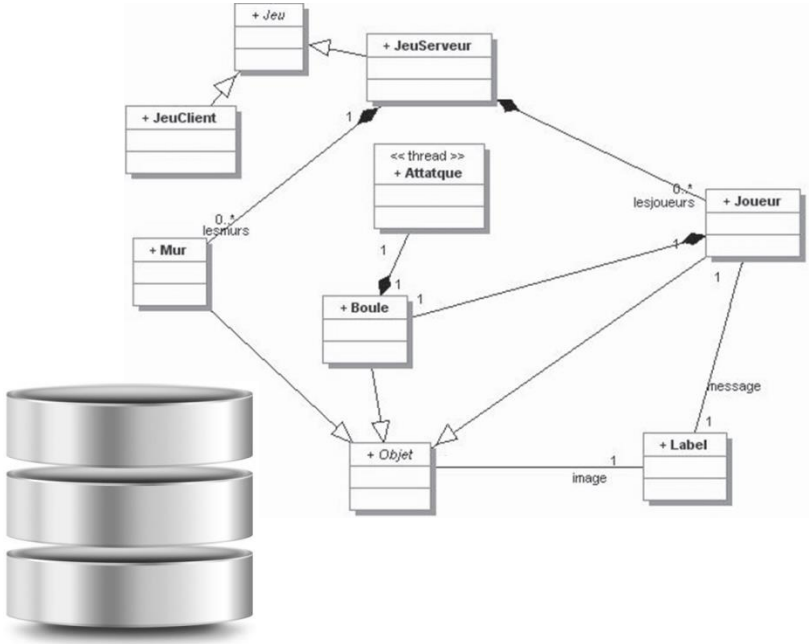
introduction à UML

Unified Modeling Language



Quelle méthode choisir ?

Face à un problème d'informatisation, plusieurs approches sont possibles pour la modélisation, dont les 2 les plus connues actuellement :

Méthode Merise

Méthode offrant une succession de modèles adaptés pour :

→ les applications fonctionnelles ou objet

→ les bases de données relationnelles

(Modélisation des acteurs, flux, données, traitements...)

1970 (environ) : naissance de Merise

Langage UML

Langage offrant un ensemble de diagrammes adaptés pour :

→ les applications objet

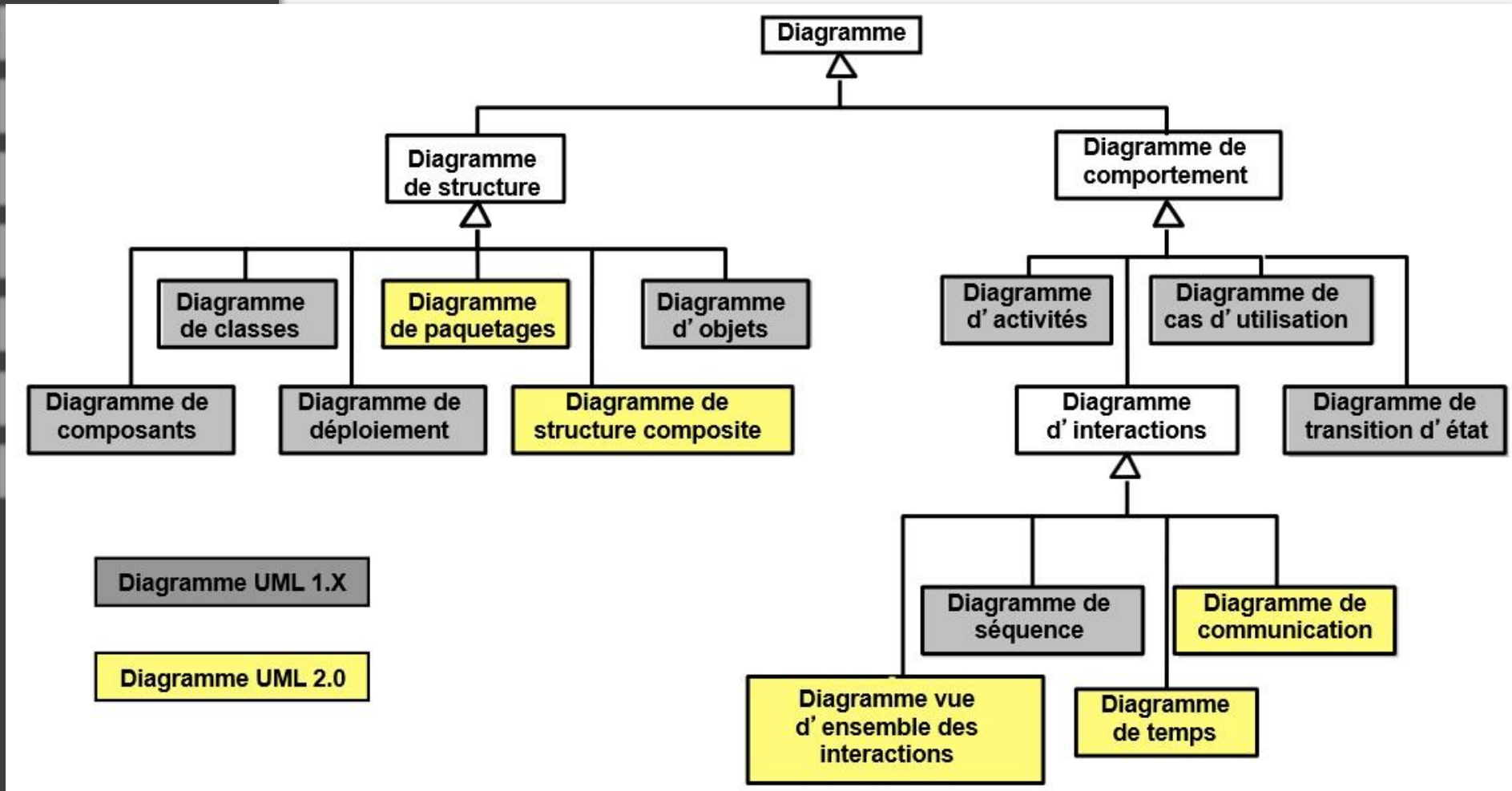
→ les bases de données relationnelles ou objet

(modélisation des cas d'utilisation, des classes, des états...)

1997 : naissance d'UML (9 diagrammes)

2006 : UML2 (13 diagrammes)

13 diagrammes



Schématisation de l'expression des besoins

Ce diagramme se présente sous 2 formes :
→ schéma regroupant les activités d'un acteur
→ tableau détaillant les actions liées à une activité

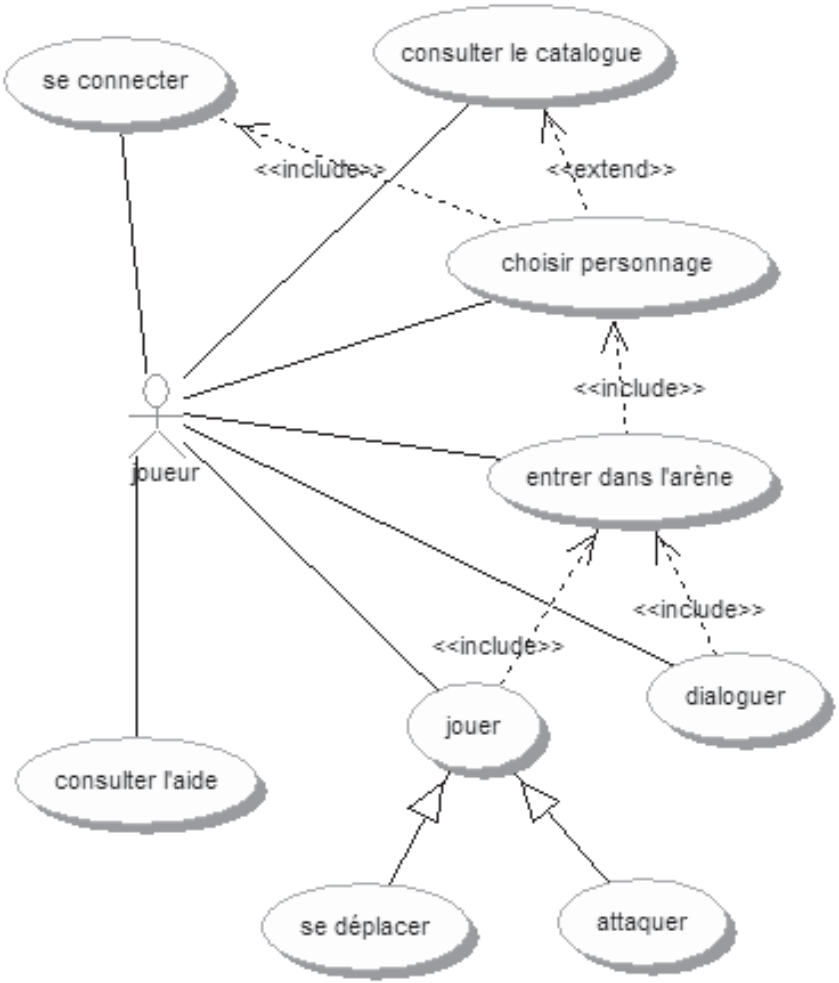
Un joueur se connecte.

Une fois connecté, il choisit un personnage après avoir éventuellement consulté le catalogue des personnages.

Il peut alors entrer dans l'arène et jouer (se déplacer, attaquer les adversaires).

Il peut aussi, pendant le jeu, dialoguer avec les autres joueurs (chat).

À tout moment (connecté ou non), le joueur peut consulter l'aide du jeu.



cas d'utilisation

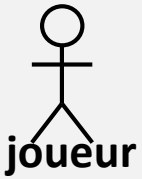
Le diagramme de cas d'utilisation correspond à :

→ une représentation globale (schéma) ou détaillée (tableau) des activités d'un acteur (vision du système par l'acteur)

→ l'ensemble des opérations possibles entre cet acteur et le système

→ une base de travail pour construire les interfaces graphiques

syntaxe

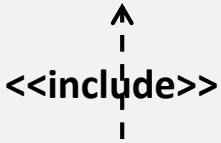


acteur

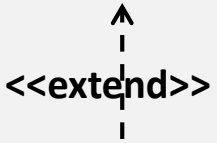
spécialisation entre acteurs

cas d'utilisation

association de communication



inclusion obligatoire



inclusion optionnelle

Tableau des opérations d'un cas d'utilisation

Le tableau détaille les opérations d'un cas d'utilisation (un cercle).
Il sert de base à la construction des interfaces.

Cas d'utilisation	Se connecter
Acteur	joueur
Év. déclencheur	néant
Intérêts	Accéder au serveur afin d'être authentifié et accéder au jeu
Pré-conditions	Serveur actif
Post-conditions	Connexion établie
Scénario nominal	1. l'utilisateur saisit l'adresse IP du serveur 2. l'utilisateur demande une connexion au serveur 3. le serveur répond
Extensions	non
Contraintes	2a. L'adresse IP n'est pas remplie : aller en 1 3a. Le serveur retourne un message d'erreur : aller en 1 3b. Le serveur ne répond pas : aller en 1

Exercice :

Sur un site commercial, un utilisateur peut sélectionner des produits éventuellement après avoir lu la fiche technique de chaque produit. Il peut ensuite visualiser son panier. S'il décide de valider son panier, il doit donner ses coordonnées et il peut alors soit choisir de payer en ligne, soit de payer à la réception. Dessiner le diagramme de cas d'utilisation correspondant.

Schématisation de classes et leurs liens

Ce diagramme représente l'organisation des classes de l'application.

Le but du jeu est de permettre à plusieurs joueurs de se connecter à un serveur et de jouer ensemble. Chaque joueur choisit un personnage, donne un pseudo et rentre dans l'arène. L'arène contient des murs. Le joueur peut se déplacer entre les murs, il peut tirer des boules pour toucher les adversaires.

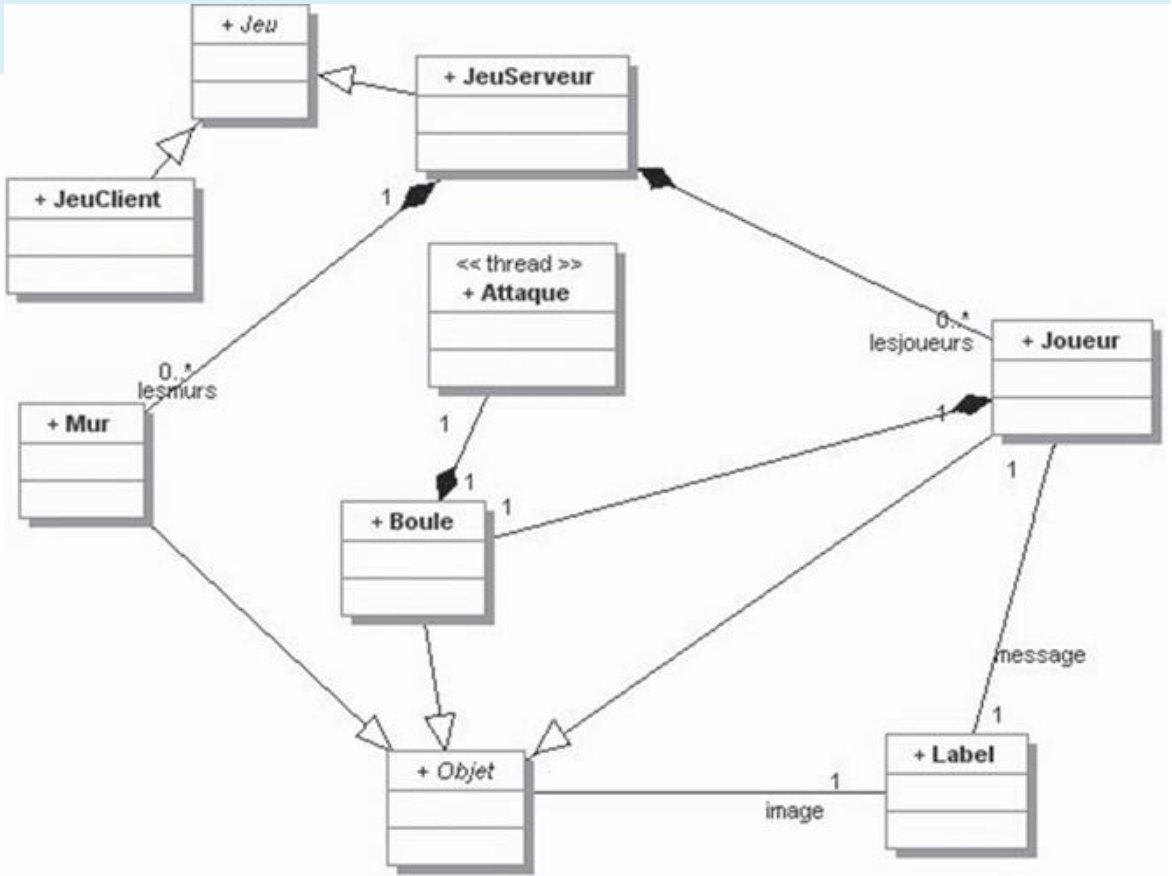
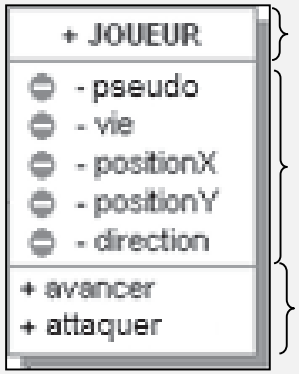


diagramme de classes

Le diagramme de classes est un schéma :
→ représentatif des classes et de leurs liens
→ présentant l'organisation des données et traitements de l'application

syntaxe

classe



nom
(italique : abstraite)

propriétés et portée
(pas les propriétés objet)

méthodes et portée
(généralement pas le constructeur)

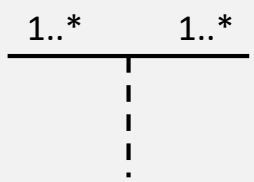
- : privé **+** : public **#** : protégé

spécialisation

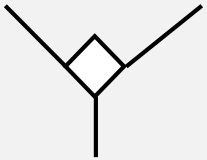
1
0..1
0..*
1..*

multiplicités
(inversées par rapport à Merise)

association non porteuse



association porteuse



association à + de 2 liens

agrégation

composition
(agrégation forte)

diagramme de classes

Exercice :

Construire le diagramme de classes correspondant.

Classe Commune
Privé
  nomCom : Chaîne
  lesSecteurs : Collection de Secteur
Public
  Commune(nomCom : Chaîne)
  volumeVannes() : Entier

Classe Secteur
Privé
  nomSecteur : Chaîne
  laCommune : Commune
  lesBranchements : Collection
        de Branchement
Public
  Secteur(nomSecteur : Chaîne,
        laCommune : Commune)
  getNomSecteur() : Chaîne

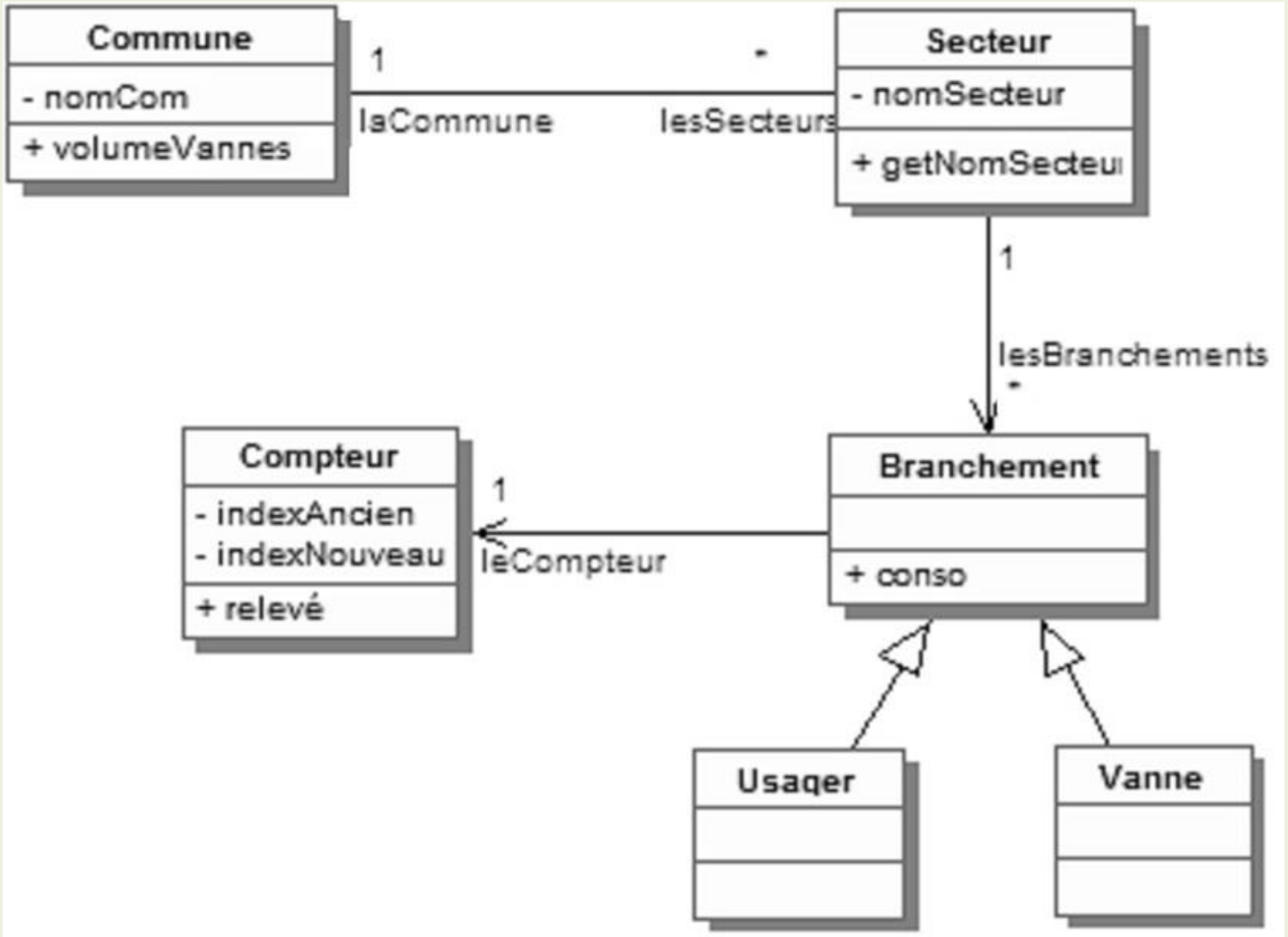
Classe Compteur
Privé
  indexAncien : Entier
  indexNouveau : Entier
Public
  Fonction relevé() : Entier

Classe Branchement
Privé
  leCompteur : Compteur
Public
  conso() : Entier

Classe Usager hérite de Branchement
  ...
Classe Vanne hérite de Branchement
  ...

Exercice :

Construire le diagramme de classes correspondant.



Schématisation de la vie d'une instance

Ce diagramme représente les états pris par un objet au cours de sa vie.

Une fois le joueur créé, le choix du personnage va permettre de valoriser les propriétés concernées. Le joueur entre dans l'arène et peut ainsi accéder aux méthodes qui vont lui permettre de se déplacer et/ou d'attaquer. Le joueur peut aussi être touché ce qui va diminuer sa vie.
Si la vie arrive à 0, le joueur meurt. L'objet est détruit.

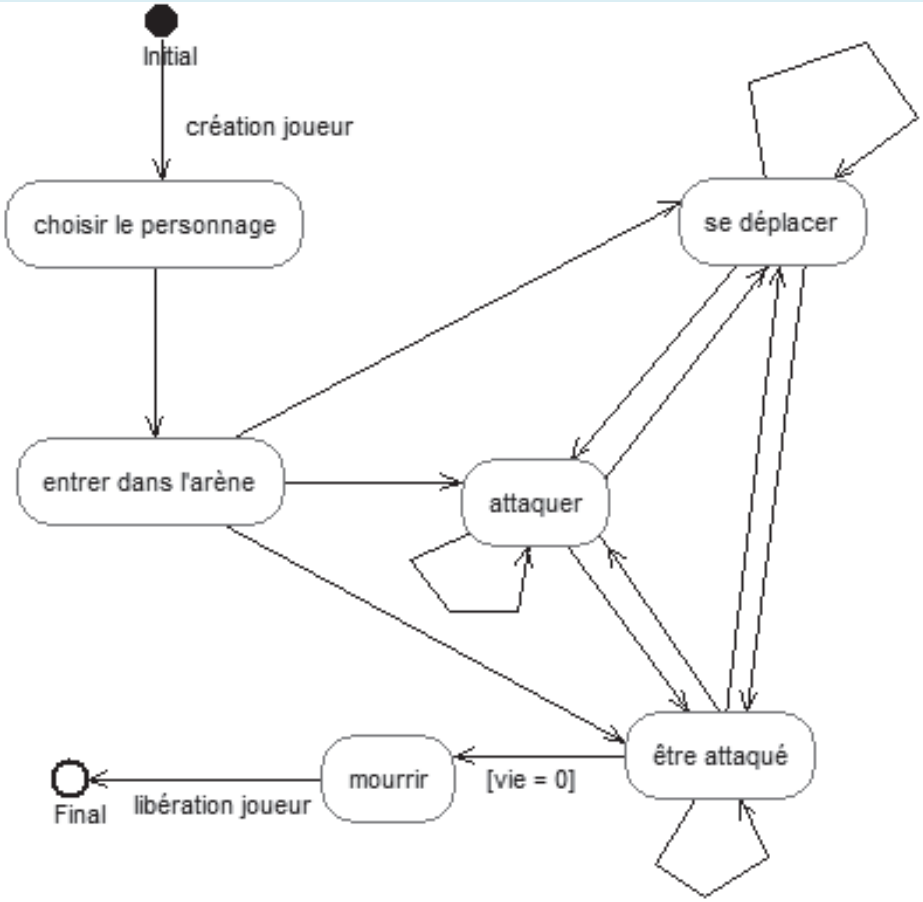


diagramme d'états

Le diagramme d'états est :
→ un schéma qui ne concerne qu'un objet précis (une instance)
→ une représentation des différents états possibles de cet objet
→ une intégration de la notion de temps (étapes de vie d'un objet)

syntaxe

état initial

état final

état transitoire

transition

création

action

[vie=0]

condition

diagramme d'états

Exercice :

Une intervention peut être dans l'état « À faire », « Planifiée », « En cours », « Interrompue », « Annulée », « Terminée », « Facturée » ou « Archivée ». Seule une intervention en cours peut être interrompue. Une fois la cause de l'interruption résolue, elle sera reprise et passera de l'état « Interrompue » à l'état « En cours ». Seule une opération planifiée peut être annulée. Une intervention annulée ne sera pas facturée mais sera archivée.

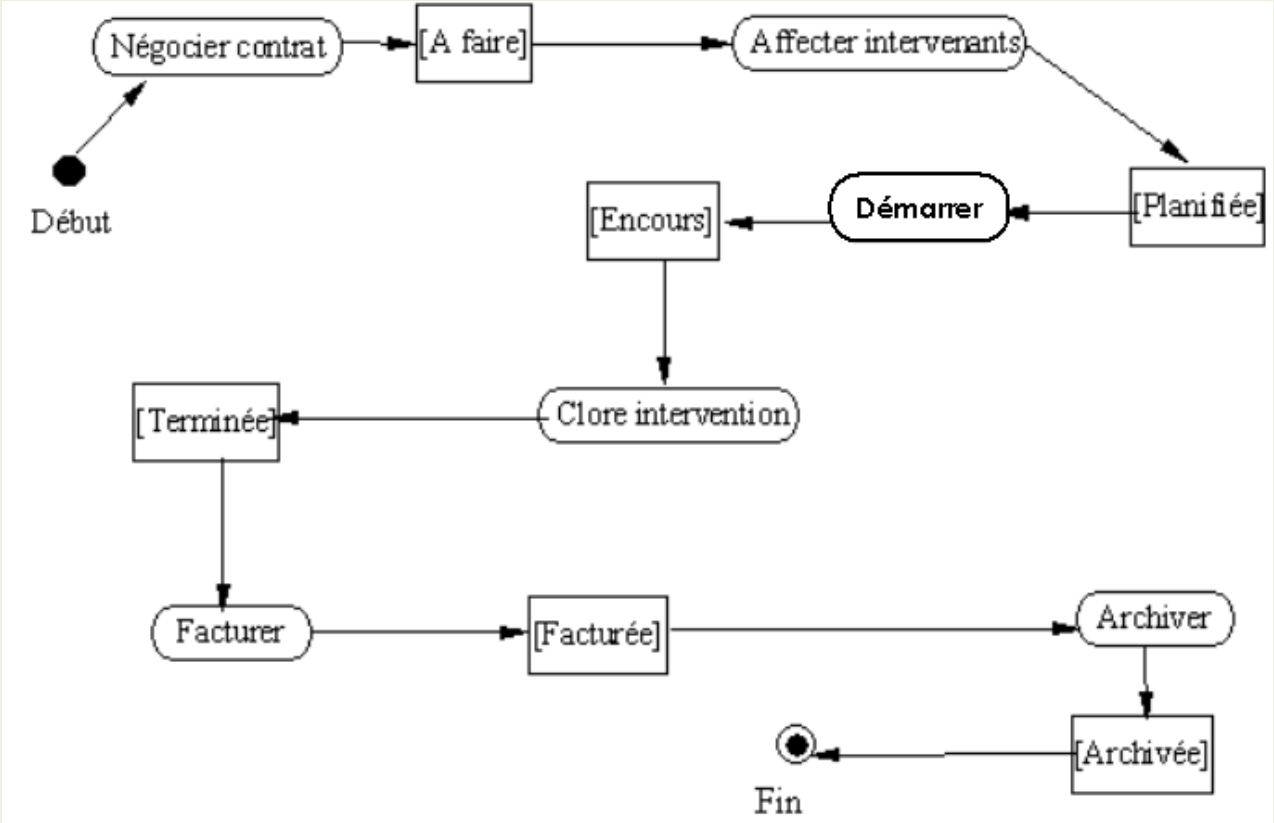
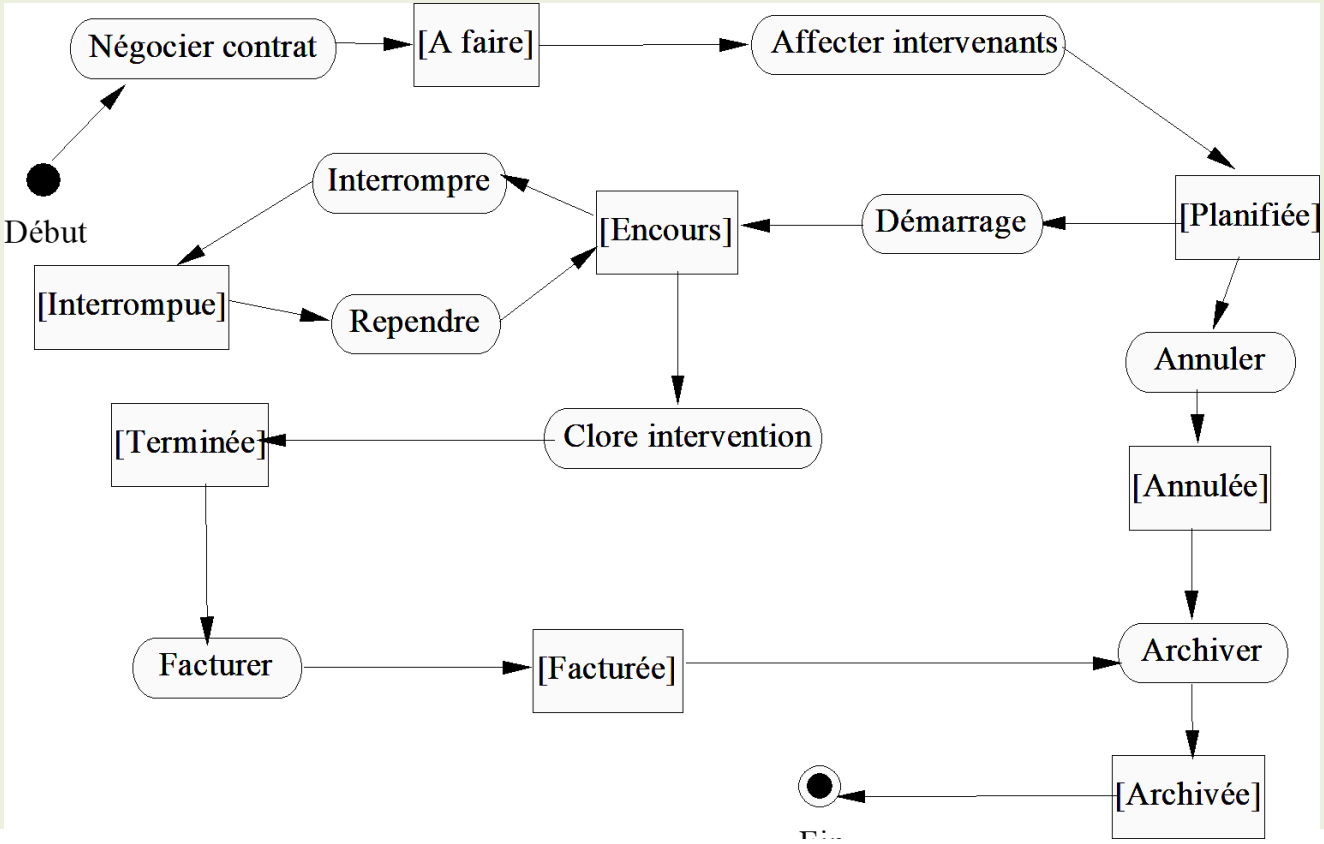


diagramme d'états

Exercice :

Une intervention peut être dans l'état « À faire », « Planifiée », « En cours », « Interrompue », « Annulée », « Terminée », « Facturée » ou « Archivée ». Seule une intervention en cours peut être interrompue. Une fois la cause de l'interruption résolue, elle sera reprise et passera de l'état « Interrompue » à l'état « En cours ». Seule une opération planifiée peut être annulée. Une intervention annulée ne sera pas facturée mais sera archivée.



Schématisation de messages entre les objets

Quand l'utilisateur se connecte, l'objet joueur et sa boule sont créés côté serveur. Quand une flèche est utilisée, l'objet joueur demande au serveur de se déplacer. Le serveur retourne l'affichage de la nouvelle position. Quand la touche espace est utilisée, l'objet joueur demande au serveur d'attaquer. Le serveur retourne l'affichage de la boule qui est lancée. Le serveur contrôle la vie des joueurs et fait mourir ceux qui sont à 0.

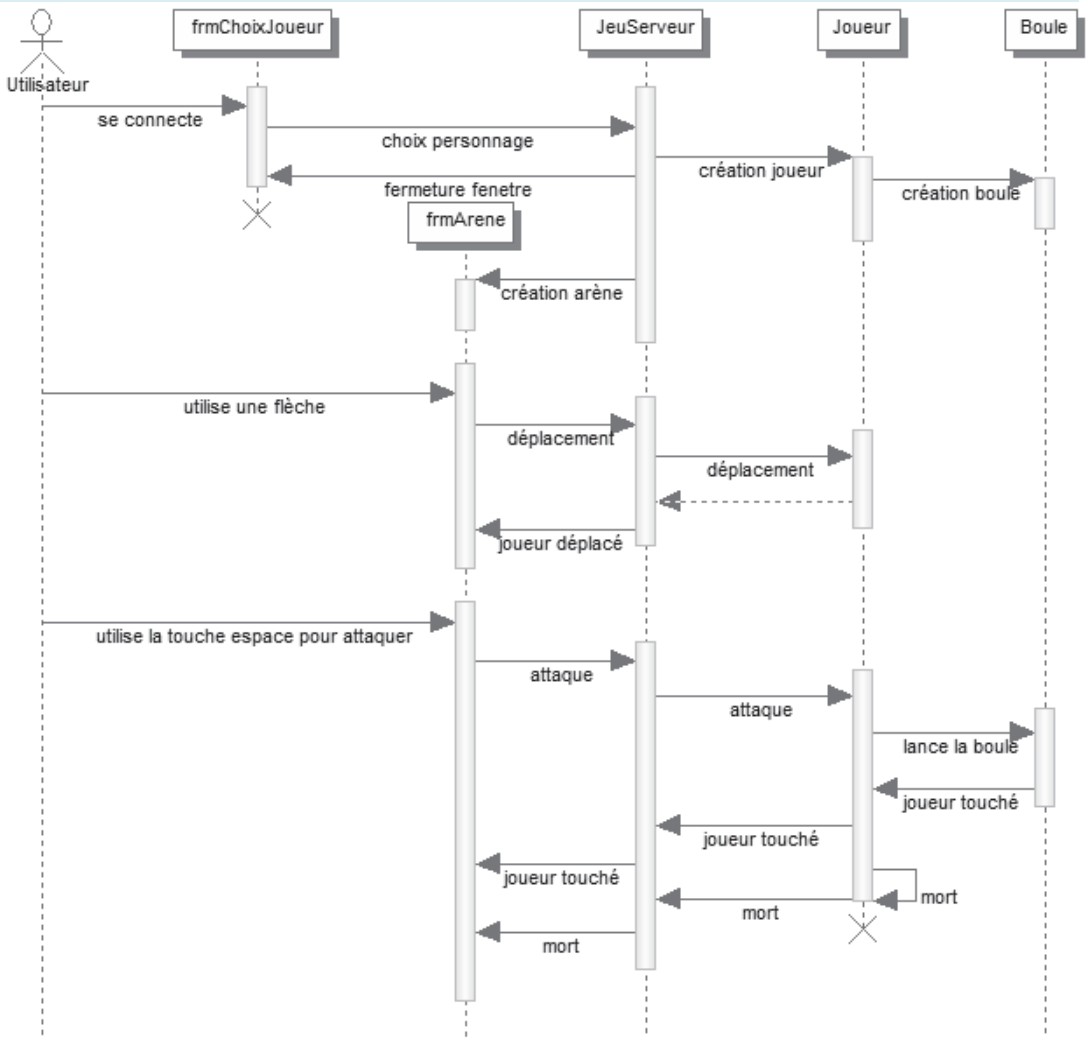
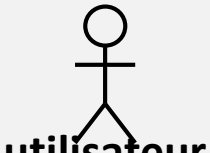


diagramme de séquences

Le diagramme de séquences est :
→ la représentation temporelle des messages entre objets
→ le cycle de vie d'un ensemble d'objets liés dans un cas d'utilisation
→ la représentation des méthodes qui agissent entre objets

syntaxe



acteur

ligne de vie

zone temporelle de sollicitation de l'objet

:Joueur

objet

message
(méthode)

réponse

destruction

diagramme de séquences

Le diagramme de séquences permet de lister les méthodes publiques, leur classe et leur rôle.

Voici un exemple de méthodes correspondant aux premiers messages échangés dans le diagramme de séquences précédent.

Nom méthode	Classe	Rôle méthode
choixPersonnage()	frmChoixJoueur	Avertit JeuServeur du choix d'un personnage et donc de l'arrivée d'un nouveau joueur.
creationJoueur()	JeuServeur	Crée une nouvelle instance de Joueur.
creationBoule()	Joueur	Crée une nouvelle instance de Boule (dès qu'un joueur est créé, sa boule est créée).
fermetureChoixJoueur()	JeuServeur	Détruit l'objet frmChoixJoueur.
creationArene()	JeuServeur	Crée une instance de frmArene
Deplacement()	frmArene	Avertit JeuServeur d'un déplacement.
Deplacement()	JeuServeur	Utilise Joueur pour enregistrer le déplacement. Joueur retourne la nouvelle position.
...		

Schématisation des objets créés

Ce diagramme détaille les instances possibles ou obligatoires de l'application.

JeuServeur contient une collection d'objets de type Joueur.

Une instance de JeuServeur sera créée (nommée jeuServeur). Elle pourra contenir par exemple, dans la collection lesjoueurs, 3 objets de type Joueur.

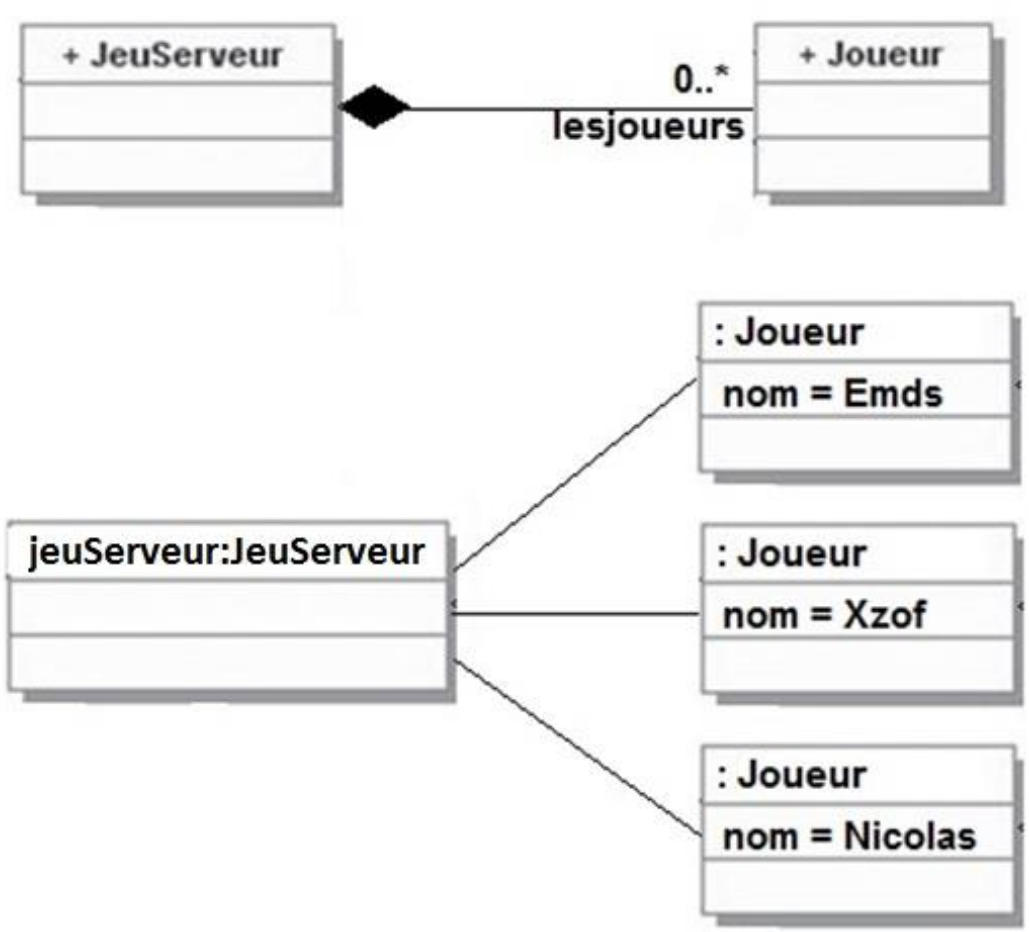


diagramme de package (UML 2)

Ce diagramme représente les différents packages de l'application et les liens entre les packages.

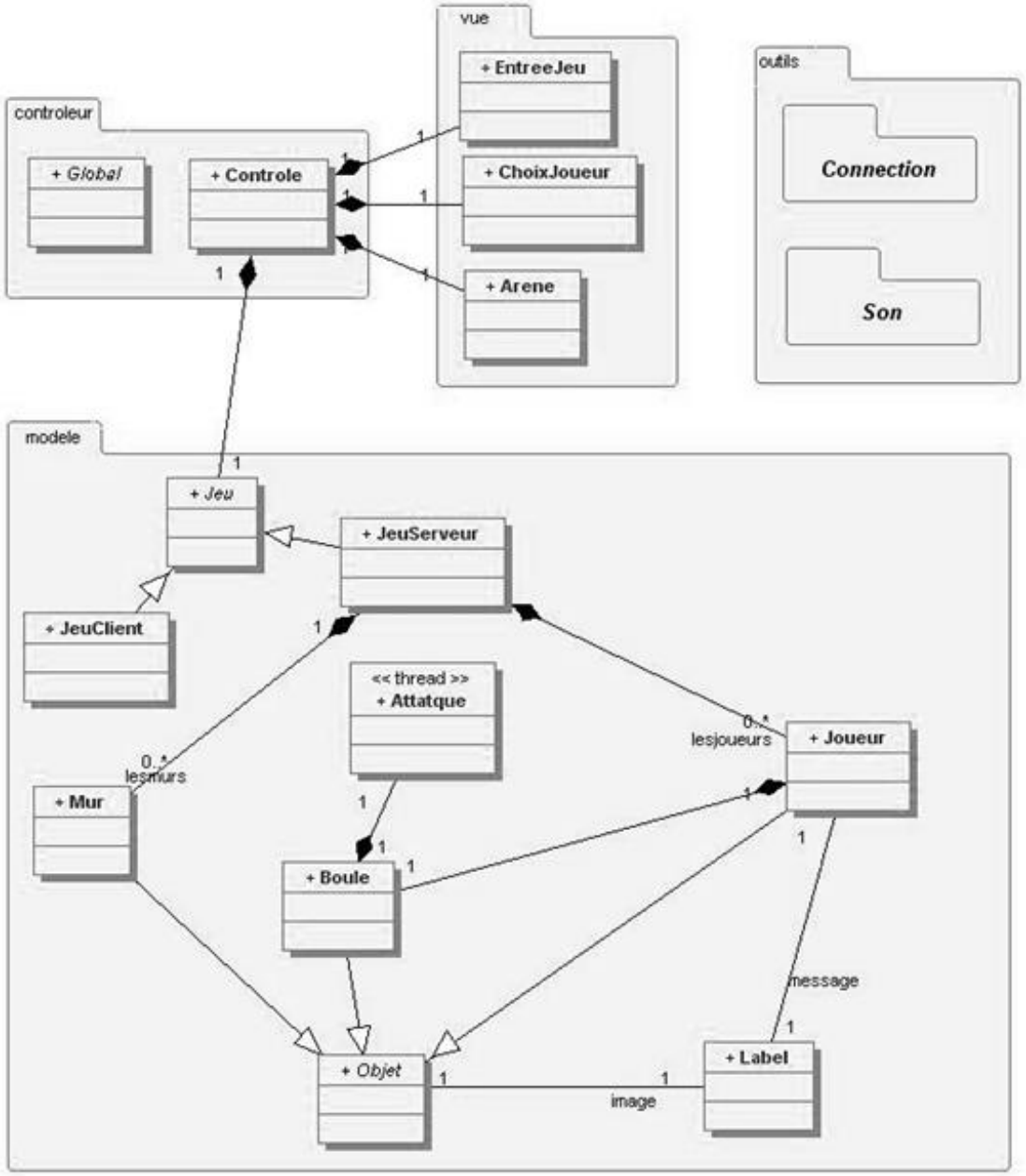


diagramme d'activité

Ce diagramme représente les règles d'enchaînement des activités par rapport aux différents acteurs.

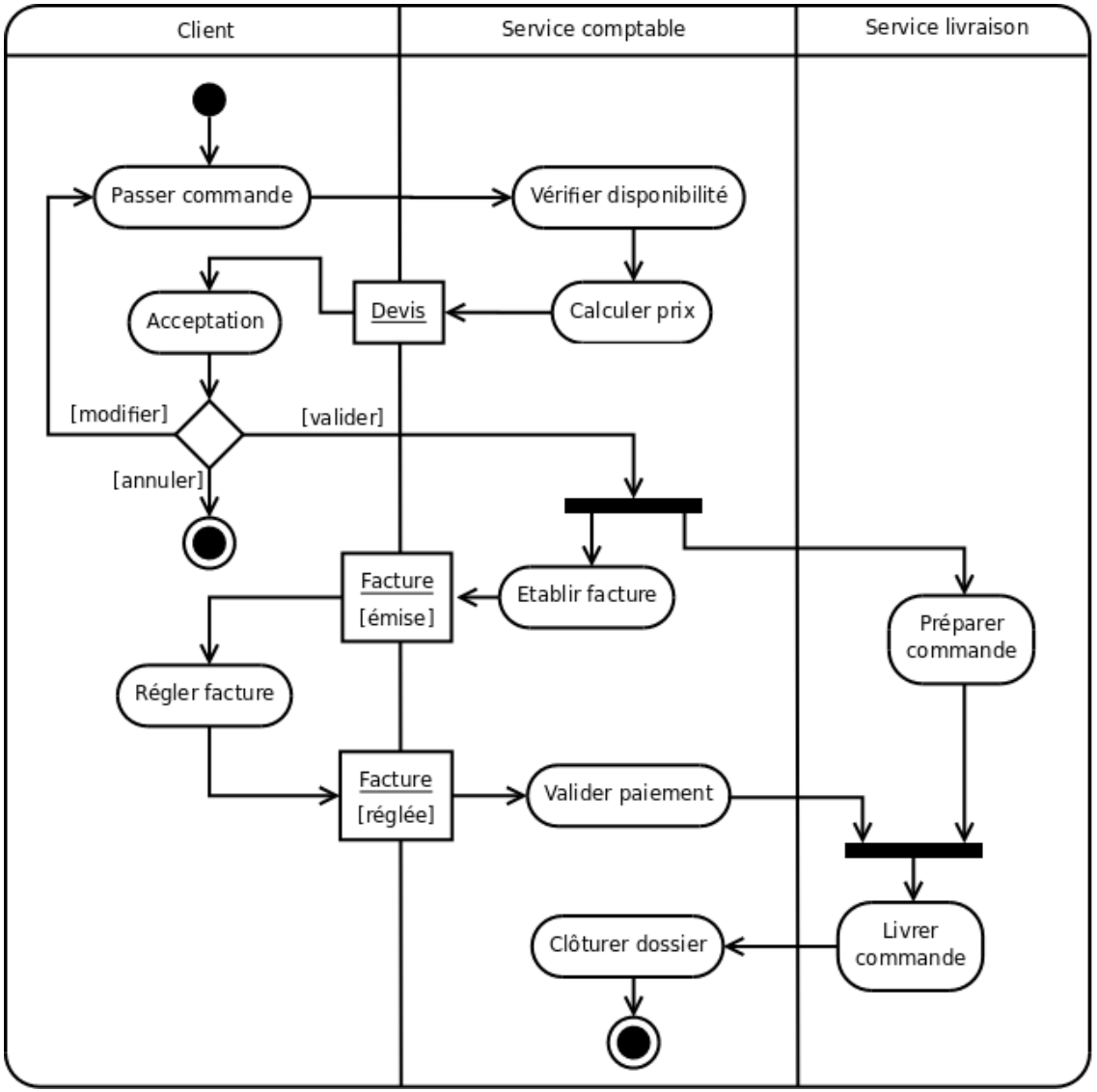


diagramme de communication (2)

Ce diagramme représente la coopération entre objets. C'est une autre représentation du diagramme de séquences en détaillant les messages.

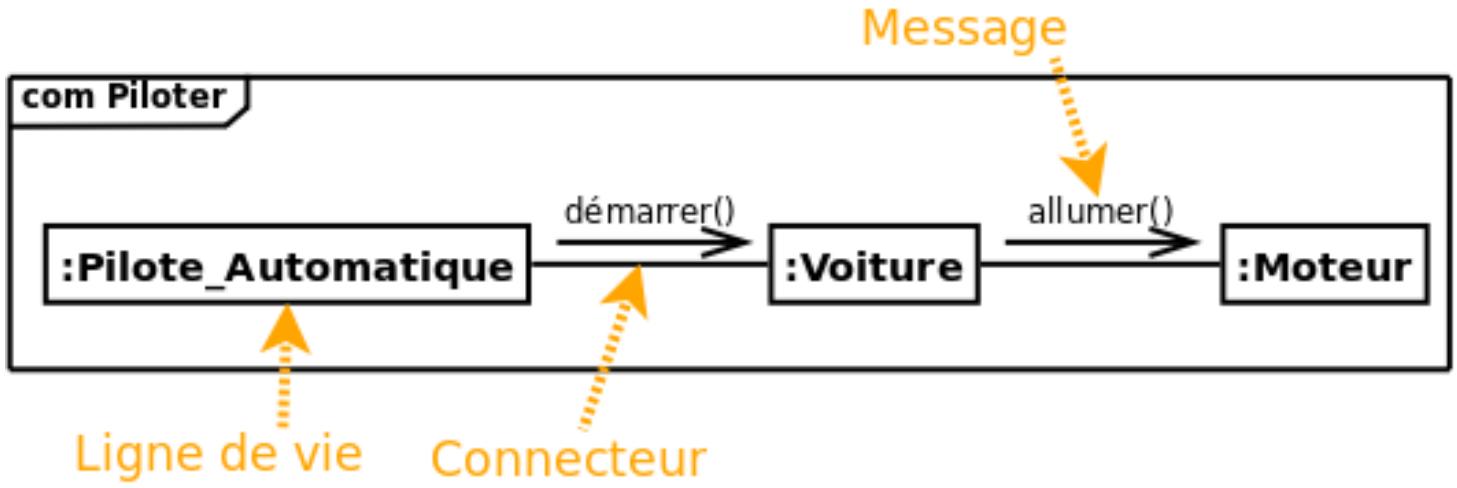


Diagramme de séquence équivalent :

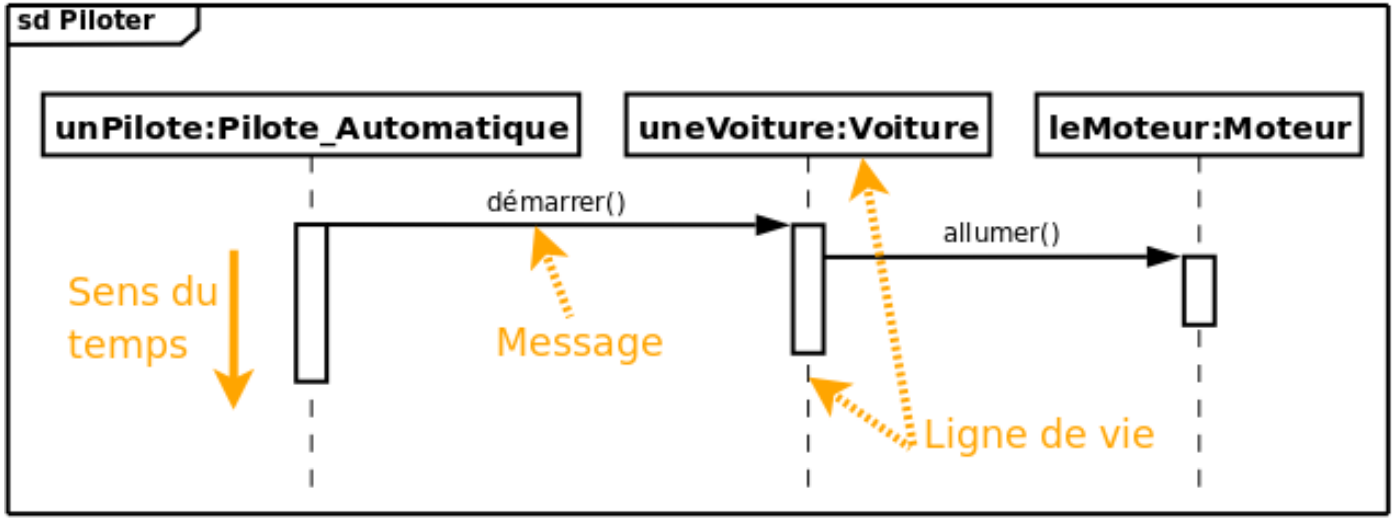


diagramme de déploiement

Ce diagramme représente l'organisation physique de la distribution des composants logiciels de l'application.

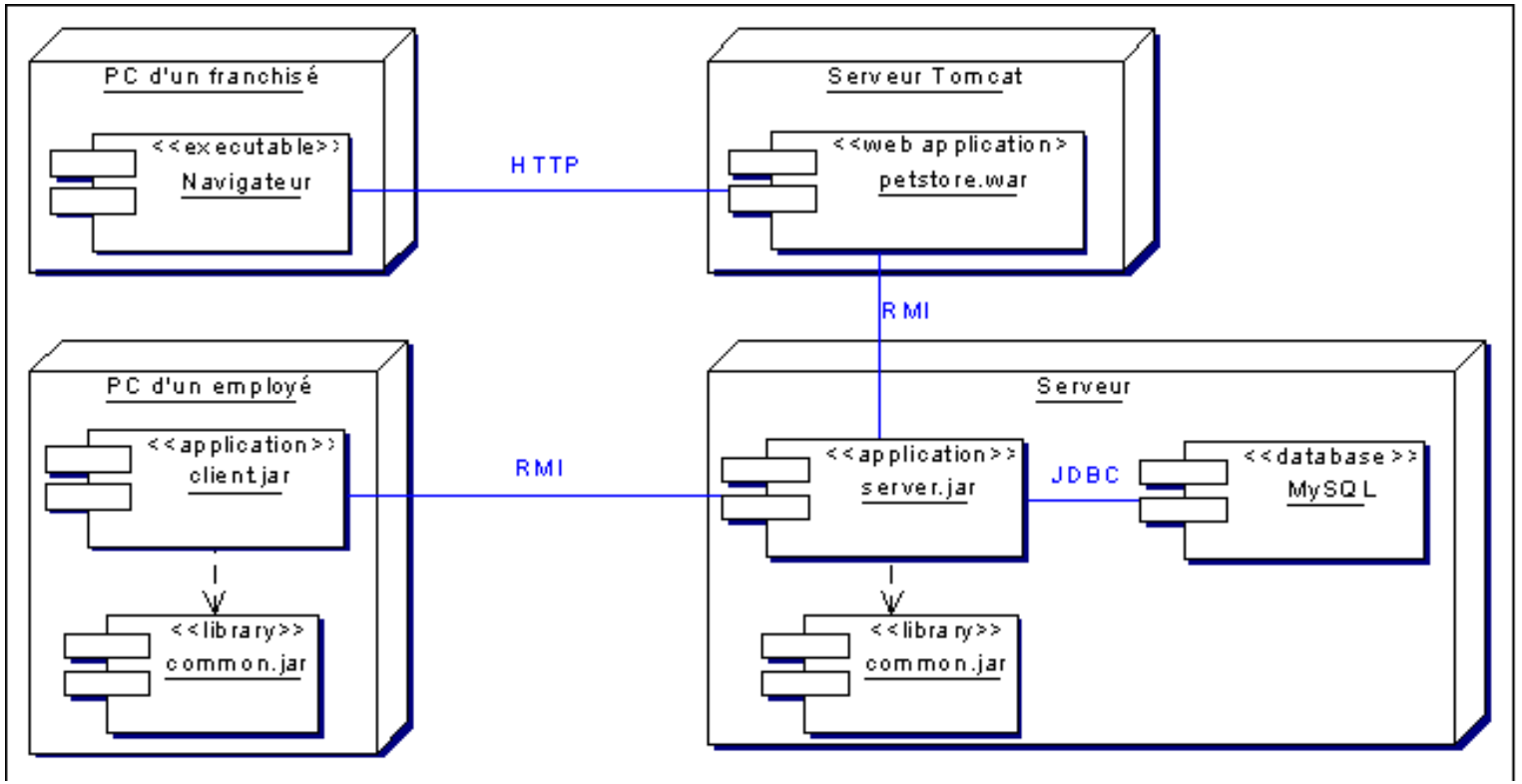


Diagramme de composants

Ce diagramme représente l'organisation des ressources au niveau logique. C'est la représentation des unités (bases de données, fichiers, librairies...) dont l'assemblage compose l'application.

Diagramme de structure composite (UML 2)

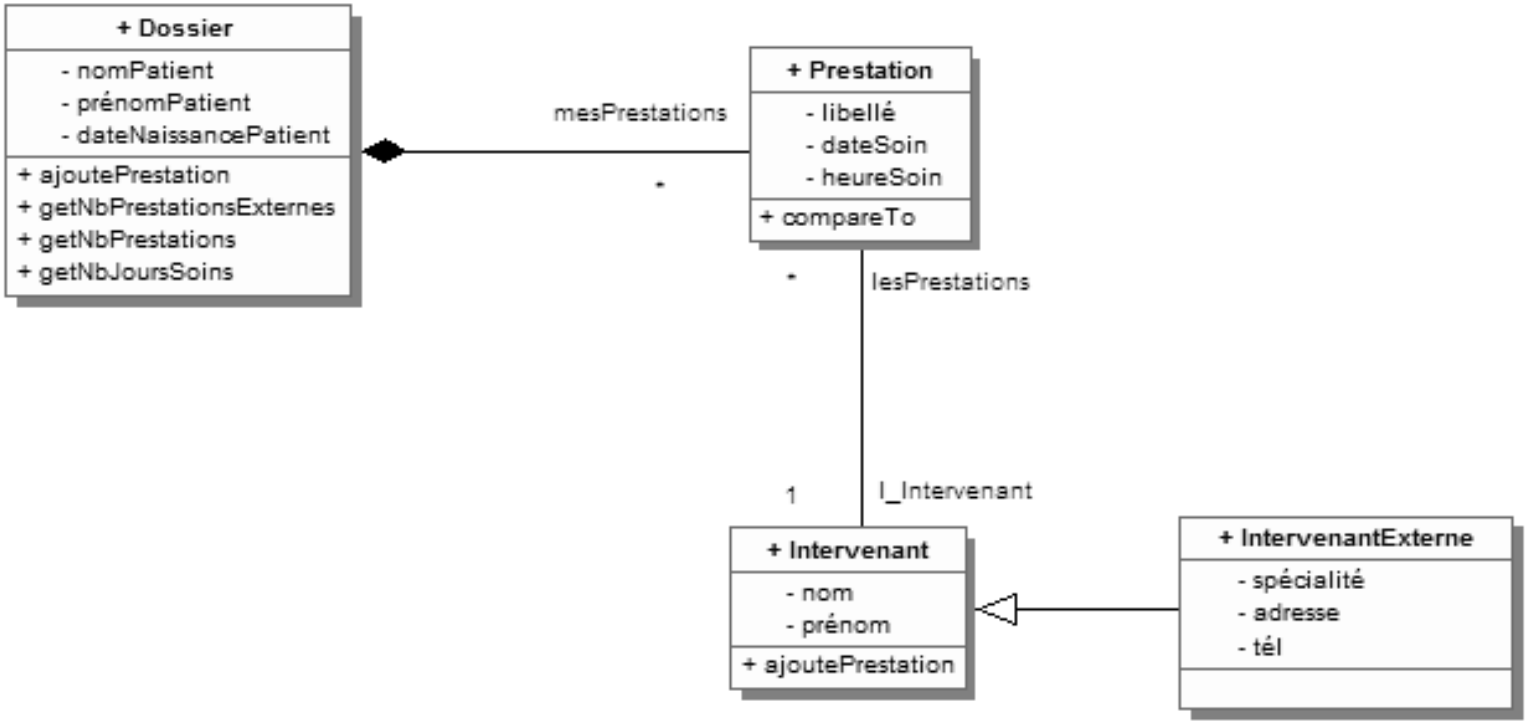
Ce diagramme représente la description de la structure interne d'un objet pendant son exécution avec les ports et connecteurs qui sont utilisés pour interagir avec d'autres instances ou avec l'extérieur.

Diagramme global d'interaction (UML 2)

Ce diagramme associe les diagrammes de séquence et d'activité. Il permet de donner une vue d'ensemble des interactions du système. Chaque élément peut ensuite être détaillé dans un diagramme de séquence ou d'activité.

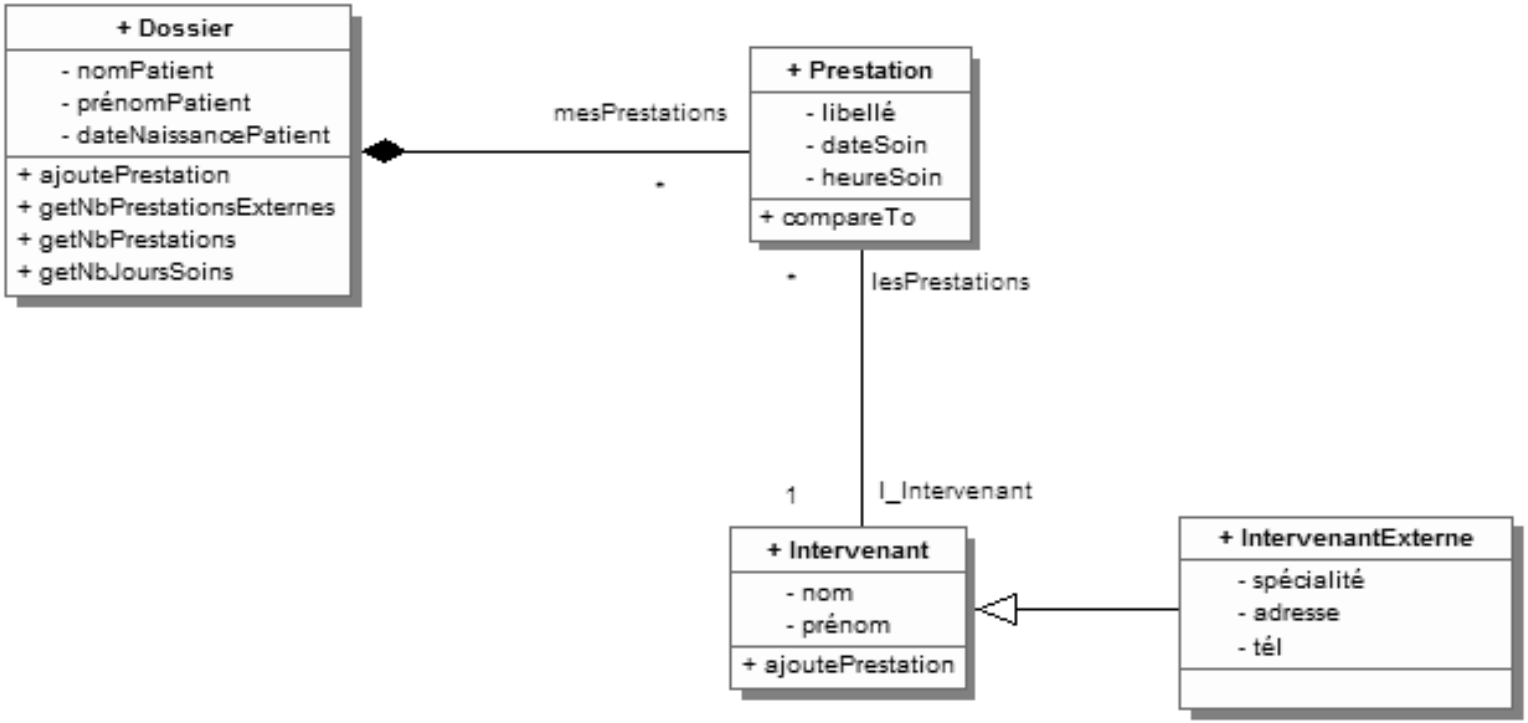
Diagramme de temps (UML 2)

Ce diagramme est utile pour les besoins en temps réel. Il modélise l'interaction entre plusieurs objets.



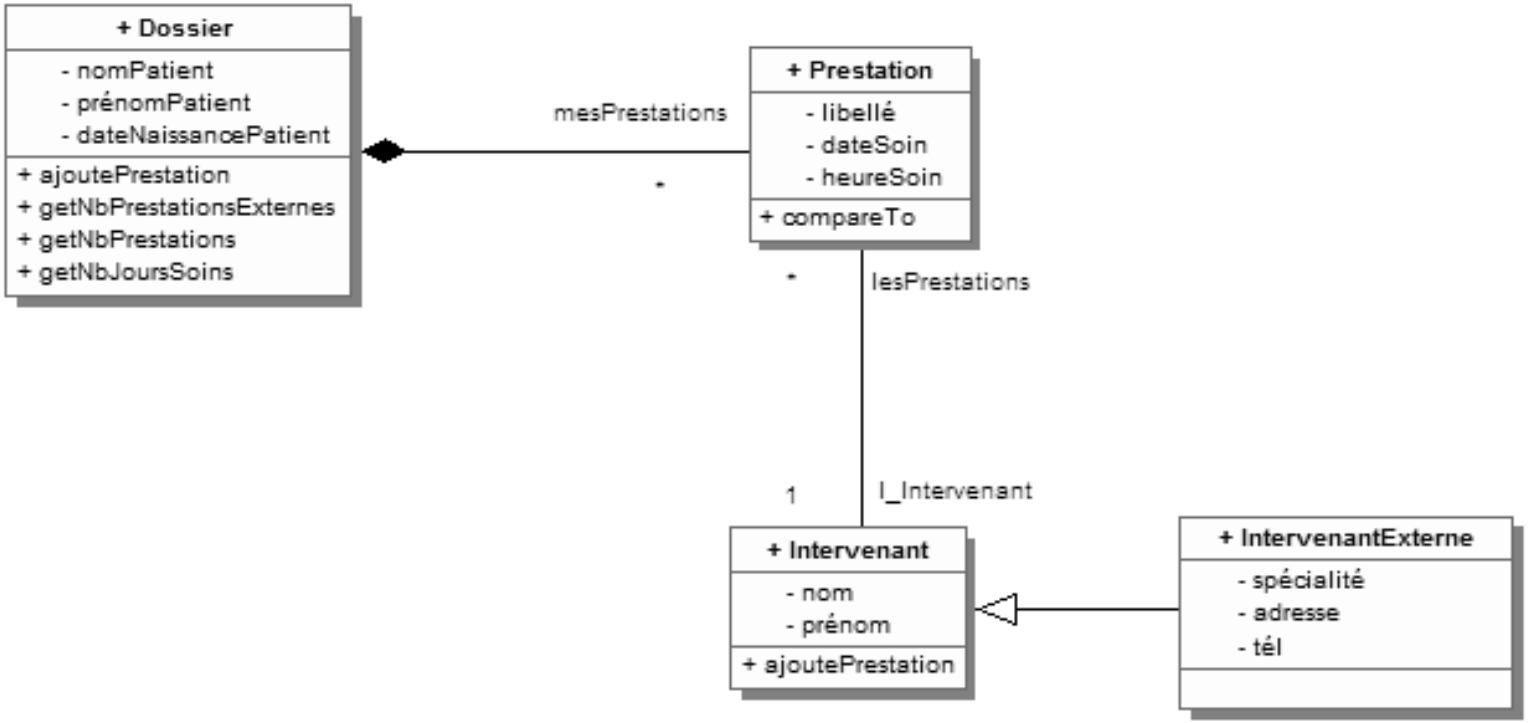
1. Expliquez comment une instance de la classe Dossier peut accéder à la méthode compareTo de la classe Prestation.

→ *En passant par un objet de la collection mesPrestations.*



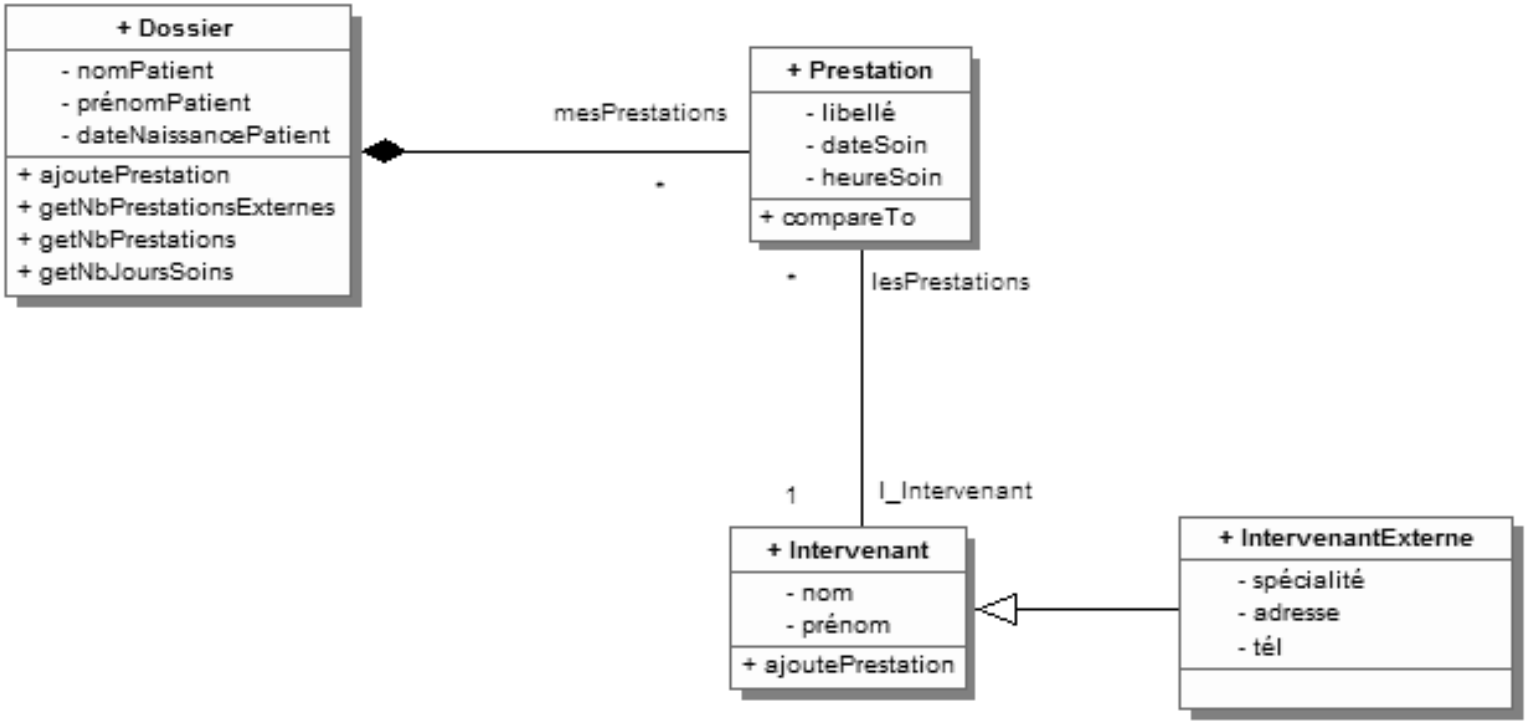
2. Quels sont les méthodes directement accessibles à partir d'une instance de la classe IntervenantExterne et pourquoi ?

→ ajoutePrestation (car c'est une méthode publique de la classe mère)



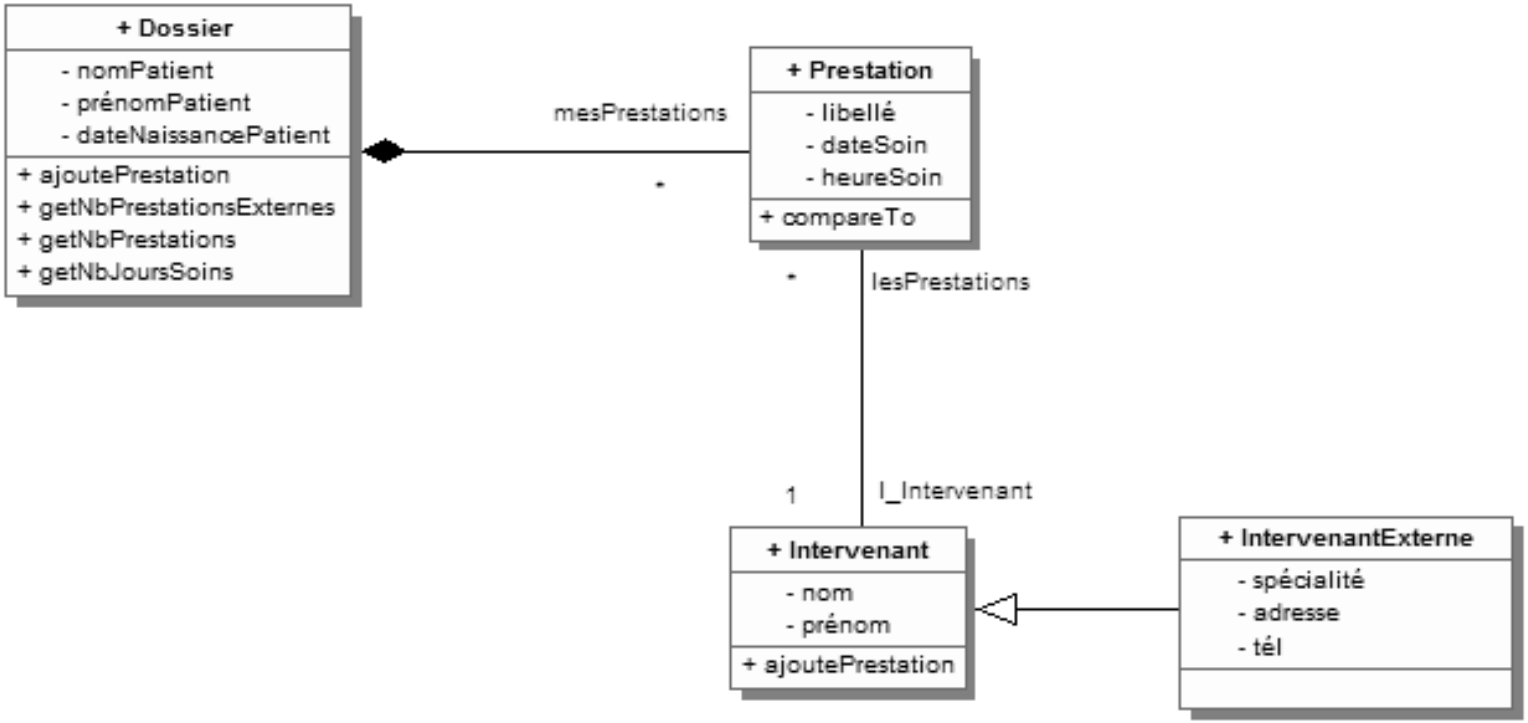
3. Quel est le nom du type de lien entre les classes Dossier et Prestation ? Expliquez ce que cela signifie.

→ *Lien de composition : lien fort qui lie les 2 classes (semblable au lien relatif de Merise). A la suppression d'une instance de Dossier, les objets de type Prestation présents dans la collection mesPrestations de ce dossier sont supprimés.*



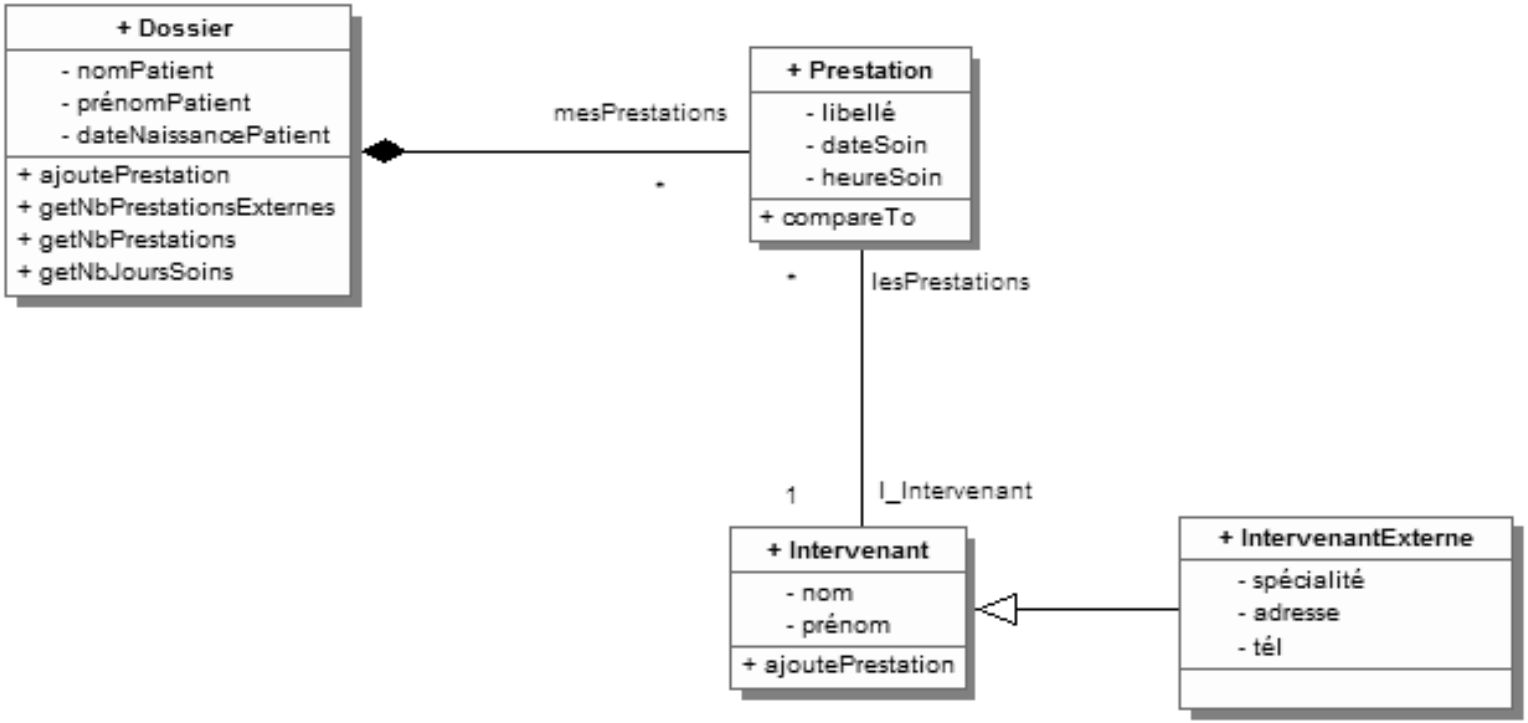
4. Que représentent "mesPrestations" et "*" écrits sur le lien et par quoi concrètement cela se traduit dans les classes ?

→ *le * signifie qu'il y a plusieurs occurrences de Prestation dans Dossier. mesPrestations est donc une propriété de type Collection de Prestation, et elle est dans Dossier.*



5. Expliquez de façon détaillée le lien entre les classes Prestation et Intervenant ainsi que toutes les informations qui sont sur le lien.

→ *Prestation contient une propriété objet l_Intervenant de type Intervenant (1 occurrence). Intervenant contient une proprité lesPrestations de type Collection de Prestation (* donc plusieurs occurrences).*



6. Expliquez textuellement ce qu'il faudrait modifier sur le schéma si un dossier était forcément suivi par un seul intervenant.

→ *Il faut ajouter une association entre Dossier et Intervenant, avec multiplicité 1 et l_Intervenant du côté fr Intervenant, et multiplicité * et lesDossiers du côté de Dossier. Il faut aussi supprimer l'association entre Prestation et Intervenant.*

exercices

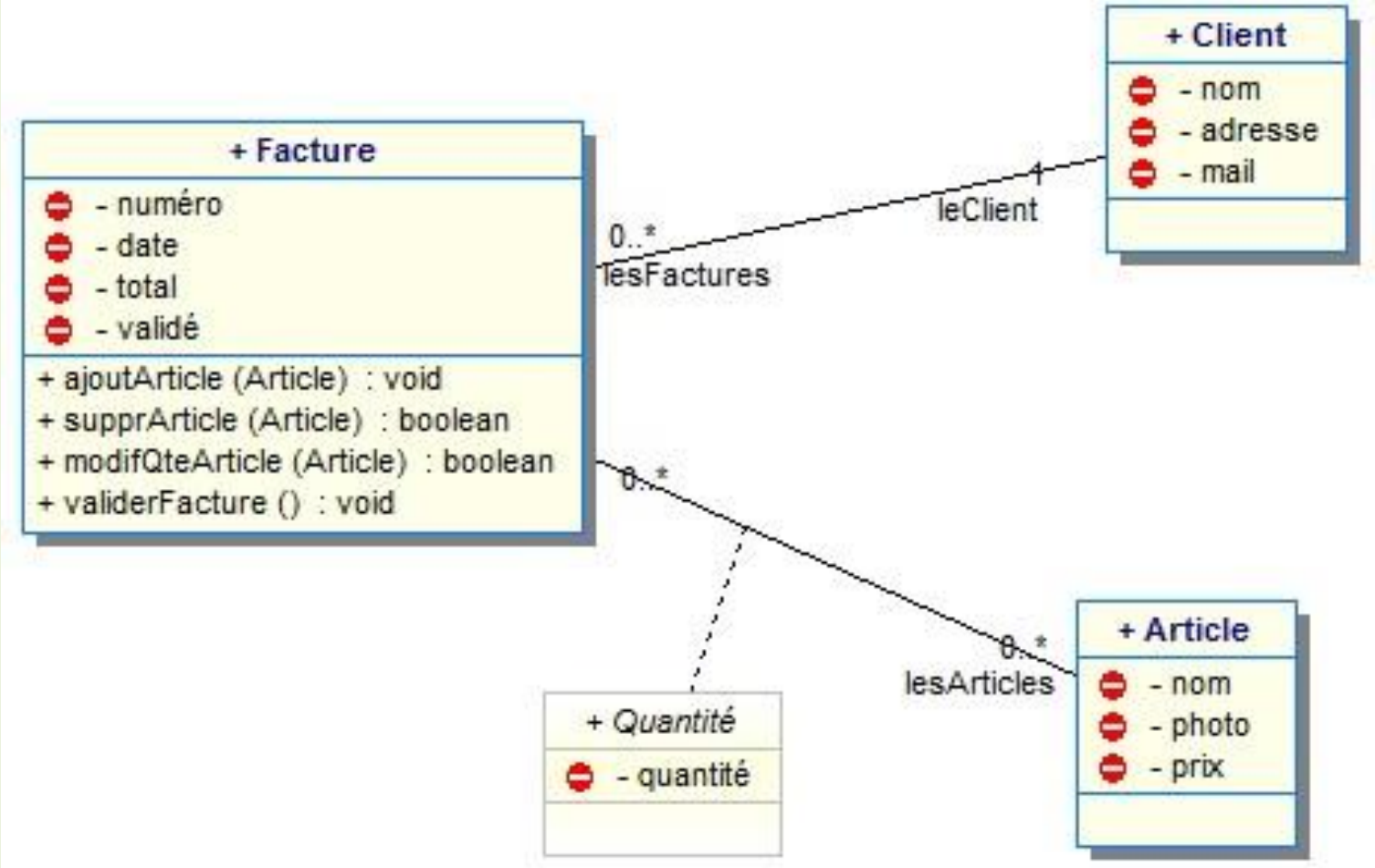
Il faut mémoriser les factures avec la date, le client concerné (nom, adresse, mail), le total et la liste des articles (nom, photo, prix). Pour chaque article présent dans une facture, la quantité est mémorisée.

Le programme doit pouvoir, entre autres, réaliser les traitements suivants :

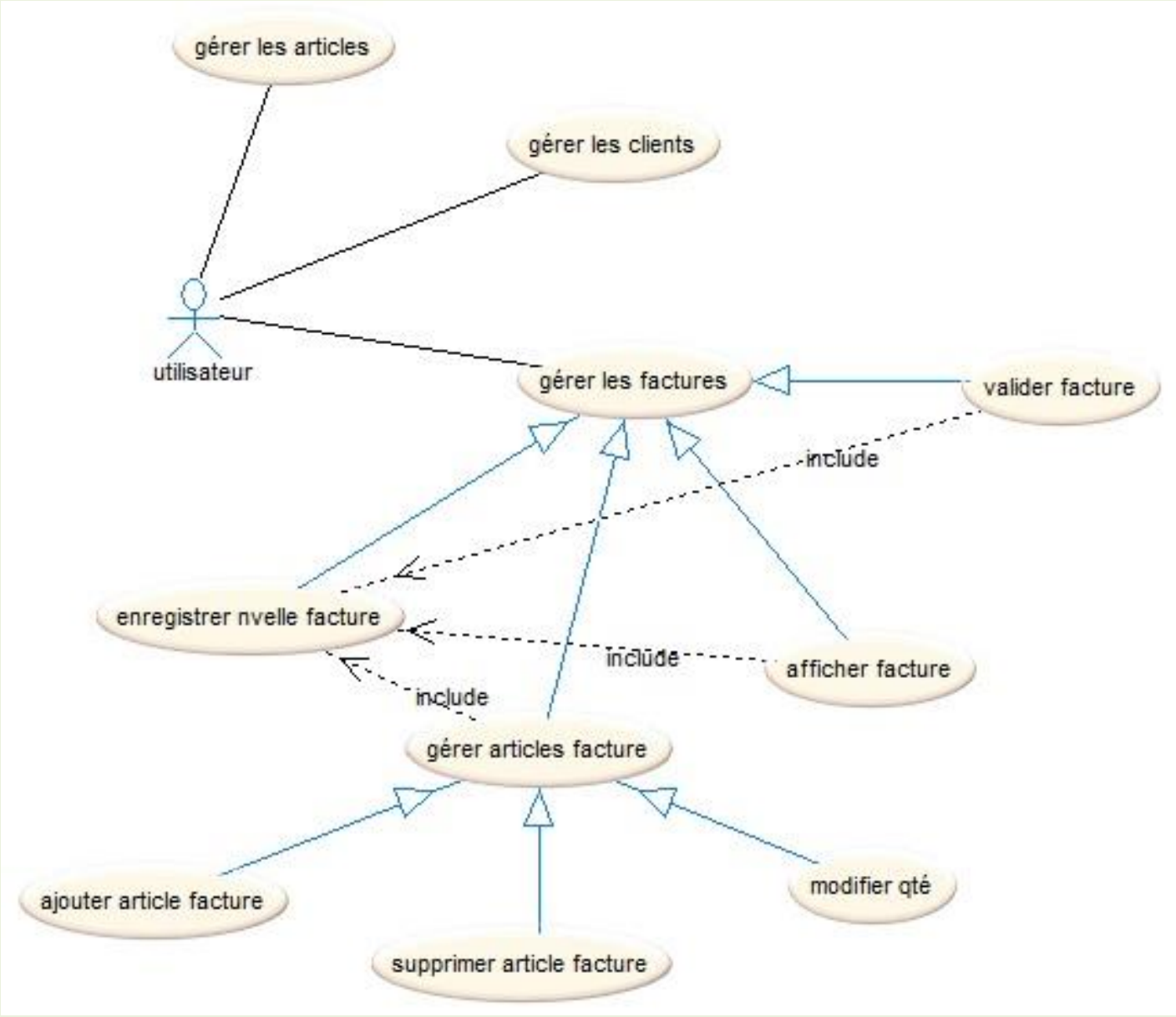
- gérer les articles (enregistrer un nouvel article, le modifier, l'afficher)
- gérer les clients (enregistrer un nouveau client, le modifier, l'afficher, gérer les factures d'un client)
- gérer les factures :
    - enregistrer une nouvelle facture vide avec le client
    - ajouter ou supprimer un article dans une facture (avec la quantité)
    - modifier la quantité d'un article dans une facture
    - afficher une facture
    - valider une facture (elle n'est alors plus modifiable)

1. **Faire une proposition de diagramme de classes (les constructeurs, getters et setters ne seront pas représentés).**

2. **Dessiner le diagramme de cas d'utilisation de l'application.**



exercices


gérer les articles
gérer les clients
utilisateur
gérer les factures
valider facture
include
enregistrer nvelle facture
include
afficher facture
include
gérer articles facture
ajouter article facture
modifier qté
supprimer article facture