Département GI

Module : UML & DotNet

Elément : UML

Historique d'UML

Début des années 1990

- Les premiers processus de développement **OO** apparaissent
- Augmentation de nombre des méthodes et notations étaient la cause de grande confusion :
  - Méthode OOD de Grady Booch (1991)
  - Méthode OMT de James Rumbaugh (1991)
  - Méthode OOSE de Ivar Jacobson (1991)

Fin 1994

- James Rumbaugh rejoint Grady Booch chez Rational Software
- OMT + OOD ➔ **Unified Method** *(oct 1995)*

Fin 1995

- Ivar Jacobson les rejoint chez Rational Software
- Unified Method + OOSE ➔ **UML 0.9** *(juin 1996)*

Début 1997

- Partenaires divers : Microsoft, Oracle, IBM, HP et autres leaders collaborent
- ➔ **UML 1.0** *(jan 1997)*

Fin 1997

- L'OMG (Object Management Group) retient UML 1.1 comme norme de modélisation

Les versions se succèdent :

- Début 1998
  - UML 1.2 puis UML 1.3
- En 2001
  - UML1.4
- En 2003
  - UML 1.5
- En 2005
  - UML 2.0
- En 2006
  - UML 2.1
- En 2013
  - UML 2.5

1 Er Chapitre : L'approche Objet

Aujourd'hui l'approche objet occupe une place clé dans le génie logiciel. En effet, ces dernières années nous avons assisté tout d'abord à une utilisation plus large des langages de programmation objet de référence comme C++, C# et Java et ensuite à l'introduction des concepts objet dans d'autres langages comme par exemple VB.NET, Perl et même Cobol.

1.1 Objet et classe

1.1.1 Concept d'objet

Un objet représente une entité du monde réel (ou du monde virtuel pour les objets immatériels) qui se caractérise par un ensemble de propriétés (attributs) des états significatifs et un comportement.

Exemple
Considérons l'employé **objet Amine, n° 105**, embauché en tant qu'ingénieur travaillant sur le site N. Cet objet est caractérisé par la liste de ses attributs et son état est représenté par les valeurs de **ses attributs** :

- **N° employé : 105,**
- **Nom : Amine,**
- **Qualification : ingénieur,**
- **Lieu de travail : site N.**

Son comportement est caractérisé par les opérations qu'il peut exécuter. Dans notre cas nous pouvons avoir **les opérations** suivantes :

- **Entrer dans l'organisme,**
- **Changer de qualification,**
- **Changer de lieu de travail,**
- **Sortir de l'organisme.**

1.1.2 Concept de classe

Une classe est l'abstraction d'un ensemble d'objets qui possèdent une structure identique (liste des attributs) et un même comportement (liste des opérations).

Un **objet** est une instance d'une et une seule classe. Une **classe abstraite** est une classe qui n'a pas d'instance. Les concepts de classe et d'objet sont interdépendants.

Exemple
Considérons la classe Employé qui représente l'ensemble des employés d'une entreprise. La description de la classe Employé comportera les éléments suivants :

• Nom de classe : Employé.
• Attributs :
  – Numéro,
  – Nom,
  – Qualification,
  – Site de travail.

• Opérations :
  – Engager un employé,
  – Consulter un employé,
  – Modifier un employé,

– Départ d'un employé.

1.2 Encapsulation

Par rapport à l'approche classique, l'approche objet se caractérise par le regroupement dans une même classe les attributs et les opérations. Ce regroupement porte le nom d'encapsulation données-traitements.

Plus précisément, les données ne sont accessibles qu'à partir d'opérations définies dans la classe. L'ensemble des opérations d'une classe rendu visible aux autres classes porte le nom d'interface.

1.3 Association et agrégation entre les classes

L'association représente une relation entre plusieurs classes. Elle correspond à l'abstraction des liens qui existent entre les objets dans le monde réel. Les multiplicités (ou cardinalités) et les rôles des objets participant aux relations complètent la description d'une association.

L'agrégation est une forme particulière d'association entre plusieurs classes. Elle exprime le fait qu'une classe est composée d'une ou plusieurs autres classes.

1.4 Généralisation et spécialisation de classe

La **généralisation** de classes consiste à factoriser dans une classe, appelée superclasse, les attributs et/ou les opérations des classes considérées.

La **spécialisation** représente la démarche inverse de la généralisation puisqu'elle consiste à créer à partir d'une classe, plusieurs classes spécialisées.

Une classe spécialisée porte aussi le nom de sous-classe. La spécialisation de classe se construit en deux temps : d'abord par héritage des opérations et des attributs d'une super-classe et ensuite par ajout d'opérations et/ou d'attributs spécifiques à la sous-classe.

1.5 Polymorphisme

Le polymorphisme est la capacité donnée à une même opération de s'exécuter différemment suivant le contexte de la classe où elle se trouve.

Ainsi une opération définie dans une super-classe peut s'exécuter de manière différente selon la sous-classe où elle est héritée.

Exemple
Soit la classe Employé et ses deux sous-classes Cadre et NonCadre.

- **Nom de classe** : Employé.
  - – Attributs :
  - – numéro,
  - – nom,
  - – salaire de base.
  - – Opérations : calculSalaire( ).
- **Nom de la sous-classe** : Cadre.
  - – Attributs : niveau d'encadrement.
  - – Opérations : calculSalaire( ).

• **Nom de la sous-classe** : NonCadre.
  – Attributs : niveau de spécialisation.
  – Opérations : calculSalaire( ).

Dans cet exemple, lorsque l'on appelle l'opération calculSalaire( ), c'est l'opération de sous-classe Cadre ou celle de la sous-classe NonCadre qui est en fait activée selon l'objet concerné. L'opération de la sous-classe fait en général appel explicitement à l'opération calculSalaire( ) de la super-classe pour bénéficier des traitements communs aux cadres et non cadres et ensuite il y aura poursuite du traitement spécifique à la sous-classe.

2 ème Chapitre: DIAGRAMME UML DES CAS D'UTILISATION

2.1 Présentation générale et concepts de base

Un diagramme de cas d'utilisation permet de décrire l'interaction entre les acteurs (utilisateurs du cas) et le système. La description de l'interaction est réalisée suivant le point de vue de l'utilisateur.

La représentation d'un cas d'utilisation se base sur trois concepts : l'acteur, le cas d'utilisation et l'interaction entre l'acteur et le cas d'utilisation.

2.1.1 Acteur

Un acteur est un utilisateur de futur système qui va utiliser ses fonctionnalités (cas d'utilisation).

Une même personne physique peut se comporter comme plusieurs acteurs différents que le nombre de rôles qu'elle joue vis-à-vis du système. Par exemple l'administrateur d'un groupe facebook il peut être considéré en tant qu'un utilisateur d'un compte facebook standard ou en tant qu'administrateur d'une page facebook.

Un acteur peut aussi être un système externe avec lequel le cas d'utilisation va interagir.

Formalisme et exemple

Un acteur peut se représenter symboliquement par un « bonhomme » et être identifié par son nom. Il peut aussi être formalisé par une classe stéréotypée « acteur » (fig. 1).

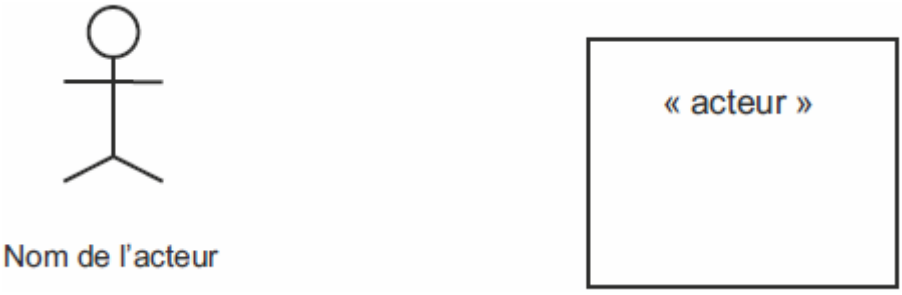


Figure 1: représentation d'acteur

2.1.2 Cas d'utilisation et interaction

Un cas d'utilisation correspond à un certain nombre de fonctionnalités que le système devra exécuter en réponse à un besoin d'un acteur.

Une interaction permet de décrire les échanges entre un acteur et un cas d'utilisation.

Formalisme et exemple

Un cas d'utilisation se représente par un ovale dans lequel figure son intitulé.

L'interaction entre un acteur et un cas d'utilisation se représente comme une association. Elle peut comporter des multiplicités comme toute association entre classes.

Le formalisme de base de représentation d'un cas d'utilisation est donné à la figure 2.

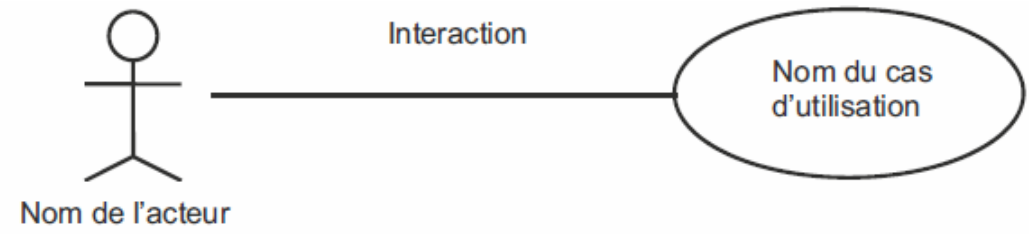


Figure 2: interaction acteur et cas d'utilisation

2.1.3 Relations entre cas d'utilisation

Trois relations peuvent être décrites entre cas d'utilisation : une relation d'inclusion (« include »), une relation d'extension (« extend ») et une relation de généralisation.

Relation d'inclusion

Une relation d'inclusion d'un cas d'utilisation A par rapport à un cas d'utilisation B signifie qu'une instance de A contient le comportement décrit dans B.

La figure 3 donne le formalisme et un exemple d'une relation d'inclusion entre cas d'utilisation.

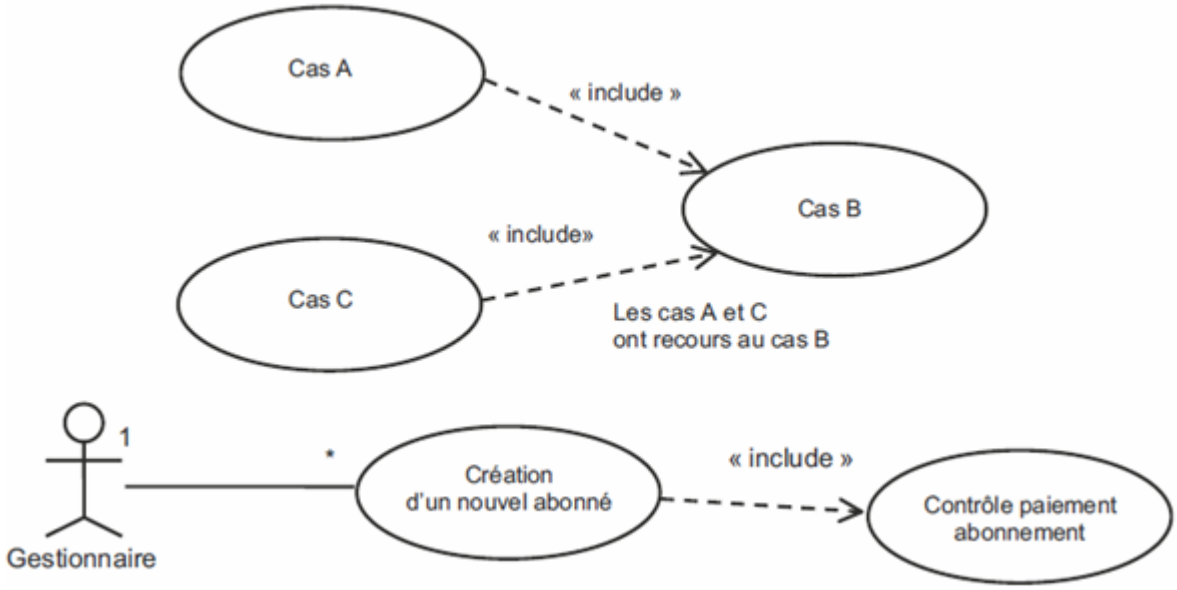


Figure 3: Formalisme et exemple de la relation d'inclusion

Relation de généralisation

Une **relation de généralisation** de cas d'utilisation peut être définie conformément au principe de la spécialisation-généralisation.

La figure 4 donne un exemple d'une relation de généralisation de cas d'utilisation

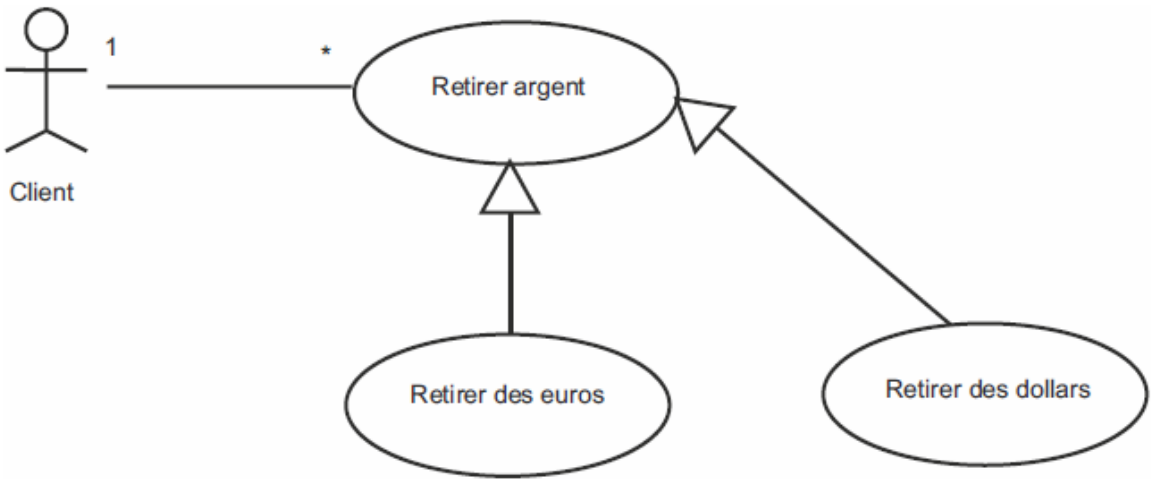


Figure 4: Exemple d'une relation de généralisation de cas d'utilisation

Relation d'extension

Une relation d'extension d'un cas d'utilisation A par un cas d'utilisation B signifie qu'une instance de A peut être étendue par le comportement décrit dans B. Deux caractéristiques sont à noter :

- Le caractère optionnel de l'extension « extend » dans le déroulement du cas d'utilisation standard.
- La mention explicite du point d'extension dans le cas d'utilisation standard.

La figure 5 donne un exemple d'une relation d'extension entre cas d'utilisation.

Une note peut être ajoutée à la représentation du cas d'utilisation permettant d'expliciter la condition.

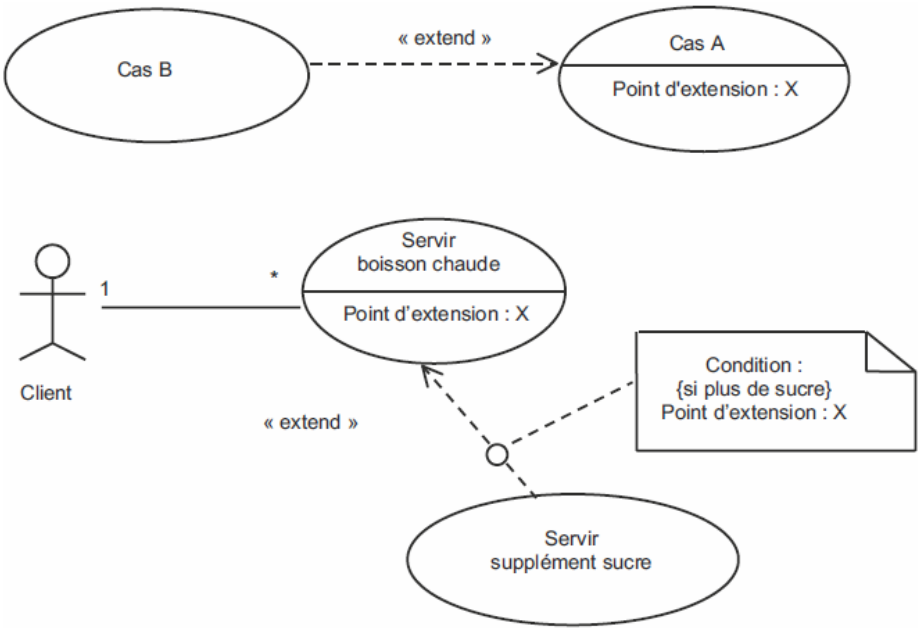


Figure 5: Formalisme et exemple d'une relation d'extension

2.2 Description textuelle d'un cas d'utilisation

À chaque cas d'utilisation doit être associée une description textuelle des interactions entre l'acteur et le système et les actions que le système doit réaliser en vue de produire les résultats attendus par les acteurs.

UML ne propose pas de présentation de cette description textuelle. Cependant, les travaux menés par Alistair Cockburn [Cockburn2001] sur ce sujet constituent une référence en la matière et tout naturellement nous reprenons ici l'essentiel de cette présentation.

La description textuelle d'un cas d'utilisation est articulée en six points :

• **Objectif** – Décrire le contexte et les résultats attendus du cas d'utilisation.

• **Acteurs concernés** – Le ou les acteurs concernés par le cas d'utilisation doivent être identifiés en précisant globalement leur rôle.

• **Pré conditions** – Si certaines conditions particulières sont requises avant l'exécution du cas, elles sont à exprimer à ce niveau.

• **Post conditions** – Par symétrie, si certaines conditions particulières doivent être réunies après l'exécution du cas, elles sont à exprimer à ce niveau.

• **Scénario nominal** – Il s'agit là du scénario principal qui doit se dérouler sans incident et qui permet d'aboutir au résultat souhaité.

• **Scénarios alternatifs** – Les autres scénarios, secondaires ou correspondants à la résolution d'anomalies, sont à décrire à ce niveau. Le lien avec le scénario principal se fait à l'aide d'une numérotation hiérarchisée (1.1a, 1.1b…) rappelant le numéro de l'action concernée.

2.3 Travaux dirigés

On désire automatiser la gestion d'une bibliothèque. La bibliothèque est administrée par un gestionnaire qui gère les inscriptions et des relances des lecteurs lorsque ces derniers n'ont pas rendu leurs ouvrages au-delà du délai autorisé. Les bibliothécaires gèrent les emprunts et le retour des ouvrages ainsi que l'approvisionnement de nouveaux ouvrages.

Il existe trois catégories d'abonné. Tout d'abord les étudiants qui doivent seulement s'acquitter d'une somme forfaitaire pour une année afin d'avoir droit à tous les services de la bibliothèque. L'accès à la bibliothèque est libre pour tous les enseignants. Enfin, il est possible d'autoriser des étudiants d'une autre université à s'inscrire exceptionnellement comme abonné moyennant le versement d'une cotisation. Le nombre d'abonné externe est limité chaque année à environ 10 % des inscrits.

Un nouveau service de consultation du catalogue général des ouvrages doit être mis en place.

Les ouvrages, souvent acquis en plusieurs exemplaires, sont rangés dans des rayons de la bibliothèque. Chaque exemplaire est repéré par une référence gérée dans le catalogue et le code du rayon où il est rangé.

Chaque abonné ne peut emprunter plus de trois ouvrages. Le délai d'emprunt d'un ouvrage est de trois semaines, il peut cependant être prolongé exceptionnellement à cinq semaines.

Il est demandé d'élaborer le diagramme des cas d'utilisation (DCU).

3 ème Chapitre : Diagrammes UML de modélisation statique

3.1 DIAGRAMME DE CLASSE (DCL) ET DIAGRAMME D'OBJET (DOB)

La description du diagramme de classe est fondée sur :

- Le concept d'objet,
- Le concept de classe comprenant les attributs et les opérations,
- Les différents types d'association entre classes.

3.1.1 Objet

Un objet est un concept, une abstraction ou une chose qui a un sens dans le contexte du système à modéliser. Chaque objet a une identité et peut être distingué des autres.

Un objet est caractérisé par les valeurs de ses propriétés qui lui attribuent des états significatifs suivant les instants considérés.

3.1.2 Classe, attribut et opération

Classe

Une classe décrit un groupe d'objets ayant les mêmes propriétés (attributs), un même comportement (opérations).

Un objet est une instance d'une classe. La classe représente l'abstraction de ses objets.

Une classe se représente à l'aide d'un rectangle comportant plusieurs compartiments.

Les trois compartiments de base sont :

- la désignation de la classe,
- la description des attributs,
- la description des opérations.

Deux autres compartiments peuvent être aussi indiqués :

- la description des responsabilités de la classe,
- la description des exceptions traitées par la classe.

Il est possible de manipuler les classes en limitant le niveau de description à un nombre réduit de compartiments selon les objectifs poursuivis par le modélisateur.

La figure 6 montre le formalisme général des compartiments d'une classe et des premiers exemples.

Attribut

Un attribut est une propriété élémentaire d'une classe. Pour chaque objet d'une classe l'attribut prend une valeur.

La figure 7 montre le formalisme et un exemple de représentation des attributs de classe.

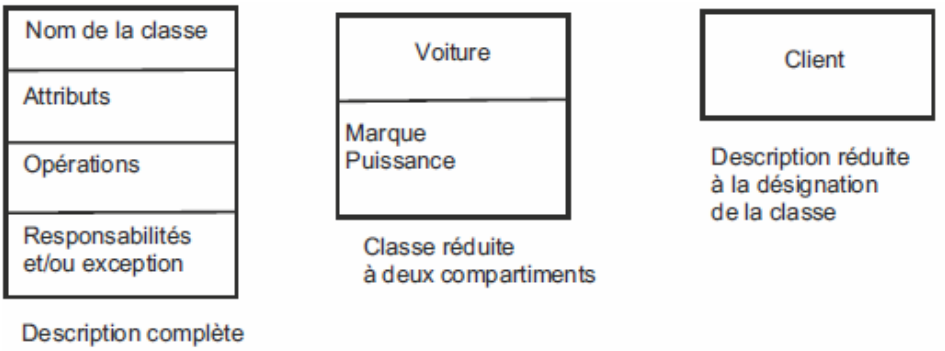


Figure 6: Formalisme général d'une classe et exemples.



Figure 7: Formalisme d'attributs de classe et exemple.

La description complète des attributs d'une classe comporte un certain nombre de caractéristiques qui doivent respecter le formalisme suivant :

Visibilité/Nom attribut : type [= valeur initiale {propriétés}]

– *Visibilité* : se reporter aux explications données plus loin sur ce point.
– *Nom d'attribut* : nom unique dans sa classe.
–*Type* : type primitif (entier, chaîne de caractères…) dépendant des types disponibles dans le langage d'implémentation ou type classe matérialisant un lien avec une autre classe.
– *Valeur initiale* : valeur facultative donnée à l'initialisation d'un objet de la classe.
– {*propriétés*} : valeurs marquées facultatives (ex. : « interdit » pour mise à jour interdite).

Un attribut peut avoir des valeurs multiples. Dans ce cas, cette caractéristique est indiquée après le nom de l'attribut (ex. : prénom [3] pour une personne qui peut avoir trois prénoms).

Un attribut dont la valeur peut être calculée à partir d'autres attributs de la classe est un attribut dérivé qui se note « /nom de l'attribut dérivé ». Un exemple d'attribut dérivé est donné à la figure 9.

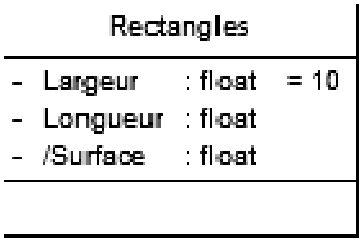


Figure 8: Exemple de représentation d'une classe

Opération

Une opération est une fonction applicable aux objets d'une classe. Une opération permet de décrire le comportement d'un objet. Une méthode est l'implémentation d'une opération.

Chaque opération est désignée soit seulement par son nom, sa liste de paramètres et son type de retour. La signature d'une méthode correspond au nom de la méthode et la liste des paramètres en entrée. La figure 9 montre le formalisme et un exemple de représentation d'opérations de classe.

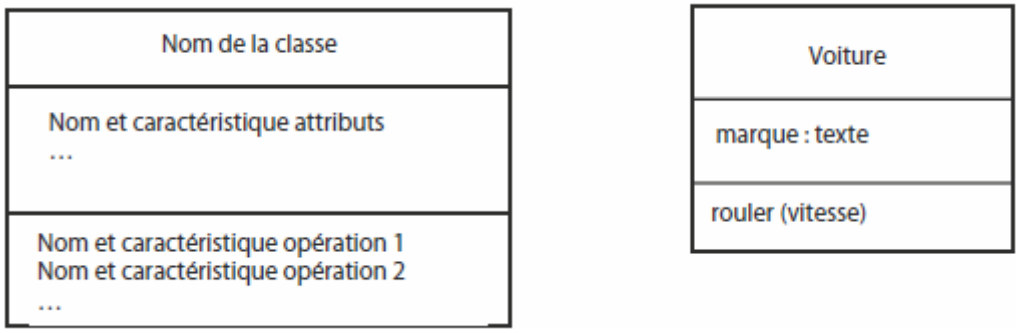


Figure 9: Formalisme et exemple d'opérations de classe.

La description complète des opérations d'une classe comporte un certain nombre de caractéristiques qui doivent respecter le formalisme suivant :

Visibilité Nom d'opération (paramètres) [:[type résultat] {propriétés}]

– Visibilité : se reporter aux explications données plus loin sur ce point.
– Nom d'opération : utiliser un verbe représentant l'action à réaliser.
– Paramètres : liste de paramètres (chaque paramètre peut être décrit, en plus de son nom, par son type et sa valeur par défaut). L'absence de paramètre est indiquée par ( ).
– Type résultat : type de (s) valeur(s) retourné(s) dépendant des types disponibles dans le langage d'implémentation. Par défaut, une opération ne retourne pas de valeur, ceci est indiqué par exemple par le mot réservé « void » dans le langage C++ ou Java.
– {propriétés} : valeurs facultatives applicables (ex. : {query} pour un comportement sans influence sur l'état du système).

Exemples de classes et représentation d'objets

Il est utile de préciser que la représentation des objets sera utilisée dans plusieurs autres diagrammes importants d'UML. C'est le cas notamment du diagramme de séquence ou encore du diagramme d'état-transition.

La figure 10 présente des exemples d'objets.

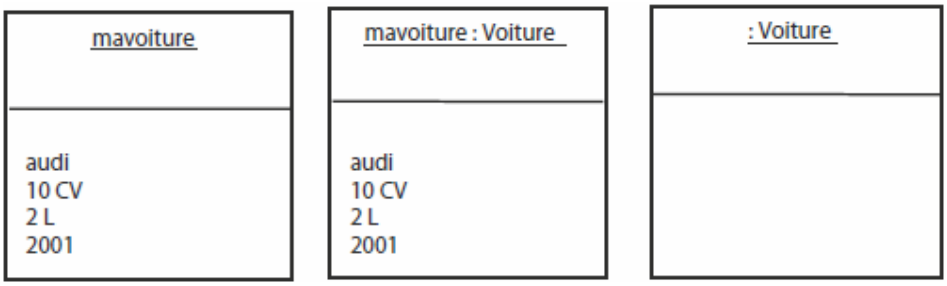


Figure 10: Exemples de représentation d'objets

Visibilité des attributs et opérations

Chaque attribut ou opération d'une classe peut être de type public, protégé, privé ou paquetage. Les symboles **+ (public), # (protégé), - (privé) et ~ (paquetage)** sont indiqués devant chaque attribut ou opération pour signifier le type de visibilité autorisé pour les autres classes.

Attribut ou opération de niveau classe

Un attribut ou une opération peut être défini non pas au niveau des instances d'une classe, mais au niveau de la classe.

Il s'agit soit d'un attribut qui est une constante pour toutes les instances d'une classe soit d'une opération d'une classe abstraite.

C'est le soulignement de l'attribut ou de l'opération qui caractérise cette propriété.

Dans l'exemple de la figure 11, l'attribut « ristourne » est de type classe et l'opération « créer » est une opération exécutable au niveau de la classe.

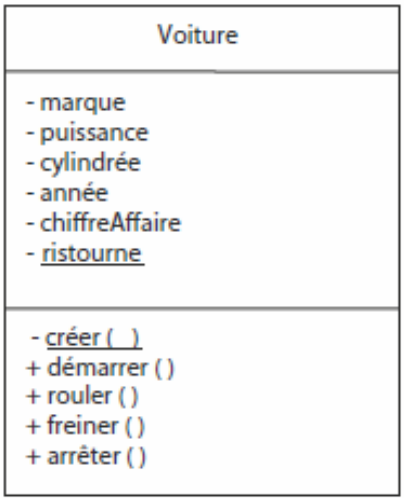


Figure 11: Exemple d'attribut ou d'opération de niveau classe.

3.1.3 Association, multiplicité, navigabilité et contraintes

Lien et association

Un lien est une connexion physique ou conceptuelle entre instances de classes donc entre objets. Une association décrit un groupe de liens ayant une même structure et une même sémantique. Un lien est une instance d'une association. Chaque association peut être identifiée par son nom.

La figure 12 donne le formalisme de l'association. Le symbole (facultatif) indique le sens de lecture de l'association. Dans cette figure est donné aussi un exemple de représentation d'une association.
Le rôle réalisé par une classe dans une association peut être précisé sur l'association.

La figure 13 donne un exemple de rôle d'association.

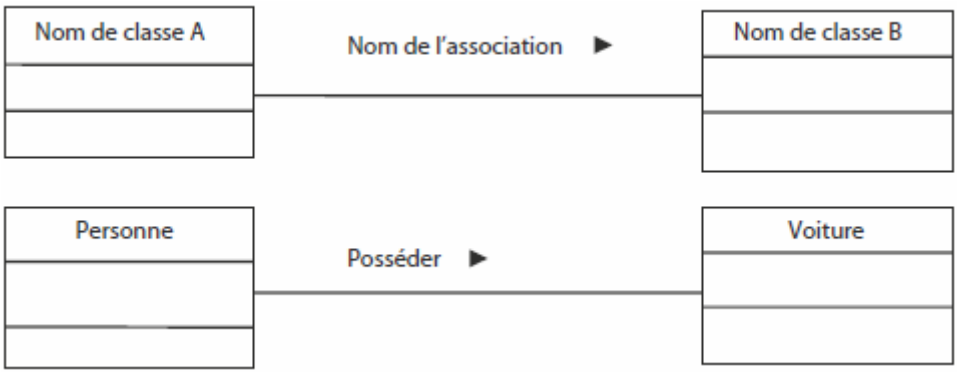


Figure 12: Formalisme et exemple d'association

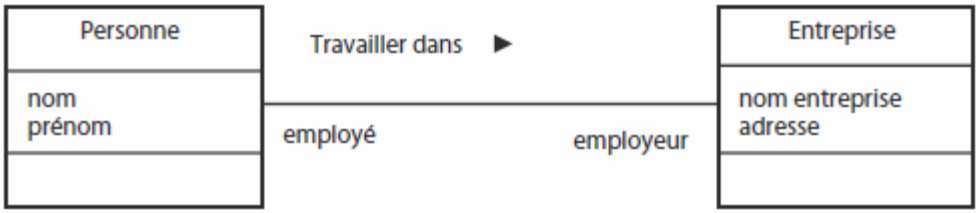


Figure 13: Exemple de rôles d'une association

Multiplicité

La **multiplicité** indique un domaine de valeurs pour préciser le nombre d'instance d'une classe vis-à-vis d'une autre classe pour une association donnée. **Le domaine de valeurs est décrit selon plusieurs formes :**

- **Intervalle fermé – Exemple : 3 ..15.**
- **Valeurs exactes – Exemple : 3, 5, 8.**
- **Valeur indéterminée notée * – Exemple : 1..*.**

– Dans le cas où l'on utilise seulement *, cela traduit une multiplicité 0..*.

Nous donnons, à la figure 14, quelques exemples des principales multiplicités définies dans UML.

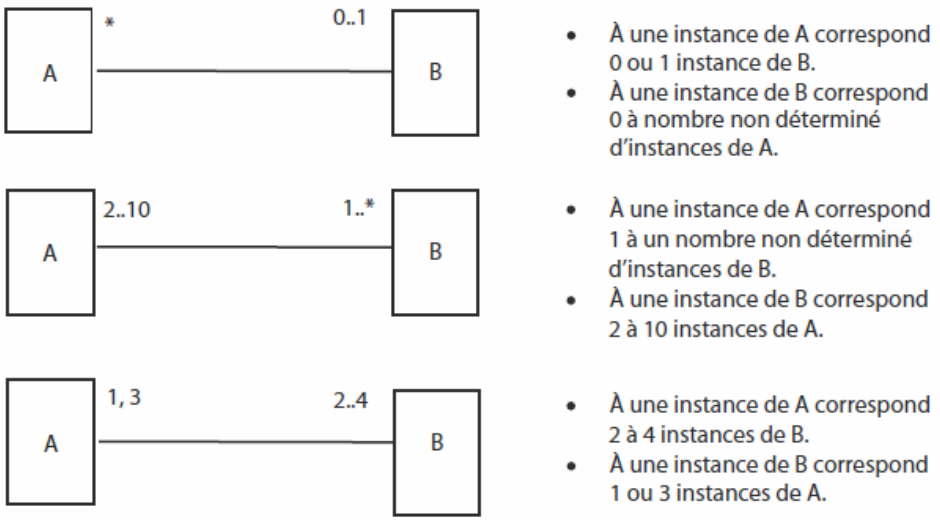


Figure 14: Exemple de multiplicités

Navigabilité

La **navigabilité** indique si l'association fonctionne de manière unidirectionnelle ou bidirectionnelle, elle est matérialisée par une ou deux extrémités fléchées. La nonnavigabilité se représente par un « X ».

Les situations possibles de navigabilité sont représentées à la figure 15.

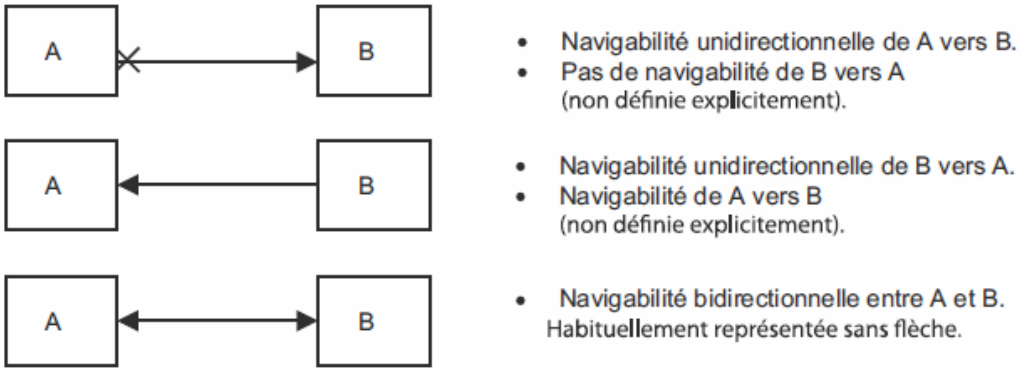


Figure 15: Représentation de la navigabilité d'association.

Par défaut, on admet qu'une navigabilité non définie correspond à une navigabilité implicite.

Dans l'exemple donné à la figure 16, a une personne sont associées ses copies d'examen mais l'inverse n'est pas possible (notamment avant la correction de la copie).

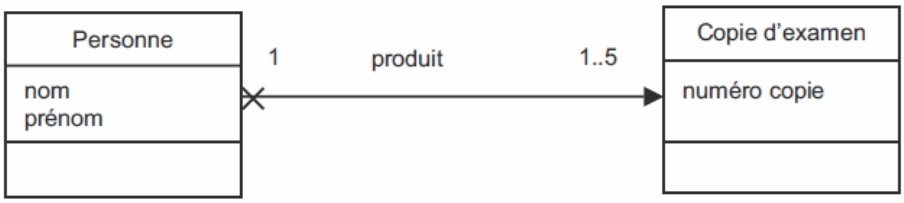


Figure 16: Exemple de navigabilité d'une association.

Contraintes

Ordre de tri

Pour une association de multiplicité supérieure à 1, les liens peuvent être :

- non ordonnés (valeur par défaut),
- ordonnés ou triés lorsque l'on est au niveau de l'implémentation (tri sur une valeur interne).

Un exemple est donné à la figure 17. Dans cet exemple, pour une entreprise donnée, les personnes seront enregistrées suivant un ordre qui correspondra à un des attributs de Personne.

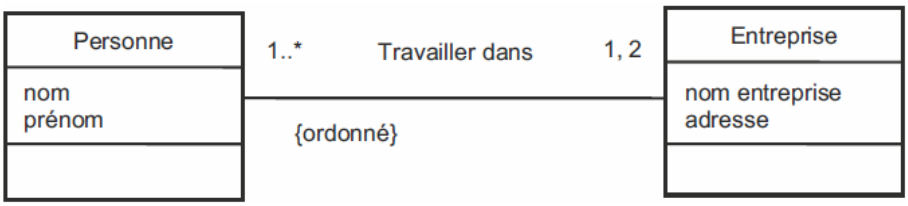


Figure 17: Exemple de contrainte d'ordre d'une association

Propriétés de mise à jour de liens

Il est possible d'indiquer des contraintes particulières relatives aux conditions de mise à jour des liens.

- {interdit} : interdit l'ajout, la suppression ou la mise à jour des liens.

Association de dimension supérieure à 2 et classe-association

Une association de dimension supérieure à 2 se représente en utilisant un losange permettant de relier toutes les classes concernées.

Une classe-association permet de décrire soit des attributs soit des opérations propres à l'association. Cette classe-association est elle-même reliée par un trait en pointillé au losange de connexion. Une classe-association peut être reliée à d'autres classes d'un diagramme de classes.

Un exemple d'une association de dimension 3 comprenant une classe-association « Affectation » est donné à la figure 18. La classe-association Affectation permet de décrire les attributs propres à l'association de dimension 3 représentée.

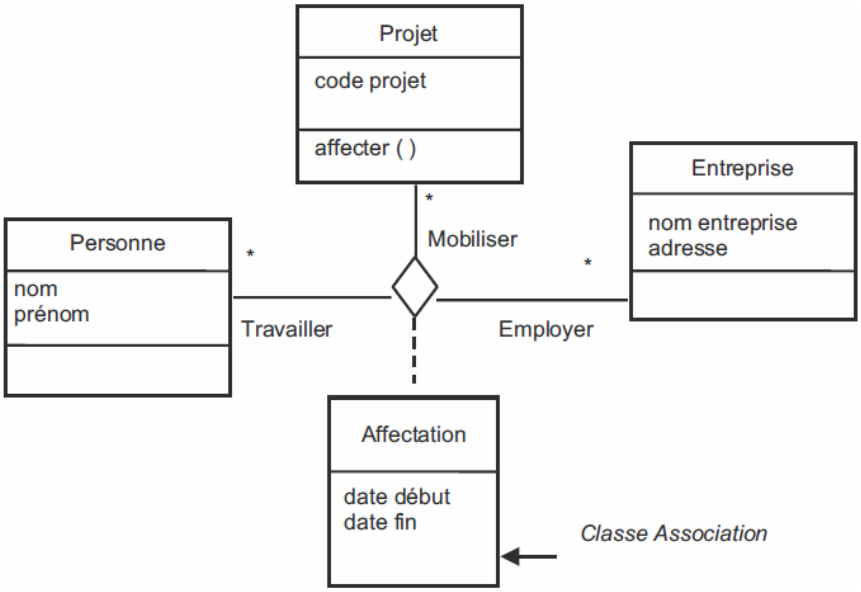


Figure 18: Exemple d'une association de dimension 3

3.1.4 Agrégation et composition entre classes

Agrégation

L'**agrégation** est une association qui permet de représenter un lien de type « ensemble » comprenant des « éléments ». Il s'agit d'une relation entre une classe représentant le niveau « ensemble » et 1 à *n* classes de niveau « éléments ».

La figure 19 montre un exemple de relation d'agrégation.

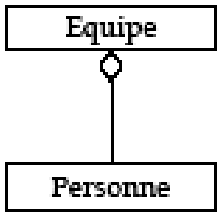


Figure 19: Exemple d'agrégation

Composition

La composition est une relation d'agrégation dans laquelle il existe une contrainte de durée de vie entre la classe « composant » et la ou les classes « composé ». Autrement dit la suppression de la classe « composé » implique la suppression de la ou des classes « composant ».

Par exemple dans la figure 20 Un document est composé de plusieurs paragraphes, qui, à son tour, est composé de plusieurs phrases.

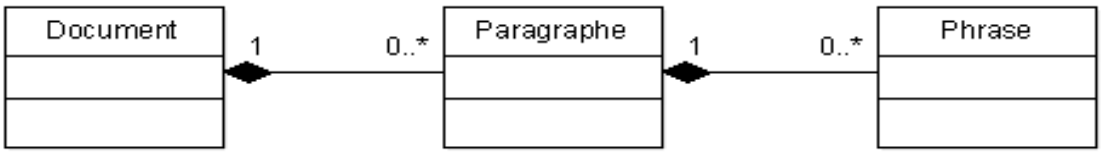


Figure 20: exemple de relation de composition

3.1.5 Association qualifiée, dépendance et classe d'interface

Qualification

La **qualification** d'une relation entre deux classes permet de préciser la sémantique de l'association et de qualifier de manière restrictive les liens entre les instances.

Seules les instances possédant l'attribut indiqué dans la qualification sont concernées par l'association. Cet attribut ne fait pas partie de l'association.

Soit la relation entre les répertoires et les fichiers appartenant à ces répertoires. À un répertoire est associé 0 à n fichiers. Si l'on veut restreindre cette association pour ne considérer qu'un fichier associé à son répertoire, la relation qualifiée est alors utilisée pour cela. La figure 21 montre la représentation de ces deux situations.

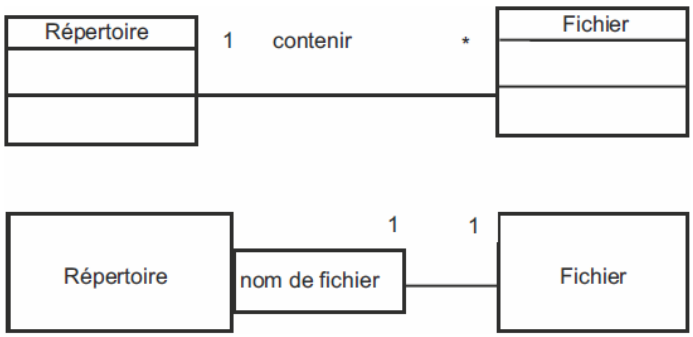


Figure 21: Formalisme et exemple d'association qualifiée

Dépendance

La **dépendance** entre deux classes permet de représenter l'existence d'un lien sémantique. Une classe B est en dépendance de la classe A si des éléments de la classe A sont nécessaires pour construire la classe B.

La relation de dépendance se représente par une flèche en pointillé (fig. 22) entre deux classes.



Figure 22: Formalisme de représentation d'un lien de dépendance

Interface

Une classe d'interface permet de décrire la vue externe d'une classe. La classe d'interface, identifiée par un nom, comporte la liste des opérations accessibles par les autres classes. Le compartiment des attributs ne fait pas partie de la description d'une interface.

L'interface peut être aussi matérialisée plus globalement par un petit cercle associé à la classe source.

La classe utilisatrice de l'interface est reliée au symbole de l'interface par une flèche en pointillé.

La figure 23 et figure 24 donnent le formalisme, sur un exemple, des deux types de représentation d'une interface.

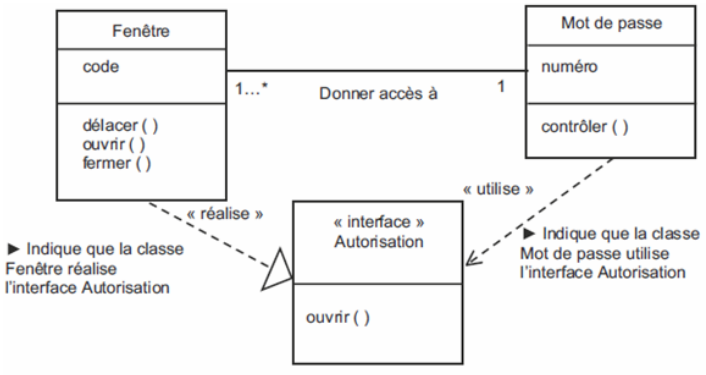


Figure 23: Exemple de description d'une classe d'interface

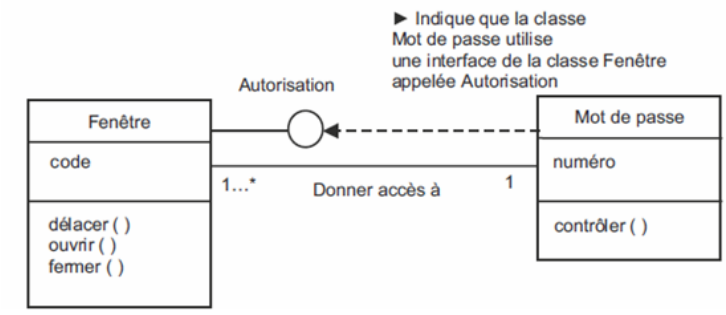


Figure 24: Exemple de description d'une classe d'interface

3.1.6 Généralisation et spécialisation

La **généralisation** est la relation entre une classe et deux autres classes ou plus partageant un sous-ensemble commun d'attributs et/ou d'opérations.

L'opération qui consiste à créer une super-classe à partir de classes s'appelle la généralisation. Inversement la spécialisation consiste à créer des sousclasses à partir d'une classe.

La figure 25 montre un exemple de relation de spécialisation. Dans cet exemple, les attributs nom, prénom et date de naissance et l'opération « calculer âge » de « Employé » sont hérités par les trois sous-classes : Employé horaire, Employé salarié, Vacataire.

Classe abstraite

Une classe abstraite est une classe qui n'a pas d'instance directe mais dont les classes descendantes ont des instances. Dans une relation d'héritage, la super-classe est par définition une classe abstraite. C'est le cas de la classe Employé dans l'exemple présenté à la figure.25.

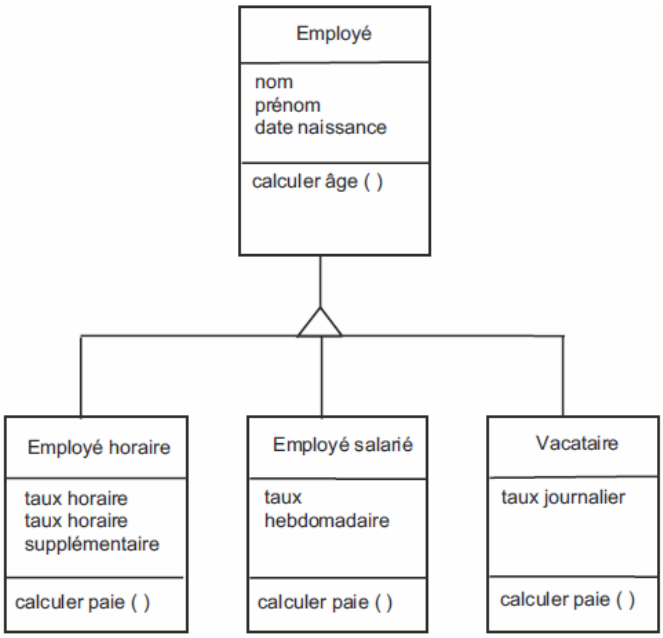


Figure 25: Exemple de relation de spécialisation

L'héritage avec recouvrement

D'une manière générale, quatre situations peuvent se rencontrer et se représentent sous forme de contraintes :

- **{chevauchement} : deux sous-classes peuvent avoir, parmi leurs instances, des instances identiques ;**
- **{disjoint}** : les instances d'une sous-classe ne peuvent être incluses dans une autre sous-classe de la même classe ;
- **{complète} : la généralisation ne peut pas être étendue ;**
- **{incomplète} : la généralisation peut être étendue.**

La figure 26 montre un exemple d'héritage avec recouvrement d'instances entre les classes Étudiant et Employé. En effet, une même personne peut être à la fois étudiante dans une université et employée dans une entreprise.

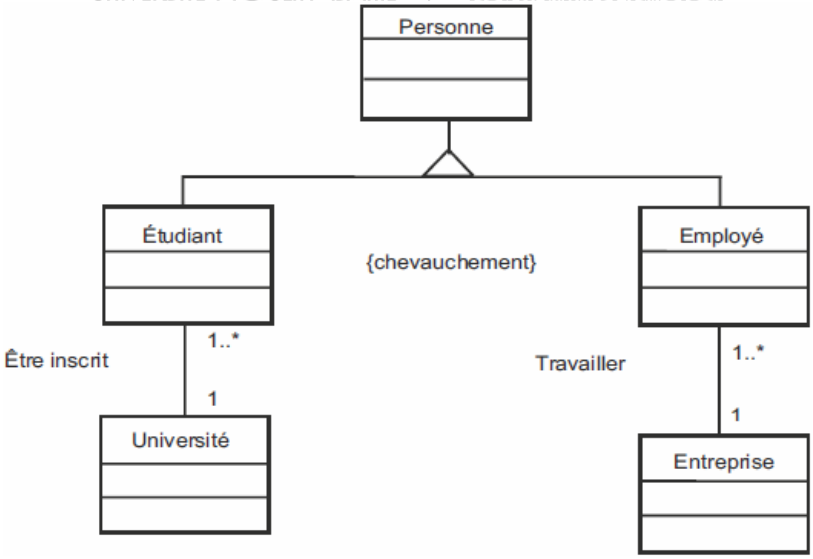


Figure 26: Exemple d'héritage avec recouvrement d'instances.

Extension et restriction de classe

L'ajout de propriétés dans une sous-classe correspond à une **extension de classe**. Le masquage de propriétés dans une sous-classe correspond à une **restriction de classe**.

La figure 27 montre un exemple d'héritage avec restriction et extension.

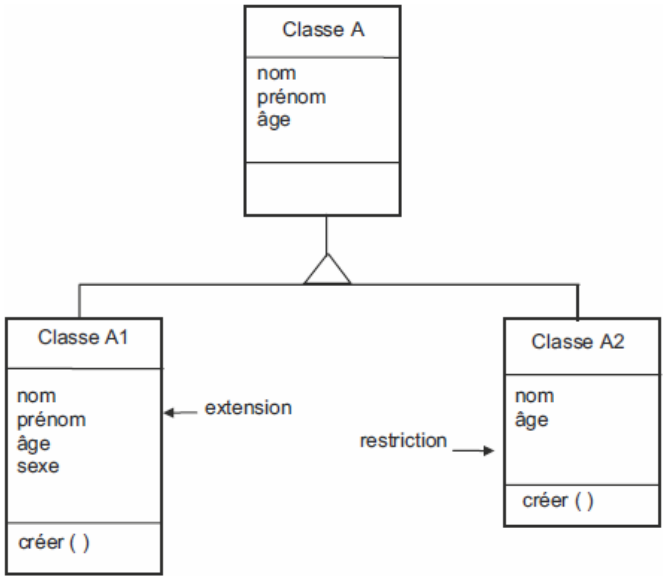


Figure 27: Exemple d'héritage avec extension et restriction de propriétés

L'héritage multiple

Dans certains cas, il est nécessaire de faire hériter une même classe de deux classes « parentes » distinctes. Ce cas correspond à un **héritage multiple**.

La figure 28 montre un exemple classique d'héritage multiple où la classe « Véhicule amphibie » hérite des classes « Véhicule terrestre » et « Véhicule marin ».

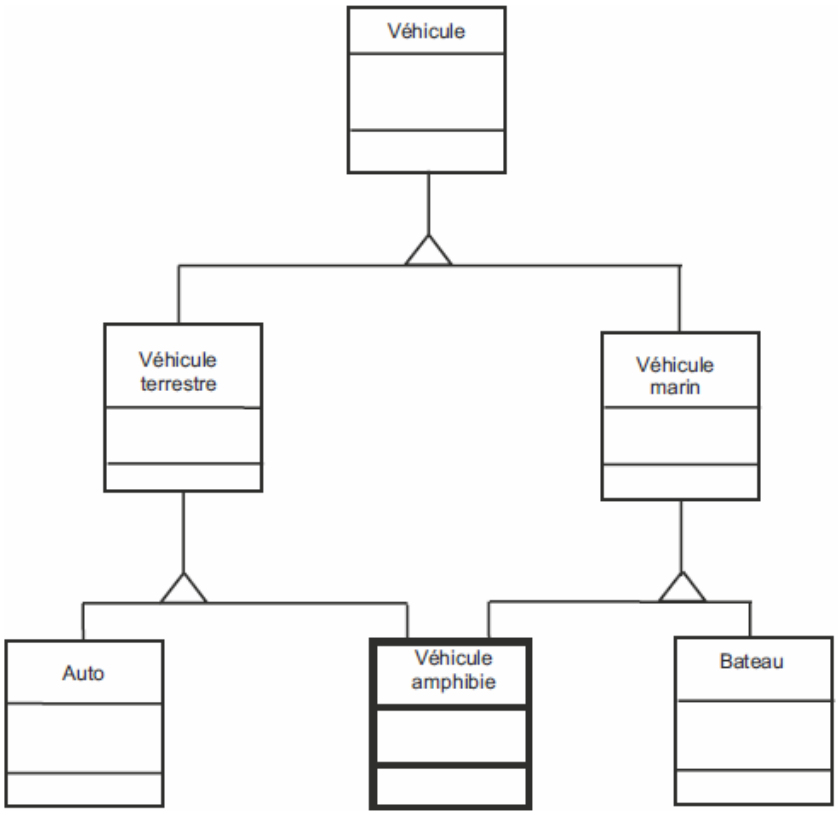


Figure 28: Exemple de relation d'héritage multiple

3.1.7 Stéréotype de classe

UML propose un certain nombre de **stéréotypes** qui permettent de qualifier les profils d'utilisation.
Parmi ces stéréotypes, nous présentons ci-après quatre d'entre eux :

- **« Classe d'implémentation » – Ce stéréotype est utilisé pour décrire des classes de niveau physique.**
- **« Type » – Ce stéréotype permet de spécifier des opérations applicables à un domaine d'objets. Exemple : Type Integer d'un langage de programmation.**
- **« Utilitaire » – Ce stéréotype qualifie toutes les fonctions utilitaires de base utilisées par les objets.**
- **« MétaClasse » – Ce stéréotype permet de regrouper des classes dans une famille de classe.**

3.1.8 Travaux dirigées

Exercice 1

Il est demandé de représenter le diagramme de classe d'une gestion technique de documents. Chaque document est composé d'un ou plusieurs feuillets. Un feuillet comporte du texte et des objets géométriques qui constituent deux types d'objets graphiques supportant des opérations de type : sélectionner, copier, couper, coller et déplacer.

Nous considérons les quatre objets géométriques suivants : cercle, ellipse, carré, rectangle. Il est demandé d'utiliser les propriétés de la généralisation et la spécialisation afin de représenter au mieux ces objets géométriques.

Exercice 2

Un hôtel est composé d'au moins deux chambres. Chaque chambre dispose d'une salle d'eau : douche ou bien baignoire. Un hôtel héberge des personnes. Il peut employer du personnel et il est impérativement dirigé par un directeur. On ne connaît que le nom et le prénom des employés, des directeurs et des occupants. Certaines personnes sont des enfants et d'autres des adultes (faire travailler des enfants est interdit).

Un hôtel a les caractéristiques suivantes : une adresse, un nombre de pièces et une catégorie. Une chambre est caractérisée par le nombre et de lits qu'elle contient, son prix et son numéro. On veut pouvoir savoir qui occupe quelle chambre à quelle date. Pour chaque jour de l'année, on veut pouvoir calculer le loyer de chaque chambre en fonction de son prix et de son occupation (le loyer est nul si la chambre est inoccupée). La somme de ces loyers permet de calculer le chiffre d'affaires de l'hôtel entre deux dates.

3.2 DIAGRAMME DE STRUCTURE COMPOSITE

Le diagramme de structure composite permet de décrire des collaborations d'instances (de classes, de composants…) constituant des fonctions particulières du système à développer.

3.2.1 Collaboration

Une collaboration représente un assemblage de rôles d'éléments qui interagissent en vue de réaliser une fonction donnée. Il existe deux manières de représenter une collaboration :

- représentation par une collaboration de rôles,
- représentation par une structure composite : le diagramme de structure composite.

3.2.2 Représentation et exemples

Représentation par une collaboration de rôles

Dans ce cas, une collaboration est formalisée par une ellipse en pointillé dans laquelle on fait figurer les rôles des éléments qui interagissent en vue de réaliser la fonction souhaitée (fig. 29). Dans cet exemple, la fonction *Persistance objets métier* résulte d'une collaboration entre deux rôles d'éléments :

- mapping : classeMétier,
- stockage : tableBDD.

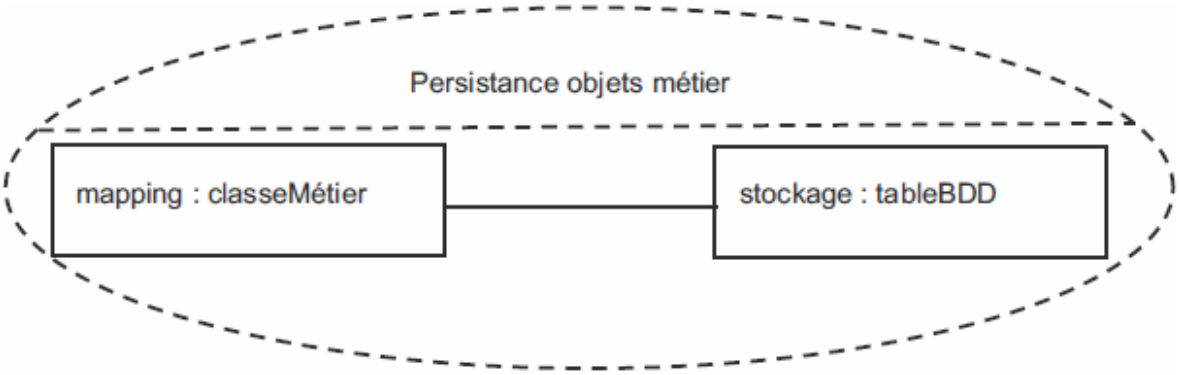


Figure 29: Exemple de représentation d'une structure composite à l'aide d'une collaboration de rôles

Représentation par un diagramme de structure composite

Cette nouvelle représentation (fig. 30) permet de montrer plus explicitement les éléments de la collaboration :

• la collaboration représentée par une ellipse en pointillé ;
• les éléments participant à la collaboration (classe, composant…) représentés à l'extérieur de la collaboration ;
• les rôles considérés dans chaque participation représentés sur les liens entre les éléments participants et la collaboration.

Dans cet exemple, la fonction Persistance objets métier résulte d'une collaboration entre la classe ClasseMétier considérée suivant le rôle mapping et la classe TableBDD considérée suivant le rôle stockage.

Cette représentation permet aussi de préciser les seuls attributs des classes participantes qui sont considérés suivant les rôles pris en compte.



Figure 30: Exemple de représentation de collaboration d'instances

4 ème chapitre: Diagrammes UML de modélisation dynamique

4.1 DIAGRAMME DE SÉQUENCE (DSE)

4.1.1 Présentation générale et concepts de base

L'objectif du diagramme de séquence est de représenter les interactions entre objets en indiquant la chronologie des échanges. Cette représentation peut se réaliser par cas d'utilisation en considérant les différents scénarios associés.

Un diagramme de séquence se représente globalement dans un grand rectangle avec indication du nom du diagramme en haut à gauche comme indiqué à la figure 31.

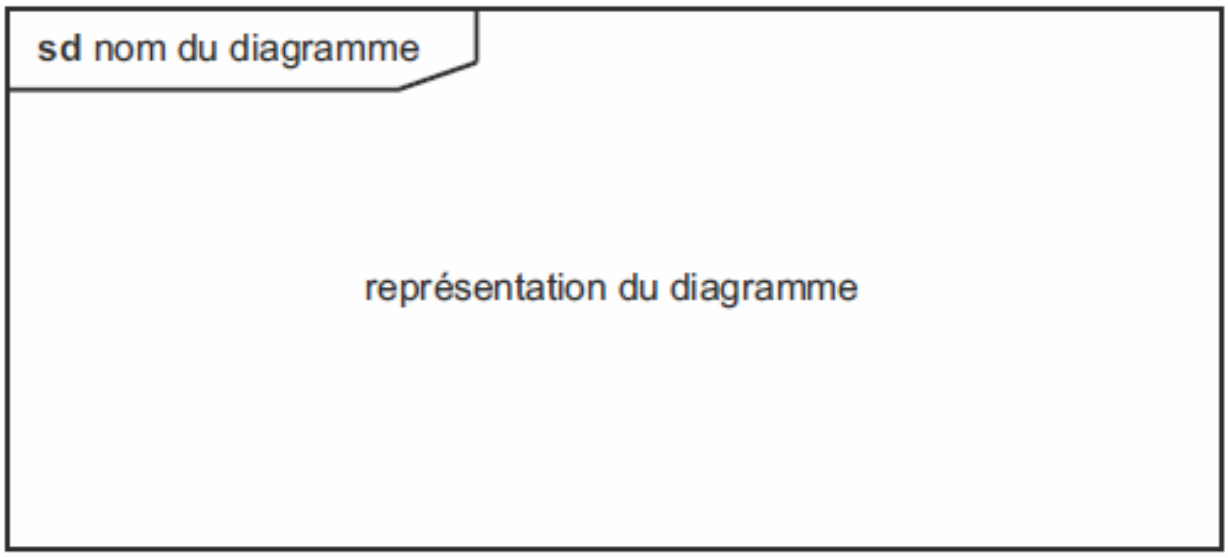


Figure 31: Formalisme général du cadre d'un diagramme de séquence.

4.1.2 Ligne de vie

Une ligne de vie représente l'ensemble des opérations exécutées par un objet. Un message reçu par un objet déclenche l'exécution d'une opération. Le retour d'information peut être implicite (cas général) ou explicite à l'aide d'un message de retour.

4.2 Opérations particulières

4.2.1 Création et destruction d'objet

Si un objet est créé par une opération, celui-ci n'apparaît qu'au moment où il est créé. Si l'objet est détruit par une opération, la destruction se représente par « X ». Un exemple type est donné à la figure 32.

Il est aussi possible dans certains outils de modélisation d'indiquer plus simplement la création d'une nouvelle instance d'objet en utilisant le mot-clé « create »

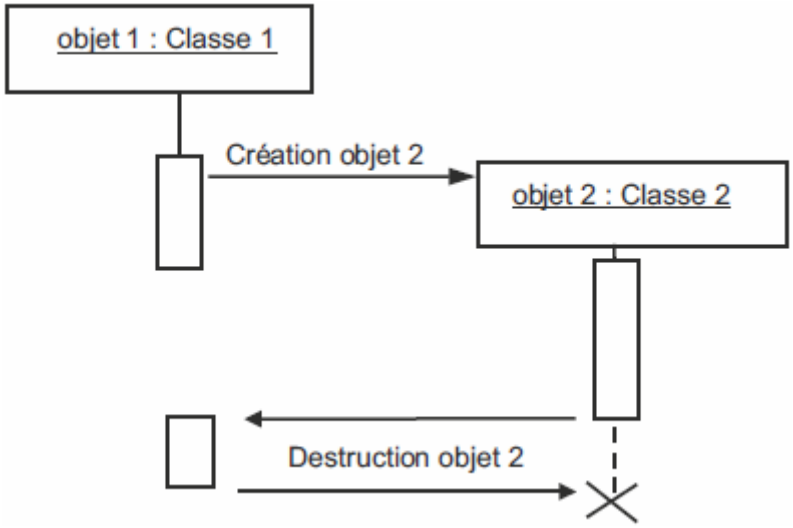


Figure 32: Exemple type de création et de destruction d'objet

4.2.2 Contrainte temporelle

Des contraintes de chronologie entre les messages peuvent être spécifiées. De plus lorsque l'émission d'un message requiert une certaine durée, il se représente sous la forme d'un trait oblique. Un exemple général de **contrainte temporelle** est donné à la figure 33.

Lorsque le diagramme de séquence est utilisé pour représenter un sous-ensemble du logiciel à réaliser, il est possible d'indiquer le pseudo-code exécuté par un objet pendant le déroulement d'une opération.

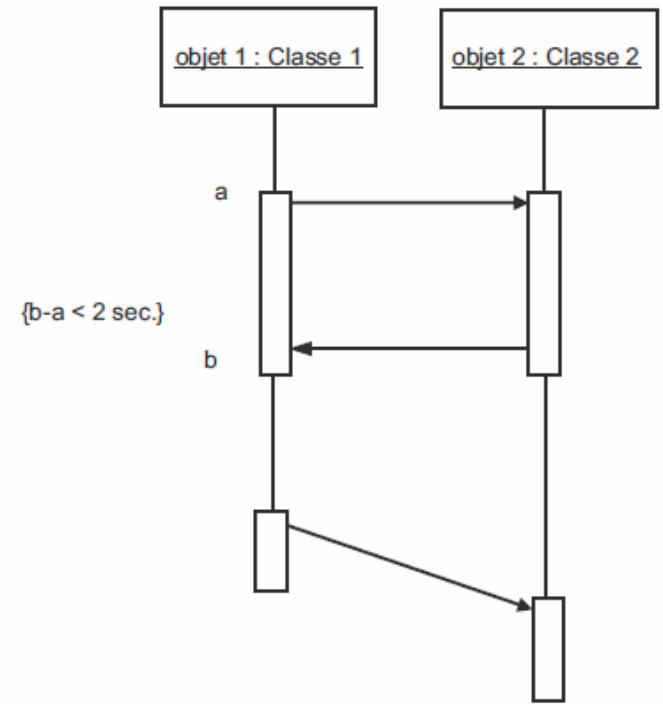


Figure 33: Exemple type de représentation de contrainte temporelle

4.3 Fragment d'interaction

Dans un diagramme de séquence, il est possible de distinguer des sous-ensembles d'interactions qui constituent des fragments.

Un **fragment d'interaction** se représente globalement comme un diagramme de séquence dans un rectangle avec indication dans le coin à gauche du nom du fragment.

Un port d'entrée et un port de sortie peuvent être indiqués pour connaître la manière dont ce fragment peut être relié au reste du diagramme comme le montre la figure 34 Dans le cas où aucun port n'est indiqué c'est l'ensemble du fragment qui est appelé pour exécution.

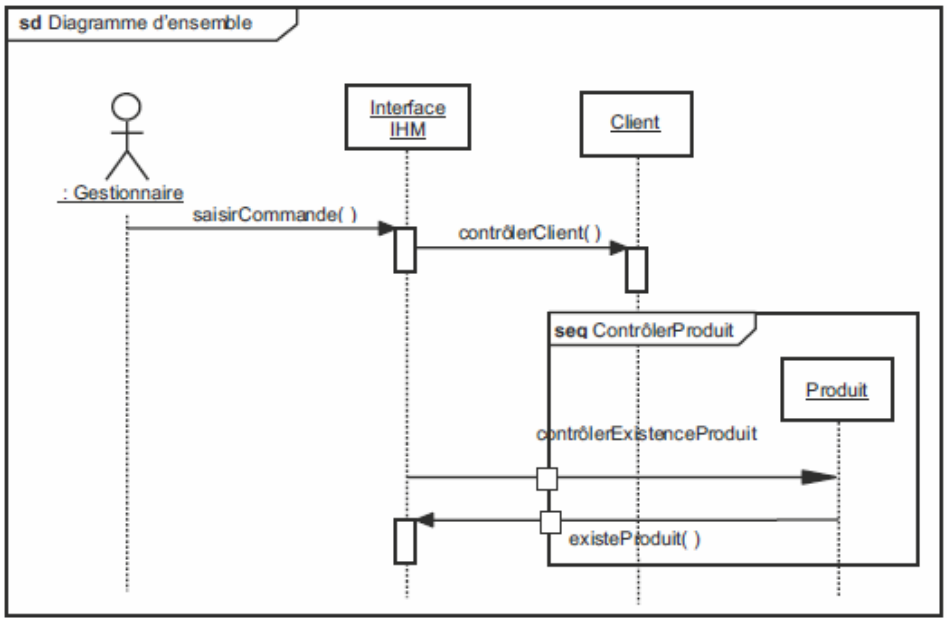


Figure 34:Exemple de fragment d'interaction avec port d'entrée et de sortie

Dans l'exemple proposé (fig. 34), le fragment « ContrôlerProduit » est représenté avec un port d'entrée et un port de sortie.

Un fragment d'interaction dit combiner correspond à un ensemble d'interaction auquel on applique un opérateur. Un fragment combiné se représente globalement comme un diagramme de séquence avec indication dans le coin à gauche du nom de l'opérateur.

Treize opérateurs ont été définis dans UML : alt, opt, loop, par, strict/weak, break, ignore/ consider, critical, negative, assertion et ref.

4.3.1 Opérateur alt

L'opérateur **alt** correspond à une instruction de test avec une ou plusieurs alternatives possibles. Il est aussi **permis** d'utiliser les clauses de type **sinon**.

L'opérateur alt se représente dans un fragment possédant au moins deux parties séparées par des pointillés. L'exemple donné (fig. 35) montre l'équivalent d'un test à deux conditions explicites.

4.3.2 Opérateur opt

L'opérateur **opt** (optional) correspond à une instruction de test **sans alternative (sinon)**.

L'opérateur opt se représente dans un fragment possédant une seule partie (fig. 36).

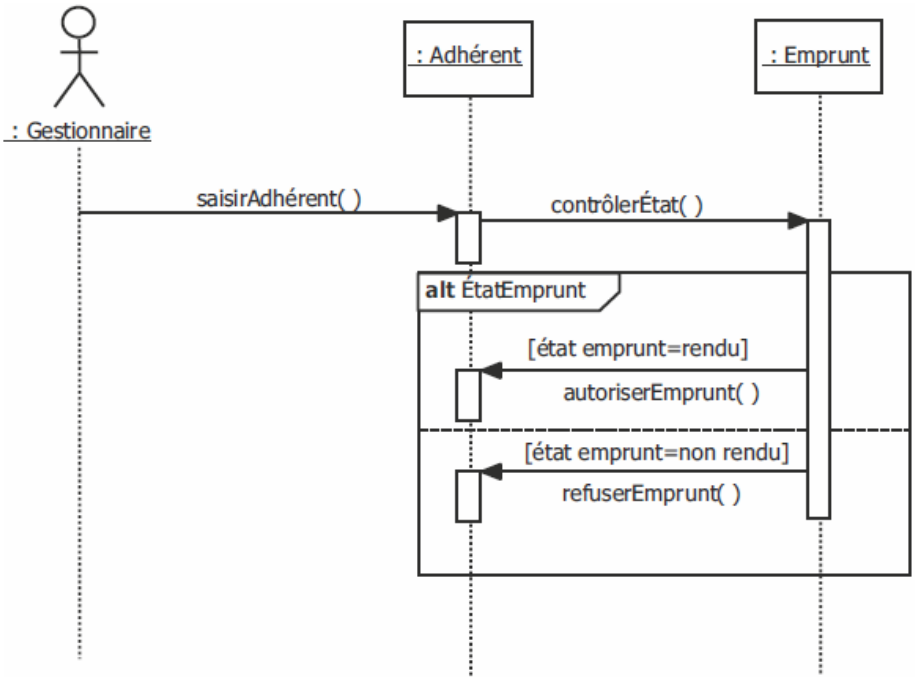


Figure 35: Exemple de fragment d'interaction avec l'opérateur alt

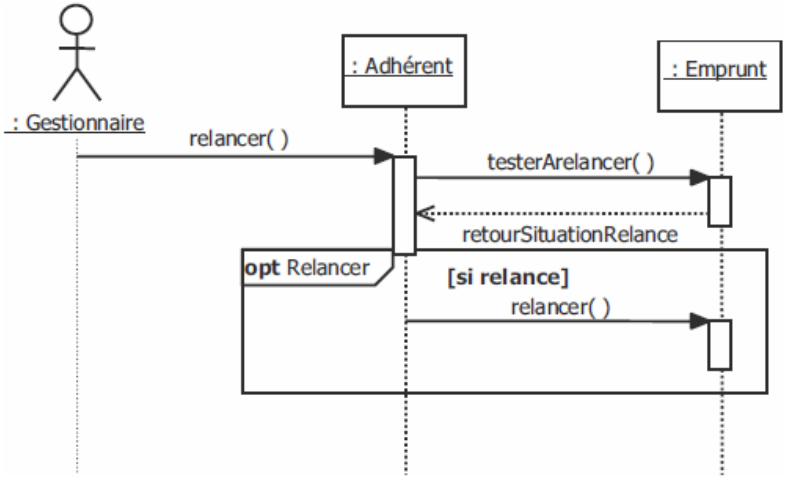


Figure 36: Exemple de fragment d'interaction avec l'opérateur opt

4.3.3 Opérateur loop

L'opérateur **loop** correspond à une instruction de boucle qui permet d'exécuter une séquence d'interaction tant qu'une condition est satisfaite.

Il est possible aussi d'utiliser une condition portant sur un nombre minimum et maximum d'exécution de la boucle en écrivant : loop min, max. Dans ce cas, la boucle s'exécutera au minimum min fois et au maximum max fois. Il est aussi possible de combiner l'option min/max avec la condition associée à la boucle.

L'opérateur loop se représente dans un fragment possédant une seule partie et englobant toutes les interactions faisant partie de la boucle. Un exemple est donné à la figure 37.

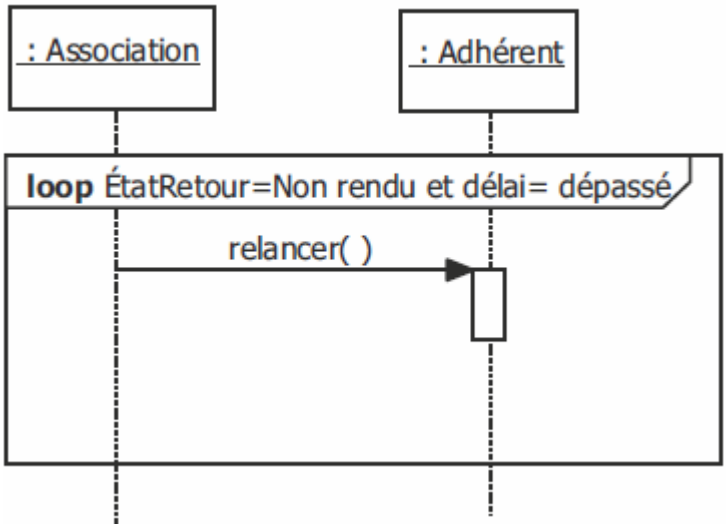


Figure 37:Exemple de fragment d'interaction avec l'opérateur loop

4.3.4 Opérateur par

L'opérateur **par** (**parallel**) permet de représenter deux séries d'interactions qui se déroulent en parallèle.

L'opérateur par se représente dans un fragment possédant deux parties séparées par une ligne en pointillé. C'est un opérateur qui est à plutôt utilisé dans l'informatique temps réel et c'est pour cela que j'ai donné qu'un exemple type (fig. 38).

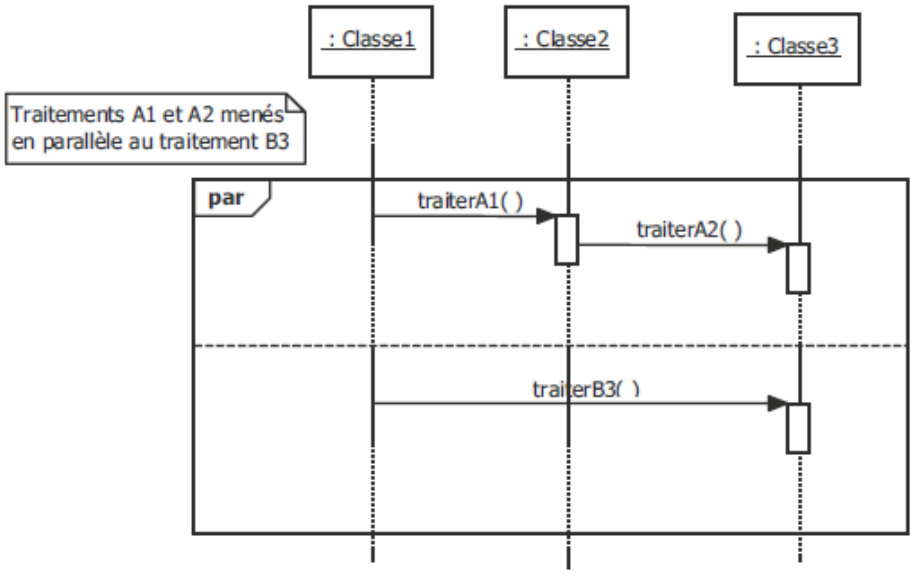


Figure 38: Exemple de fragment d'interaction avec l'opérateur par

4.3.5 Opérateurs strict et weak sequencing

Les opérateurs **srict** et **weak** permettent de représenter une série d'interactions dont certaines s'opèrent sur des objets indépendants :

• L'opérateur strict est utilisé quand **l'ordre d'exécution** des opérations doit être strictement respecté.

• L'opérateur weak est utilisé quand **l'ordre d'exécution** des opérations n'a pas d'importance.
L'exemple présenté figure 39 montre que les opérations A1, A2, B1, B2 et A3 doivent être exécutées dans cet ordre puisqu'elles font partie du fragment d'interaction comportant l'opérateur strict.

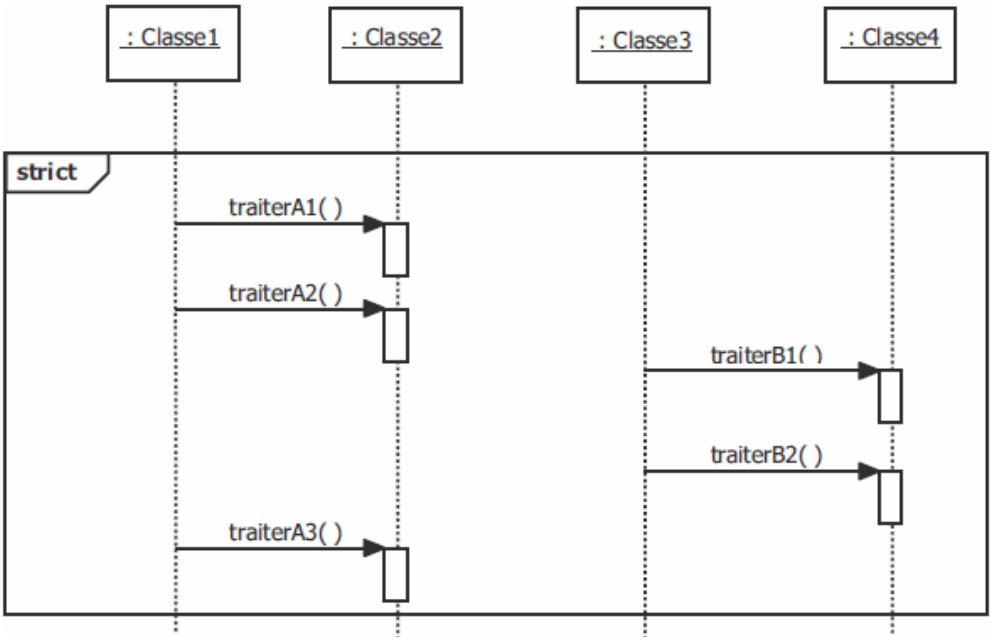


Figure 39: Exemple de fragment d'interaction avec l'opérateur strict

4.3.6 Opérateur break

L'opérateur **break** permet de représenter une situation exceptionnelle correspondant à un scénario de rupture par rapport au scénario général. **Le scénario de rupture s'exécute si la condition de garde est satisfaite.**
L'exemple présenté figure 40 montre que les opérations annulerOp1( ), annulerOp2( ) et afficherAide( ) ne seront exécutées que si la touche F1 est activée sinon le fragment est ignoré et la séquence de traitement passe directement de l'opération Op2( ) à l'opération Op3( ).

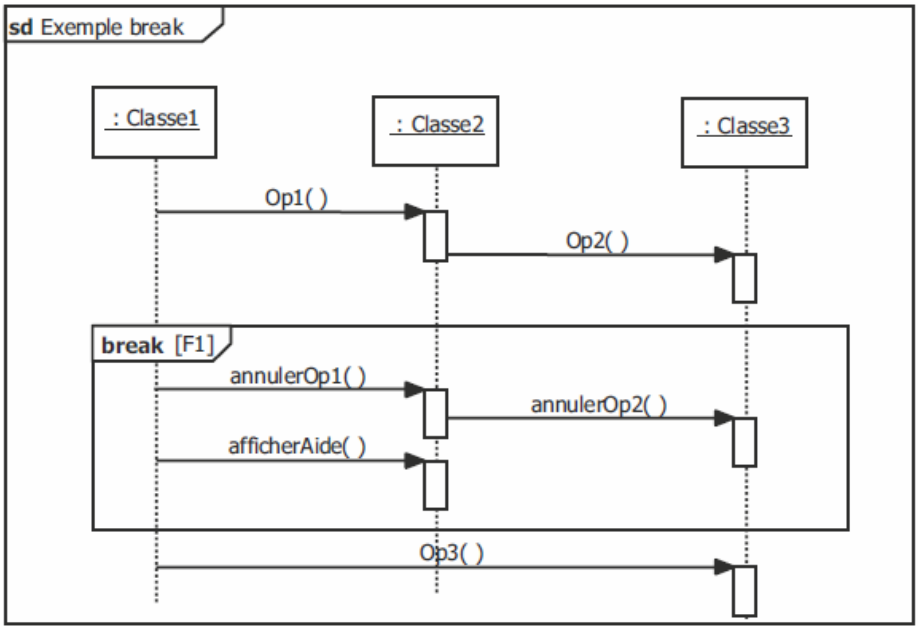


Figure 40: Exemple de fragment d'interaction avec l'opérateur break

4.4 Opérateurs ignore et consider

Les opérateurs **ignore** et **consider** sont utilisés pour des fragments d'interactions dans lesquels on veut montrer que certains messages peuvent **être soit absents sans avoir d'incidence sur le déroulement des interactions (ignore), soit obligatoirement présents (consider).**

L'exemple présenté figure 41 montre que :

• dans le fragment consider, les messages Op1, Op2 et Op5 doivent être obligatoirement présents lors de l'exécution du fragment sinon le fragment n'est pas exécuté,
• dans le fragment ignore, les messages Op2 et Op3 peuvent être absents lors de l'exécution du fragment.

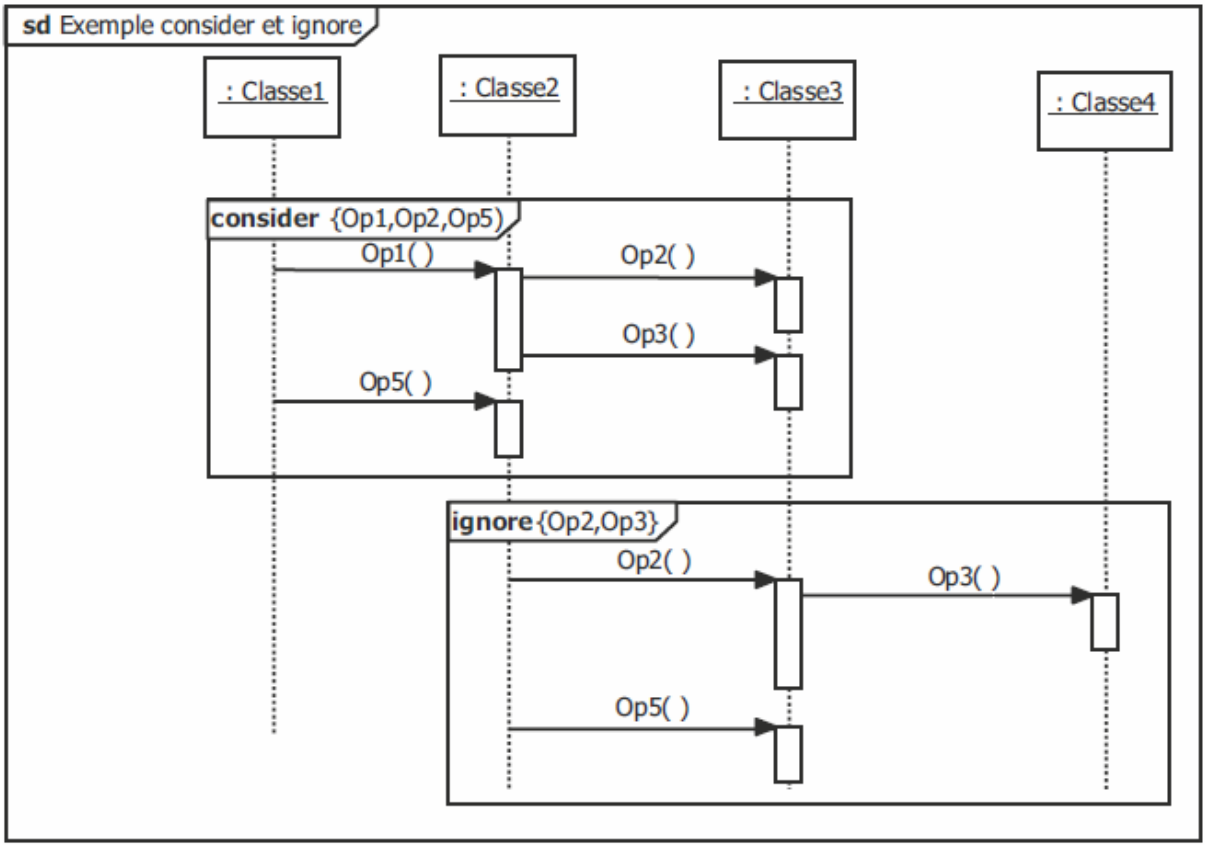


Figure 41: Exemple de fragment d'interaction

4.4.1 Opérateur critical

L'opérateur **critical** permet d'indiquer qu'une séquence d'interactions ne peut être interrompue compte tenu du caractère critique des opérations traitées. On considère que le traitement des interactions comprises dans la séquence critique est atomique.

L'exemple présenté figure 42 montre que les opérations Op1( ), Op2( ) et Op3( ) du fragment **critical doivent s'exécuter sans interruption**.

4.4.2 Opérateur negative

L'opérateur **neg** (negative) permet d'indiquer qu'une **séquence d'interactions est invalide**.

L'exemple présenté figure 43 montre que les opérations Op1( ) et Op2( ) du fragment neg sont invalides. Une erreur sera déclenchée dans ce cas à l'exécution du fragment.

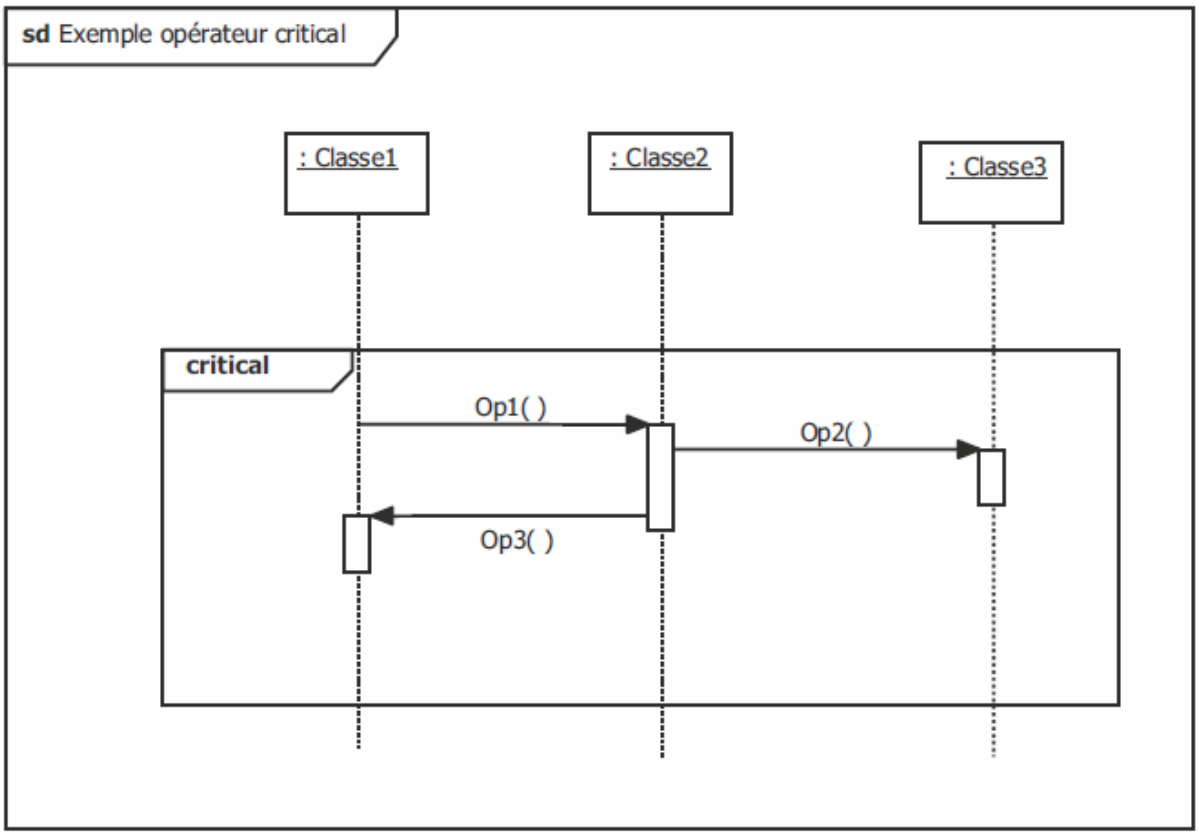


Figure 42: Exemple de fragment d'interaction avec l'opérateur critical

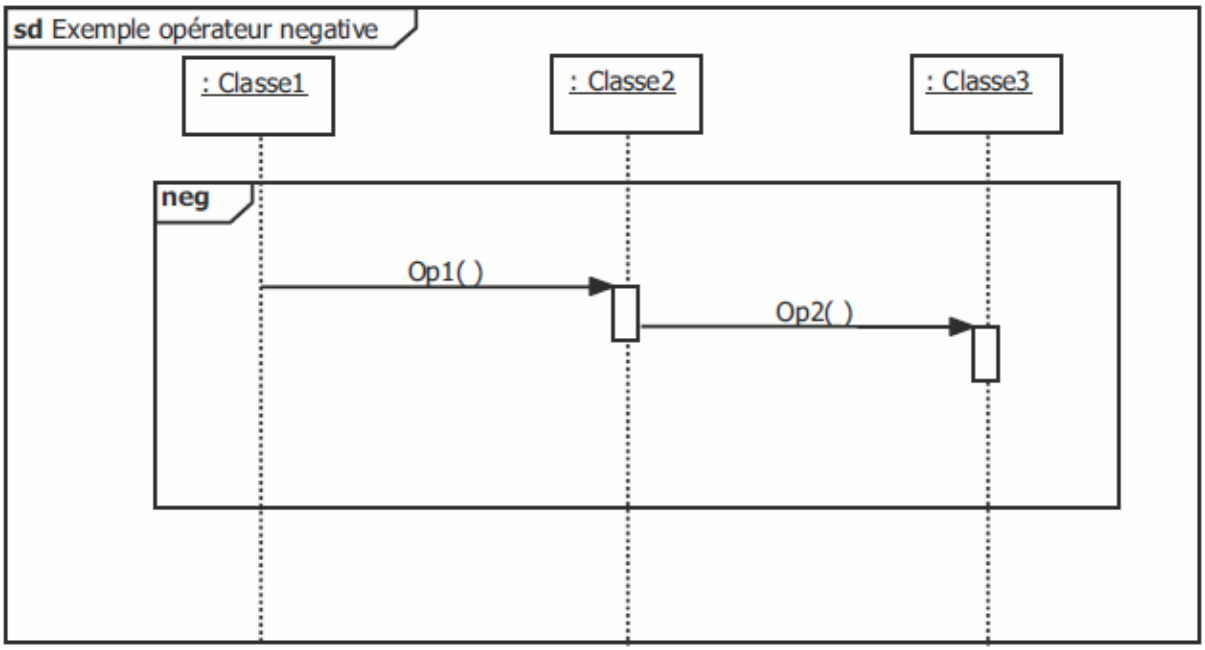


Figure 43: Exemple de fragment d'interaction avec l'opérateur neg

4.4.3 Opérateur assertion

L'opérateur **assert** (assertion) permet d'indiquer qu'**une séquence d'interactions est l'unique séquence possible** en considérant les messages échangés dans le fragment. Toute autre configuration de message est invalide.

L'exemple présenté figure 44 montre que le fragment assert ne s'exécutera que si l'unique séquence de traitement Op1( ), Op2( ) et Op3( ) se réalise en respectant l'ensemble des caractéristiques de ces opérations (paramètre d'entrée, type de résultat…). Toute autre situation sera considérée invalide.

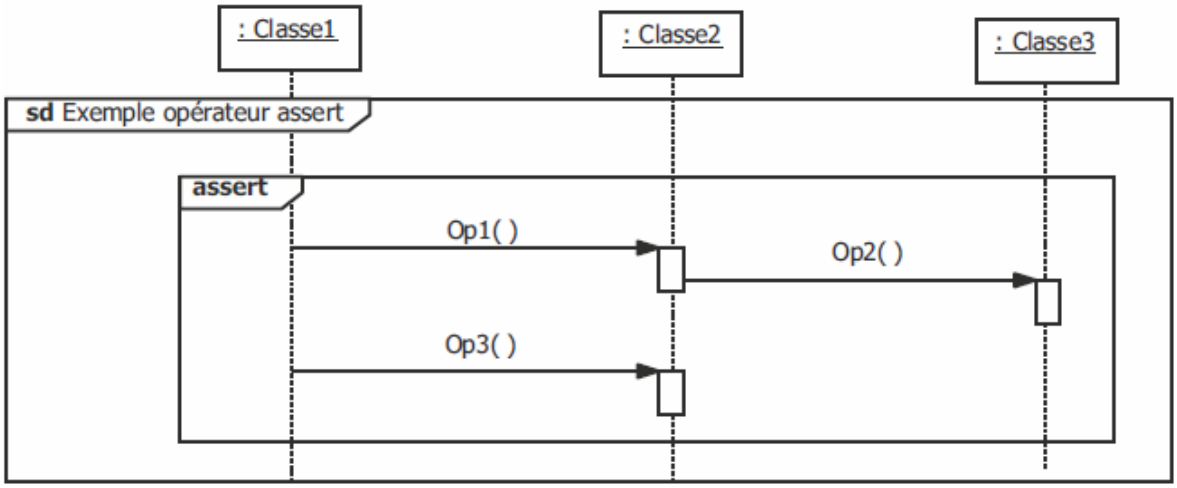


Figure 44: Exemple de fragment d'interaction avec l'opérateur assert

4.4.4 Opérateur ref

L'opérateur ref permet d'appeler une séquence d'interactions décrite par ailleurs constituant ainsi une sorte de sous-diagramme de séquence.
L'exemple présenté figure. 45 montre que l'on fait appel à un fragment « Contrôle,des droits » qui est décrit par ailleurs.

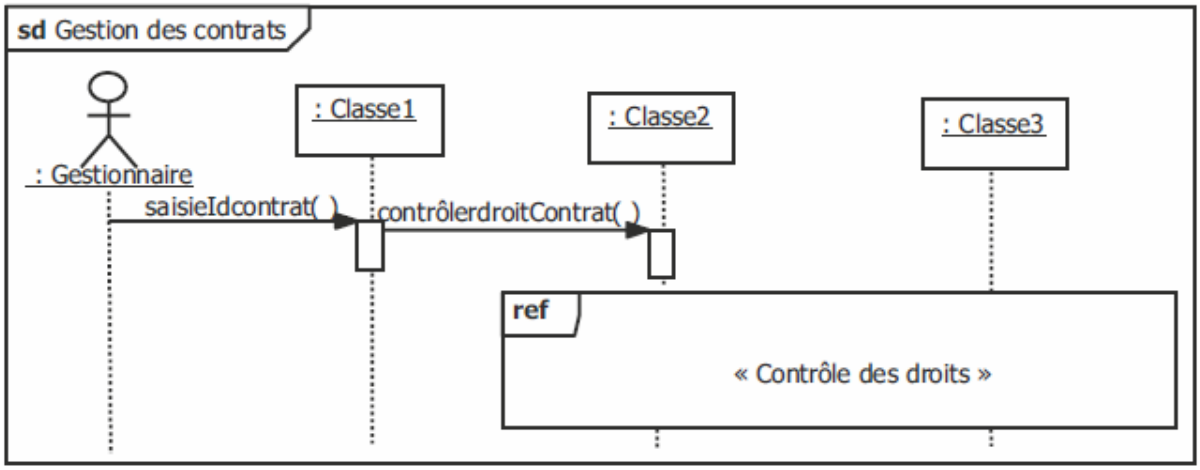


Figure 45: Exemple de fragment d'interaction avec l'opérateur ref

4.5 Autre utilisation du diagramme de séquence

Le diagramme de séquence peut être aussi utilisé pour documenter un cas d'utilisation. Les interactions entre objets représentent, dans ce cas, des flux d'informations échangés et non pas de véritables messages entre les opérations des objets. Un exemple de cette utilisation du diagramme de séquence est donné à la figure 46.

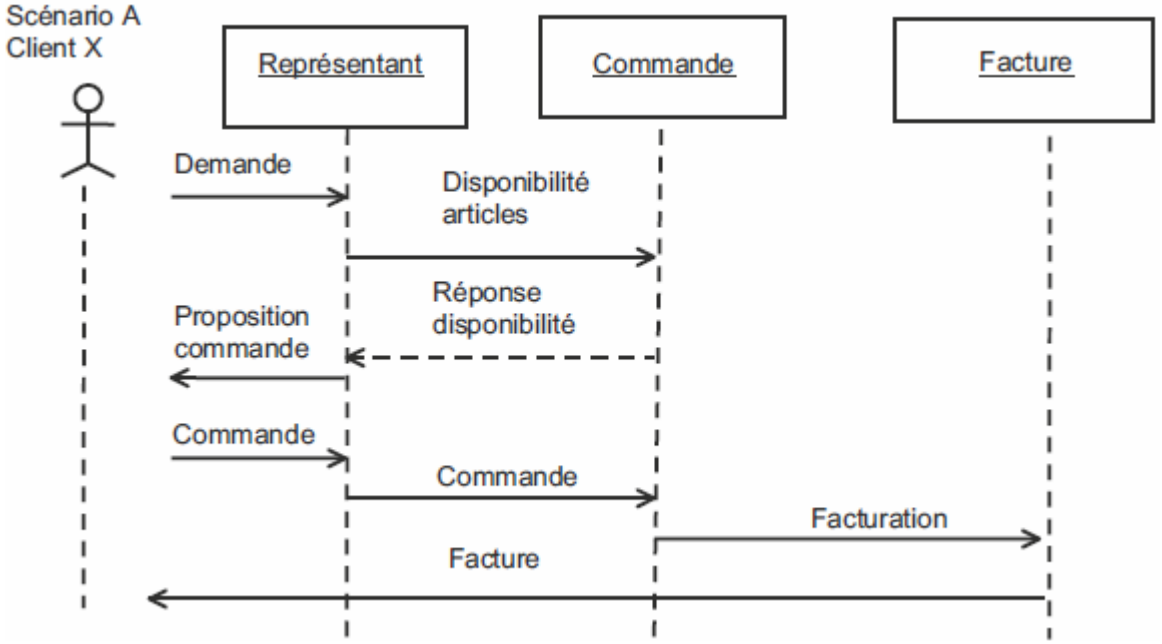
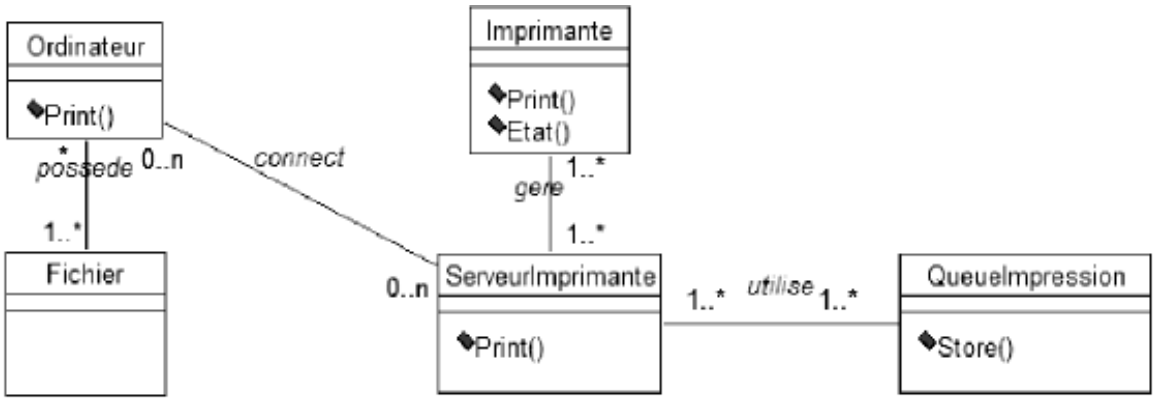


Figure 46: Exemple de diagramme de séquence associé à un cas d'utilisation

4.5.1 Travaux dirigés

Exercice 1



A partir du diagramme de classe ci-dessus rédigez un diagramme de séquence pour modéliser le scénario où un utilisateur voudrait imprimer un fichier.

Exercice 2

On considère le cas d'utilisation gestion des emprunts. Le fonctionnement de la bibliothèque est le suivant : une bibliothèque propose à ses adhérents des ouvrages littéraires. Les ouvrages peuvent être présents en plusieurs exemplaires. Un adhérent peut emprunter jusqu'à trois livres. Le scénario nominal d'emprunt est comme suit : Le bibliothécaire recherche l'adhérent dans le système, vérifie si l'adhérent est autorisé à emprunter, recherche l'ouvrage, vérifie s'il y a un exemplaire disponible, décrémente le nombre d'exemplaires et attribue l'exemplaire à l'adhérent.

Représenter par un diagramme de séquence système, le scénario nominal d'emprunt.

4.6 DIAGRAMME GLOBAL D'INTERACTION

4.6.1 Présentation générale et concepts de base

Le **diagramme global d'interaction** permet de représenter une vue générale des interactions décrites dans le diagramme de séquence et des flots de contrôle décrits dans le diagramme d'activité.

Le diagramme global d'interaction privilégie la vue générale des flux de contrôle dans lesquels les noeuds sont des interactions ou des utilisations d'interactions (opérateur ref).

Autrement dit, le diagramme global d'interaction est un diagramme d'activité dans lequel on représente des fragments d'interaction ou des utilisations d'interactions. Ainsi, il est possible de représenter :

• des choix de fragments d'interactions (fusion) ;
• des déroulements parallèles de fragments d'interactions (débranchement et jonction) ;
• des boucles de fragments d'interaction.

Les lignes de vie concernées par le diagramme global d'interaction peuvent être citées dans l'en-tête du diagramme mais ne sont pas à représenter graphiquement.

Concepts manipulés

Le diagramme global d'interaction utilise les concepts du diagramme d'activité auquel on ajoute deux compléments :

- Les fragments d'interaction du diagramme de séquence – Il s'agit comme le montre la figure 47 de la notion de fragment d'interaction vue dans le diagramme de séquence mais qui ne doit pas être détaillé à ce niveau.

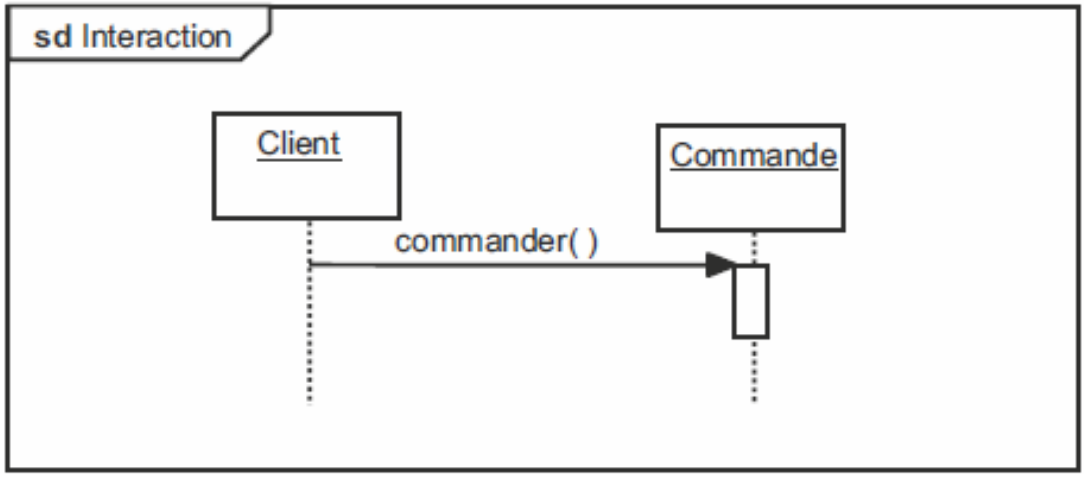


Figure 47: Exemple de fragment d'interaction

- Les utilisations de fragments d'interaction – Il est aussi possible de faire appel à des fragments d'interaction à l'aide de l'opérateur ref comme le montre la figure 48.

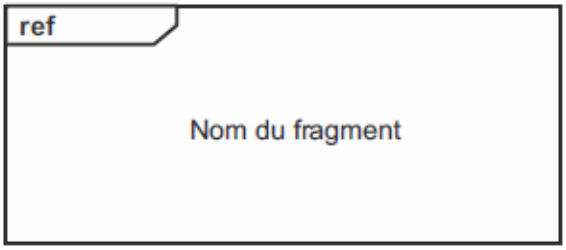


Figure 48: Exemple de fragment d'interaction avec l'opérateur ref

4.6.2 Représentation et exemple

La figure 49 donne un exemple de diagramme global d'interaction.

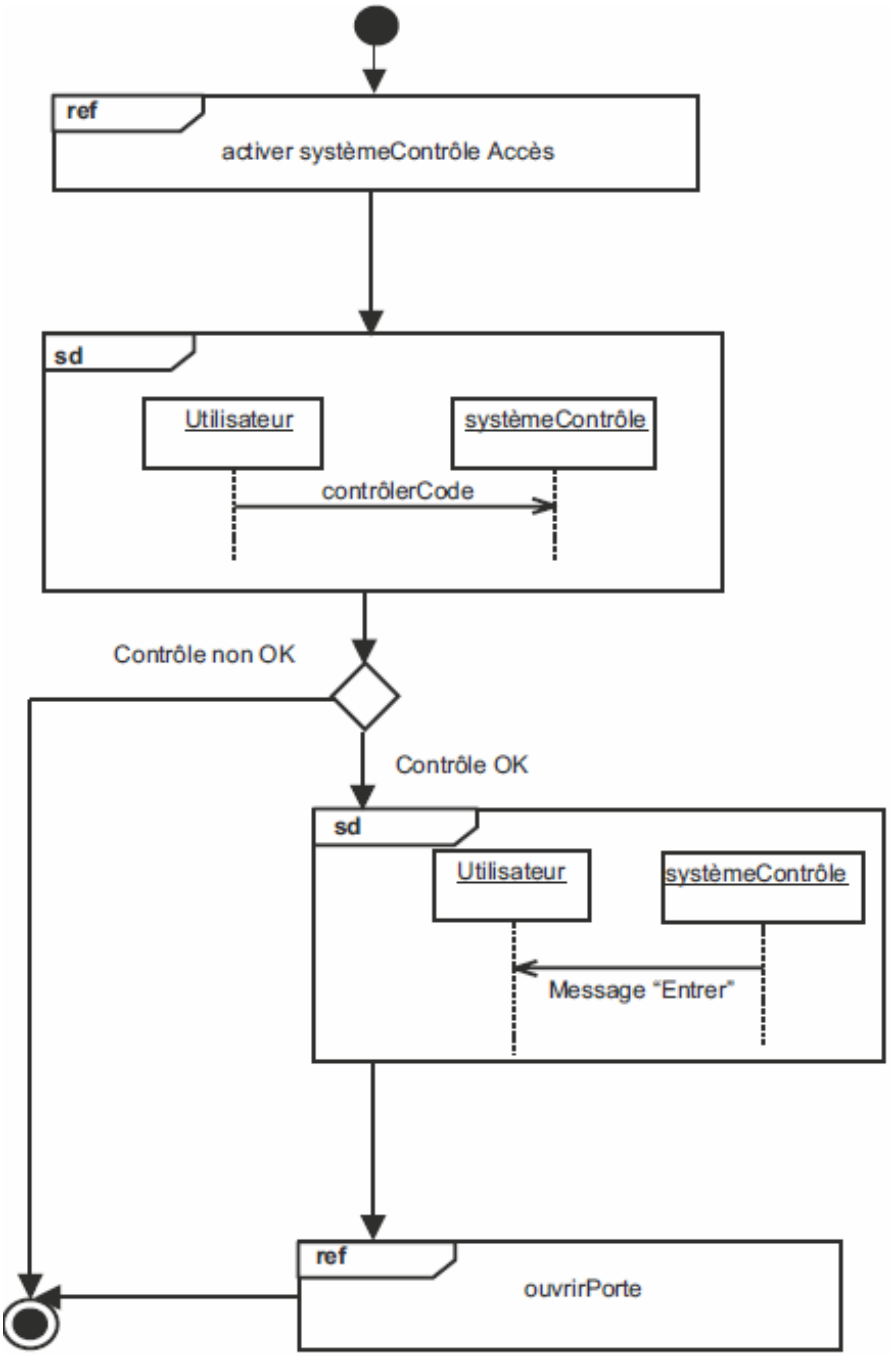


Figure 49: Exemple de diagramme global d'interaction

4.7 DIAGRAMME DE COMMUNICATION

4.7.1 Présentation générale et concepts de base

Le diagramme de communication constitue une autre représentation des interactions que celle du diagramme de séquence. En effet, le diagramme de communication met plus l'accent sur l'aspect spatial des échanges que l'aspect temporel.

4.7.1.1 *Rôle*

Chaque participant à un échange de message correspondant à une ligne de vie (objet) dans le diagramme de séquence se représente sous forme d'un rôle dans le diagramme de communication. Un rôle est identifié par :

<nom de rôle> : <nom du type>

Une des deux parties de cette identification est obligatoire ainsi que le séparateur « : ». Le nom du rôle correspond au nom de l'objet dans le cas où l'acteur ou la classe ont un rôle unique par rapport au système. Le nom du type correspond au nom de la classe lorsque l'on manipule des objets.

Exemple

administrateur : utilisateur

Pour un utilisateur qui est vu au travers de son rôle d'administrateur.

4.7.1.2 *Message*

Un message correspond à un appel d'opération effectué par un rôle émetteur vers un rôle récepteur. Le sens du message est donné par une flèche portée au-dessus du lien reliant les participants au message (origine et destinataire). Chaque message est identifié par :

<numéro> : nom ( )

Plus précisément l'identification d'un message doit respecter la syntaxe suivante **: [n° du message préc. reçu] « . » n° du message *[clause d'itération] [condition] « : » nom du message.**

- Numéro du message précédent reçu : permet d'indiquer la chronologie des messages.
- Numéro du message : numéro hiérarchique du message de type 1.1, 1.2… avec utilisation de lettre pour indiquer la simultanéité d'envoi de message.
- Clause d'itération : indique si l'envoi du message est répété. La syntaxe est *[spécification de l'itération].
- Condition : indique si l'envoi du message est soumis à une condition à satisfaire.

Exemples

1.2.1 * [3 fois] pour un message à adresser trois fois de suite.
1.2a et 1.2b pour deux messages envoyés en même temps.
Exemple récapitulatif de désignation de message :

1.2a.1.1[si t > 100] : lancer( )

Ce message signifie :
1.2a : numéro du message reçu avant l'envoi du message courant.
1.1 : numéro de message courant à envoyer.
[si t > 100] : message à envoyer si t > 100.
lancer( ) : nom du message à envoyer.

Formalisme et exemple

Les rôles correspondent à des objets. Le lien entre les rôles est représenté par un trait matérialisant le support des messages échangés. La figure 22 donne le formalisme de base du diagramme de communication.

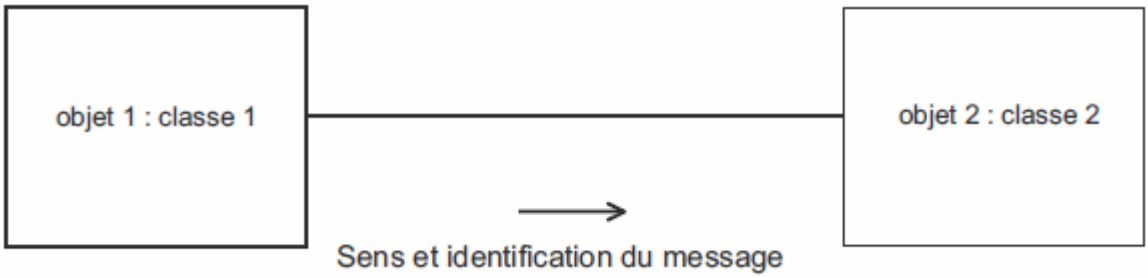


Figure 50: Formalisme de base du diagramme de communication

Un premier exemple de diagramme de communication est donné à la figure 23.

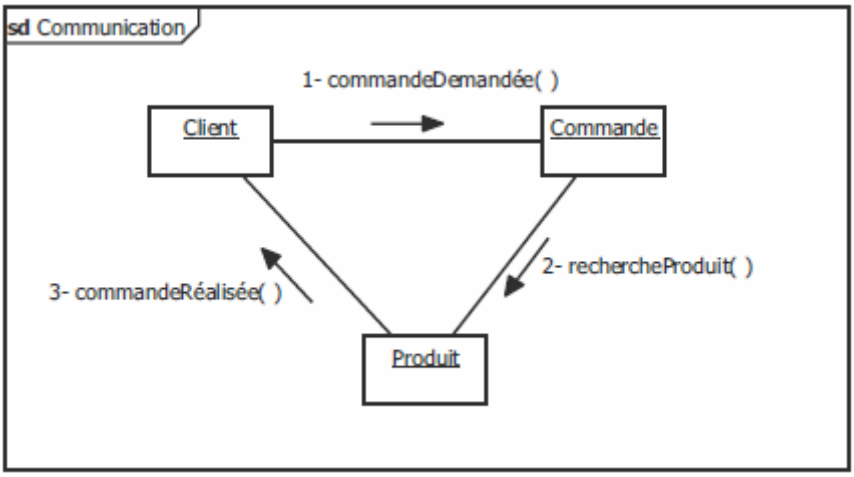


Figure 51: Exemple 1 de diagramme de communication

Un deuxième exemple de diagramme de communication est donné à la figure 24.

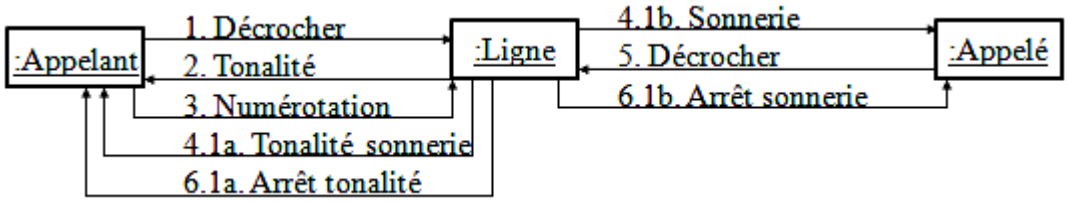


Figure 52: Exemple 2 de diagramme de communication

Un deuxième exemple de diagramme de communication est donné à la figure 25.

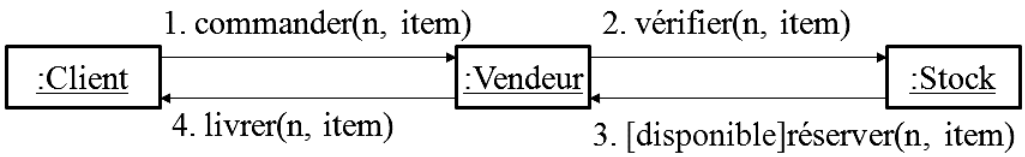
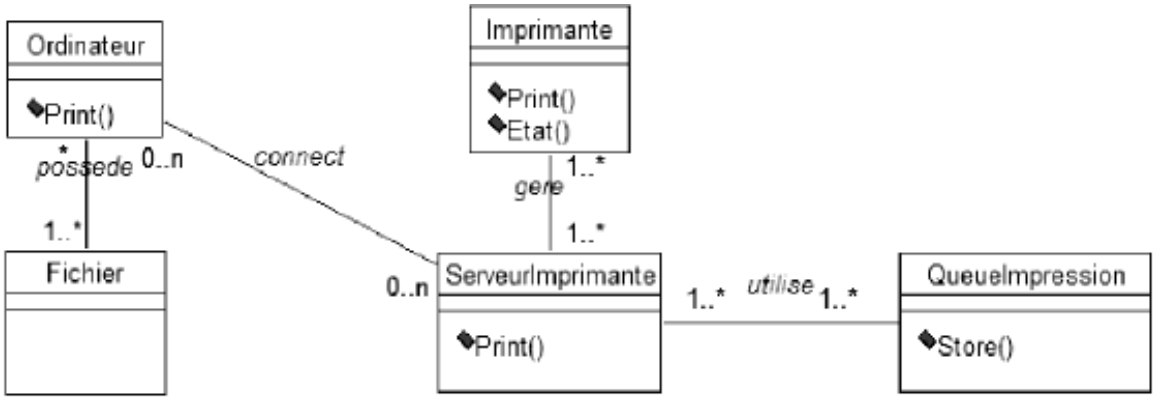


Figure 53: Exemple 3 de diagramme de communication

4.7.2 Travaux pratiques



A partir du diagramme de classe ci-dessus rédigez un diagramme de communication pour modéliser le scénario où un utilisateur voudrait imprimer un fichier.

4.8 DIAGRAMME D'ÉTAT-TRANSITION (DET)

4.8.1 Présentation générale et concepts de base

4.8.1.1 *État-transition et événement*

L'état d'un objet est défini, à un instant donné, par l'ensemble des valeurs de ses propriétés. Seuls certains états caractéristiques du domaine étudié sont considérés.

Le passage d'un état à un autre état s'appelle transition. Un événement est un fait survenu qui déclenche une transition.

Il existe quatre types d'événements :

Type appel de méthode (call) – C'est le type le plus courant que nous traiterons dans la suite de la présentation.

Type signal – Exemple : clic de souris, interruption d'entrées-sorties… La modélisation de la réception ou l'émission d'un signal est traitée dans le diagramme d'activité.

Type changement de valeur (vrai/faux) – C'est le cas de l'évaluation d'une expression booléenne.

Type écoulement du temps – C'est un événement lié à une condition de type after (durée) ou when (date).

Formalisme et exemple

Un objet reste dans un état pendant une certaine durée. La durée d'un état correspond au temps qui s'écoule entre le début d'un état déclenché par une transition i et la fin de l'état déclenché par la transition i+1. Une condition, appelée « garde », peut être associée à une transition.

Le formalisme de représentation d'état-transition est donné à la figure 54.

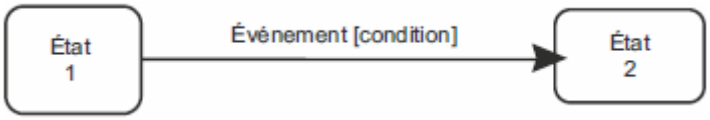


Figure 54: Formalisme d'état-transition

La figure 55 donne un premier exemple d'état-transition. Dans cet exemple, pour un employé donné d'une entreprise, nous pouvons considérer les deux états significatifs suivants : état recruté, état en activité.

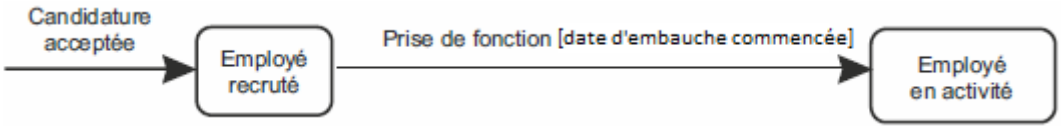


Figure 55: Exemple d'état-transition

4.8.1.2 *Action et activité*

Une action est une opération instantanée qui ne peut être interrompue ; elle est associée à une transition.

Une activité est une opération d'une certaine durée qui peut être interrompue, elle est associée à un état d'un objet.

Formalisme et exemple

Le formalisme de représentation d'état-transition comprenant la représentation d'action et/ou activité est donné à la figure 56.

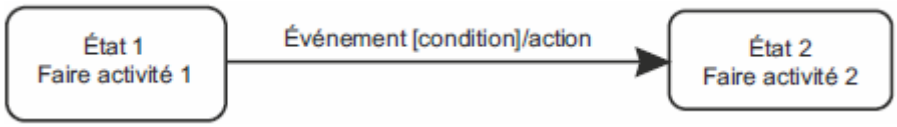


Figure 56: Formalisme d'état-transition avec action et activité

La figure 57 montre un exemple des actions et activités d'états ainsi que la description complète d'une transition.

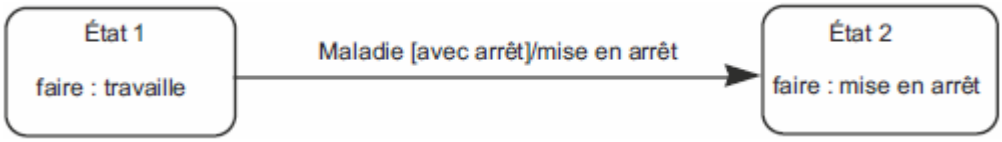


Figure 57: Exemple d'état-transition avec action et activité

4.8.2 Représentation du diagramme d'état-transition d'un objet

L'enchaînement de tous les états caractéristiques d'un objet constitue le diagramme d'état. Un diagramme d'états débute toujours par un état initial et se termine par un ou plusieurs états finaux sauf dans le cas où le diagramme d'états représente une boucle. À un événement peut être associé un message composé d'attributs.

Formalisme et exemple

Le formalisme de représentation des états initial et final est donné à la figure 58

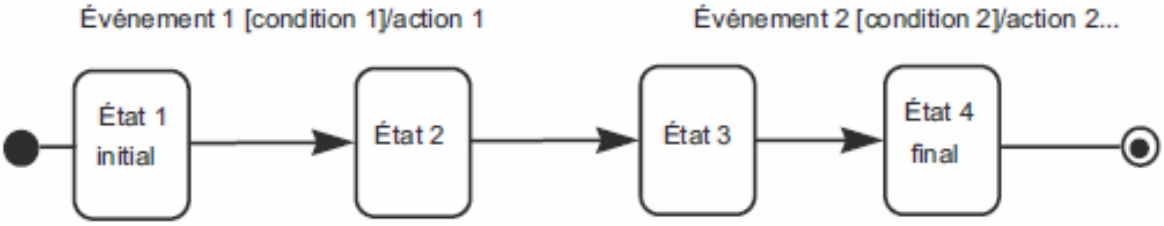


Figure 58: Formalisme de représentation des états initial et final

Afin de nous rapprocher des situations réelles, nous proposons à la figure 59 un premier exemple tiré d'une gestion commerciale qui montre le diagramme d'état-transition de l'objet client.

Nous proposons comme second exemple, à la figure 60, le diagramme d'état-transition de l'objet « personnel » qui se caractérise par trois états :

• En prévision d'arrivée : si la date prévisionnelle est supérieur à la date du jour.

• En activité : état qui correspond à un personnel ayant une date d'arrivée renseignée.

• Parti : état qui correspond à un personnel ayant une date de départ renseignée.

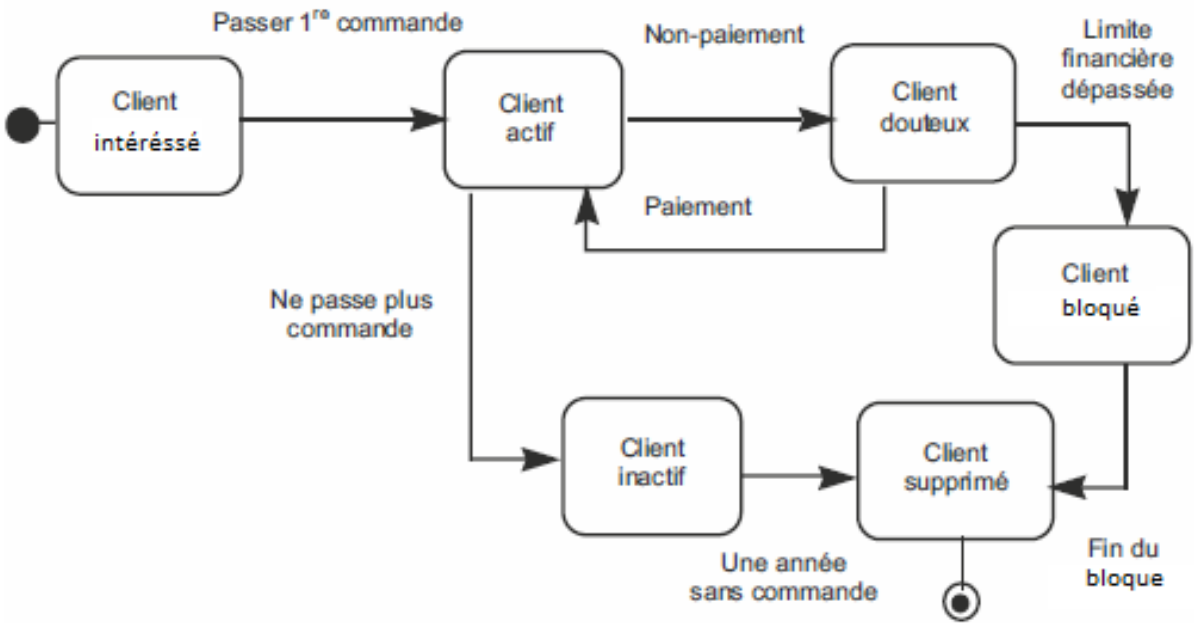


Figure 59: Diagramme d'état-transition de l'objet client d'une gestion commerciale

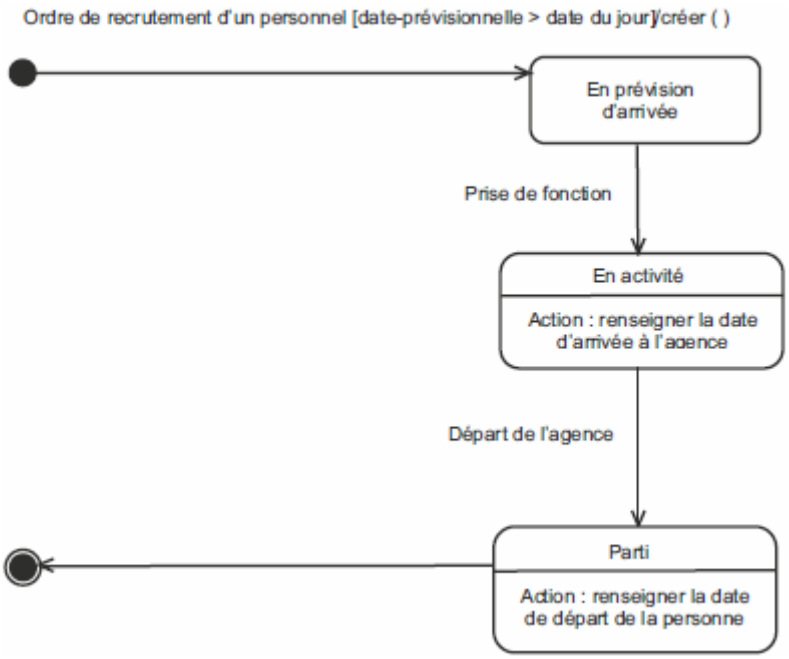


Figure 60: Exemple de diagramme d'état-transition

4.8.3 Compléments sur le diagramme d'état-transition

4.8.3.1 *Composition et décomposition d'état*

Il est possible de décrire un diagramme d'état-transition à plusieurs niveaux. Ainsi, à un premier niveau, le diagramme comprendra des états élémentaires et des états composites. Les états composites seront ensuite décrits à un niveau élémentaire dans un autre diagramme. On peut aussi parler d'état composé et d'état composant.

Formalisme et exemple

Le formalisme de représentation d'états composites est donné à la figure 61.

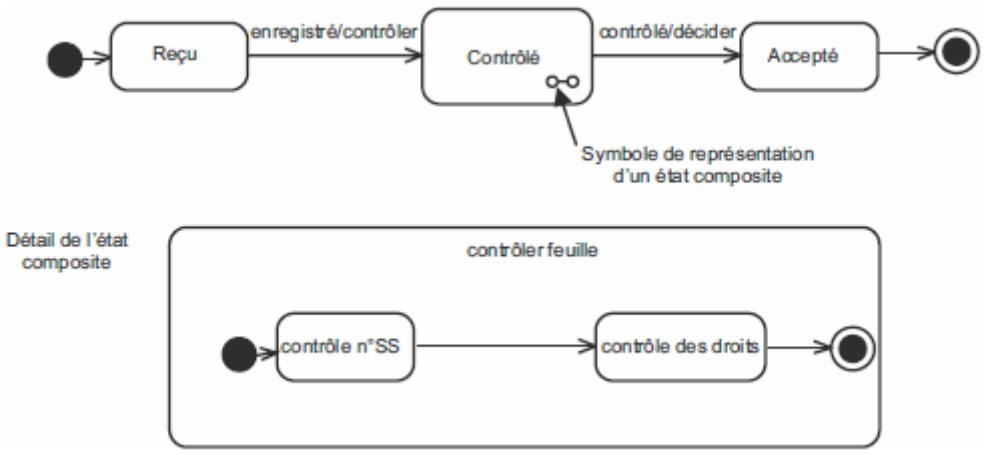


Figure 61: Exemple d'état composite

Dans cet exemple, l'état contrôlé est un état composite qui fait l'objet d'une description individualisée à un second niveau que l'on appelle aussi sous-machine d'état.

4.8.3.2 *Point d'entrée et de sortie*

Sur une sous-machine d'état, il est possible de repérer un point d'entrée et un point de sortie particuliers.

Formalisme et exemple

Le formalisme de représentation d'une sous-machine d'état avec point d'entrée et de sortie est donné à la figure 62.

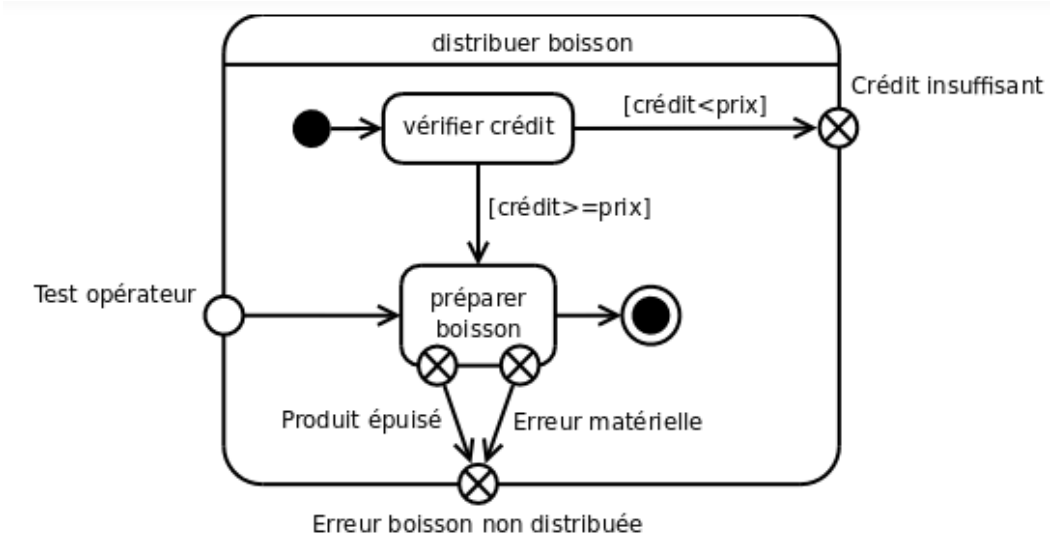


Figure 62: Exemple d'une sous-machine d'état avec point d'entrée et de sortie

4.8.3.3 *Point de jonction*

Lorsque l'on veut relier plusieurs états vers d'autres états, un **point de jonction** permet de décomposer une transition en deux parties en indiquant si nécessaire les gardes propres à chaque segment de la transition.
À l'exécution, un seul parcours sera emprunté, c'est celui pour lequel toutes les conditions de garde seront satisfaites.

Formalisme et exemple

Le formalisme de représentation d'états-transitions avec point de jonction est donné à la figure 63.

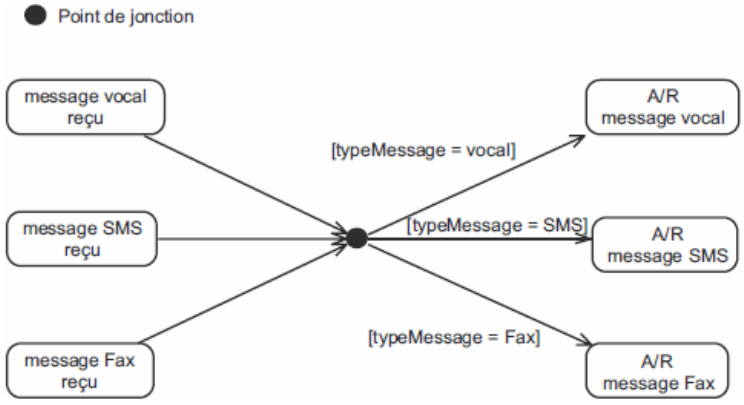


Figure 63: Exemple d'états-transitions avec point de jonction

4.8.3.4 *Point de choix*

Le point de choix se comporte comme un test de type : si condition faire action1 sinon faire action2.

Formalisme et exemple

Le formalisme de représentation d'états composites est donné à la figure 64.

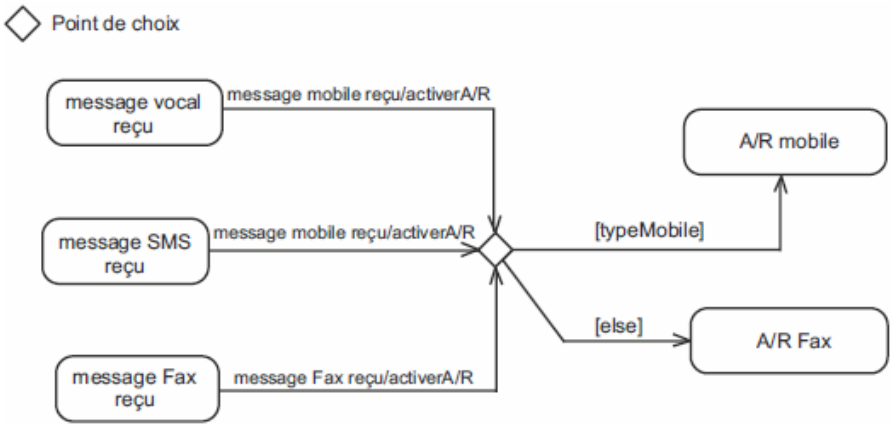


Figure 64:Exemple d'états-transitions avec point de choix

4.8.3.5 *État historique*

La mention de l'historisation d'un état composite permet de pouvoir indiquer la réutilisation du dernier état historisé en cas de besoin.

Formalisme et exemple

Le formalisme de représentation d'états historisés est donné à la figure 65.

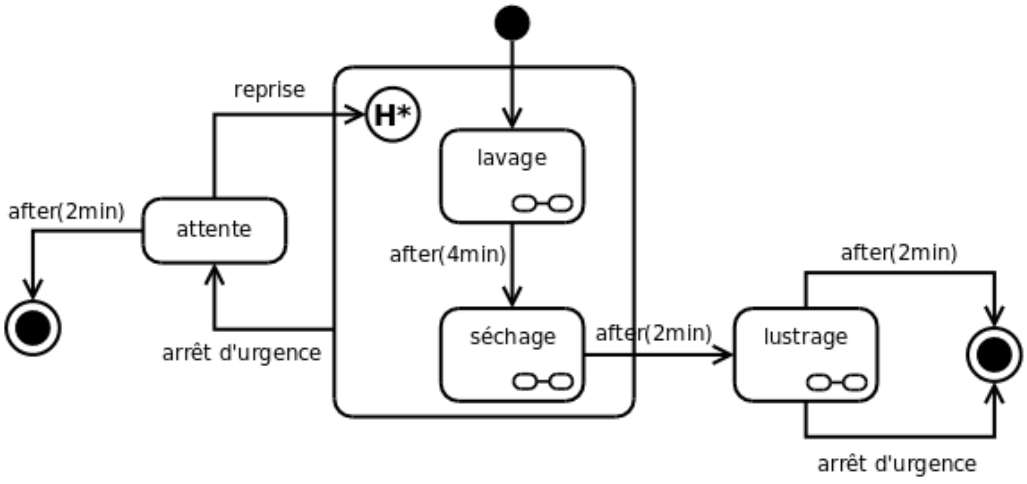


Figure 65: Exemple d'états-transitions historisés

4.8.4 Travaux pratiques

Soit à représenter le diagramme d'état-transition d'un objet personnel en suivant les événements de gestion depuis le recrutement jusqu'à la mise en retraite. Après le recrutement, une personne est considérée en activité dès sa prise de fonction dans l'entreprise. Au cours de sa carrière, nous retiendrons seulement les événements : congé de maladie et prise de congé annuel. En fin de carrière, nous retiendrons deux situations : la démission et la retraite.

4.9 DIAGRAMME D'ACTIVITÉ (DAC)

4.9.1 Présentation générale et concepts de base

Le diagramme d'activité présente un certain nombre de points communs avec le diagramme d'état-transition puisqu'il concerne le comportement interne des opérations ou des cas d'utilisation. Cependant le comportement visé ici s'applique aux flots de contrôle et aux flots de données propres à un ensemble d'activités et non plus relativement à une seule classe.

Les concepts communs ou très proches entre le diagramme d'activité et le diagramme d'état-transition sont :

- **Transition,**
- **Nœud initial (état initial),**
- **Nœud final (état final),**
- **Nœud de fin flot (état de sortie),**
- **Nœud de décision (choix).**

Le formalisme reste identique pour ces nœuds de contrôle.

Les concepts spécifiques au diagramme d'activité sont :

- ✓ nœud de bifurcation,
- ✓ nœud de jonction,
- ✓ nœud de fusion,
- ✓ pin d'entrée et de sortie,
- ✓ flot d'objet,
- ✓ partition,

4.9.1.1 *Action*

Une action correspond à un traitement qui modifie l'état du système. Cette action peut être appréhendée soit à un niveau élémentaire proche d'une instruction en termes de programmation soit à un niveau plus global correspondant à une ou plusieurs opérations.

Formalisme et exemple

Une action est représentée par un rectangle dont les coins sont arrondis comme pour les états du diagramme d'état-transition (fig. 66).



Figure 66: Formalisme et exemple d'une action

4.9.1.2 *Transition et flot de contrôle*

Dès qu'une action est achevée, une transition automatique est déclenchée vers l'action suivante. Il n'y a donc pas d'événement associé à la transition.

L'enchaînement des actions constitue le flot de contrôle.

Formalisme et exemple

Le formalisme de représentation d'une transition est donné à la figure 67.

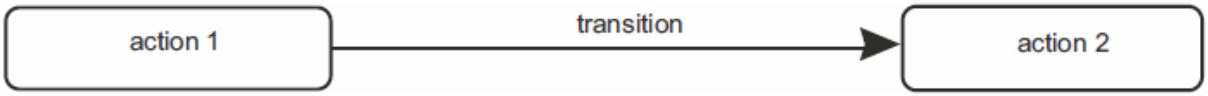


Figure 67:Formalisme de base du diagramme d'activité : actions et transition

4.9.1.3 *Activité*

Une activité représente le comportement d'une partie du système en termes d'actions et de transitions. Une activité est composée de trois types de noeuds :

- ✓ noeud d'exécution (action, transition),
- ✓ noeud de contrôle (noeud initial, noeud final, flux de sortie, noeud de bifurcation, noeud de jonction, noeud de fusion-test, noeud de test-décision, pin d'entrée et de sortie),
- ✓ noeud d'objet.

Une activité peut recevoir des paramètres en entrée et en produire en sortie.

Formalisme et exemple

Nous donnons une première représentation simple à la figure 68.

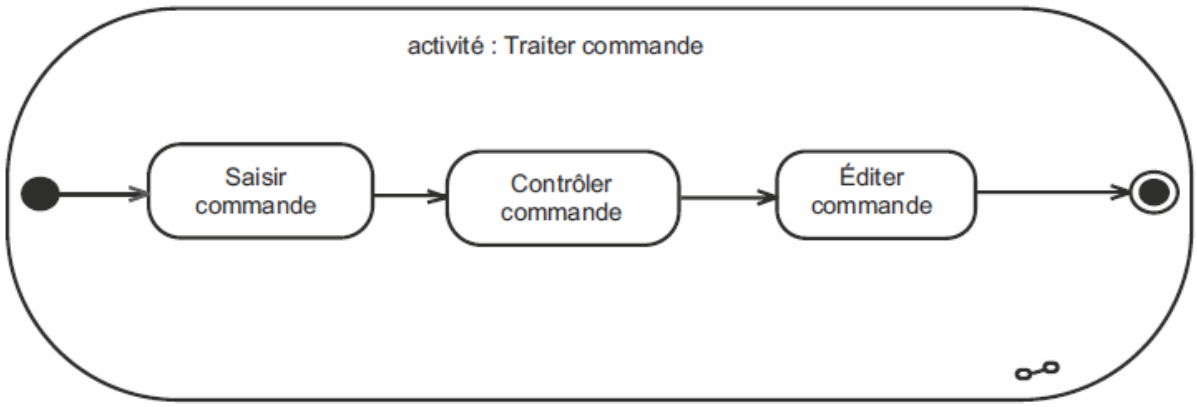


Figure 68: Exemple de représentation d'une activité

4.9.1.4 *Nœud de bifurcation (fourche)*

Un nœud de bifurcation (fourche) permet à partir d'un flot unique entrant de créer plusieurs flots concurrents en sortie de la barre de synchronisation.

Formalisme et exemple

Le formalisme de représentation de nœud de bifurcation ainsi qu'un premier exemple sont donnés à la figure 69.

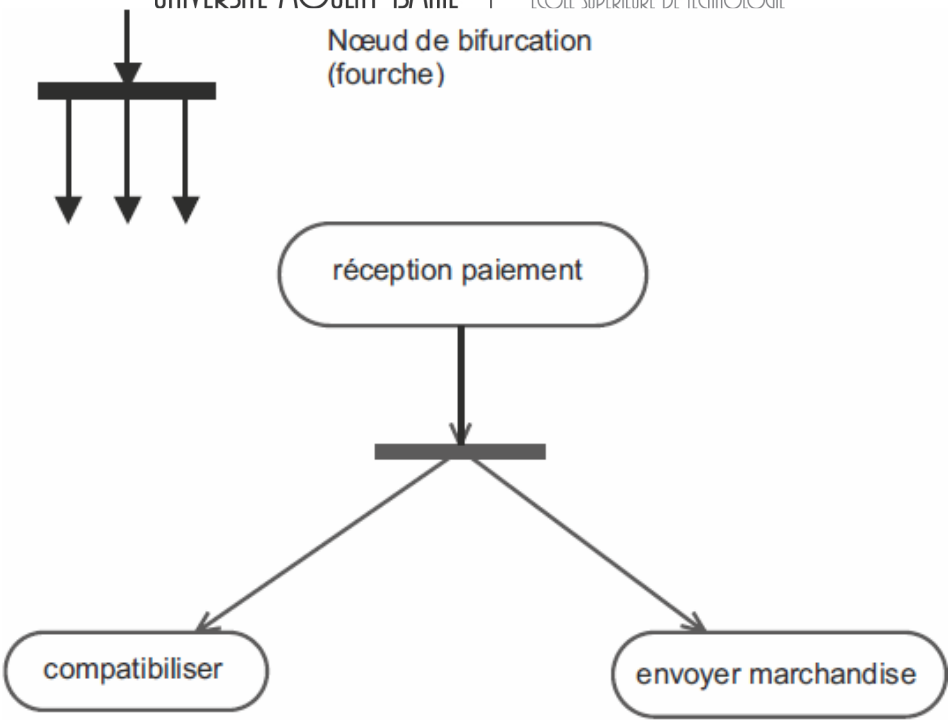


Figure 69: Exemple 1 d'activités avec nœud de bifurcation

4.9.1.5 *Noeud de jonction (synchronisation)*

Un **noeud de jonction** (synchronisation) permet, à partir de plusieurs flots concurrents en entrée de la synchronisation, de produire un flot unique sortant. Le nœud de jonction est le symétrique du noeud de bifurcation.

Formalisme et exemple

Le formalisme de représentation d'un noeud de jonction est donné à la figure 65.

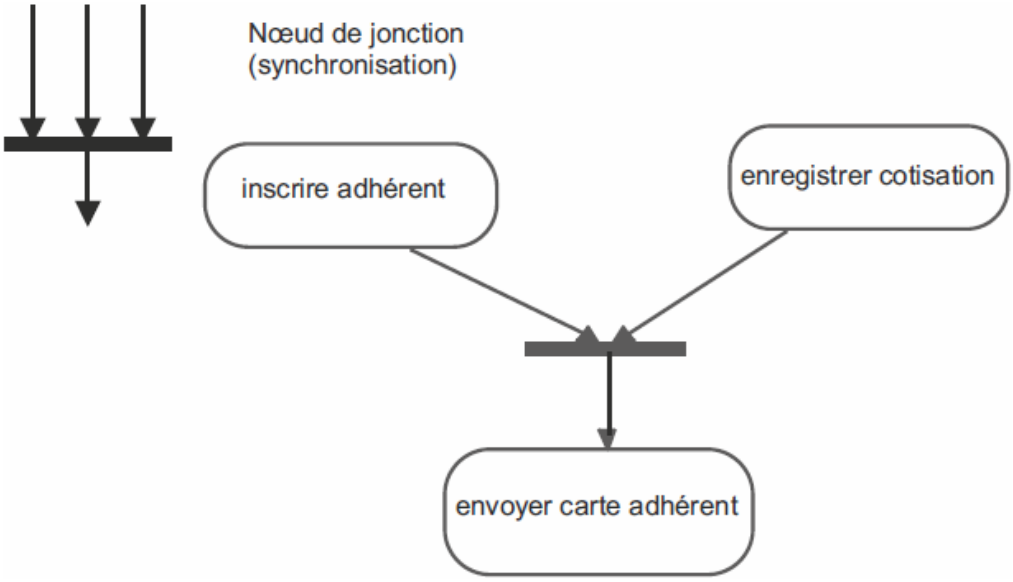


Figure 70: Exemple d'activités avec noeud de jonction

4.9.1.6 *Noeud de test-décision*

Un noeud de test-décision permet de faire un choix entre plusieurs flots sortants en fonction des conditions de garde de chaque flot. Un noeud de test-décision n'a qu'un seul flot en entrée. On peut aussi utiliser seulement deux flots de sortie : le premier correspondant à la condition vérifiée et l'autre traitant le cas sinon.

Formalisme et exemple

Le formalisme de représentation d'un nœud de test-décision ainsi qu'un premier exemple sont donnés à la figure 71. Un second exemple avec noeud de test-décision est donné à la figure 72.

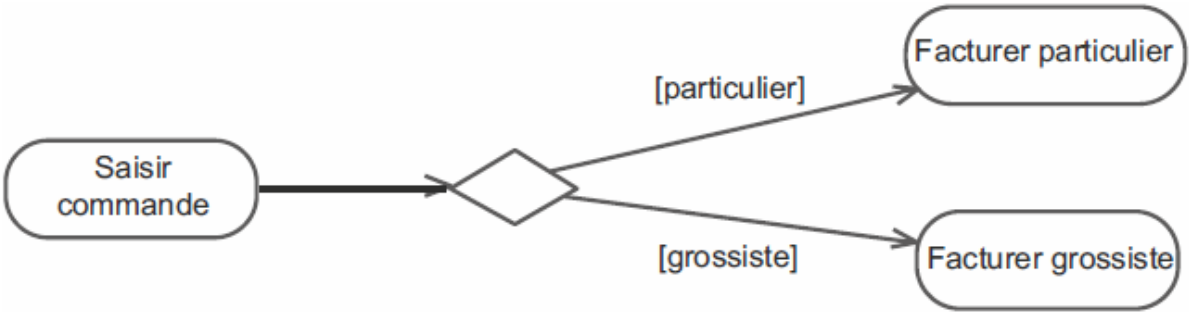


Figure 71: Formalisme et exemple 1 d'activités avec noeud de test-décision

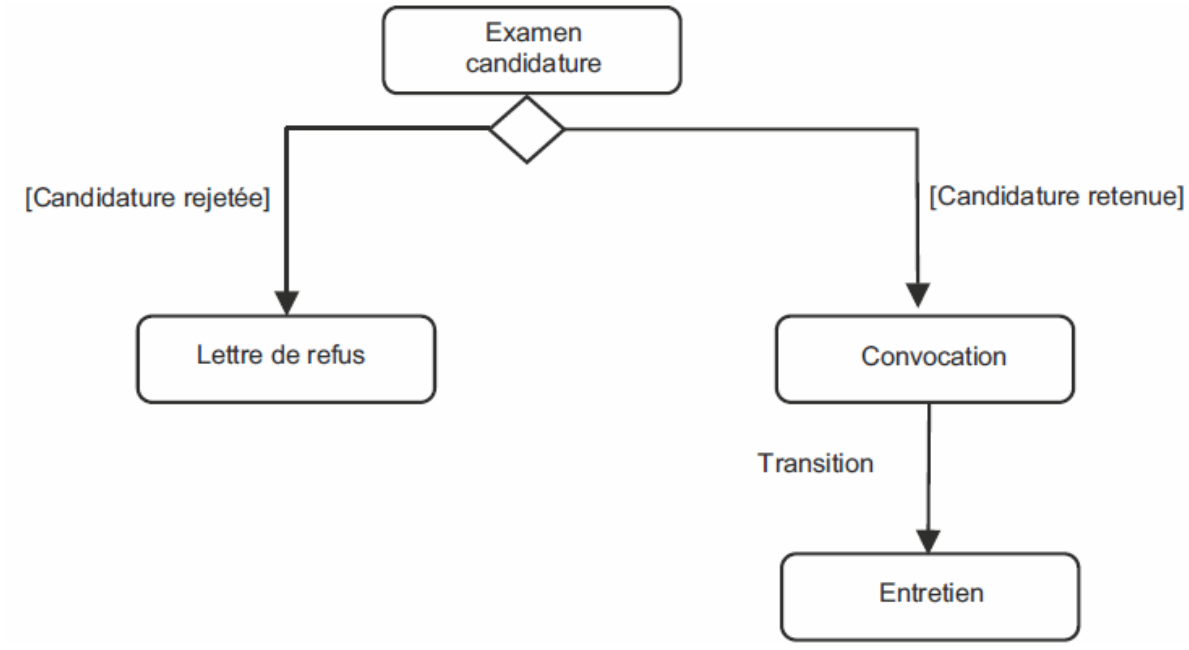


Figure 72: Exemple 2 de diagramme d'activités avec un noeud de test-décision

4.9.1.7 *Noeud de fusion-test*

Un **noeud de fusion-test** permet d'avoir plusieurs flots entrants possibles et un seul flot sortant. Le flot sortant est donc exécuté dès qu'un des flots entrants est activé.

Formalisme et exemple

Le formalisme de représentation d'un nœud de fusion-test ainsi qu'un exemple sont donnés à la figure 73.

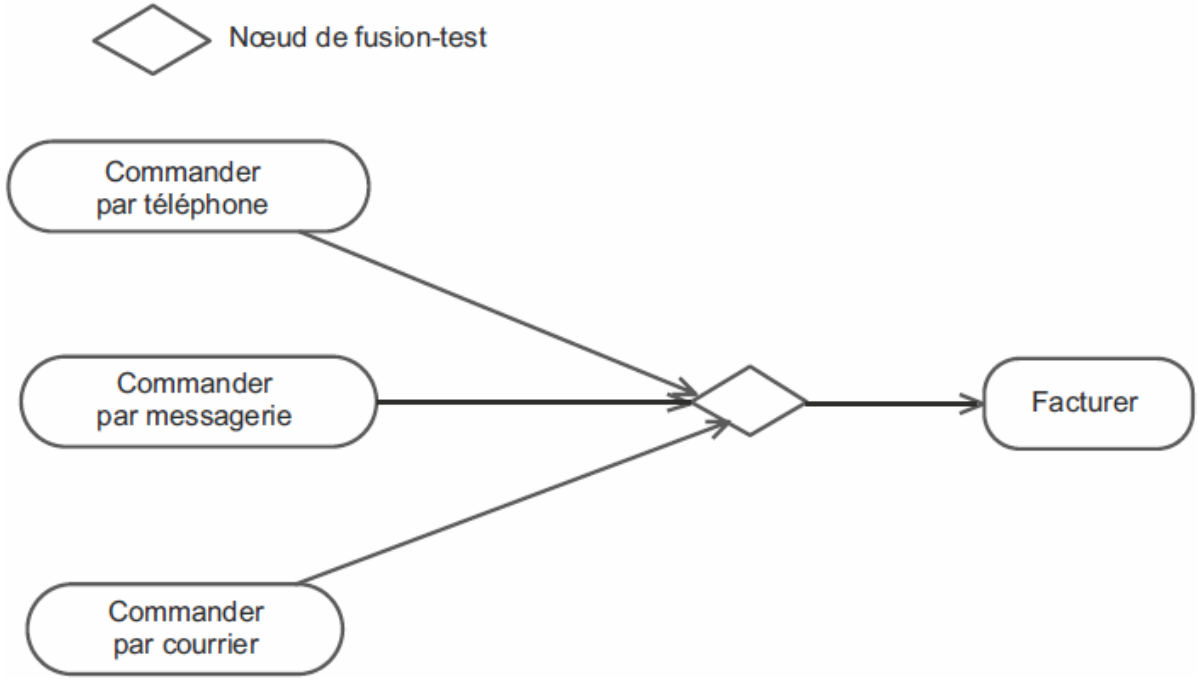


Figure 73: Formalisme et exemple de diagramme d'activités

4.9.1.8 *Pin d'entrée et de sortie*

Un **pin d'entrée** ou **de sortie** représente un paramètre que l'on peut spécifier en entrée ou en sortie d'une action. Un nom de donnée et un type de donnée peuvent être associés au pin. Un paramètre peut être de type objet.

Formalisme et exemple

Chaque paramètre se représente dans un petit rectangle. Le nom du paramètre ainsi que son type sont aussi à indiquer. Le formalisme de représentation de pin d'entrée ou de sortie ainsi qu'un exemple sont donnés à la figure 74.

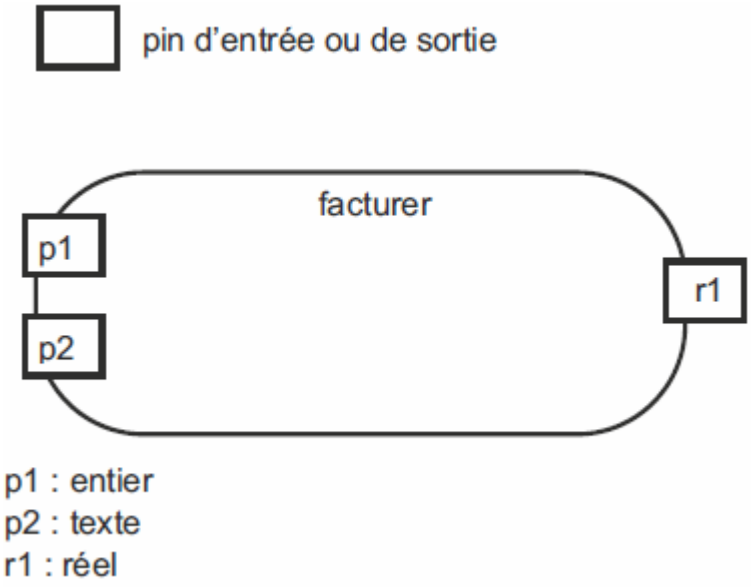


Figure 74: Formalisme et exemple d'activité avec pin d'entrée et de sortie

4.9.1.9 *Flot de données et noeud d'objet*

Un **noeud d'objet** permet de représenter le **flot de données** véhiculé entre les actions. Les objets peuvent se représenter de deux manières différentes : soit en utilisant le pin d'objet soit en représentant explicitement un objet.

Formalisme et exemple

Le formalisme de représentation de flot de données et noeud d'objet est donné directement au travers d'un exemple (fig. 70).

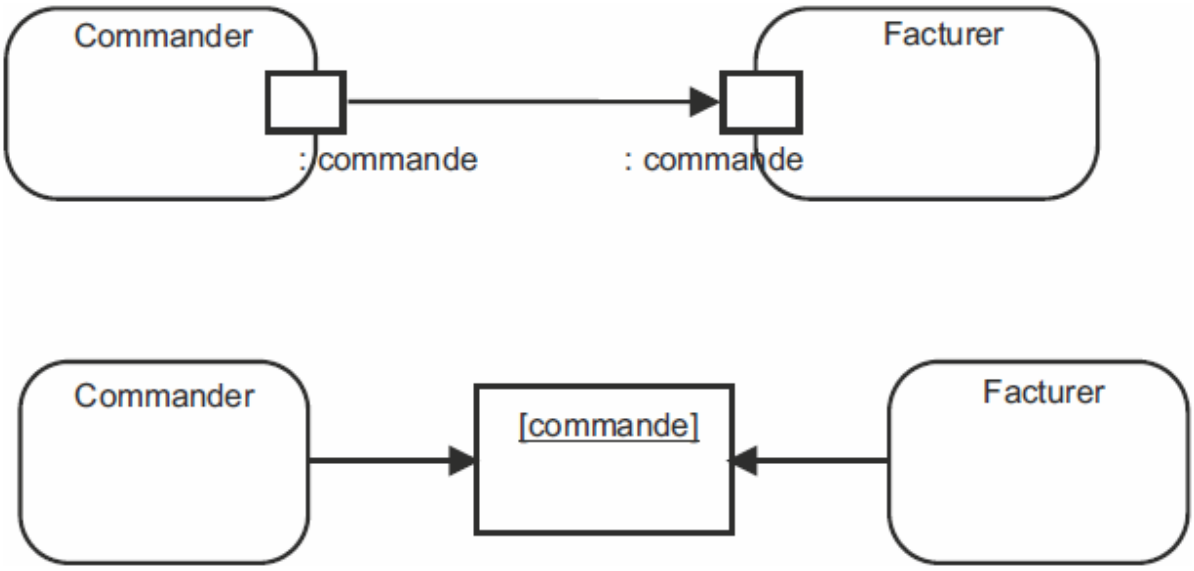


Figure 75: Exemple de flot de données et de noeud d'objets

4.9.1.10 *Partition*

UML permet aussi d'organiser la présentation du diagramme d'activité en couloir d'activités. Chaque couloir correspond à un domaine de responsabilité d'un certain nombre d'actions.

Les flots d'objets sont aussi représentés dans le diagramme. L'ordre relatif des couloirs de responsabilité n'est pas significatif.

4.9.2 Représentation du diagramme d'activité

Un exemple général de diagramme d'activité est donné à la figure 76.

Un autre exemple général de diagramme d'activité avec partitions est donné à la figure 77.

4.9.3 Représentation d'actions de communication

Dans un diagramme d'activité, comme dans un diagramme de temps, des interactions de communication liées à certains types d'événement peuvent se représenter.

Les types d'événement concernés sont :

- signal,
- Écoulement du temps.

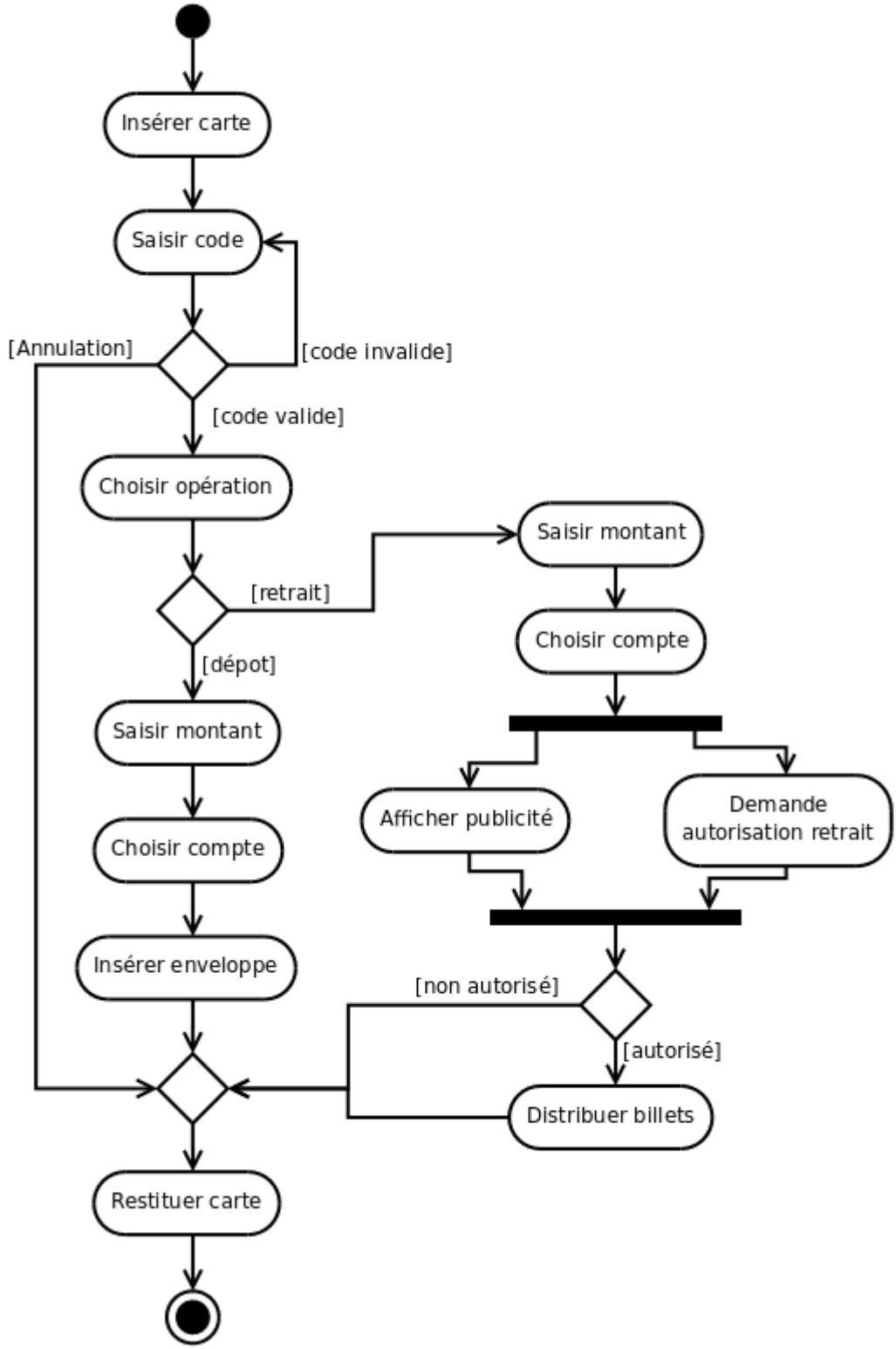


Figure 76: Un exemple général de diagramme d'activité

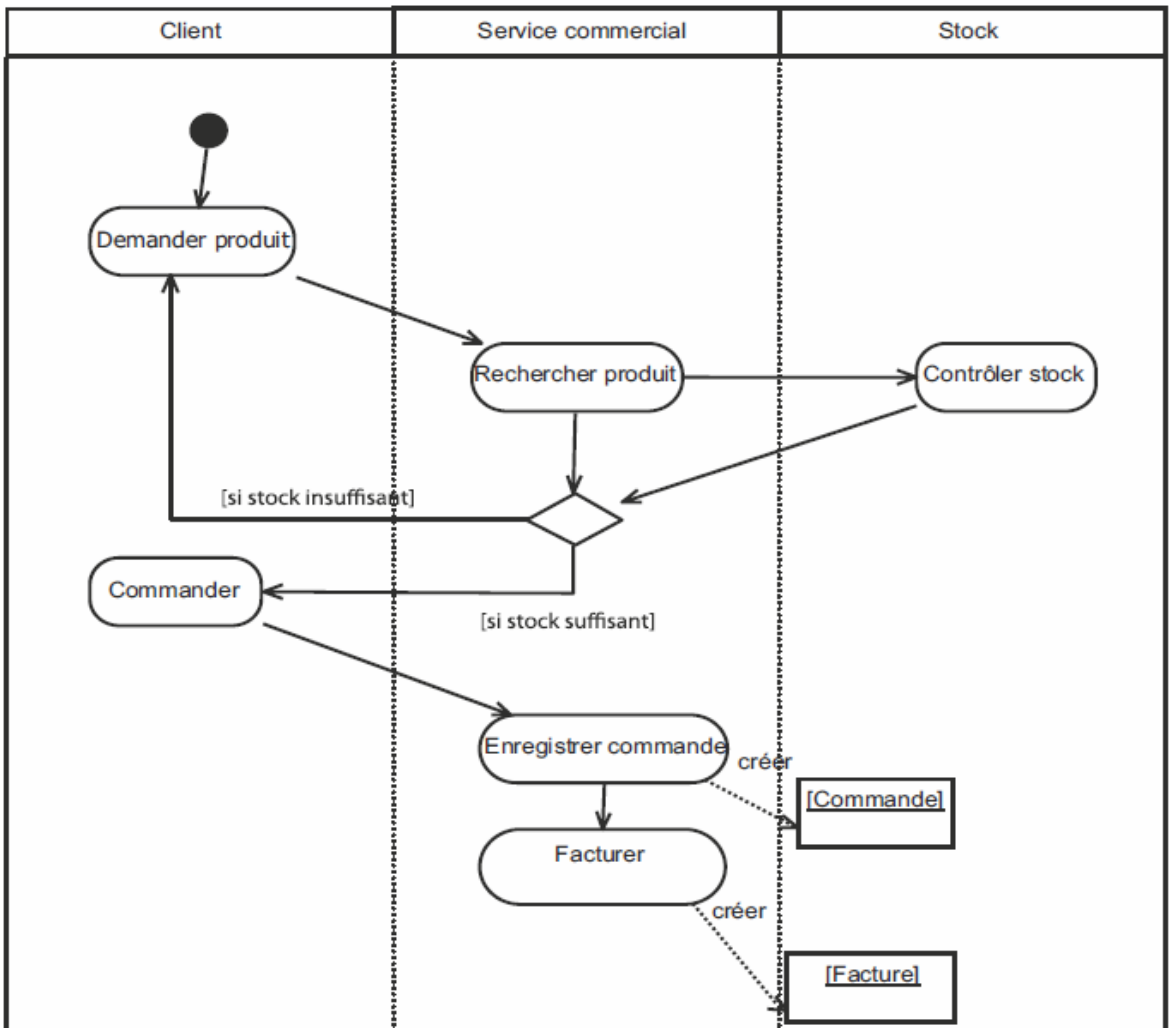


Figure 77: Un exemple général avec partitions de diagramme d'activité

Formalisme et exemple

Le formalisme de représentation ainsi qu'un exemple d'actions de communication sont donnés à la figure 78.



Figure 78: Formalisme et exemple de diagramme d'activité

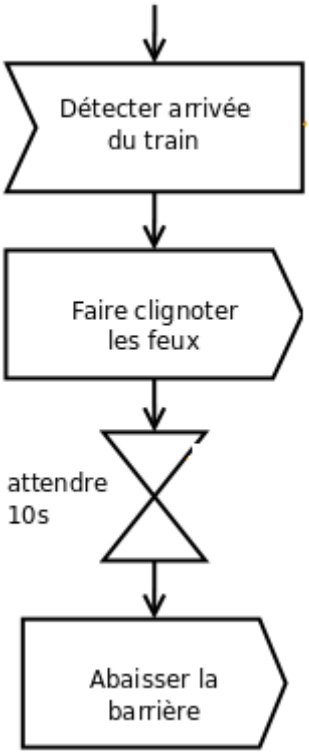


Figure 79: Formalisme et exemple de diagramme d'activité

4.9.4 Travaux dirigés

Exercice 1

En reprenant l'exercice relatif à la gestion de la bibliothèque traité dans les cas d'utilisation nous pouvons élaborer le diagramme d'activité correspondant.

Deux acteurs ont été identifiés :

• Bibliothécaire chargé de l'approvisionnement des ouvrages, de la gestion du catalogue et de l'enregistrement des emprunts et retours d'ouvrages ;

• Gestionnaire, chargé de l'inscription des adhérents et de la relance des adhérents ayant dépassé le délai de restitution des ouvrages.

4.10 DIAGRAMME DE TEMPS

4.10.1 Présentation générale et concepts de base

Le **diagramme de temps** permet de représenter les états et les interactions d'objets dans un contexte où le temps a une forte influence sur le comportement du système à gérer.

Autrement dit, le diagramme de temps permet de mieux représenter des changements d'états et des interactions entre objets liés à des contraintes de temps.

Pour cela, le diagramme de temps utilise en plus des lignes de vie, les concepts suivants :

• Des états ou des lignes de temps conditionnées avec deux représentations graphiques possibles.
• Des représentations propres aux aspects temporels : échelle de temps, contrainte de durée, événements…

4.10.1.1 *Concepts manipulés*

Le diagramme de temps utilise trois concepts de base :

• **Ligne de vie** – Elle représente l'objet que l'on veut décrire. Elle se dessine de manière horizontale. Plusieurs lignes de vie peuvent figurer dans un diagramme de temps.

• **État ou ligne de temps conditionnée** – Les différents états que peut prendre l'objet d'étude sont listés en colonne permettant ainsi de suivre le comportement de l'objet ligne par ligne (une ligne pour un état).

• **États linéaires** – Il s'agit du même concept que le précédent, mais la représentation de la succession des états est faite de manière linéaire à l'aide d'un graphisme particulier.

4.10.2 Représentation et exemples

Soit à représenter le dispositif de chauffe d'un fer à repasser à vapeur au moment de sa mise en service selon les règles suivantes :

• la pompe à eau qui remplit la chambre de chauffe s'active dès que le témoin d'eau interne le demande ;
• la pompe à eau se désactive dès que le niveau d'eau nécessaire est atteint ;
• le chauffage de l'eau, permettant de produire la vapeur, se met en action à la première mise en service du fer à repasser dès que le niveau d'eau de la chambre de chauffe est suffisant ;
• le chauffage initial de l'eau dure 3 mm permettant ainsi de produire la vapeur.

Dans cet exemple, nous avons deux objets à étudier : pompe à eau et chauffage de l'eau. Nous allons considérer pour chacun d'entre eux deux états significatifs : activé et désactivé.

La figure 80 donne la représentation du diagramme de temps en utilisant le formalisme des états « en escalier » et la figure 81 fournit la représentation linéaire des états.

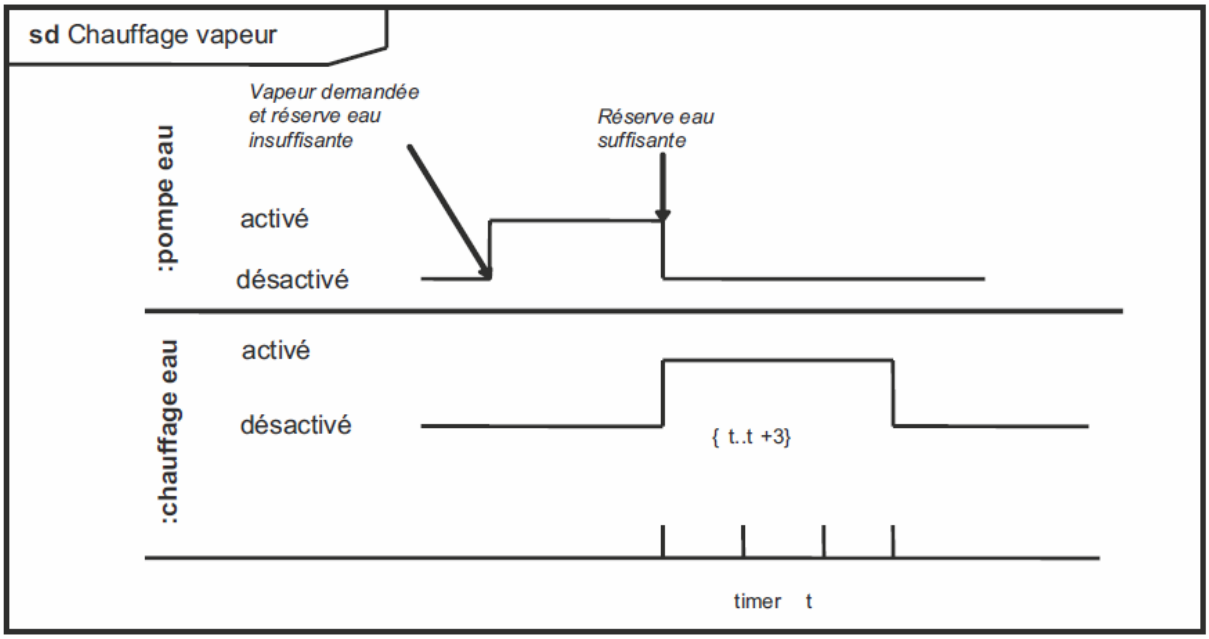


Figure 80: Exemple de diagramme de temps

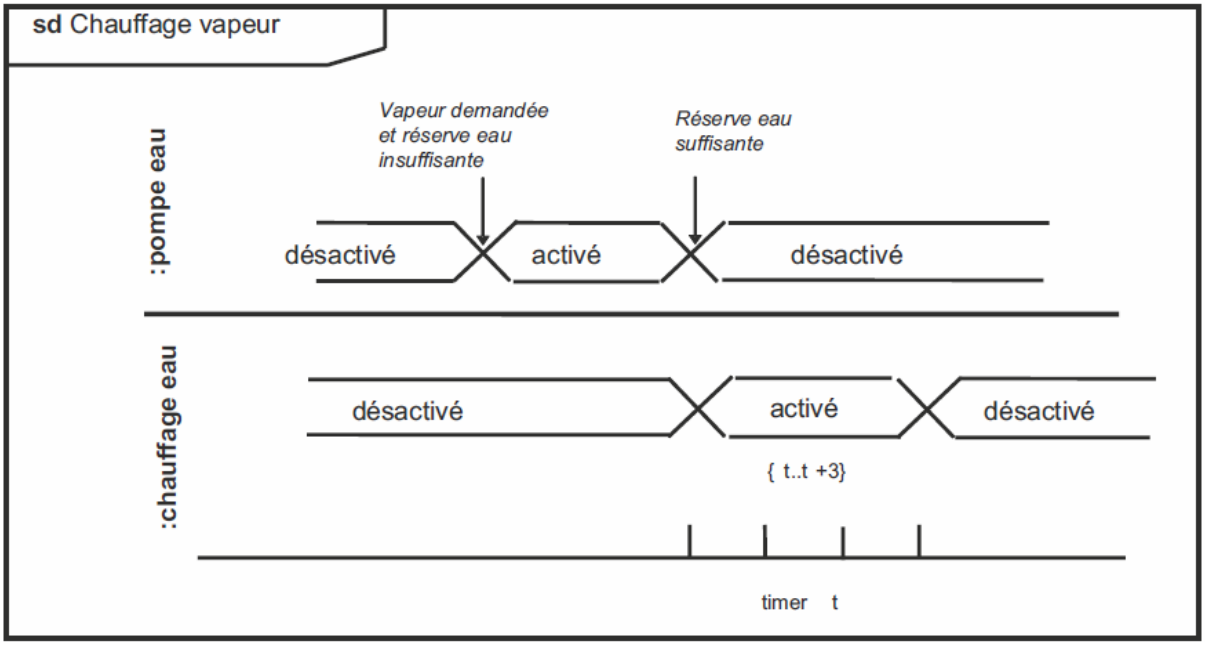


Figure 81: Exemple de diagramme de temps avec représentation linéaire

5 ème chapitre : L'architecture logicielle et matérielle du système

5.1 DIAGRAMME DE PAQUETAGE

5.1.1 Paquetage

Un paquetage regroupe des éléments de la modélisation appelés aussi membres, portant sur un sous-ensemble du système. Le découpage en paquetage doit traduire un découpage logique du système à construire qui corresponde à des espaces de nommage homogènes.

Les éléments d'un paquetage peuvent avoir une visibilité déclarée soit de type public (+) soit privé (-).

Un paquetage peut importer des éléments d'un autre paquetage. Un paquetage peut être fusionné avec un autre paquetage.

Formalisme et exemple

La figure 82 montre le formalisme général d'un paquetage et les trois manières de présenter un paquetage.

- **Représentation globale** – Le nom du paquetage se trouve à l'intérieur du grand rectangle.
- **Représentation détaillée** – Les membres du paquetage sont représentés et le nom du paquetage d'ensemble s'inscrit dans le petit rectangle.
- **Représentation éclatée** – Les membres du paquetage sont reliés par un lien connecté au paquetage par le symbole ⊕.

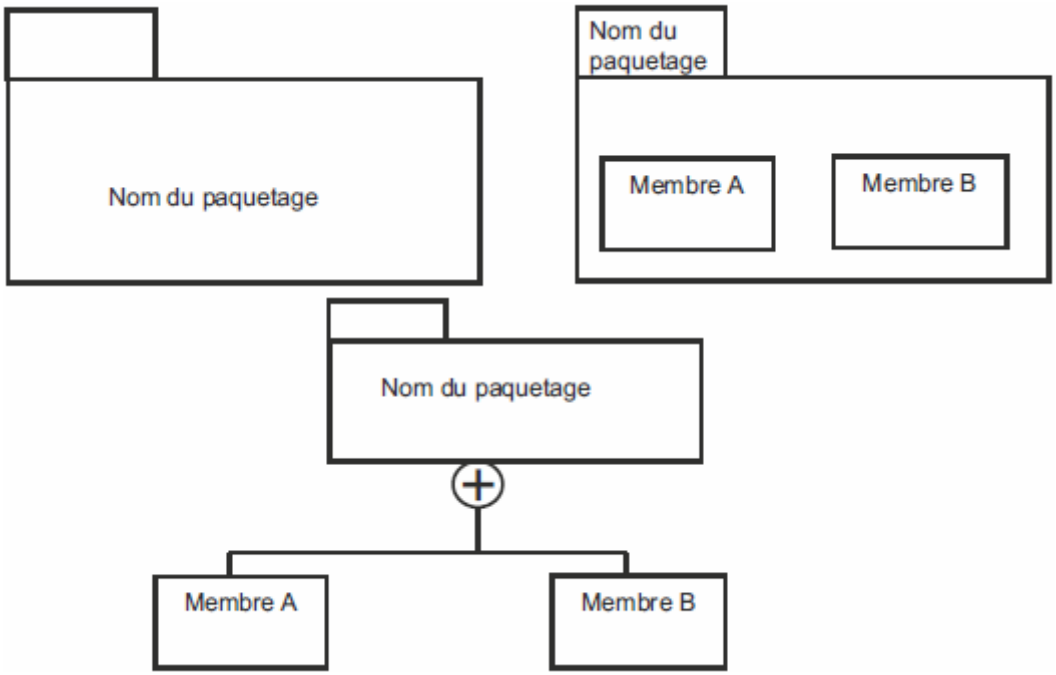


Figure 82: Formalisme de représentation de paquetages

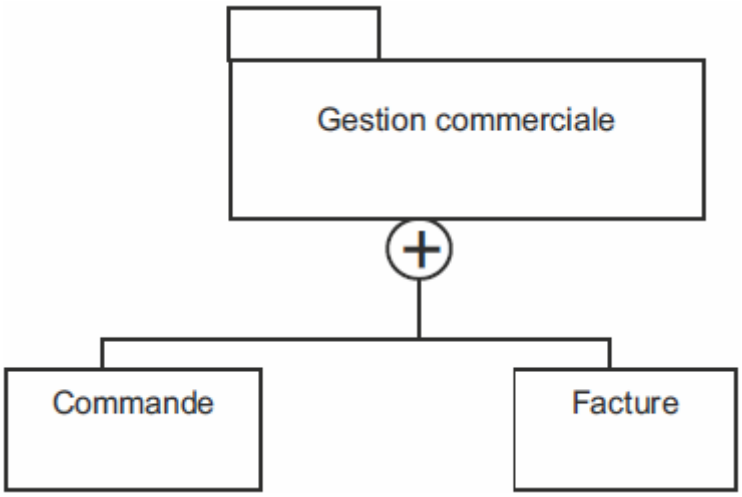


Figure 83: Exemple de représentation éclatée d'un paquetage

5.1.2 Dépendance entre paquetages

La dépendance entre paquetages peut être qualifiée par un niveau de visibilité qui est soit public soit privé. Par défaut le type de visibilité est public.

À chaque type de visibilité est associé un lien de dépendance. Les deux types de dépendances entre paquetages sont :

- **« import »** – Ce type de dépendance permet, pour un paquetage donné, d'importer l'espace de nommage d'un autre paquetage. Ainsi tous les membres du paquetage donné ont accès à tous les noms des membres du paquetage importé sans avoir à utiliser explicitement le nom du paquetage concerné. Ce type de dépendance correspond à un lien ayant une visibilité « public ».
- **« access »** – Ce type de dépendance permet, pour un paquetage donné, d'avoir accès à l'espace de nommage d'un paquetage cible. L'espace de nommage n'est donc pas importé et ne peut être transmis à d'autres paquetages par transitivité. Ce type de dépendance correspond à un lien ayant une visibilité « privé ».

Un exemple de dépendance entre paquetages mettant en jeu les niveaux de visibilité est donné à la figure 84.

Dans cet exemple, les éléments de Clients externes sont importés dans Domaine client et ensuite dans Domaine tiers. Cependant, les éléments de Clients internes sont seulement accessibles par le paquetage Domaine client et donc pas à partir du paquetage Domaine tiers.

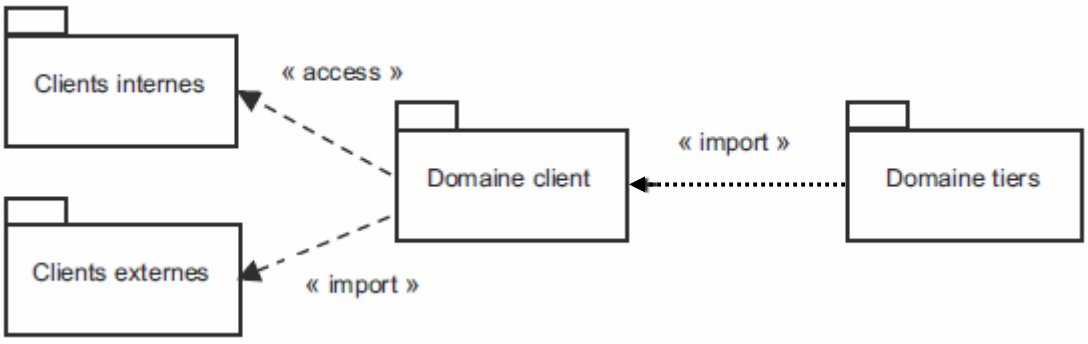


Figure 84: Exemple de liens de dépendance entre paquetages

5.1.3 Représentation et exemples

La figure (fig. 85) est une représentation des liens de dépendance entre paquetages.

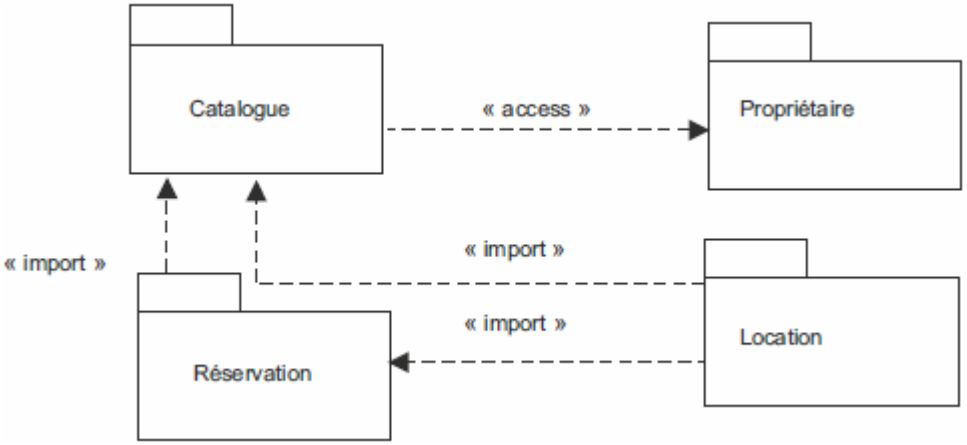


Figure 85: Exemple de liens de dépendance entre paquetages de Locagite

Enfin, UML propose aussi une opération de fusion entre deux paquetages. Le lien de dépendance comporte dans ce cas le mot-clé « merge ».

Ce type de lien permet de fusionner deux paquetages et d'obtenir ainsi un paquetage contenant la fusion des deux paquetages d'origine. Pour bien clarifier cette opération, il est important de qualifier par des rôles les paquetages dans cette fusion. Ainsi trois rôles sont à distinguer :

- le paquetage à fusionner (entrant dans la fusion, mais préservé après la fusion) ;
- le paquetage recevant (paquetage d'origine avant la fusion, mais non conservé après la fusion) ;
- le paquetage résultat (paquetage contenant le résultat de la fusion et écrasant le contenu du paquetage d'origine).

Un exemple type de fusion entre deux paquetages est donné à la figure 86.

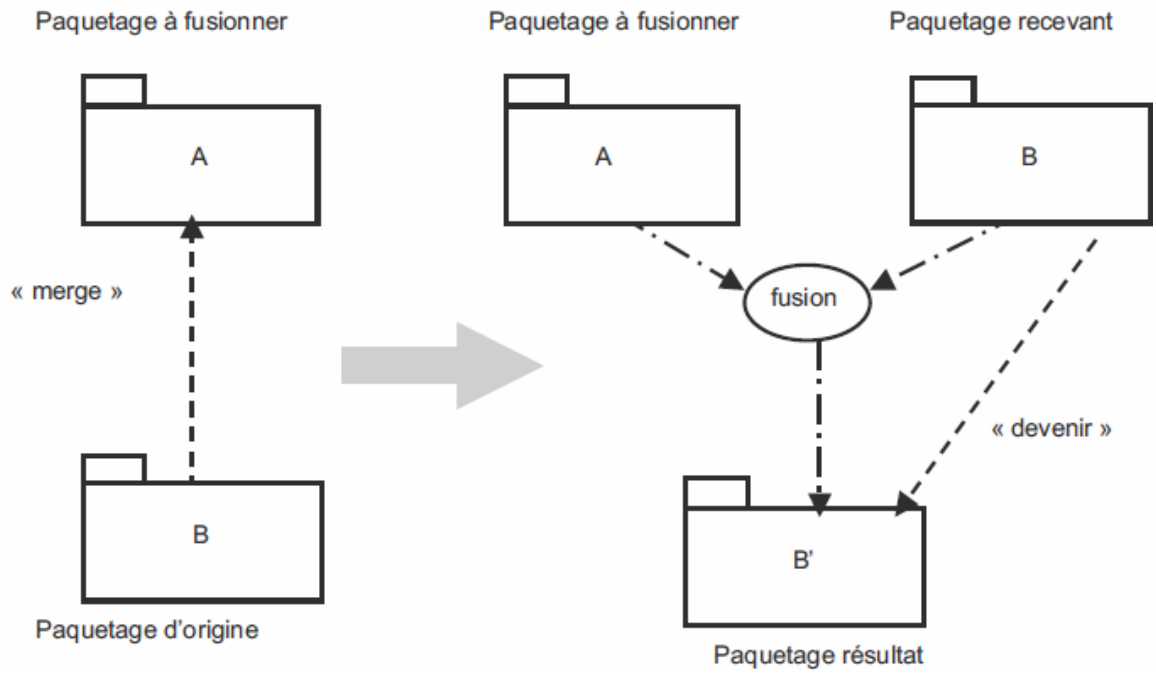


Figure 86: Exemple de liens de fusion entre paquetages

5.1.4 Travaux dirigés

Soit le cas ''Réservation de vols dans une agence de voyage''

1. Des compagnies aériennes proposent différents vols.
2. Un vol est ouvert à la réservation et fermé sur ordre de la compagnie.
3. Un client peut réserver un ou plusieurs vols, pour des passagers différents.
4. Une réservation concerne un seul vol, et un seul passager.
5. Une réservation peut être annulée ou confirmée.
6. Un vol a un aéroport de départ et un aéroport d'arrivée.
7. Un vol a un jour et une heure de départ et un jour et une heure d'arrivée.
8. Un vol peut comporter des escales dans des aéroports
9. Une escale a une heure d'arrivée et une heure de départ.
10. 10° Chaque aéroport dessert une ou plusieurs villes

Proposer un digramme de classe pour la description ci-dessus, puis transformer ce digramme en diagramme de package en regroupant chaque catégorie de classes en package.

Après le travail de l'étude de cas réservation de vol. Nous souhaitons élargir le champ du diagramme de package en proposant également des voyages en bus, que des transporteurs assurent.

1. Un voyage en bus a une ville de départ et une ville d'arrivée, avec des dates et heures associées.
2. Le trajet peut comporter des arrêts dans des villes intermédiaires.
3. Un client peut réserver un ou plusieurs voyages, pour un ou plusieurs passagers.

5.2 DIAGRAMME DE COMPOSANT (DCP)

Le **diagramme de composant** permet de représenter les composants logiciels d'un système ainsi que les liens existant entre ces composants.
Les composants logiciels peuvent être de deux origines : soit des composants métiers propres à une entreprise soit des composants disponibles sur le marché comme par exemple les composants EJB, CORBA, .NET, WSDL.

5.2.1 Composant

Chaque composant est assimilé à un élément exécutable du système. Il est caractérisé par :

• un nom ;
• une spécification externe sous forme soit d'une ou plusieurs interfaces requises, soit d'une ou plusieurs interfaces fournies ;
• un port de connexion.

Le port d'un composant représente le point de connexion entre le composant et une interface. L'identification d'un port permet d'assurer une certaine indépendance entre le composant et son environnement extérieur.

Formalisme général

Un composant est représenté (fig. 87) par un classeur avec le mot-clé « composant » ou bien par un classeur comportant une icône représentant un module.



Figure 87: Formalisme général d'un composant

5.2.2 Les deux types de représentation et exemples

Deux types de représentation sont disponibles pour modéliser les composants : une représentation « boîte noire » et une représentation « boîte blanche ». Pour chaque représentation, plusieurs modélisations des composants sont proposées.

5.2.2.1 *Représentation « boîte noire »*

C'est une vue externe du composant qui présente ses interfaces fournies et requises sans entrer dans le détail de l'implémentation du composant. Une boîte noire peut se représenter de différentes manières.

Connecteur d'assemblage

Une interface fournie se représente à l'aide d'un trait et d'un petit cercle et une interface requise à l'aide d'un trait et d'un demi-cercle. Ce sont les connecteurs d'assemblage.

Un exemple de modélisation avec les connecteurs d'assemblage est donné à la figure 75, le composant Commande possède deux interfaces fournies et deux interfaces requises.



Figure 88: Représentation d'un connecteur d'assemblage

Connecteur d'interfaces

Une autre représentation peut être aussi utilisée en ayant recours aux dépendances d'interfaces *utilise* et *réalise* :

• pour une interface fournie, c'est une relation de réalisation partant du composant et allant vers l'interface ;
• pour une interface requise, c'est une dépendance avec le mot-clé « utilise » partant du composant et allant vers l'interface.

Un exemple de modélisation avec connecteurs d'interfaces est donné à la figure 76. Le composant Commande possède une interface fournie « GestionCommande » et une interface requise « Produit ».

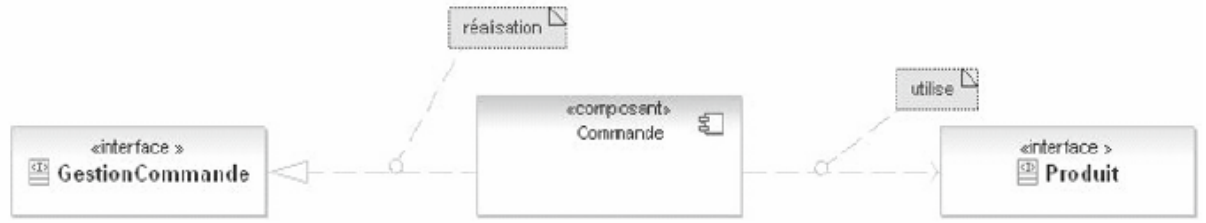


Figure 89: Représentation d'un composant avec connecteur d'interfaces

Compartiment

Une dernière manière de modéliser un composant avec une représentation boîte noire est de décrire sous forme textuelle les interfaces fournies et requises à l'intérieur d'un second compartiment.

Un exemple de modélisation avec les compartiments est donné à la figure 77.

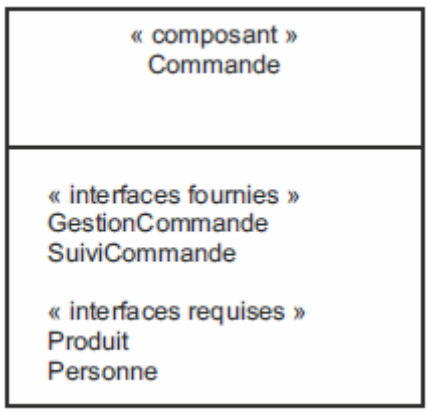


Figure 90: Représentation d'un composant avec compartiments

5.2.2.2 *Représentation « boîte blanche »*

C'est une vue interne du composant qui décrit son implémentation à l'aide de classificateurs (classes, autres composants) qui le composent. Plusieurs modélisations sont possibles pour la représentation boîte blanche.

Compartiment

Une manière de modéliser un composant avec une représentation boîte blanche est de décrire sous forme textuelle les interfaces fournies et requises à l'intérieur d'un compartiment, les classificateurs (classes, autres composants) dans un autre compartiment, les artefacts (élément logiciel : jar, war, ear, dll) qui représentent physiquement le composant dans un dernier compartiment.

Un exemple de modélisation avec les compartiments est donné à la figure 91.

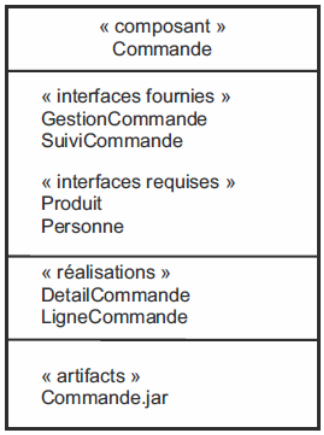


Figure 91: Représentation boîte blanche avec compartiments

Dépendance

Une autre représentation interne du composant peut être aussi utilisée en ayant recours aux dépendances. Ainsi, les classificateurs qui composent le composant sont reliés à celui-ci par une relation de dépendance.

Les relations entre les classificateurs (association, composition, agrégation) sont aussi modélisées. Néanmoins, si elles sont trop complexes, elles peuvent être représentées sur un diagramme de classe relié au composant par une note.

Un exemple de modélisation boîte blanche avec les dépendances est donné à la figure 92.

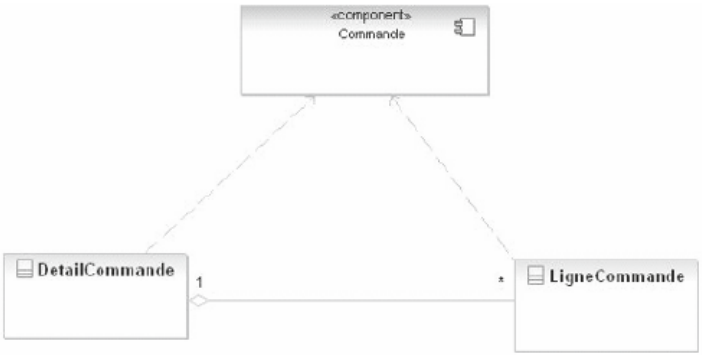


Figure 92: Représentation boîte blanche avec dépendances

Ports et connecteurs

Le port est représenté par un petit carré sur le composant. Les connecteurs permettent de relier les ports aux classificateurs. Ils sont représentés par une association navigable et indiquent que toute information arrivée au port est transmise au classificateur.

Un exemple de représentation avec port et connecteur du composant Commande est donné à la figure 92.

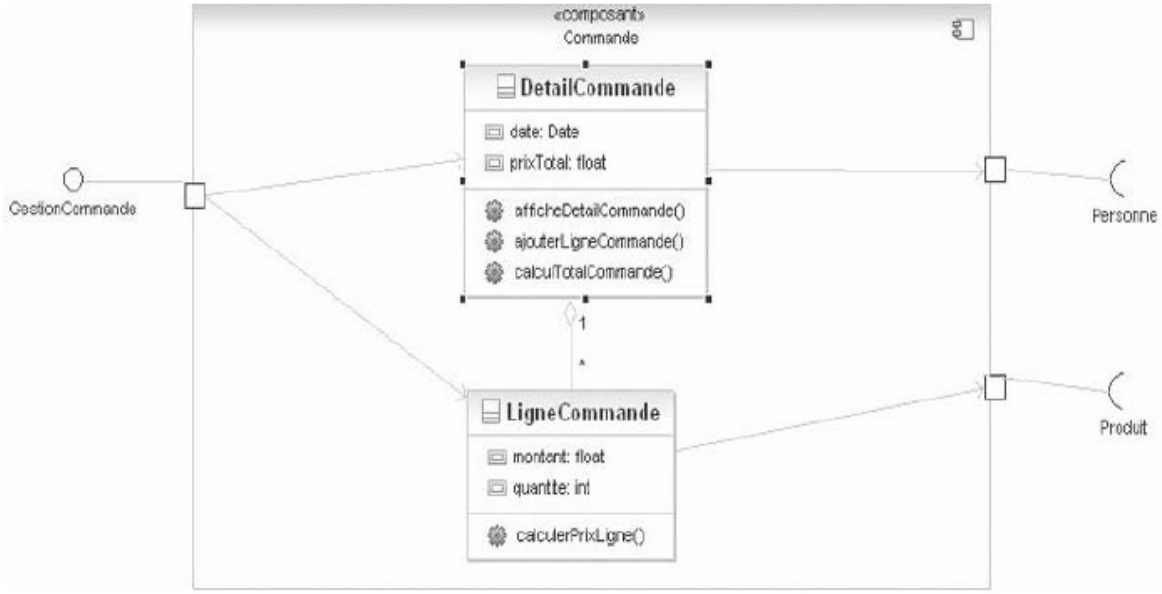


Figure 93: Représentation boîte blanche avec connecteurs

Dans l'exemple de la figure 92, le composant Commande est constitué de deux classes (classificateur) reliées par une agrégation : DetailCommande et LigneCommande.

L'interface fournie GestionCommande est accessible de l'extérieur *via* un port et permet d'accéder *via* les connecteurs aux opérations des deux classes DetailCommande et LigneCommande (ajouterLigneCommande, calculTotal-Commande, calculPrixLigne).

L'interface requise Personne est nécessaire pour l'affichage du détail de la commande et est accessible *via* un port du composant Commande.

L'interface requise Produit est nécessaire pour le calcul du prix de la ligne de commande et est accessible *via* un port du composant Commande.

5.3 DIAGRAMME DE DÉPLOIEMENT

Le **diagramme de déploiement** permet de représenter l'architecture physique supportant l'exploitation du système. Cette architecture comprend des noeuds correspondant aux supports physiques (serveurs, routeurs…) ainsi que la répartition des artefacts logiciels (bibliothèques, exécutables…) sur ces noeuds. C'est un véritable réseau constitué de noeuds et de connexions entre ces noeuds qui modélise cette architecture.

5.3.1 Noeud

Un noeud correspond à une ressource matérielle de traitement sur laquelle des artefacts seront mis en oeuvre pour l'exploitation du système. Les noeuds peuvent être interconnectés pour former un réseau d'éléments physiques.

Formalisme et exemple

Un noeud ou une instance de noeud se représente par un cube ou parallélépipède (fig 90).

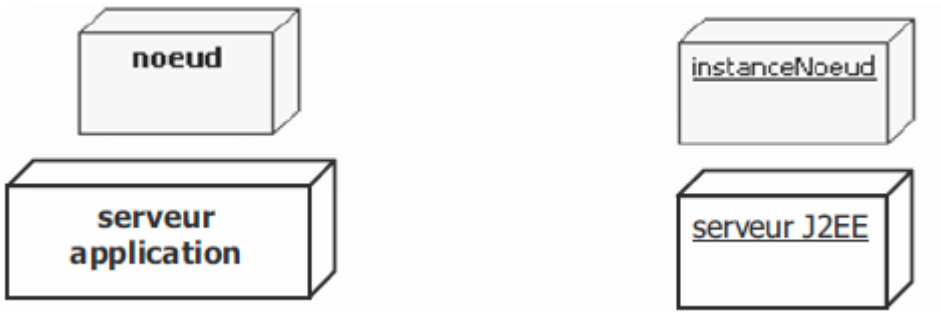


Figure 94: Représentation de nœuds

Compléments sur la description d'un nœud

Il est possible de représenter des noeuds spécialisés. UML propose en standard les deux types de noeuds suivants :

- Unité de traitement – Ce noeud est une unité physique disposant de capacité de traitement sur laquelle des artefacts peuvent être déployés. Une unité de traitement est un noeud spécialisé caractérisé par le mot-clé « device ».
- Environnement d'exécution – Ce noeud représente un environnement d'exécution particulier sur lequel certains artefacts peuvent être exécutés. Un environnement d'exécution est un noeud spécialisé caractérisé par le motclé « executionEnvironment ».

5.3.2 Artefact

Un artefact est la spécification d'un élément physique qui est utilisé ou produit par le processus de développement du logiciel ou par le déploiement du système. C'est donc un élément concret comme par exemple : un fichier, un exécutable ou une table d'une base de données.

Un artefact peut être relié à d'autres artefacts par notamment des liens de dépendance.

Formalisme et exemple

Un artefact se représente par un rectangle (fig. 95) caractérisé par le mot-clé « artifact » et/ou une icône particulière dans le coin droit du rectangle.



Figure 95: Représentation d'un artefact

5.3.3 Spécification de déploiement

Une spécification de déploiement peut être associée à chaque artefact. **Elle permet de préciser les conditions de déploiement de l'artefact sur le noeud** sur lequel il va être implanté.

Formalisme et exemple

Une spécification de déploiement se représente par un rectangle avec le mot-clé « deployment spec ». Un exemple d'un artefact avec une spécification de déploiement est donné à la figure 96.

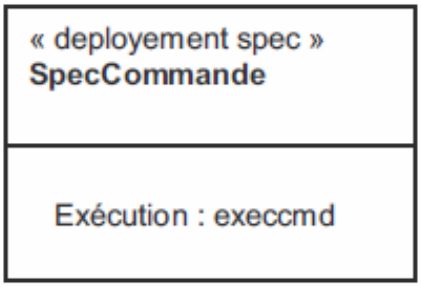


Figure 96: Exemple d'un artefact avec spécification de déploiement

5.3.4 Liens entre un artefact et les autres éléments du diagramme

Il existe deux manières de représenter le lien entre un artefact et son nœud d'appartenance :

• **Représentation inclusive – Dans cette représentation, un artefact est représenté à l'intérieur du noeud auquel il se situe physiquement**. Un exemple est donné à la figure 97.

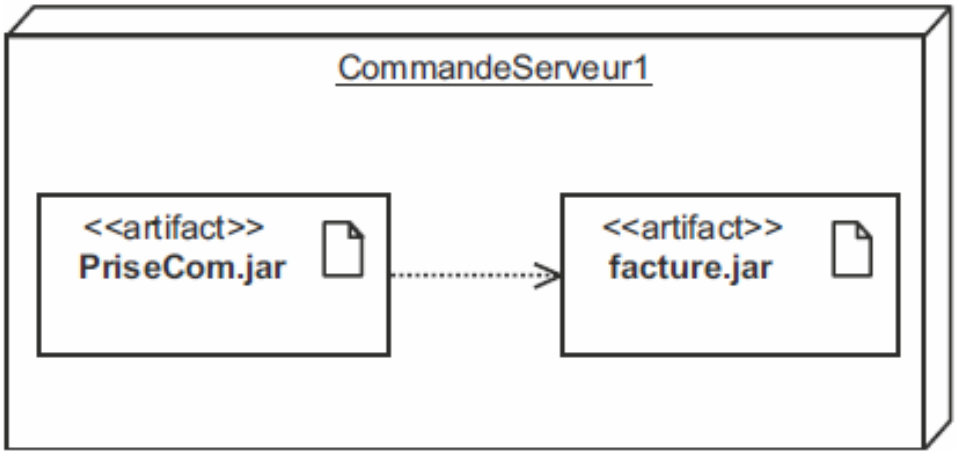


Figure 97: Exemple de représentation inclusive d'artefact

Représentation avec un lien de dépendance typé « deploy » – Dans ce cas l'artefact est représenté à l'extérieur du noeud auquel il appartient avec un lien de dépendance entre l'artefact et le noeud typé avec le mot-clé « deploy ». Un exemple est donné à la figure 98.

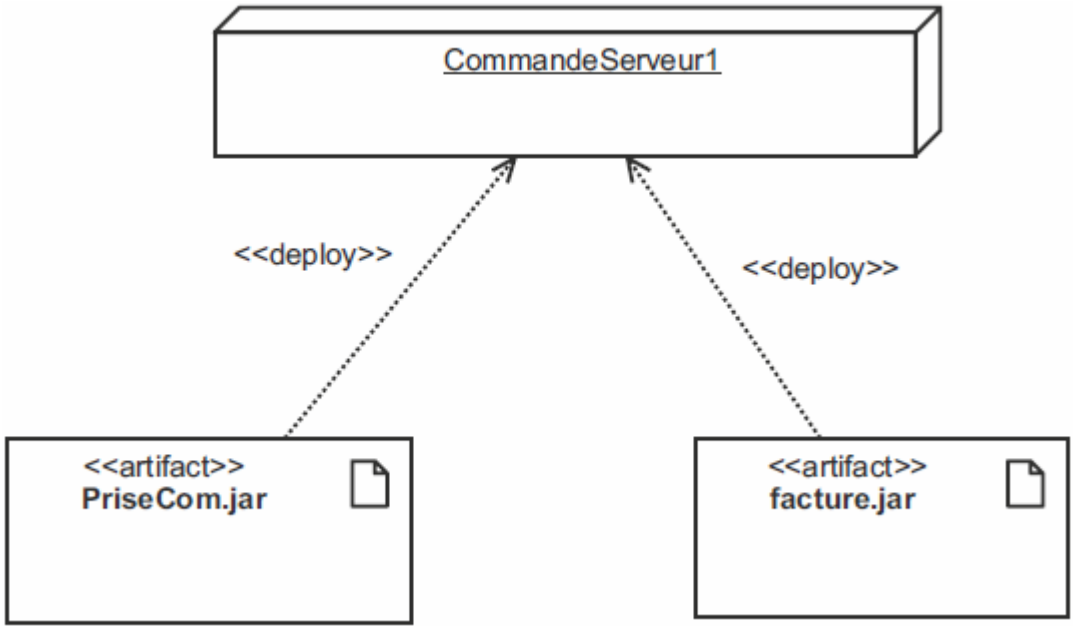


Figure 98: Exemple de représentation d'artefact avec lien de dépendance

Un artefact peut représenter un ou plusieurs éléments d'un modèle. Le qualificatif « manifest » permet d'indiquer ce type de dépendance.

5.3.5 Représentation et exemples

Le diagramme de déploiement représente les noeuds de l'architecture physique ainsi que l'affectation des artefacts sur les noeuds conformément aux règles de déploiement définies. Un premier exemple est donné à la figure 99.

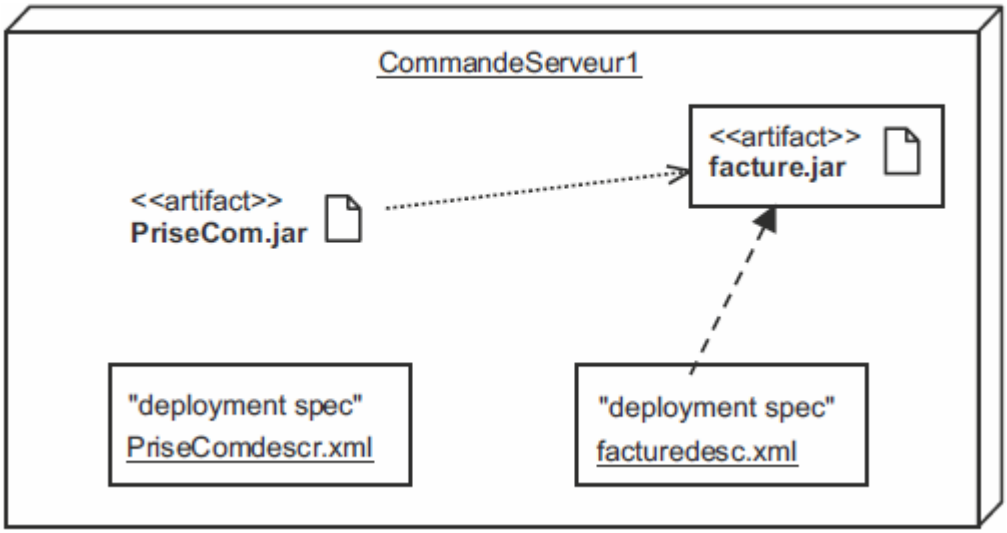


Figure 99: Exemple de représentation d'un diagramme de déploiement