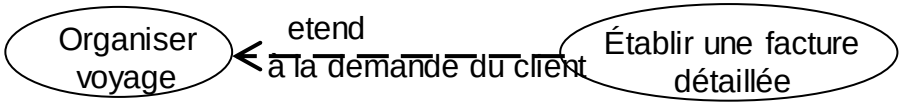


- La relation «*inclut*» est une relation entre 2 instances de cas d'utilisation telle que la réalisation de l'un nécessite la réalisation de l'autre.



Le use case "retirer_argent" va faire appel *systématiquement* au use case "verifier_solvabilité_client" afin de vérifier que le client dispose effectivement de la somme d'argent demandée.

- La relation *«etend»* est une relation entre 2 instances de cas d'utilisation telle que A extend B signifie que le comportement de B peut être complété par le comportement de A. La relation *etend* indique une possibilité, un complément possible.



Le use case "organiser_voyage", peut *éventuellement* faire appel à un autre use case "établir_une_facture_détaillée" uniquement sur demande du client.

- Remarques

➢ Inclusion ou extension ?

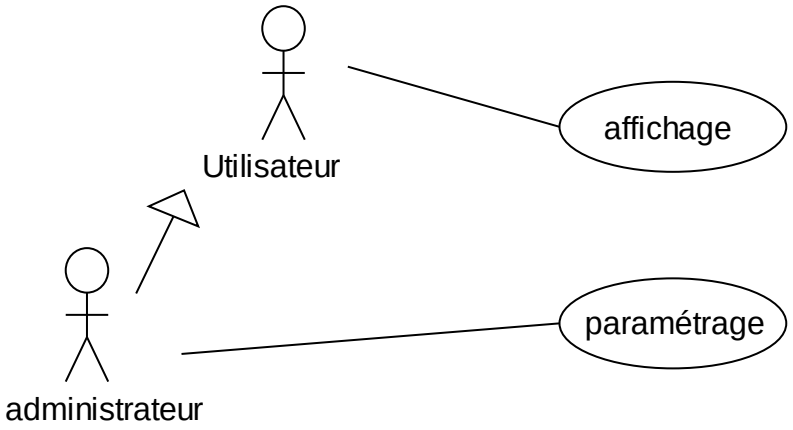
-
- *destination*

  source
-

➢ Relations entre acteurs

- Une seule relation possible entre les acteurs : la généralisation

Tous les administrateurs sont des utilisateurs.
Les administrateurs accèdent aux cas des utilisateurs : ils accèdent à « affichage ».
Par contre, les utilisateurs pas au paramétrage



- Remarques

➢ Scénarios ?

-
-
- day scénario == nominal)
- Puis on a un scénario à chaque point de décision et chaque exception
- On écrira les différents scénarii possibles : ils constitueront la bases des jeux de tests

➢ Intérêts ?

-
- Que doit faire le système
- Qui interagit avec le système
- Quelle interfaces doit posséder le système
- Il permet de vérifier que les développeurs ont bien compris le besoin

Diagramme de classes

- Diagramme de classes

- Décrit le système
- Montre les classes et la façon dont elles sont associées
- une vision statique et structurelle

Dans un diagramme ont trouve des classes et des relations (des associations) entre ces classes :

association simple,
généralisation (héritage)
dépendance,
agrégation,
composition
Implémentation (interface).

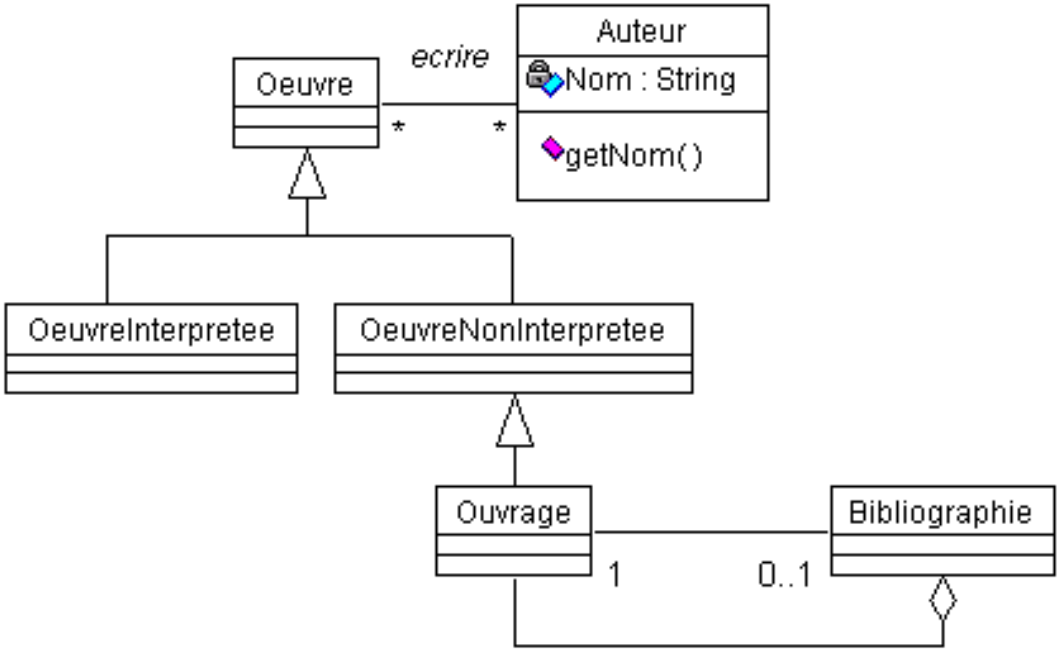
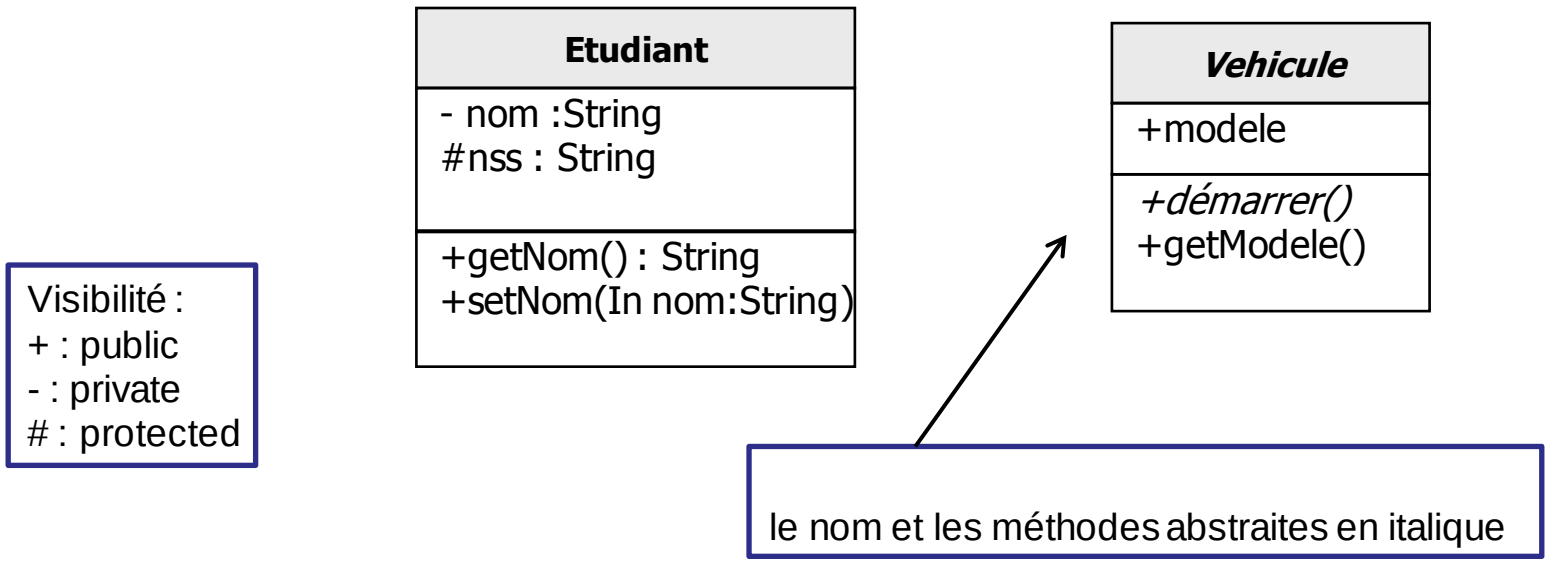


Diagramme de classes

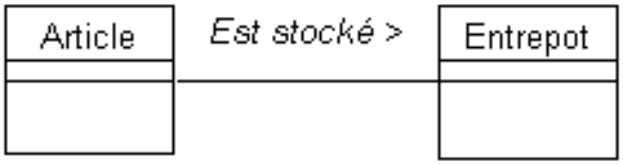
- Classe et classe abstraite

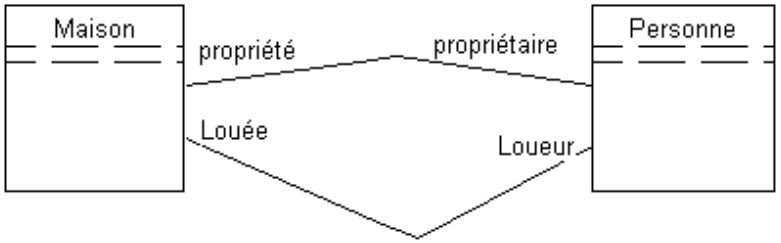
- La classe est un **prototype** qui définit les variables (états) et les méthodes communs à tous les objets même type (comportements)
- *Classe abstraite: classe ne pouvant pas être instanciée directement. Une telle classe sert de spécification pour* des objets instances de ses sous-classe



Relations entre classes

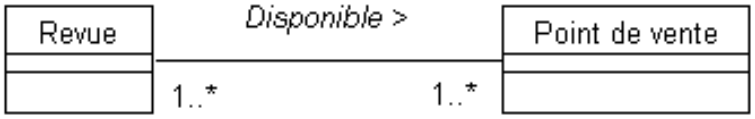
- Association : c'est la relation la plus simple entre deux classes.
- Elle existe à partir du moment où l'une des deux classes sert de type à un attribut de l'autre, et que cet autre envoi des messages à la première (condition nécessaire pour une association).
- Simplement, une association indique que deux classes communiquent entre elles (dans un sens ou dans les deux sens).
- En UML, une association est représentée par une ligne entre deux classes, possiblement accompagnée d'une flèche si l'association n'est pas bidirectionnelle.



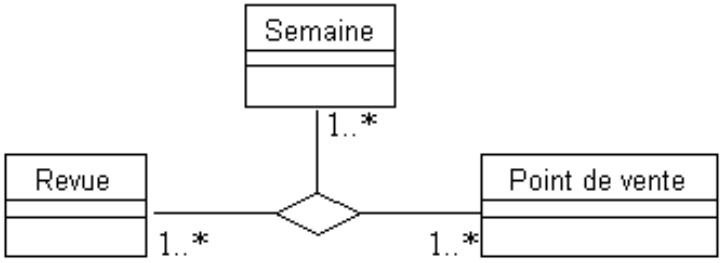


Relations entre classes

- Association (multiplicité):
- Si l'une des deux classes a une association multiple avec classe on parlera alors de cardinalité (multiplicité).
- peut relier deux classes (binaire) ou plusieurs classes
- En UML, la multiplicité est représentée par des valeurs : 0..1, 0..*(*), 1..1 (1), 1..*, m..n



Une revue est disponible dans un ou plusieurs points de vente. Un point de vente distribue de une à plusieurs revues.



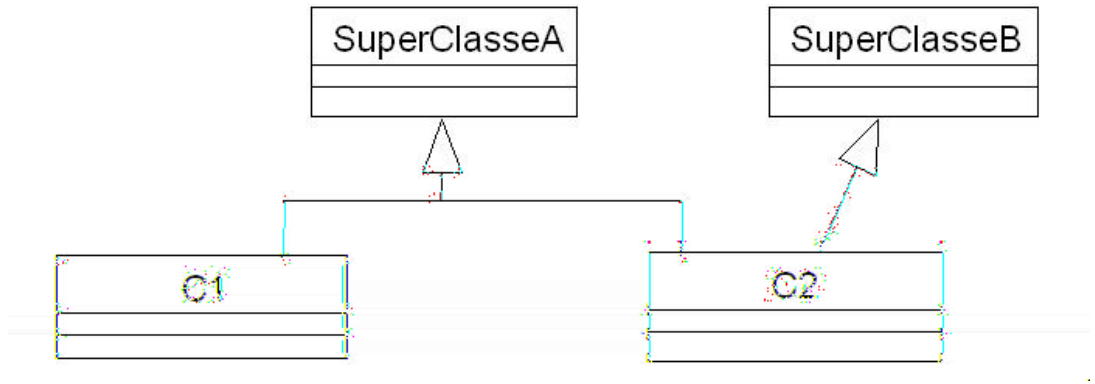
Association ternaire pour suivre les disponibilités hebdomadaires

Relations entre classes

- Généralisation (Héritage) : elle présente une classe spécifique comme descendante d'une classe plus générique. Cette classe spécifique propose des méthodes dont la classe générique ne dispose pas, tout en conservant la plupart des méthodes de cette classe "parente".

- Par exemple, une classe *EtudiantVA* et une sous classe de la classe Etudiant.

- En UML, une généralisation est représentée par une ligne, terminée par une flèche évidée.

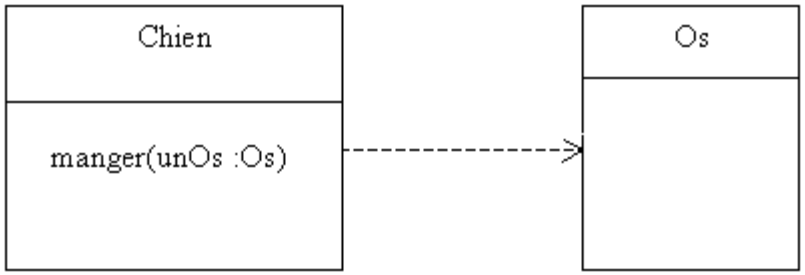


Relations entre classes

- Dépendance : elle présente l'utilisation que fait une classe d'une autre. Une classe dépend d'une autre si ses méthodes manipulent l'objet de cette classe.

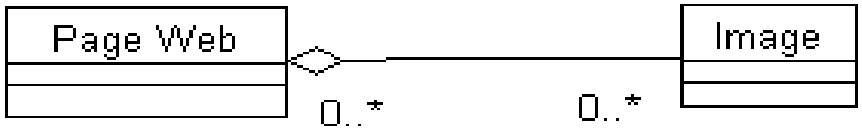
- Par exemple, une classe Réservation ne pourrait exister que si la classe Compte indique les coordonnées de la personne...

- En UML, une dépendance est représentée par une ligne en tirets, terminée par une simple flèche.



Relations entre classes

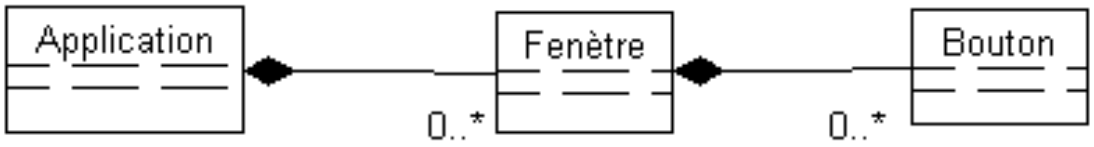
- Agrégation : cette relation indique un principe de subordination entre l'agrégat (classe qui regroupe les classes agrégées) et les agrégées.
- Concrètement, elle indique une "possession" : l'agrégat peut contenir plusieurs objets d'un type.
- Par exemple, une classe Réservation peut contenir un ou plusieurs objets de type Place de Cinéma.
- En UML, une agrégation est représentée par une ligne entre deux classes, terminée par un losange vide ("diamant") du côté de l'agrégat.



- Une page peut contenir des images mais celles-ci peuvent appartenir à d'autres pages.
- La destruction d'une page n'entraîne pas celle de l'image mais seulement la suppression du lien.

Relations entre classes

- Composition "agrégation forte" ou "agrégation par valeur " : il s'agit en fait d'une agrégation à laquelle on impose des contraintes internes
- Un seul objet peut faire partie d'un composite (l'agrégat de la composition), et celui-ci doit gérer toutes ses parties. En clair, les composants sont totalement dépendants du composite.
- Les «parties» sont créées et détruites en même temps que le «tout».
- En UML, la composition est représentée de la même manière que l'agrégation, mais le diamant est plein..



- Une application contient de 0 à n fenêtres qui contiennent de 0 à n boutons.
- La fermeture de l'application entraîne la destruction des fenêtres qui entraîne la destruction des boutons.
- La non-présence des valeurs de multiplicités est synonyme de 1..1.

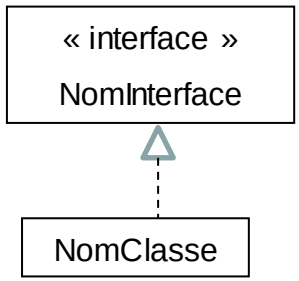
Relations entre classes

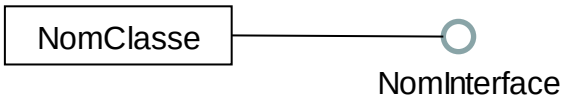
- Exemple :

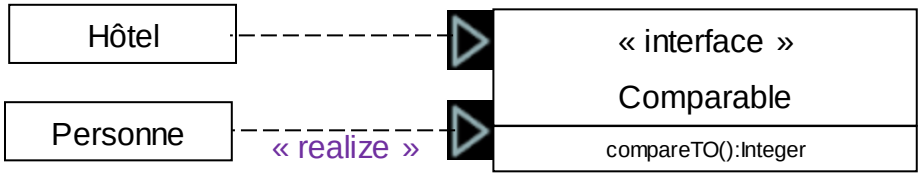
- Le châssis est un élément indissociable d'une voiture, d'où la composition.
- On estime que le moteur et les roues peuvent être utilisés dans d'autres voitures.
- Les valeurs 4..4 caractérisent plus précisément les valeurs de multiplicité.
- Les absences de cardinalité sont assimilable à 1

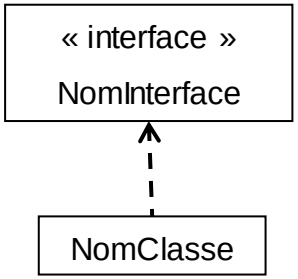
Interface

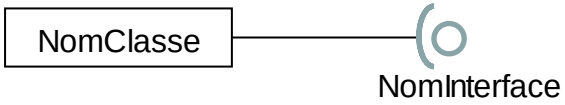
- Une interface est définie comme une classe, avec les mêmes compartiments
- et toutes ses méthodes sont, par définition, abstraites





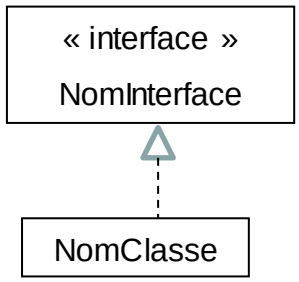


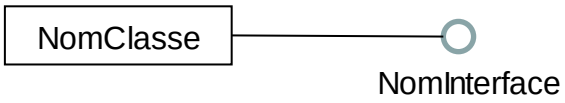


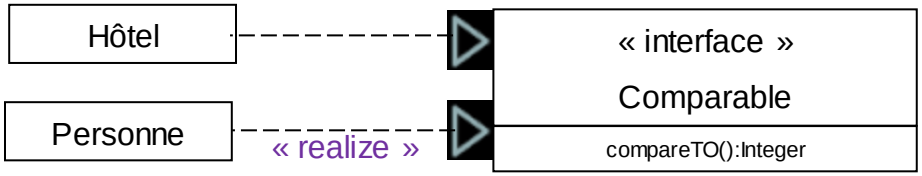


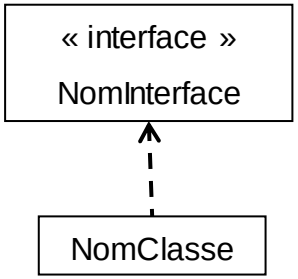
Interface

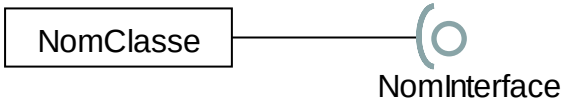
- Une interface est définie comme une classe, avec les mêmes compartiments
- et toutes ses méthodes sont, par définition, abstraites







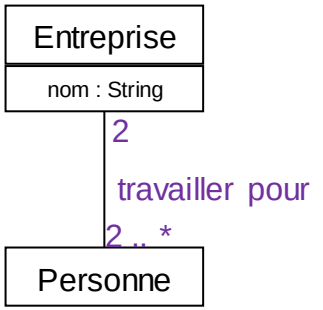




permet, selon des situations :

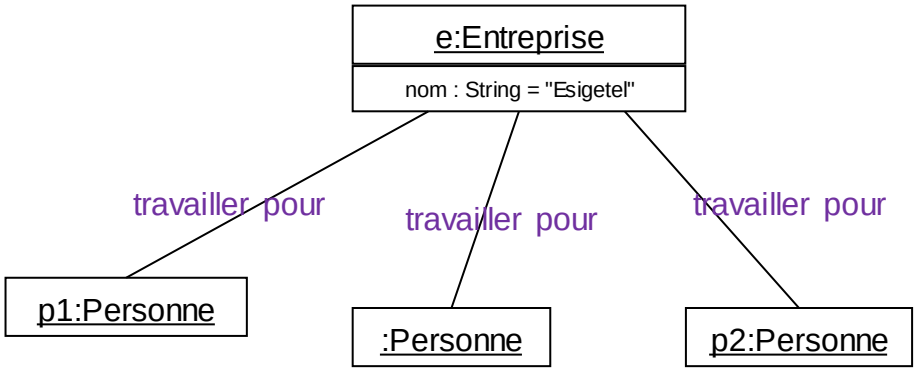
- le modèle de classes en montrant un exemple qui explique le modèle,
- de préciser certains aspects du système,
- une exception
- de une image système à un instant donné.

Diagramme de classes



- Une entreprise emploie au moins une personne et personne travaille dans au plus deux entreprises.

Diagramme



- Une entreprise qui emploie trois personnes

- Montrent dynamique système
- Ils permettent de :
  - décrire des scénarios pour des cas
  - vérifier la cohérence entre les modèles, i.e. montrer comment des instances de classes interagissent pour réaliser les cas
  - décrire le fonctionnement opération.
- Ils sont utilisés tout au long du cycle de vie projet, ils peuvent être affinés.

- les **diagrammes de séquences** qui montrent un séquencement temporel de messages,
- les **diagrammes de communication** décrivent la structure spatiale des participants a une interaction,
- les **diagramme de timing**.

Diagrammes de séquences

- Diagrammes de séquences :

modélisent un comportement dynamique entre objets. Ils se traduisent par l'envoi de messages entre objets. Un diagramme de séquence représente une interaction entre objets, en insistant sur la chronologie des envois de messages

- Montre des interactions entre objets selon un point de vue temporel
- Sert à modéliser les aspects dynamiques des systèmes temps réels et des scénarios complexes mettant en peu d'objets
- L'accent est mis sur la chronologie des envois de messages
- La représentation se concentre sur l'expression des interactions et non pas sur l'état ou le contexte des objets
- Utilisé pour illustrer les diagrammes de cas d'utilisation
- Les scénarios correspondent aux instances concrètes cas
- Il faut décrire chaque scénario
- On peut pour cela soit faire une fiche écrite, soit faire un diagramme UML de séquence

Diagrammes de séquences

- Diagrammes de séquences : Fiche écrite

TÍEMC /P <</MCID 17>> BDC B21 0 0 1 51.75 15.55 Tm377Tm 0.0432 432 4366 0.0432 432 4366(M)20

Diagrammes de séquences

- Diagrammes de séquences : Fiche écrite

Enchaînements :

1.Le guichetier saisit le numéro de compte client

Post-conditions:

Diagrammes de séquences

- Diagrammes de séquences : types de message

Message synchrone

Message asynchrone

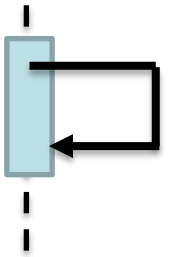
fait ultérieurement.

Retour de procédure

Le retour des messages sont le plus souvent implicite, mais parfois ils sont indiqués (cas asynchrone)

Message réflexif

Les objets peuvent produire des messages pour eux-mêmes

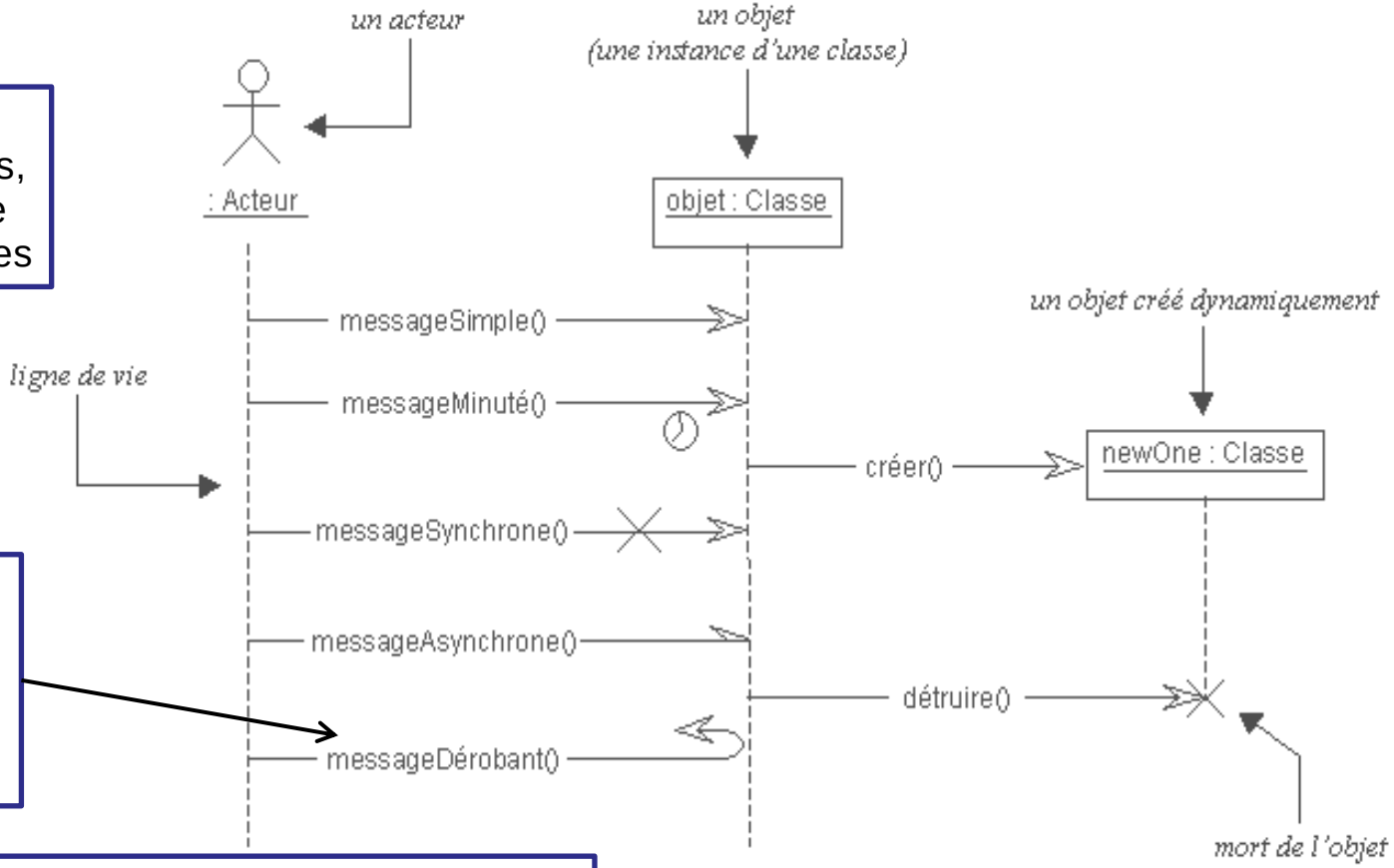


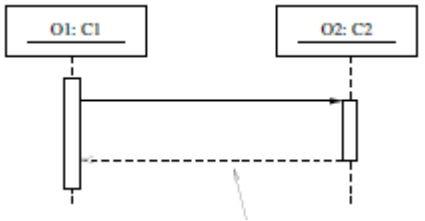
Diagrammes de séquences

Ligne de vie peut porter toutes sortes de contraintes, une contrainte est indiquée par un texte entre accolades

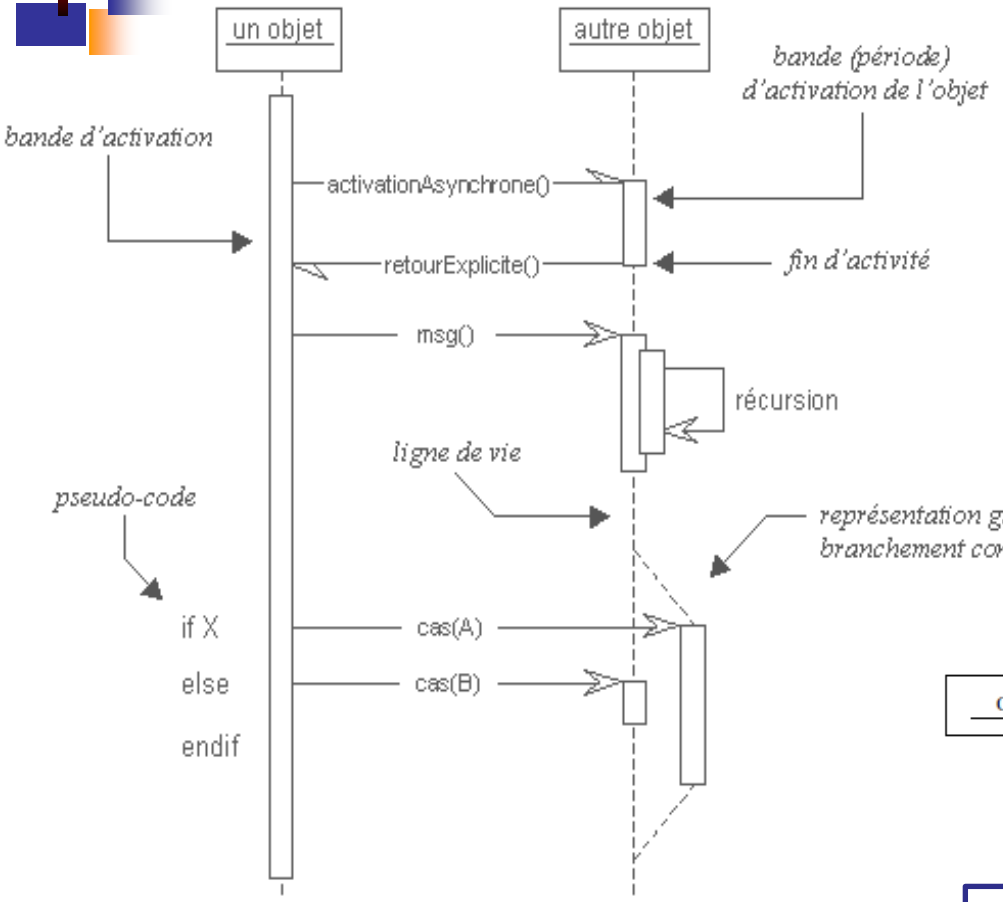
N'interrompt pas l'exécution de l'expéditeur et ne déclenche une opération chez le récepteur que s'il s'est préalablement mis en attente de ce message.

Message:
représentation d'une communication au cours de laquelle des informations sont échangées.





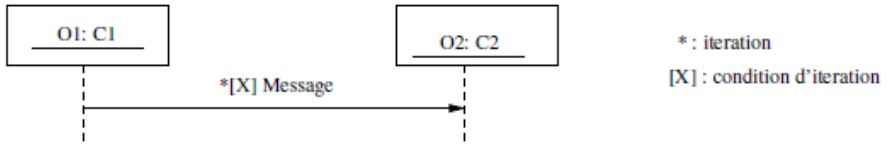
Diagrammes de séquences



Période d'activation: correspond au temps pendant lequel un objet effectue une action, soit directement, soit par l'intermédiaire d'un autre objet.

Un objet peut être actif plusieurs fois au cours de son existence

Boucle « while »



Branchement conditionnel « if then else »

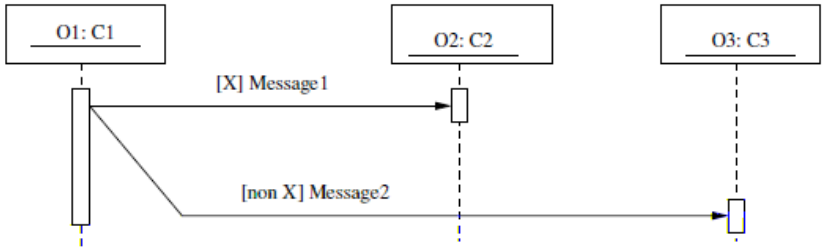


Diagramme des classes

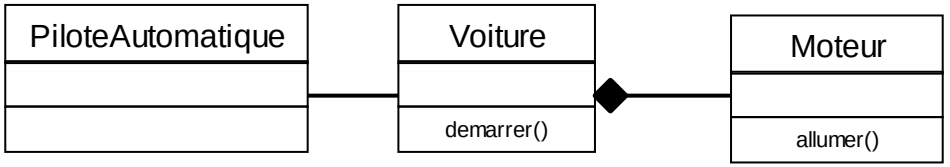


Diagramme de communication

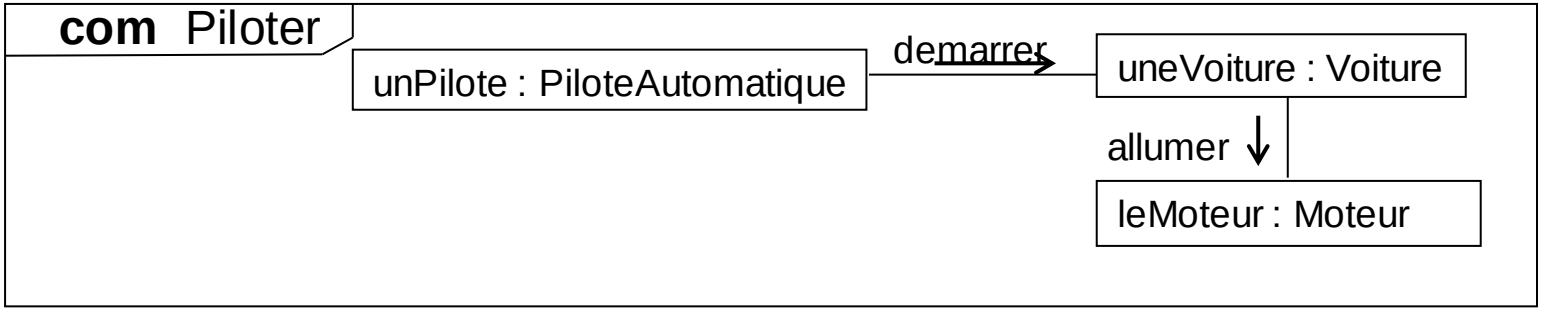
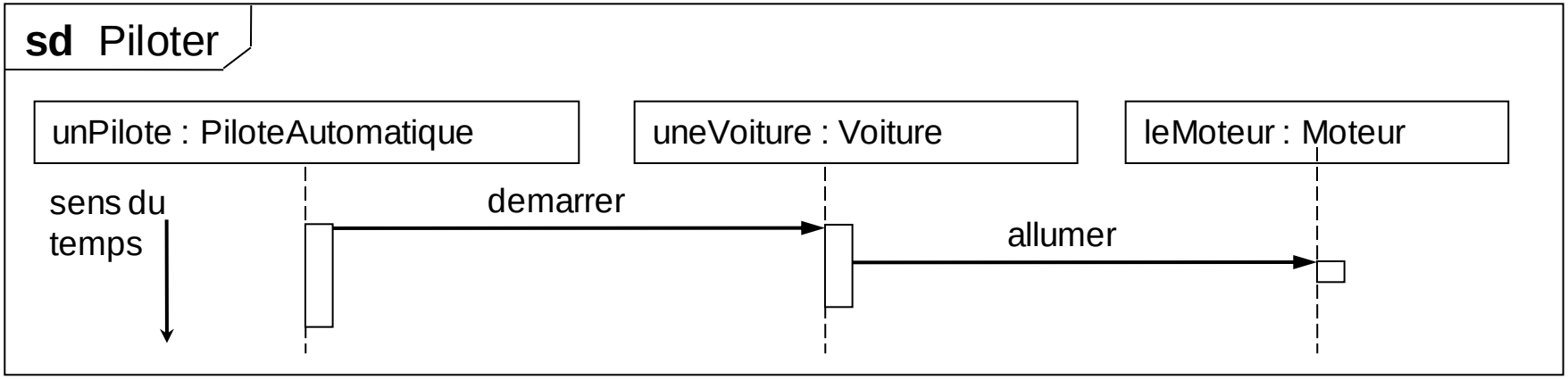


Diagramme de séquence



Interaction Operand) :

- Les opérateurs de choix et de boucle :
  - *alternative, option, break, loop*
-
  - *parallel, critical region*
-
  - *ignore, consider, assertion, negative*
-
  - *weak sequencing, strict sequencing*

Interaction Operand) :

- Les opérateurs de choix et de boucle :
  - *Alternative :*

    désigne une alternative, équivalent à *SI...ALORS...SINON*
  - *Option*

    représente un comportement qui peut se produire ou pas. équivalent à *SI...ALORS*
  - *Break*

    représente des scenarii d'*exception*. Les interactions de ce fragment seront exécutées à la place des interactions décrites en dessous (interruption du flot "normal" des interactions).
  - *Loop*

    représente un ensemble d'interactions qui s'exécutent *en boucle*

    on peut utiliser une contrainte appelée *garde* pour indiquer le nombre de répétitions (minimum et maximum) ou bien une condition booléenne à respecter.

Interaction Operand) :

- *Parallel :*

  représente des interactions ayant lieu en *parallèle*.

- *Critical region :*

  désigne une *section critique* (Traitement atomique)

- *Ignore :*

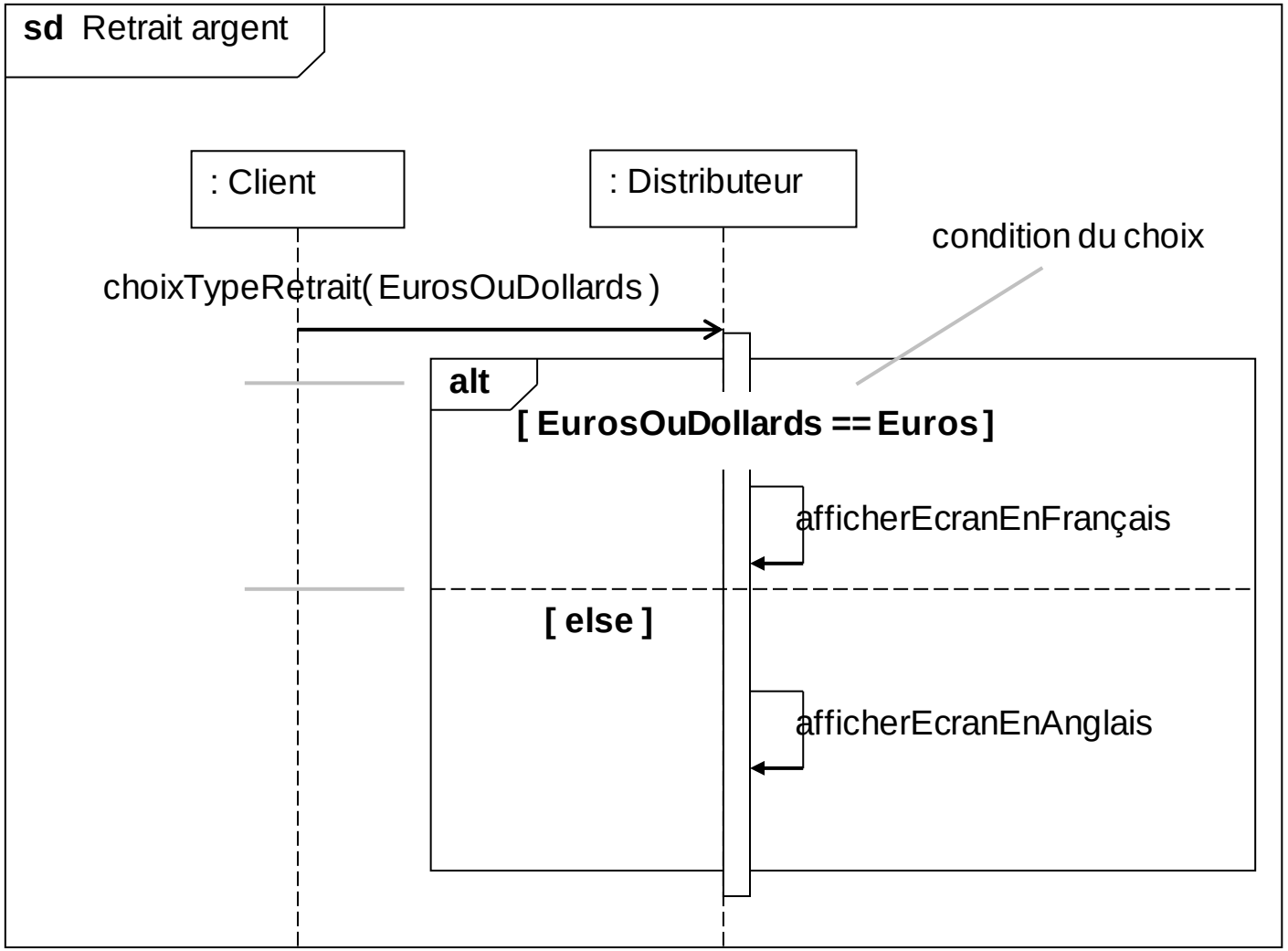
  indique qu'il existe des messages qui ne sont pas présents dans le fragment de cet opérateur (messages que l'on peut qualifier d'insignifiants, ex. confirmation de connexion)

- *Consider :*

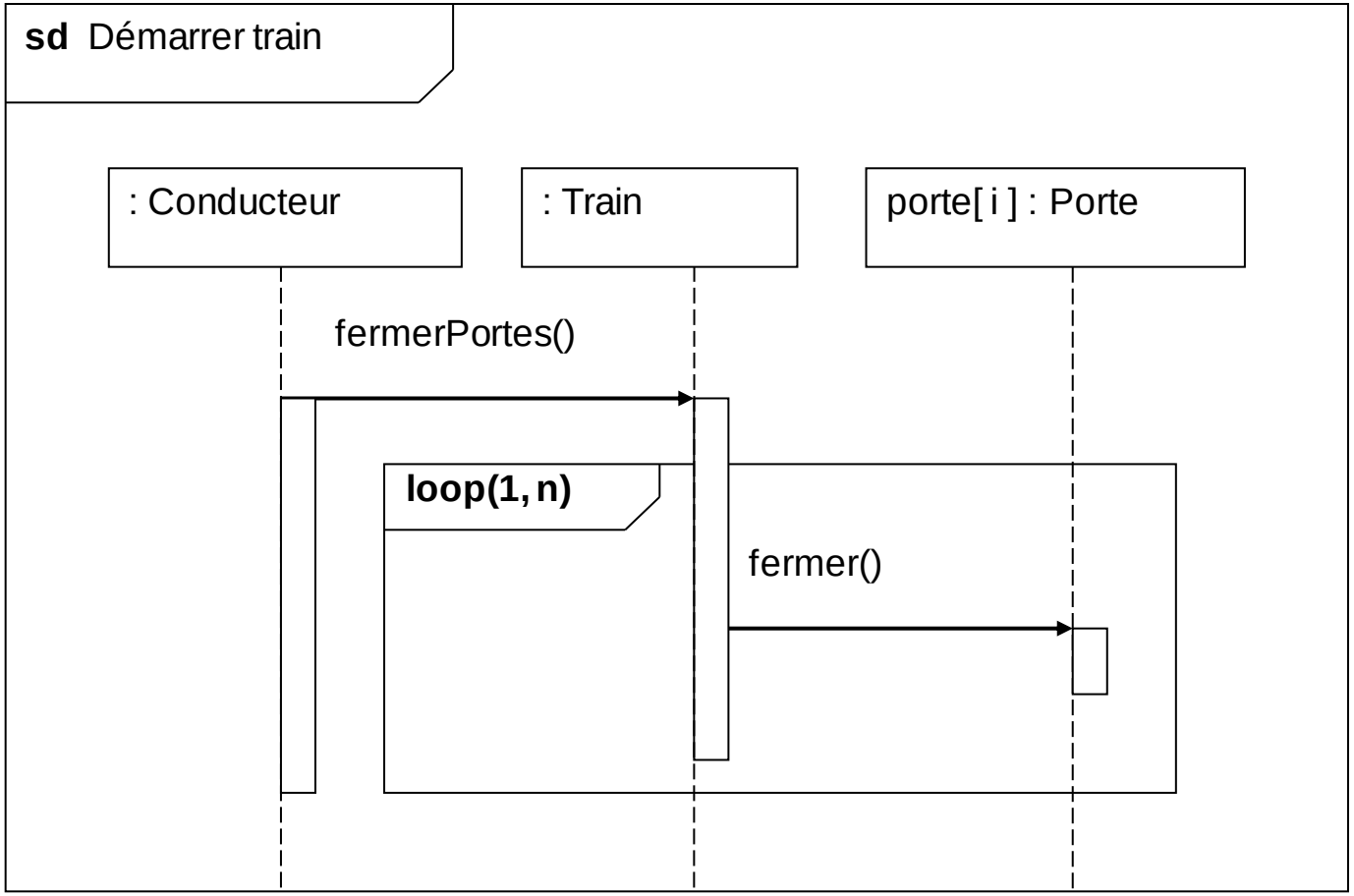
  désigne les interactions à prendre en compte dans la séquence.

- *Assertion :* indique que le fragment est une assertion
- *Negative :* désigne un ensemble d'interactions invalides

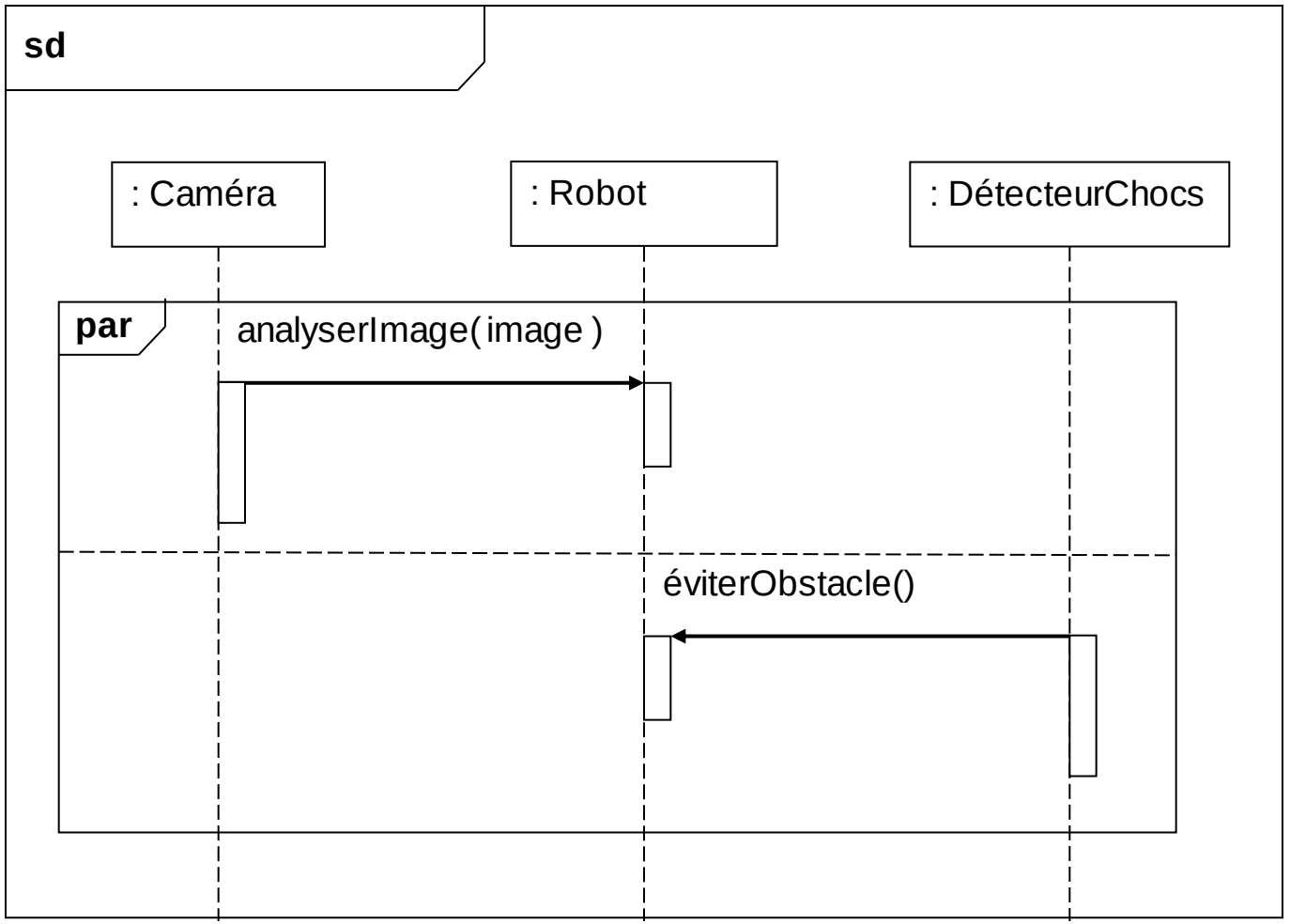
Diagrammes de séquences -



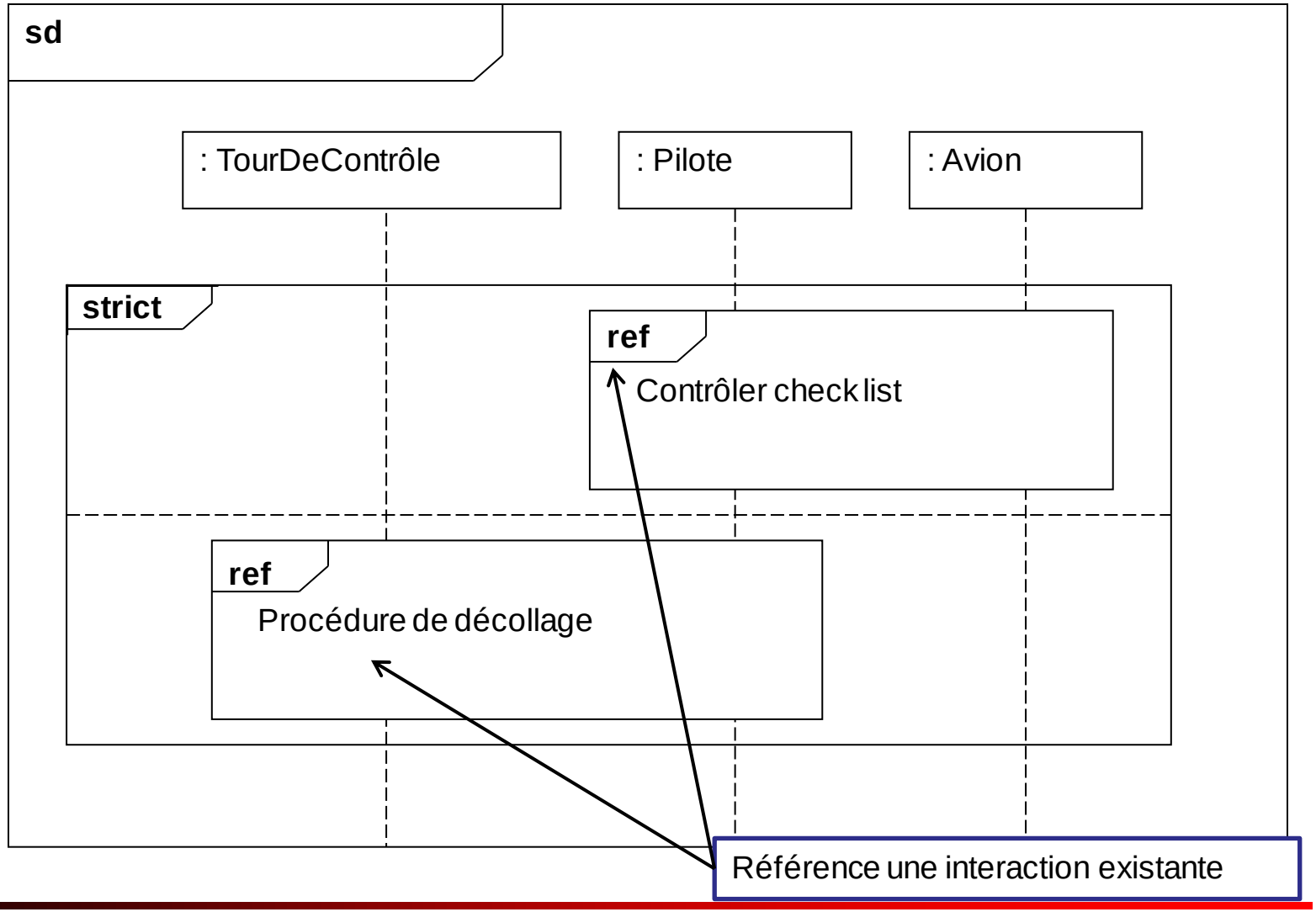
Diagrammes de séquences - les boucles



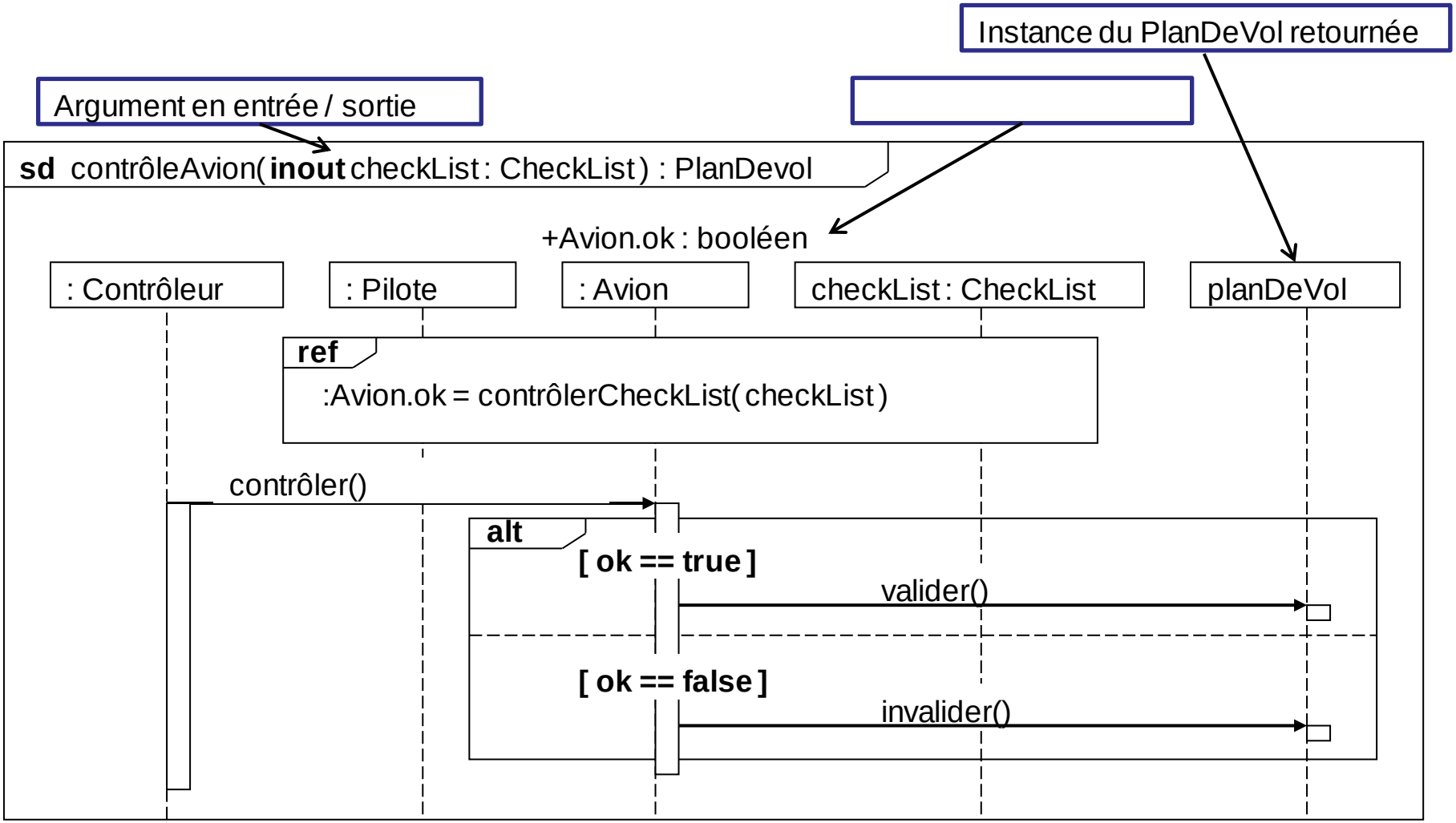
Diagrammes de séquences - Messages envoyés en parallèle



Diagrammes de séquences - Référencement de diagramme existant et ordre strict de déroulement de séquences



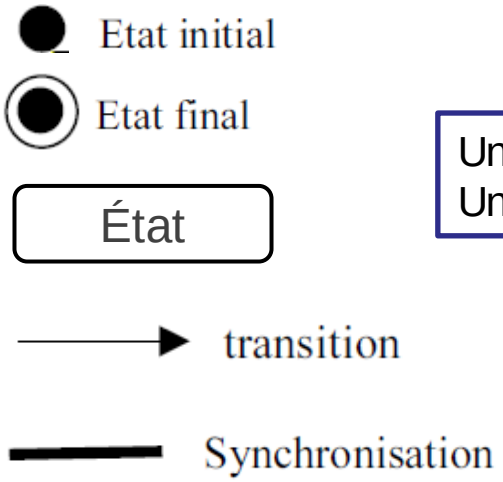
Diagrammes de séquences - Exemple contrôleur aérien



-transitions

- Permettent de décrire le comportements interne objet à automate à états finis
- Ce diagramme est utilisé pour montrer l'historique de la vie d'une classe donnée : des évènements font passer d'un état à un autre état qui doit déclencher des actions

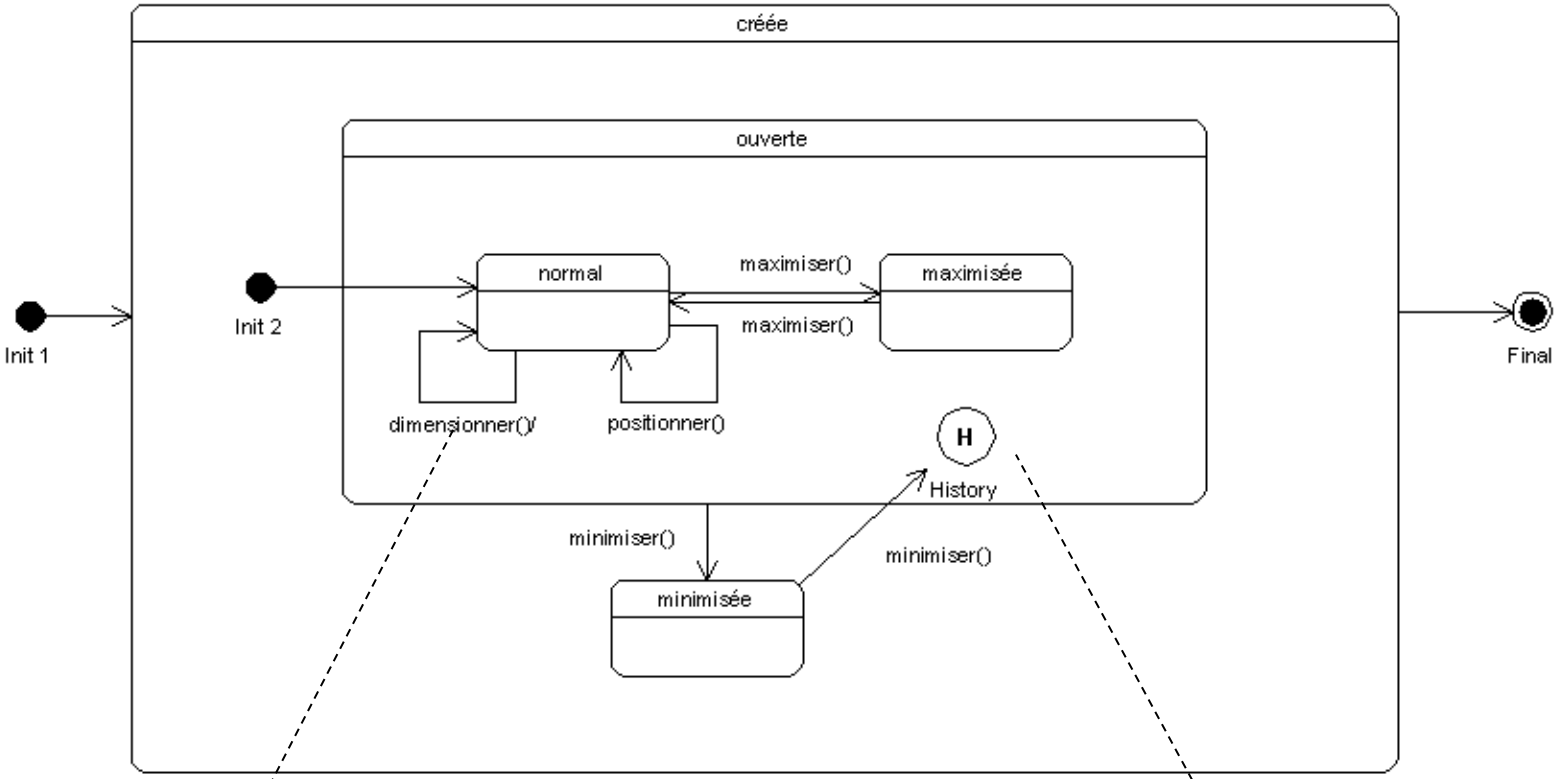
- Ils peuvent être utilisés aussi pour :
  - décrire le comportement interne cas
  - un sous-système,
  - une opération.



Une activité dure et peut être interrompue
Une action est immédiate et ne doit pas être interrompue

-transitions

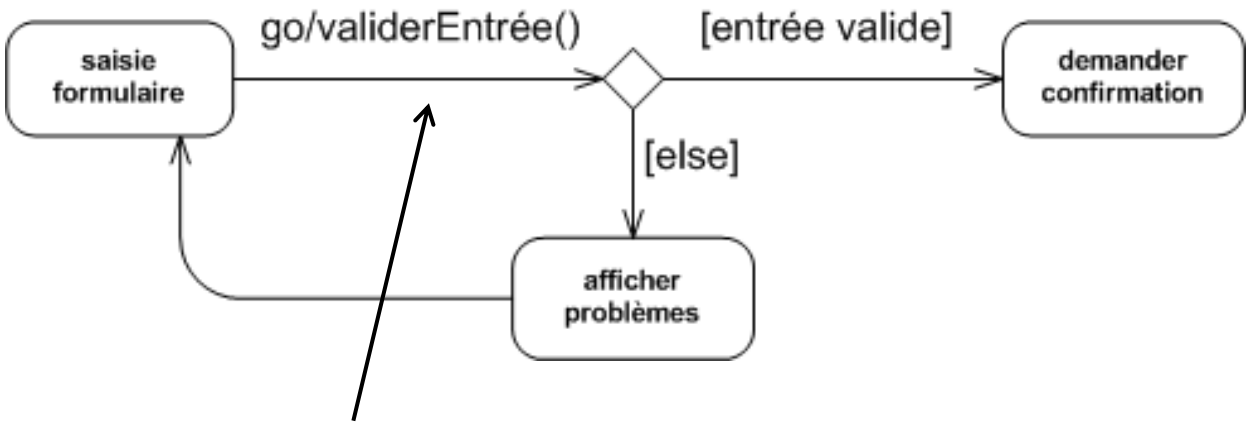
- Une fenêtre peut être dans trois états : réduite, normale, agrandie



Les états créée et ouverte sont appelés : « état composite »

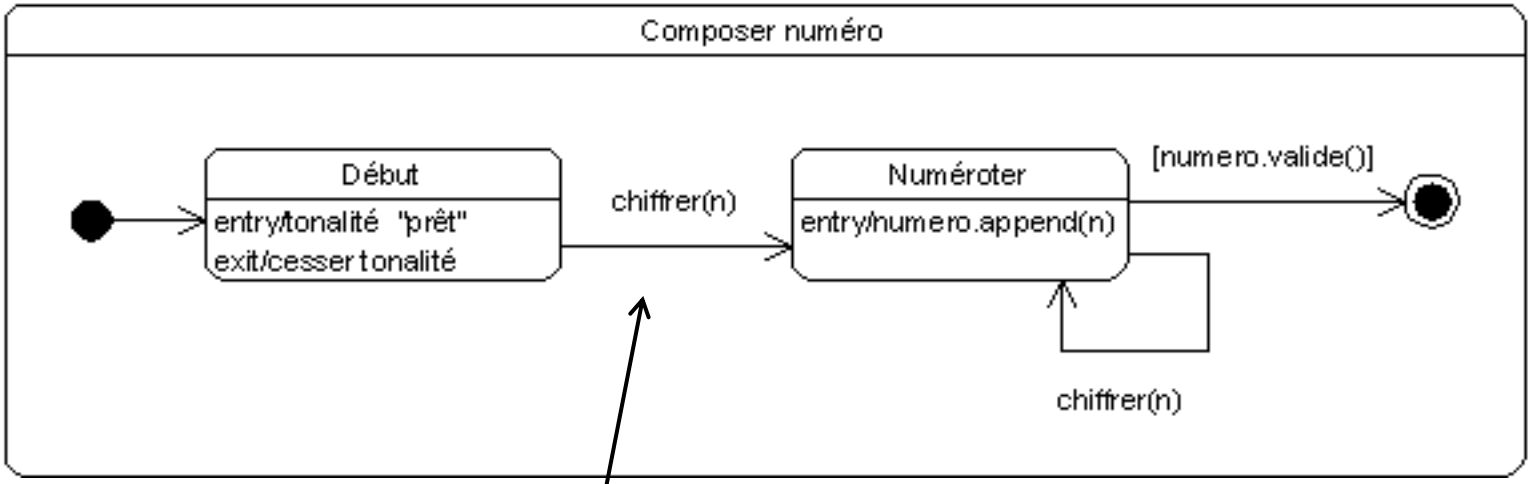
Le pseudo état *history* permet de retrouver à la fenêtre son état précédent

Exemple 2: formulaire en ligne



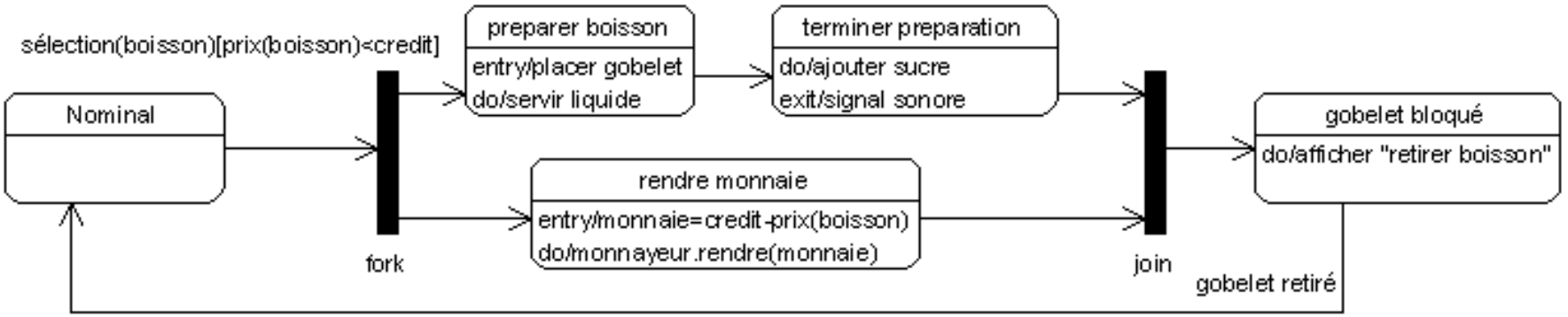
Point de choix dynamique:
La validation se fait sur le segment avant le point de choix

-transitions



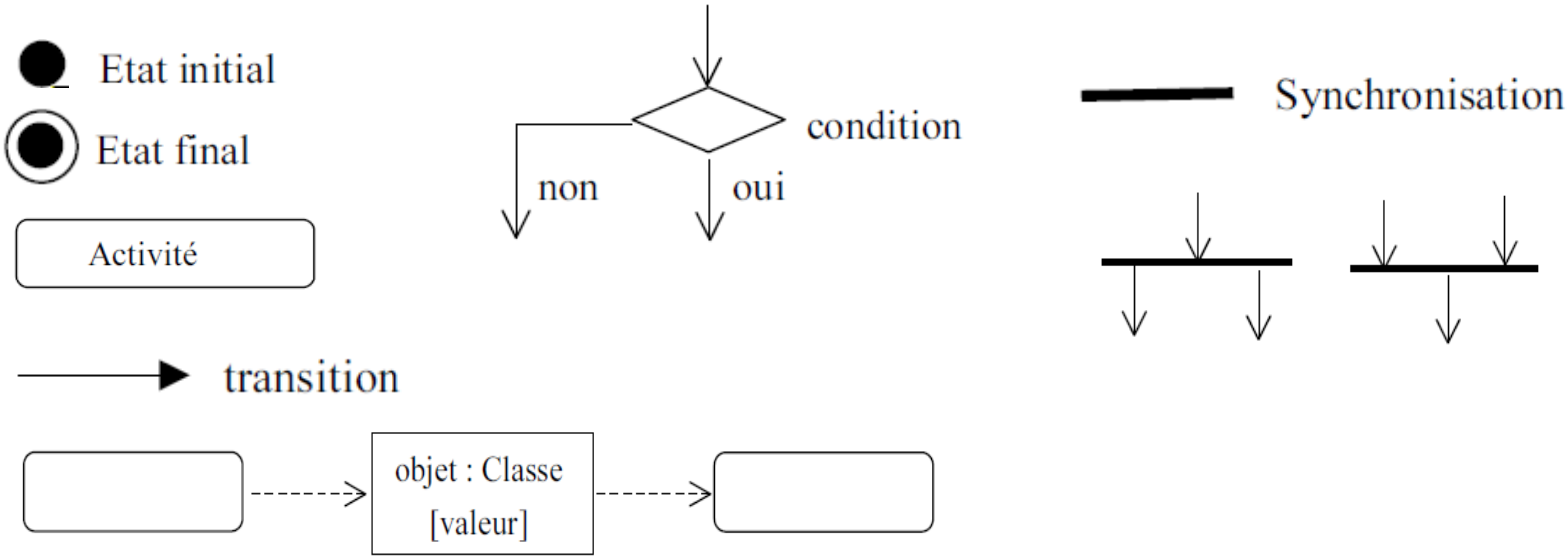
chiffrer

Exemple 4: transitions concurrentes

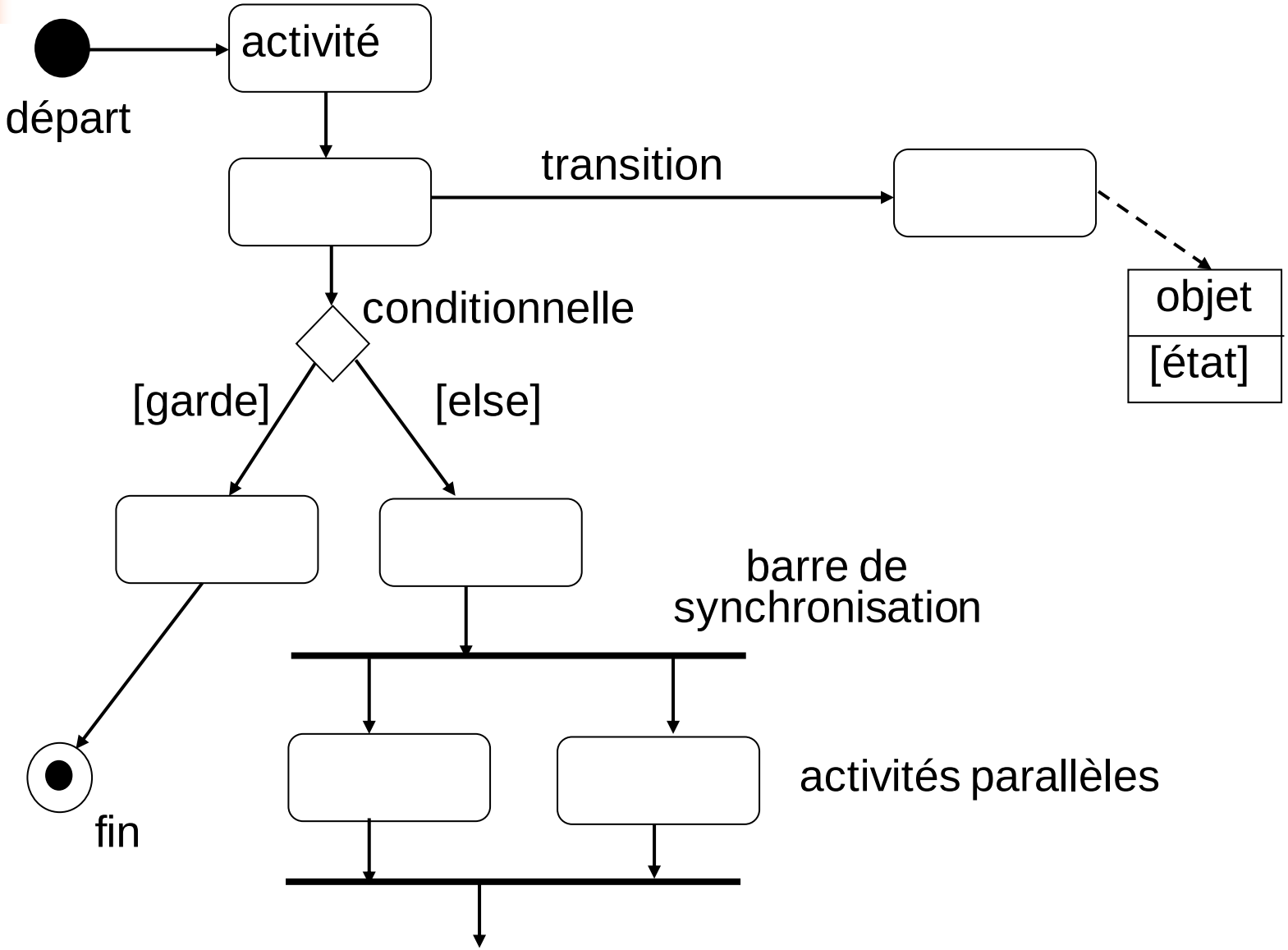


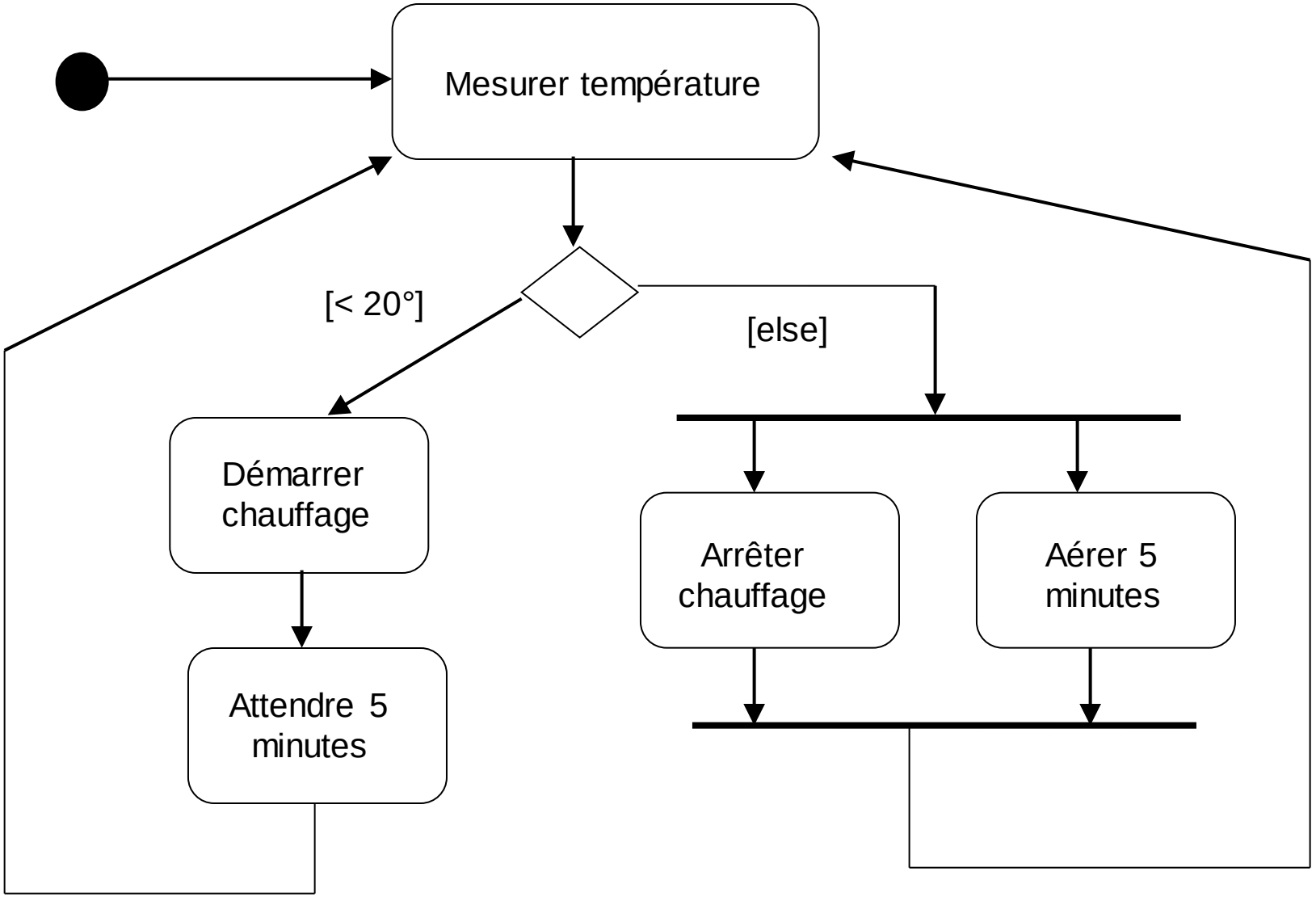
- Permettent de décrire le comportements interne objet à *automate à états finis*

- Ils peuvent être utilisés aussi pour :
  - Adaptés pour décrire tout traitements séquentiels,
  - ils servent à décrire des cas et des algorithmes complexes,
  - les traitements peuvent être décrits par des diagrammes .

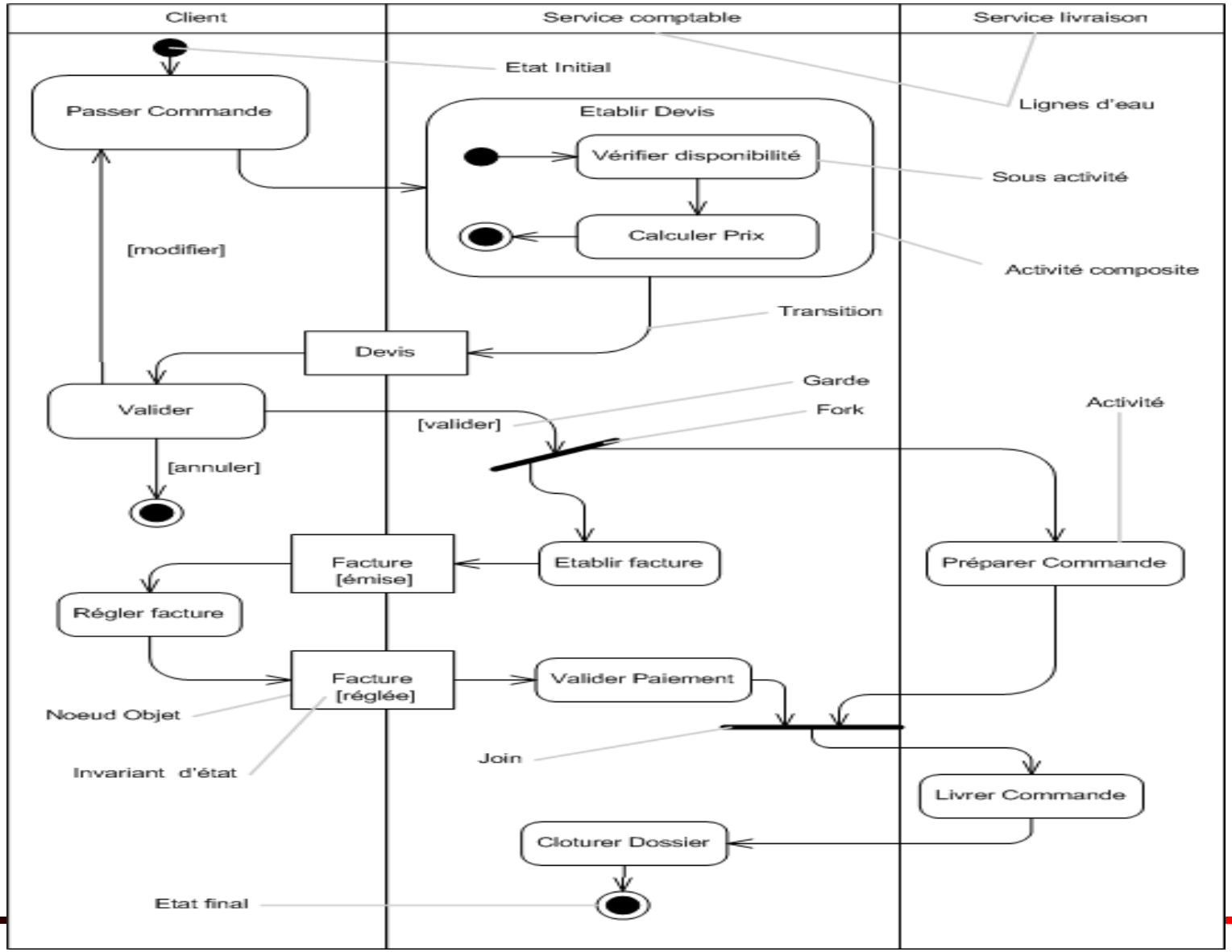


Transition à objet

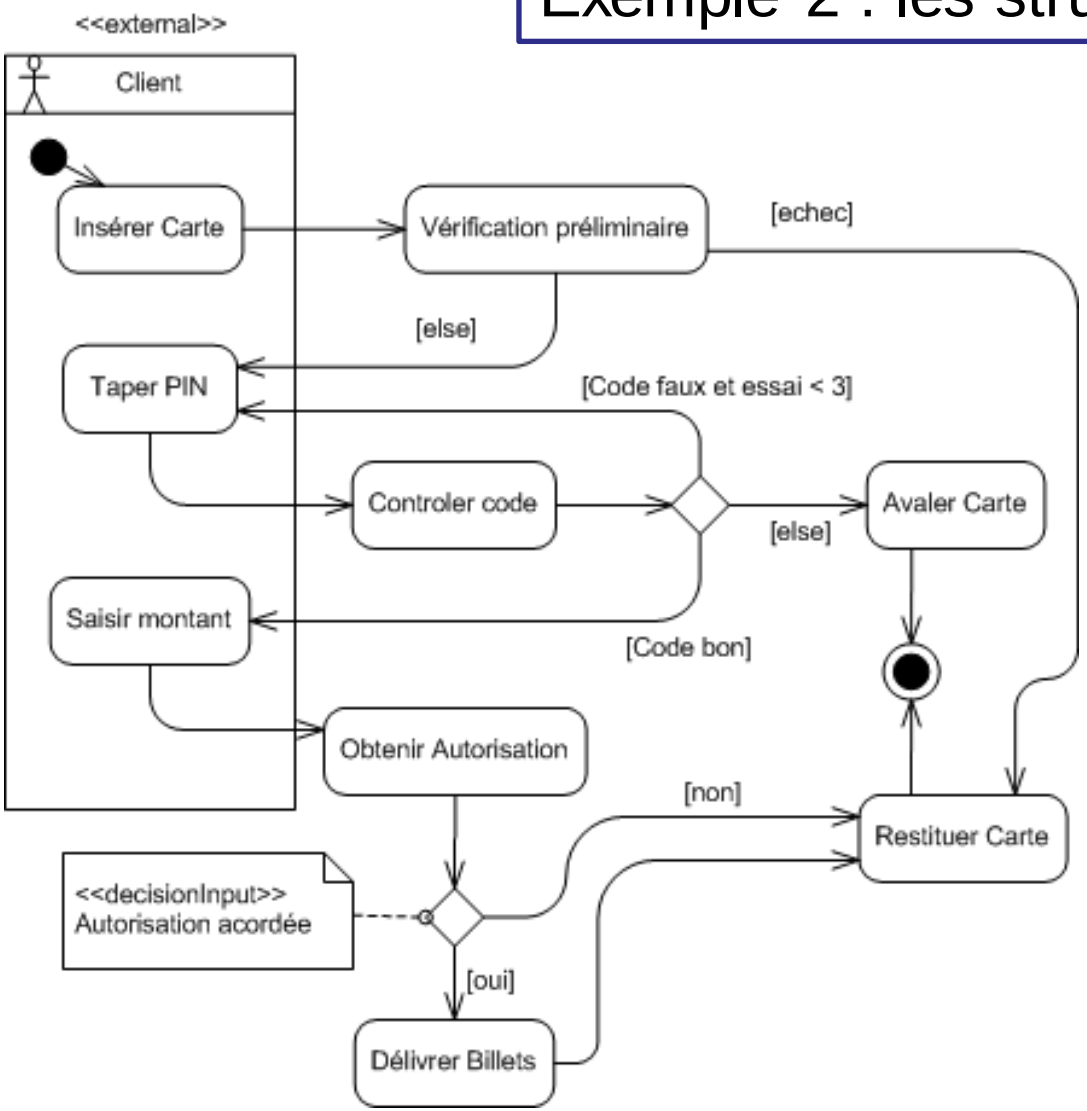

activité
départ
transition
objet
[état]
conditionnelle
[garde]
[else]
barre de
synchronisation
fin
activités parallèles


Mesurer température
[< 20°]
[else]
Démarrer chauffage
Attendre 5 minutes
Arrêter chauffage
Aérer 5 minutes

Passer Commande »



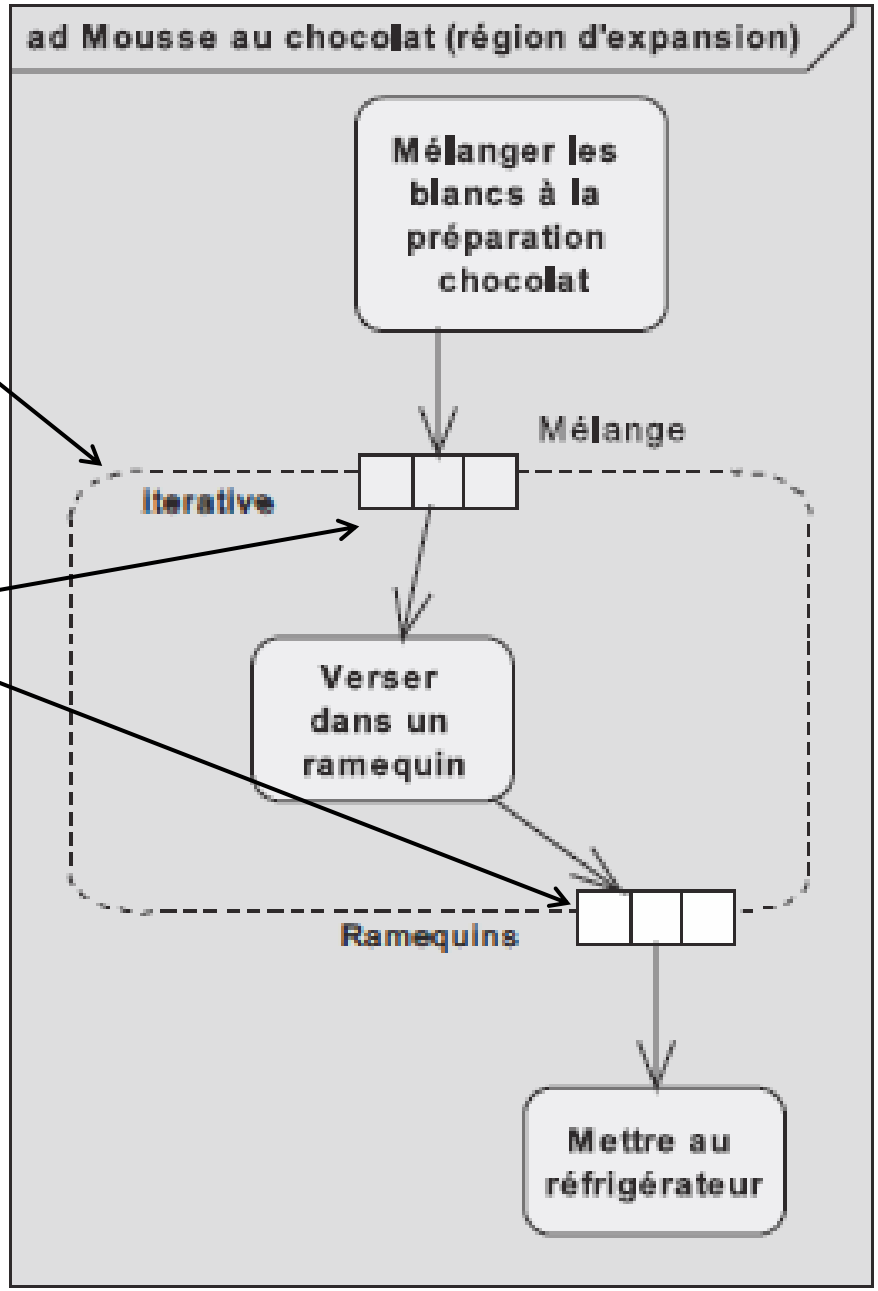
Exemple 2 : les structures de contrôle



RÉGION C

Une région est un structuré qui une fois pour chaque élément dans une collection .

Les entrées et sorties de la collection sont matérialisés sur la frontière de la boîte par des petits rectangles divisés en plusieurs compartiments.

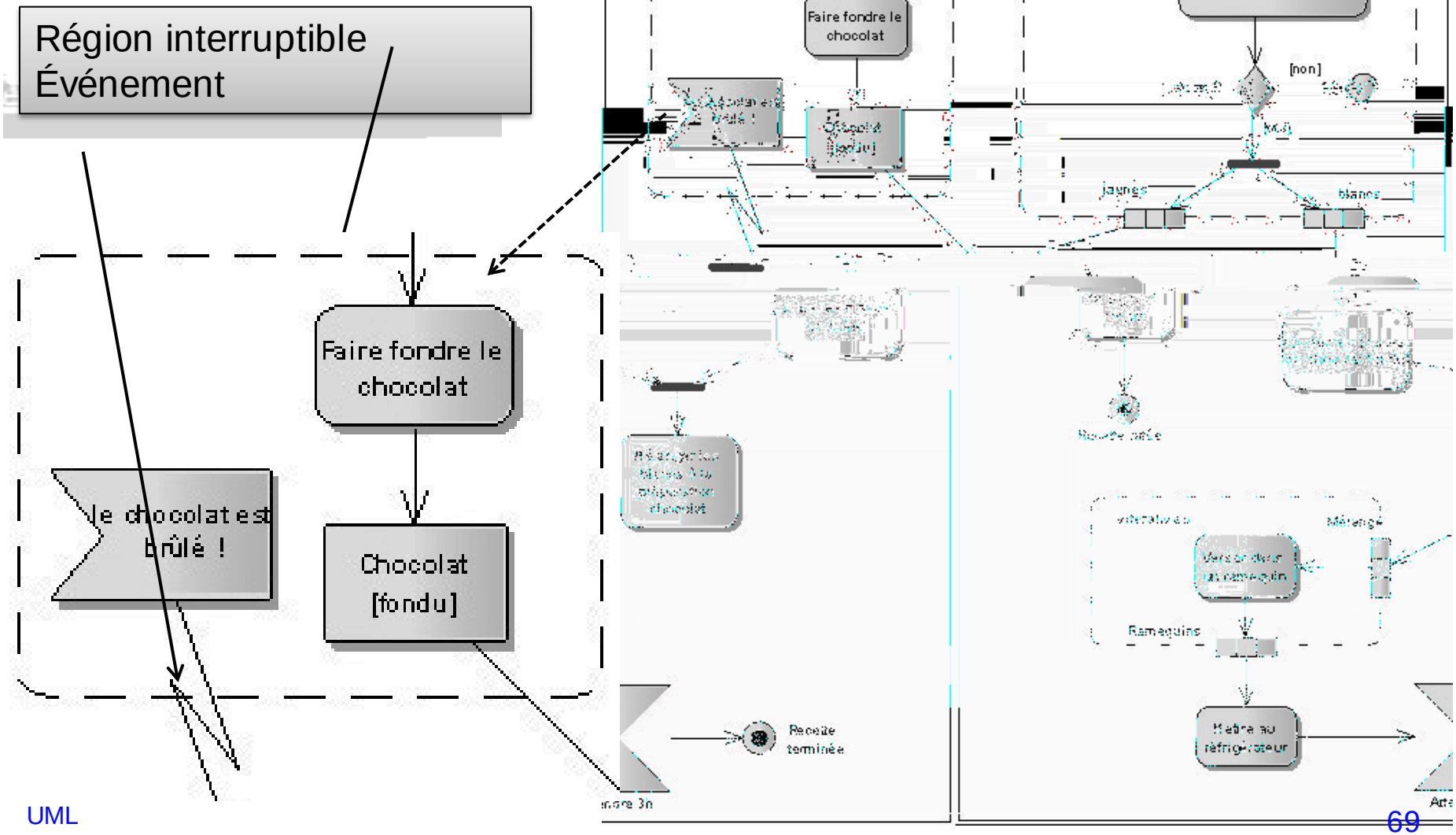


RÉGION INTERRUPTIBLE

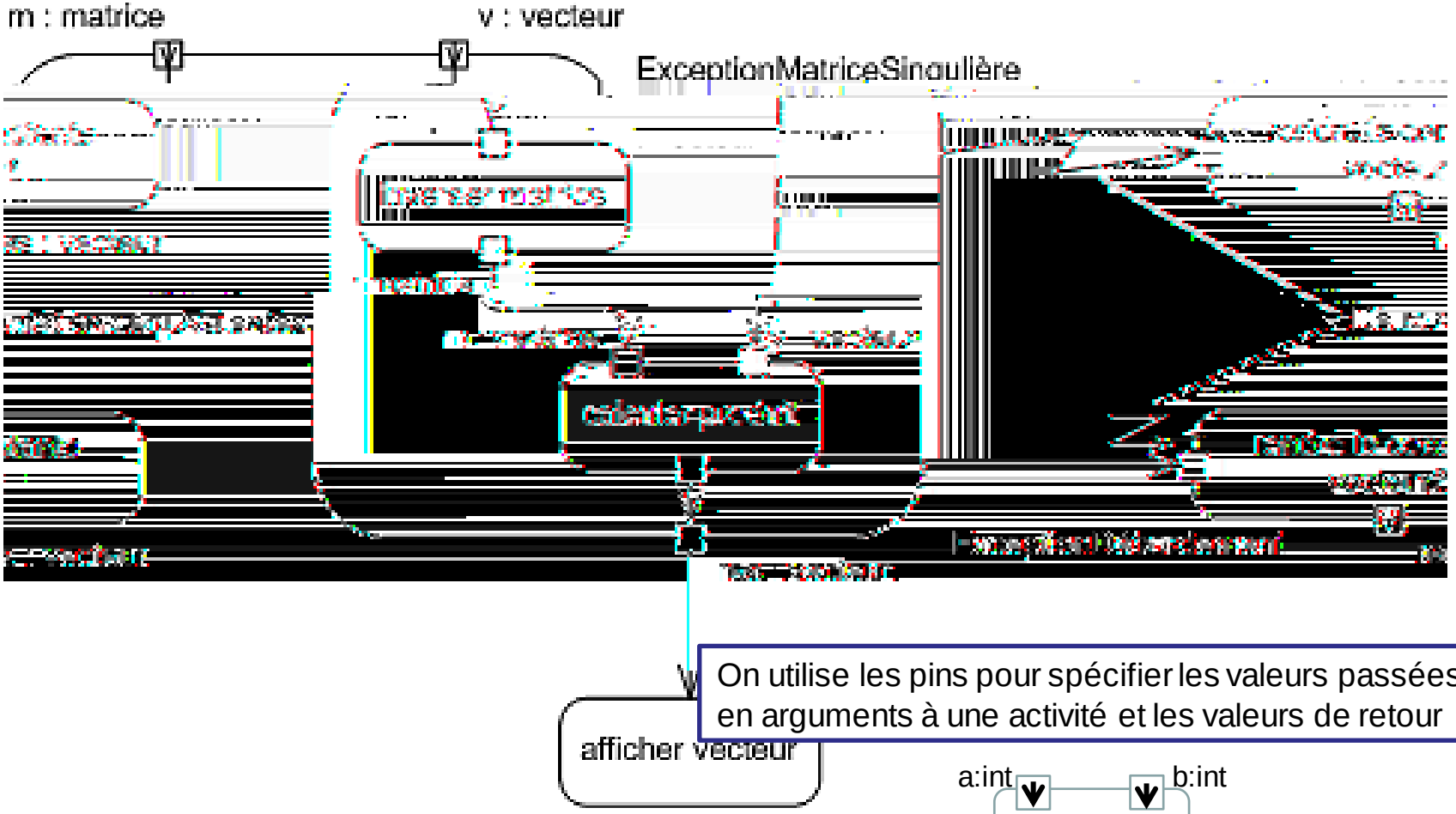
Une région interruptible est un ensemble de et au sein duquel se termine si désigné se produit.

Cet événement apparaît comme une flèche «éclair » partant de de la région interruptible

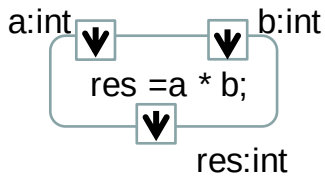
Diagrammes


Région interruptible
Événement
Faire fondre le chocolat
le chocolat est brûlé !
Chocolat [fondu]
act Mousse au chocolat (exception)
Casser le chocolat
Faire fondre le chocolat
oeufs
«iterative»
Casser l'oeuf et séparer le blanc du jaune
[non]
jaunes
blancs
Ramequins
Mettre au réfrigérateur
Recette terminée

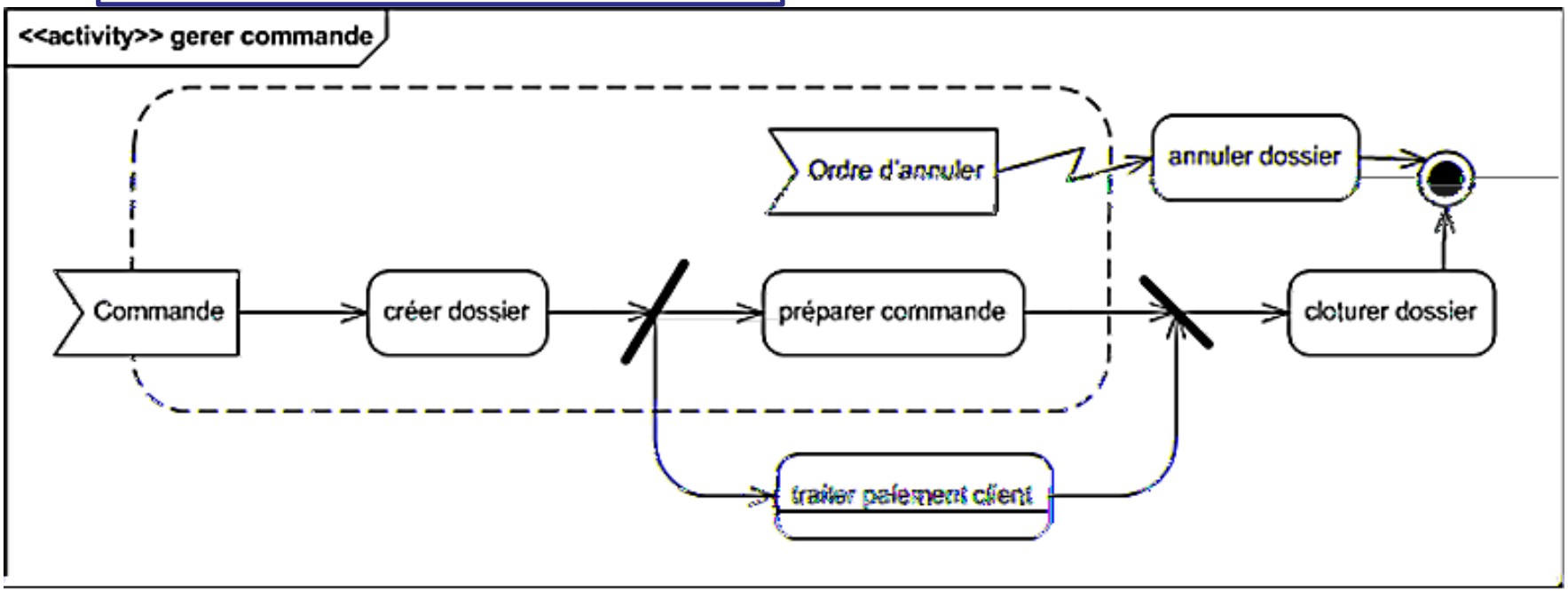
Exemple 3 : les pins et les exceptions



On utilise les pins pour spécifier les valeurs passées en arguments à une activité et les valeurs de retour



Exemple 4 : les actions



send signal

accept event

accept time event

- des diagrammes est de modéliser le comportement interne, classe, cas ou opération sous forme succession .

- Le diagramme d'activité est une variante des diagrammes d'états-transitions

- Le diagramme d'états-transitions permet de visualiser les automates d'états finis, du point de vue des états et des transitions

- Le diagramme d'état est plus concentré sur le cycle de vie de l'objet

Un G ¶ p W D W V O ¶ p Y R O X W L R Q G ¶ X Q G ¶ X Q H

± ±

Il G ¶ p O p P H Q W V

Un G ¶ D F W L Y L W p

Analyse du domaine

Analyse du domaine

- Trouver les classes du domaine,
- Trouver les associations entre les classes,
- Trouver les attributs des classes,
- Organiser et simplifier le diagramme en utilisant
- Construire le diagramme des classes
- Itérer et affiner le modèle.

Modèle des états du domaine

- Identifier des classes du domaine ayant des états complexes,
- Définir les états,
- Trouver les événements qui vont engendrer des transitions entre états,
- Construire les diagrammes

Analyse

Modèle des interactions

- Déterminer les limites du système,
- Trouver les acteurs
- Trouver les cas
- Construire le diagramme des cas ,
- Préparer les scénarios pour décrire les cas à de diagrammes de séquence.
- Ajouter des séquences alternatives et des séquences aux scénarios.

Confrontation des deux modèles

Reprendre les diagrammes de séquence et en déduire des opérations à ajouter dans les classes du domaine.

Conseils de modélisation

Diagrammes de

Si vous voulez décrire la fonctionnalité d'un système (ou d'une partie d'un système)

Diagrammes d'états

Si vous êtes intéressés par un objet sur plusieurs

Diagrammes d'activité

Si vous êtes intéressés par de nombreux objets sur plusieurs cas

Diagrammes de séquence

Quand vous voulez clarifier et explorer des individuels impliquant plusieurs objets

Diagrammes de collaboration (ou diagramme de séquence)

Si vous voulez décrire comment les objets collaborent

Diagrammes de classe

Quand vous êtes intéressés dans la structure d'un système

Outils de modélisation

- Ils existent plusieurs outils gratuits:
  - ArgoUML (Windows/Mac/Linux)
  - StarUML (Windows)
  - Papyrus UML : basé sur le plugin UML2 de Eclipse (Windows/Mac/Linux)
  - BoUML (Windows/Mac/Linux)
  -
- payants:
  - Rational Rose (leader Mondial en outil de Modélisation UML IBM) - Windows
  - Poseidon (Gentleware) - Windows/Mac/Linux
  - Modelio