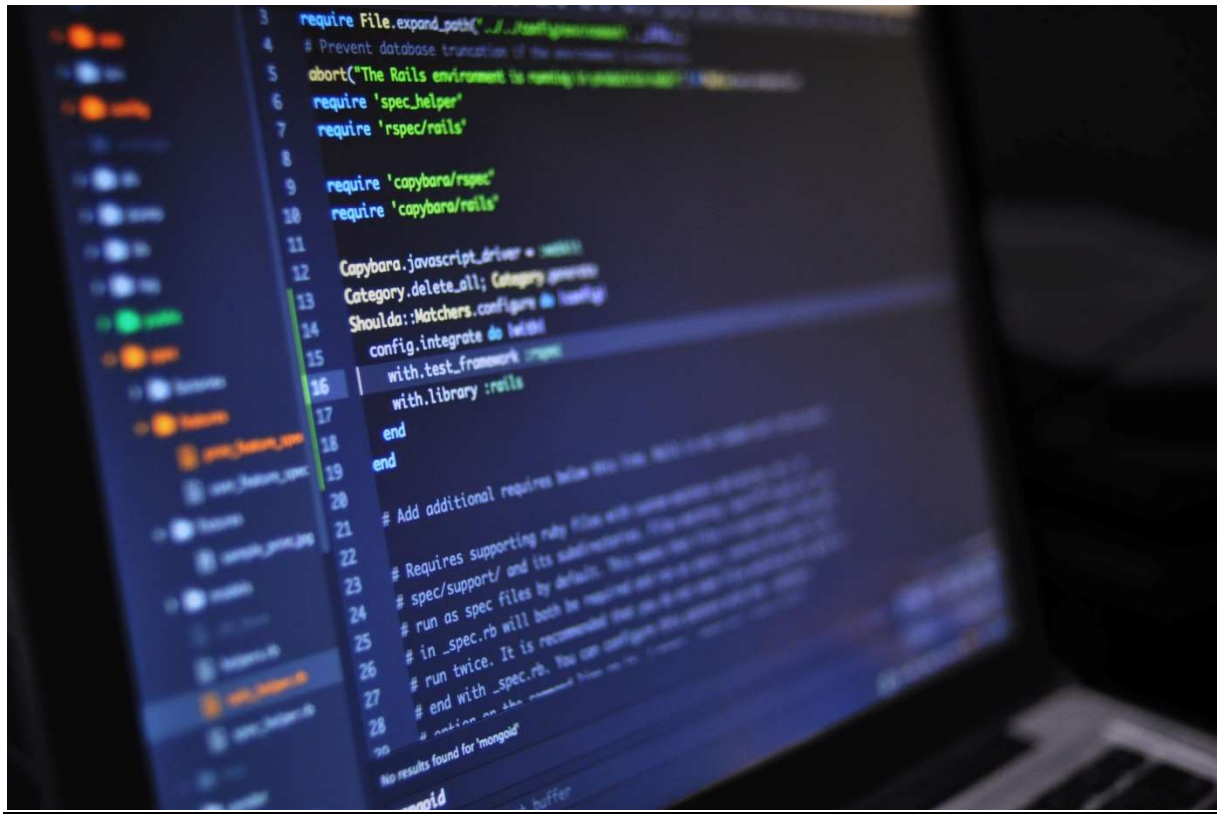
جامعة مولاي إسماعيل
UNIVERSITÉ MOULAY ISMAÏL

EST MEKNÈS
المدرسة العليا للتكنولوجيا
ÉCOLE SUPÉRIEURE DE TECHNOLOGIE

Département GI

Module : UML & développement mobile sous Android

Elément : UML



Prof. Dr. Yassine Rhazali

Partie Introduction au Génie logiciel

Logiciel

Définition

Logiciel : l'ensemble des instructions qui agissent sur des structures des données.

Classifications

Il y a plusieurs manières de classification des logiciels :

Les deux principales catégories :

- **Logiciel de base** : c'est un logiciel de base d'autres logiciel, ainsi les systèmes d'exploitation se sont des logiciels de base qui permettent de fonctionner directement le matériel.
- **Logiciel d'application** : c'est un logiciel qui réalise une application en se basant sur un logiciel de base, ainsi photoshop, office et mozila se sont des logiciels d'application.

Puis il y a :

- **Application** permet de réaliser une tâche.
- **Pilote** permet d'utiliser un matériel informatique.

Ensuite Il y a :

- **Un logiciel spécifique** permet de répondre à la demande d'un client.
- **Un logiciel standard** (**progiciel**) son objectif est d'être vendu en grande distribution.

Ensuite selon les droits du contrat de licence, on parle de :

- **Logiciel propriétaire** l'auteur se réserve le droit de diffuser le logiciel.
- **Partagiciel** l'auteur autorise d'autre à diffuser le logiciel.
- **logiciel libre** (**open source**) l'auteur autorise d'accéder au code source.
- **Logiciel gratuit** c'est un logiciel propriétaire qui peut être diffuser sans frais.

Enfin il y a :

- **Logiciel bureau** : qui sont utilisés localement dans l'ordinateur tel que bloc note
- **Logiciel web** : se sont des logiciels client-serveur installés dans le serveur et utilisés depuis les navigateurs des clients tel que facebook
- **Logiciel mobile** : sont des logiciels développés pour une utilisation au niveau de smartphone ils peuvent être des logiciels client-serveur ou web tel que numberbook et candy camera.

Middleware

Un **middleware** (**intergiciel)** est un logiciel d'échange d'informations entre différentes applications informatiques. Ainsi on trouve CORBA d'OMG, RMI de Java et DCOM de Microsoft.

Protection de logiciel

Le code source des logiciels sont protégés par la convention de **Berne**.

Qualité des logiciels

La norme ISO 9126 définit six groupes d'indicateurs de qualité des logiciels :

- la capacité fonctionnelle : la capacité de répondre aux exigences de clients.
- la facilité d'utilisation : l'effort nécessaire au client pour utiliser le logiciel.
- la fiabilité : la capacité d'un logiciel d'avoir des résultats corrects dans n'importe quelles conditions d'exploitation.
- la performance : le rapport entre la quantité de ressources utilisées et la quantité de résultats.
- la maintenabilité : l'effort nécessaire à corriger ou transformer le logiciel.
- la portabilité : la capacité d'un logiciel de fonctionner dans un nouveau environnement matériel ou logiciel.

Génie logiciel

Définition et objectifs

Le **génie logiciel** désigne l'ensemble des méthodes, des techniques et outils permettant la production d'un logiciel.

L'objectif principal du génie logiciel c'est minimiser le cout, minimiser le temps, minimiser les risques et réaliser un logiciel de qualité.

Cycle de vie d'un logiciel

Le « **cycle de vie d'un logiciel** » représente toutes les étapes du développement d'un logiciel.

Le découpage permet de détecter les erreurs très tôt et ainsi de maîtriser le cout, le temps, et la qualité du logiciel.

Le cycle de vie du logiciel comprend les activités suivantes :

- **Analyse** : permet d'identifier le problème à étudier. Le résultat de l'analyse est le cahier de charges. Cette étape est informelle. Cette phase contient 3 sous phases :

- **Faisabilité** : Est-ce que le logiciel est réalisable ? Est-ce que le développement proposé mérite la mise en œuvre ?
- **Spécification des besoins** : Permet de définir ce que doit faire le logiciel et non comment il le fait. Quatre types de spécifications :
    - Spécification générale : Objectifs à atteindre, Contraintes.
    - Spécification fonctionnelles est la description des fonctionnalités du futur logiciel de manière détaillée que possible.
    - Spécification d'interface décrit les interfaces du logiciel avec le monde extérieur : homme (IHM), autres logiciel (Middleware), machines (rebot)
    - Spécification technique :(Étude de l'existant) : Moyens d'accès (local, distant, Internet, …), Quantité d'informations à stocker (choix du SGBDR, …)
- **Organisation du projet :** Permet de déterminer la manière de développer le logiciel : contient plusieurs étapes :
    - planification : permet de : découper le projet en tâches, puis décrire leur enchaînement dans le temps, ensuite affecter à chacune une durée et un effort.
    - Analyse des coûts: estimation du prix du projet

- **Conception** : Permet de représenter une description fonctionnelle (formelle) du système en utilisant une méthode. Deux types de conception :

    - **Conception générale :** permet de représenter l'architecture générale du système : décomposition du système en sous système.
    - **Conception détaillée :** permet de représenter les procédures, fonctions et des structures de données (conception classique). Permet de représenter les classes en termes de propriétés et de méthodes (conception Orientée Objet).

- **implémentation (Réalisation) :** développé le logiciel. Plusieurs types langages sont utilisés: Langages classiques et Langages orientés objets.

- **Tests :**

    - **Test unitaire**: tester chaque module à part
    - **Test d'intégration**: tester pendant l'intégration des modules
    - **Test système**: tester dans un environnement proche à celui de production
    - **Test alpha**: tests faits par le client sur le site de développement
    - **Test bêta**: tests faits par le client sur le site de production
    - **Test de régression** : exécuté après la correction des erreurs pour vérifier si d'autres erreurs n'ont pas été introduites au cours de la correction.

- **Livraison :** Permet de fournir au client une solution logicielle qui fonctionne correctement.

    - **Installation**: rendre le logiciel opérationnel sur le site du client
    - **Formation**: enseigner les utilisateurs comment utiliser logiciel
    - **Assistance**: répondre aux questions de l'utilisateur

- **Maintenance**
  - **Corrective :** correction des erreurs.
  - **Adaptative** : adaptation à de nouveaux environnements
  - **Évolutive ou Perfective** : ajout de nouvelles fonctionnalités

Documents

- Cahier des charges
- Spécifications : use case
- Plan de test
- Calendrier du projet
- Code source
- Rapport des tests
- Rapport des défauts
- Manuel d'utilisateur

Modèles de cycles de vie

Modèle en cascade

Permet de représenter des phases séquentielles, à la fin de chaque phase des documents sont produits.

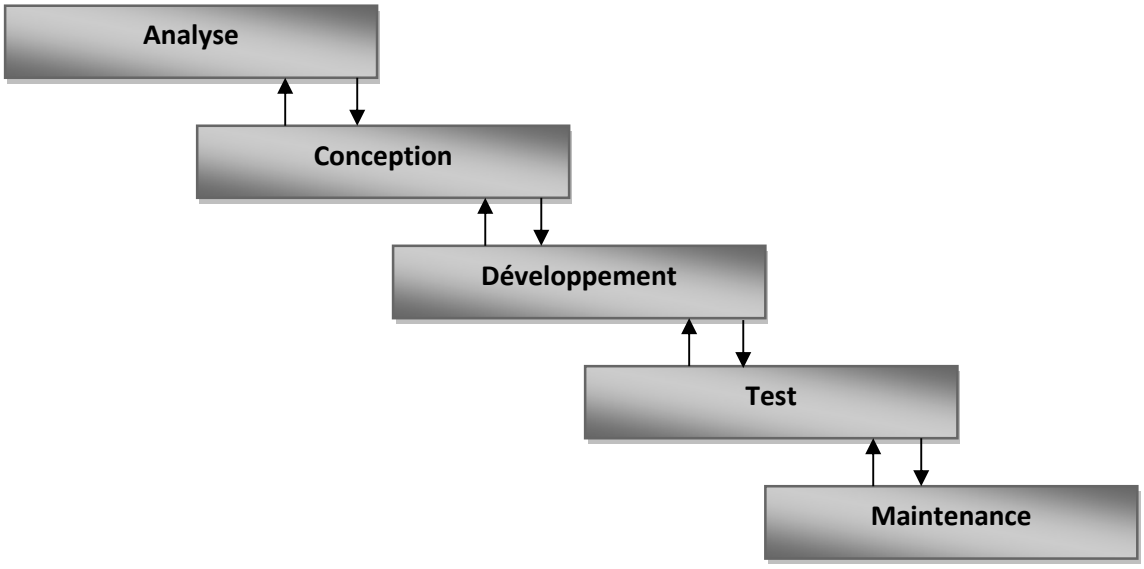


Figure 1: cycle en cascade

Avantages du modèle en cascade : Facilite l'établissement du planning et on sait à l'avance les livrable.

Inconvénients du modèle en cascade : le principal inconvénient c'est que les erreurs sont détectées tardivement, ce qui résulte un coût important en cas d'anomalie.

Modèle en V

Permet de définir les procédures de vérification de la conformité du logiciel aux spécifications depuis les premières phases.

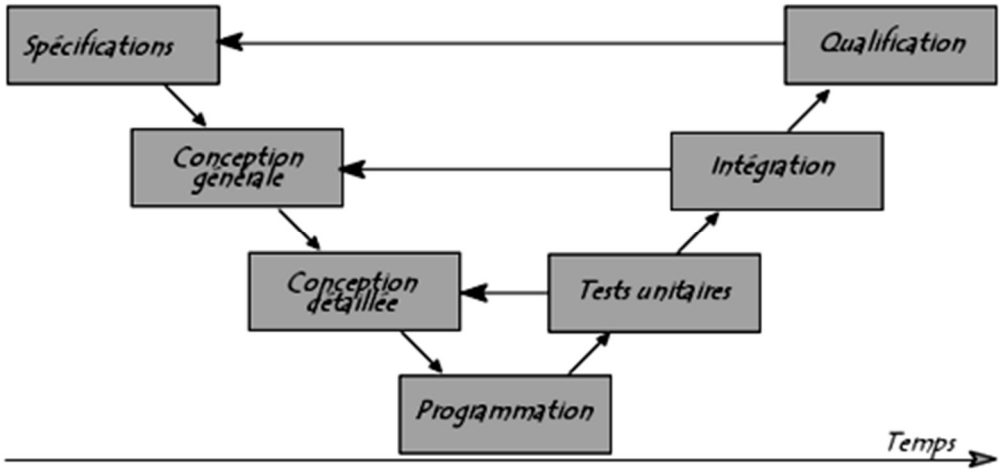


Figure 2: cycle en V

Avantages du modèle en V : permet de construire un livrable final parfait, puisque les étapes de test sont nombreuses.

Inconvénients du modèle en V: ce modèle est utilisé juste pour les projets de petites tailles.

Cycle en spirale

Permet la définition de **versions successives**, en se basant sur la **gestion des risques.**

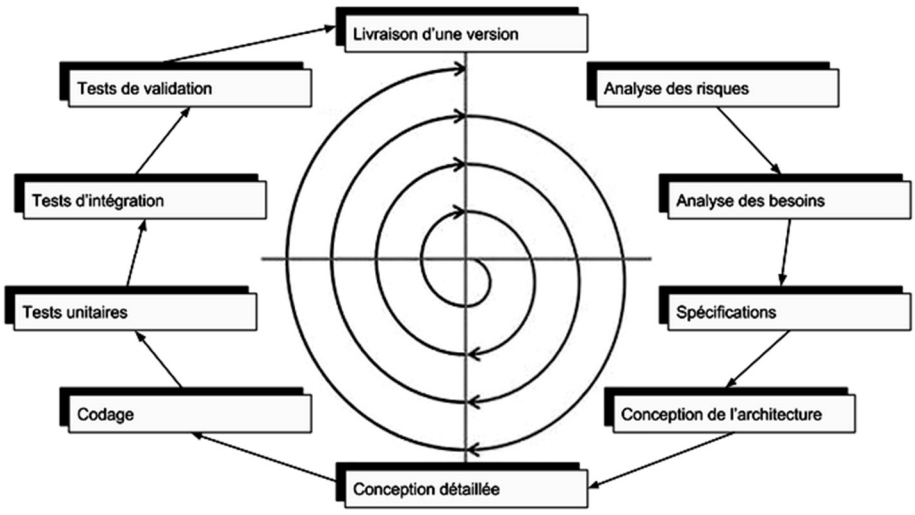


Figure 3: cycle en spirale

On distingue quatre phases :

1. déterminer les objectifs ;
2. déterminer des risques ;
3. développement et test ;
4. planification de l'itération suivante.

Avantages du modèle en spirale : la gestion des risques.

Inconvénients du modèle en spirale : utilisé juste dans les grands projets

Cycle itératif

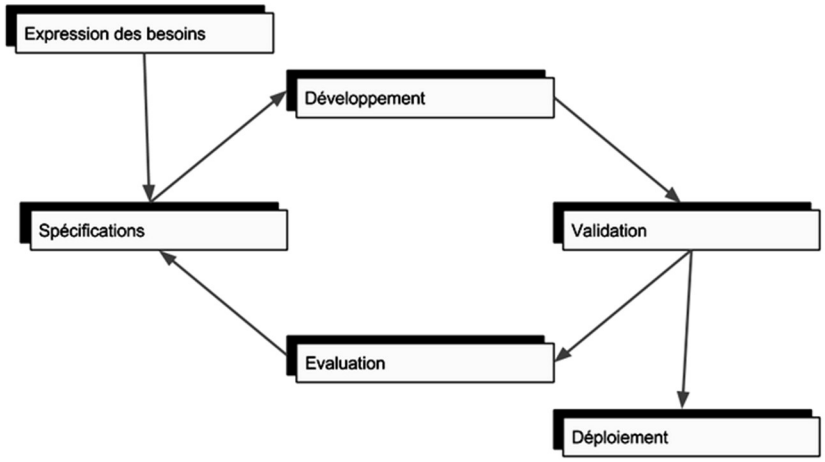


Figure 4: Cycle itératif

Avantages du modèle itératif : Ce type de cycle de développement est le plus souple. Ainsi le projet fini peut varier du besoin qui a été exprimé à l'origine.

Inconvénients du modèle itératif : Ce modèle néglige le test d'intégration.

Partie 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
- En 2017
  - UML 2.5.1

1er Chapitre: DIAGRAMME UML DES CAS D'UTILISATION

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.

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 5: représentation d'acteur

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.

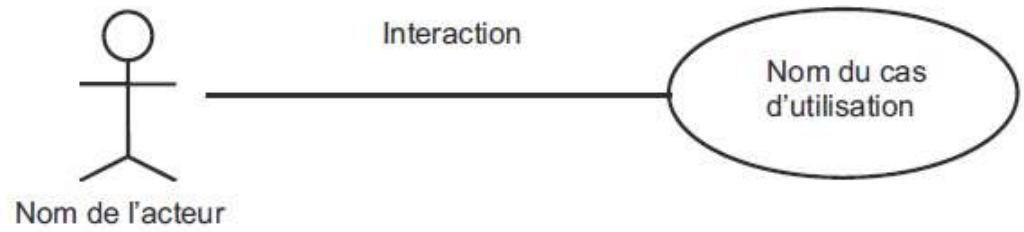


Figure 6: interaction acteur et cas d'utilisation

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.

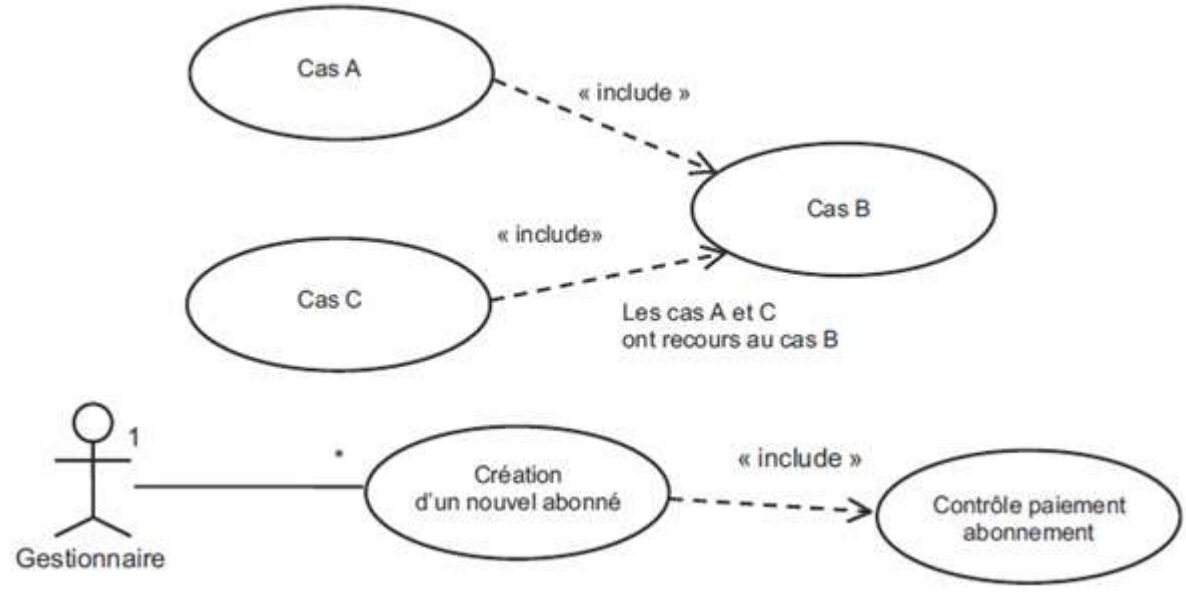


Figure 7: Formalisme et exemple de la relation d'inclusion

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

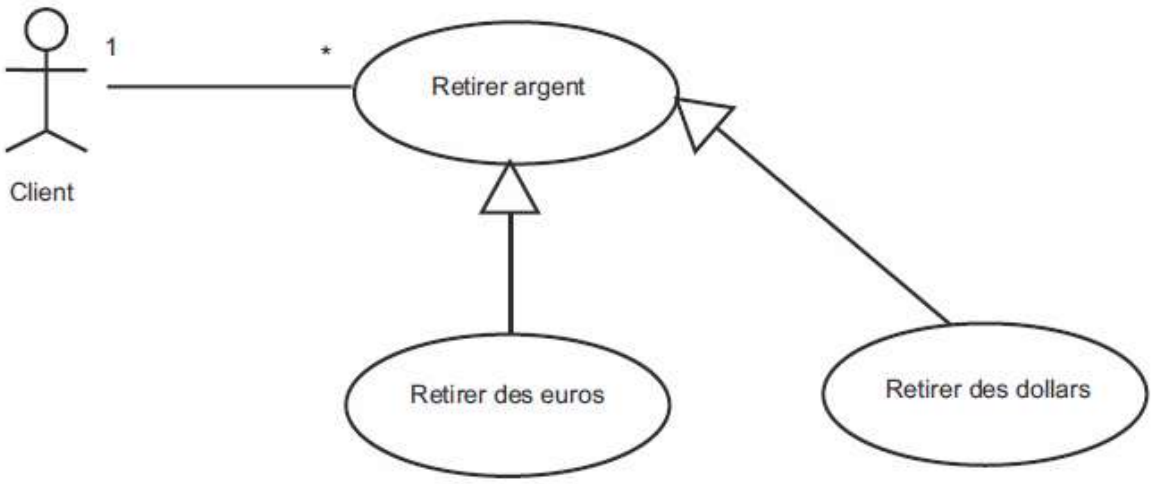


Figure 8: Exemple d'une relation de généralisation de cas d'utilisation

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.

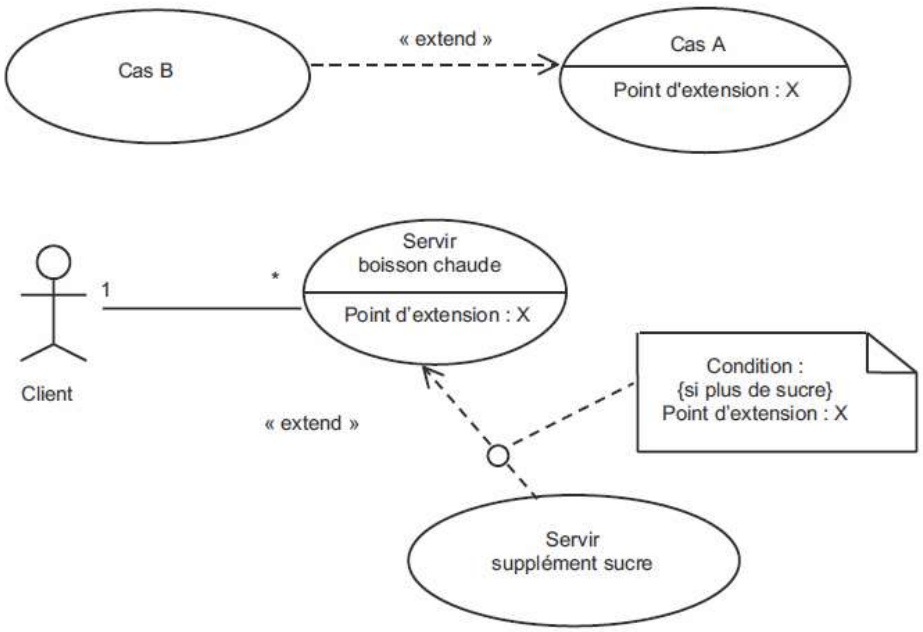


Figure 9: Formalisme et exemple d'une relation d'extension

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.

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).

2ème chapitre : DIAGRAMME DE SÉQUENCE (DSE)

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 6.

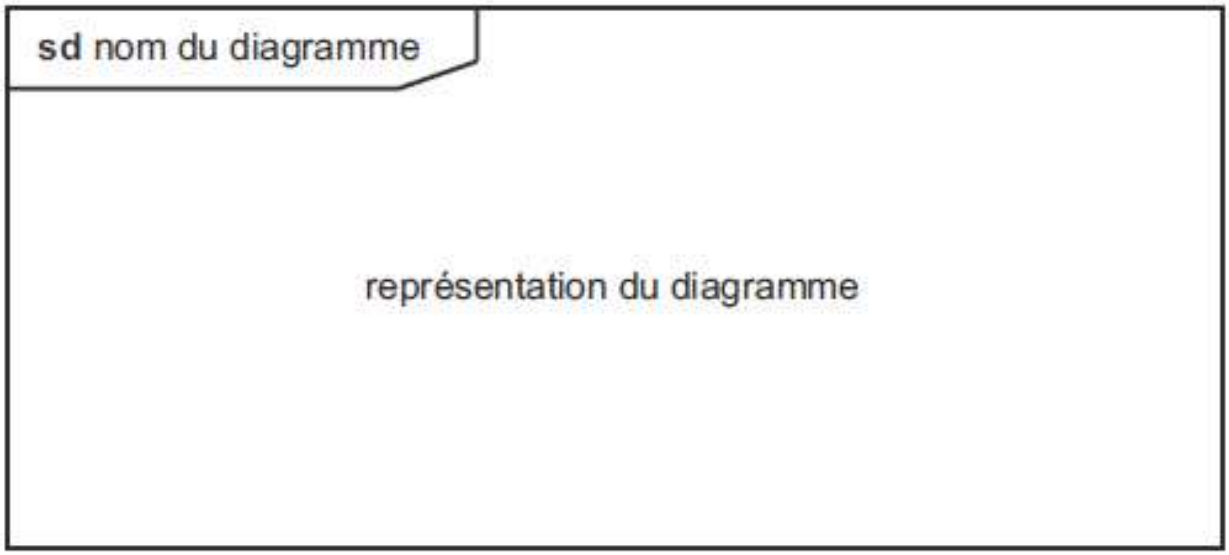


Figure 6: Formalisme général du cadre d'un diagramme de séquence.

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.

Opérations particulières

1.1.1 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 7.

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 »

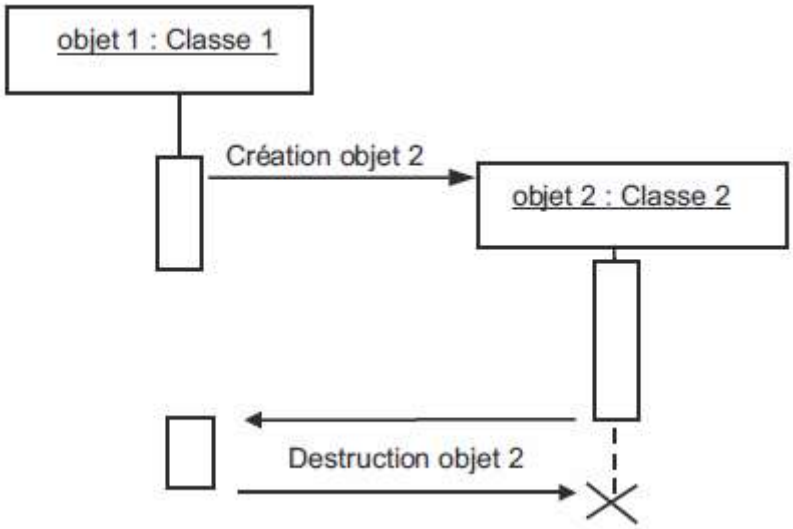


Figure 7: Exemple type de création et de destruction d'objet

1.1.2 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 8.

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.

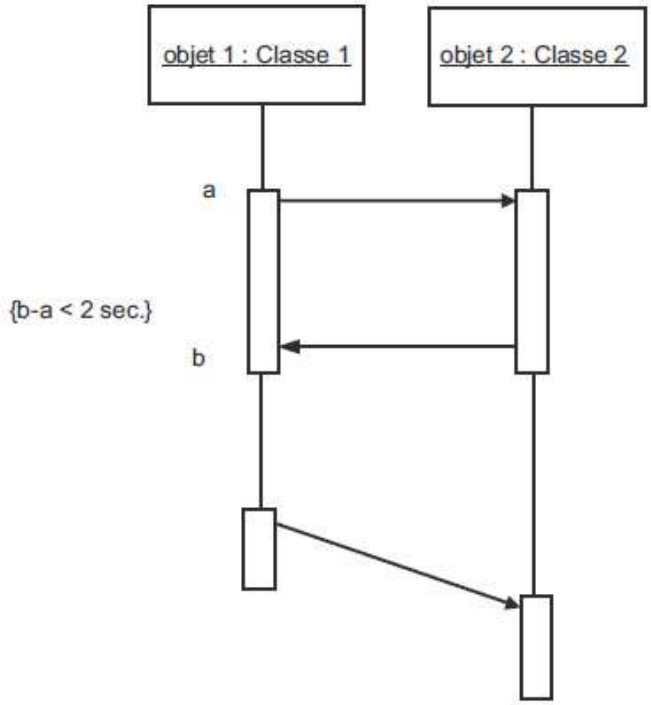


Figure 8: Exemple type de représentation de contrainte temporelle

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.

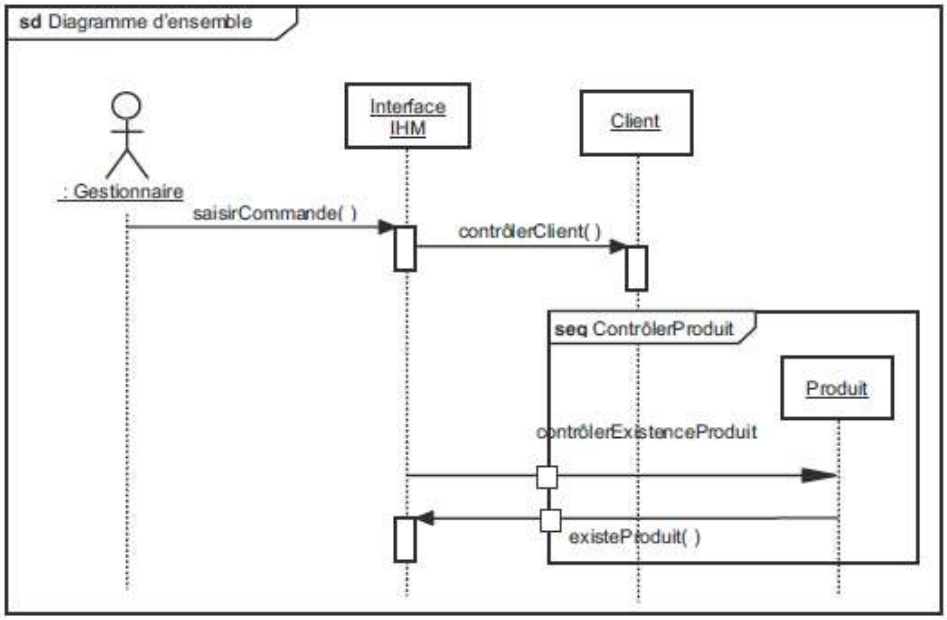


Figure 9: Exemple de fragment d'interaction avec port d'entrée et de sortie

Dans l'exemple proposé (fig. 9), 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.

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. 10) montre l'équivalent d'un test à deux conditions explicites.

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. 11).

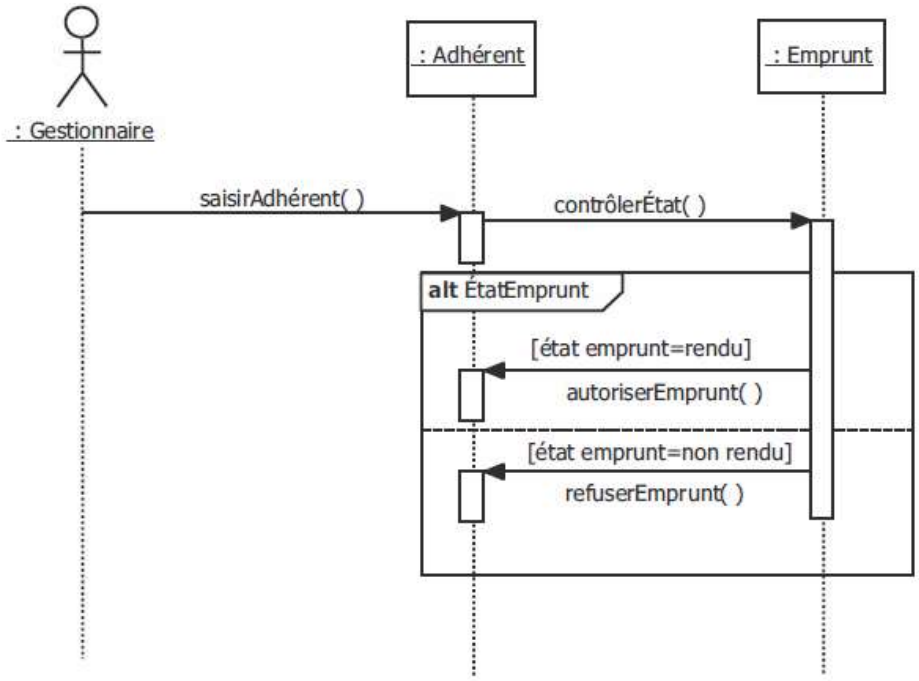


Figure 10 Exemple de fragment d'interaction avec l'opérateur alt

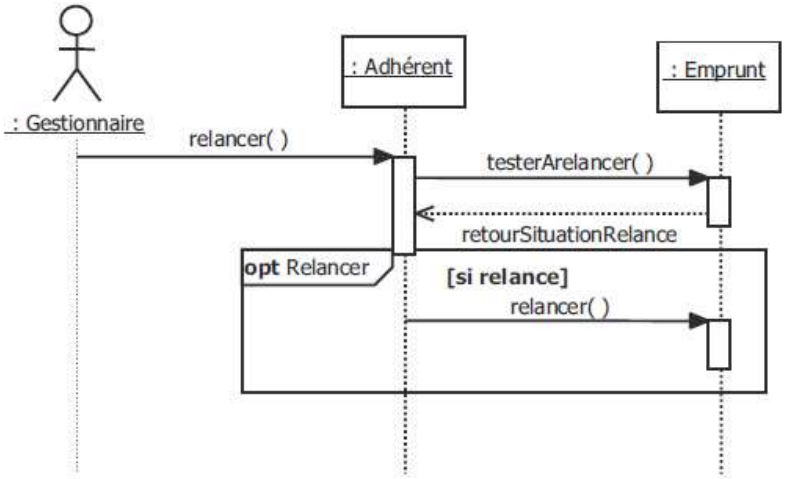


Figure 11: Exemple de fragment d'interaction avec l'opérateur opt

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 12.

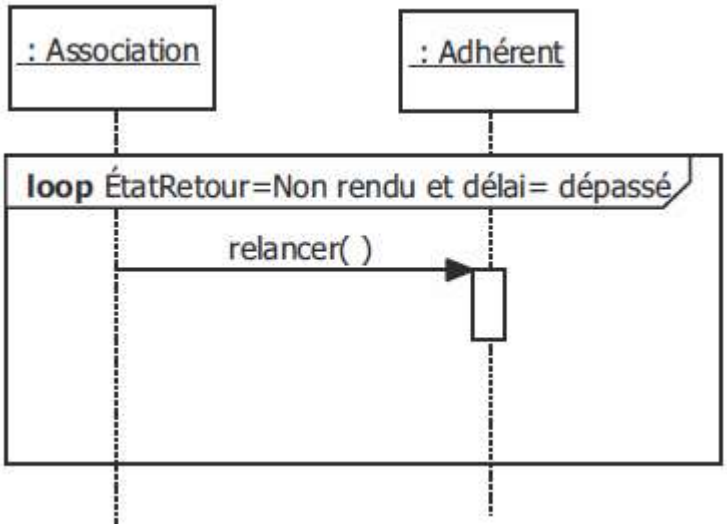


Figure 10:Exemple de fragment d'interaction avec l'opérateur loop

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).

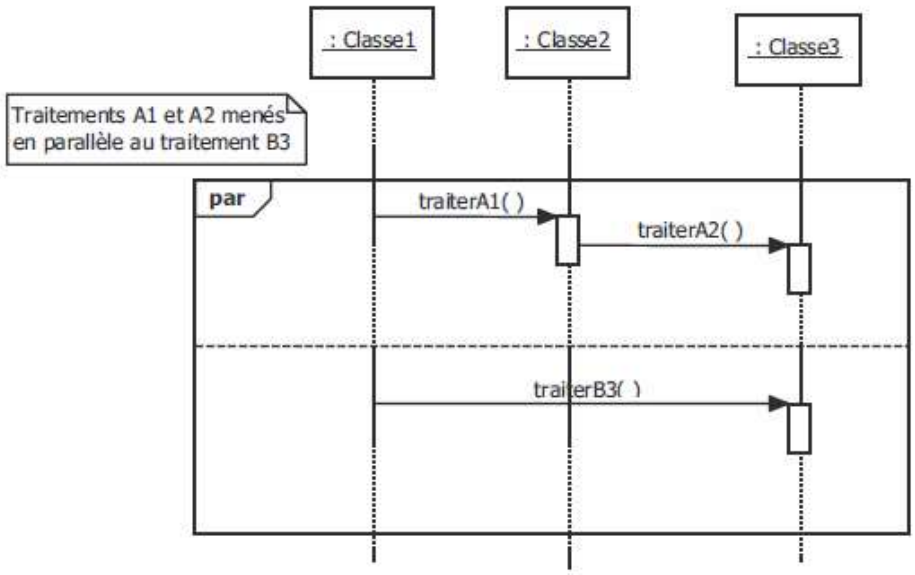


Figure 11: Exemple de fragment d'interaction avec l'opérateur par

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.

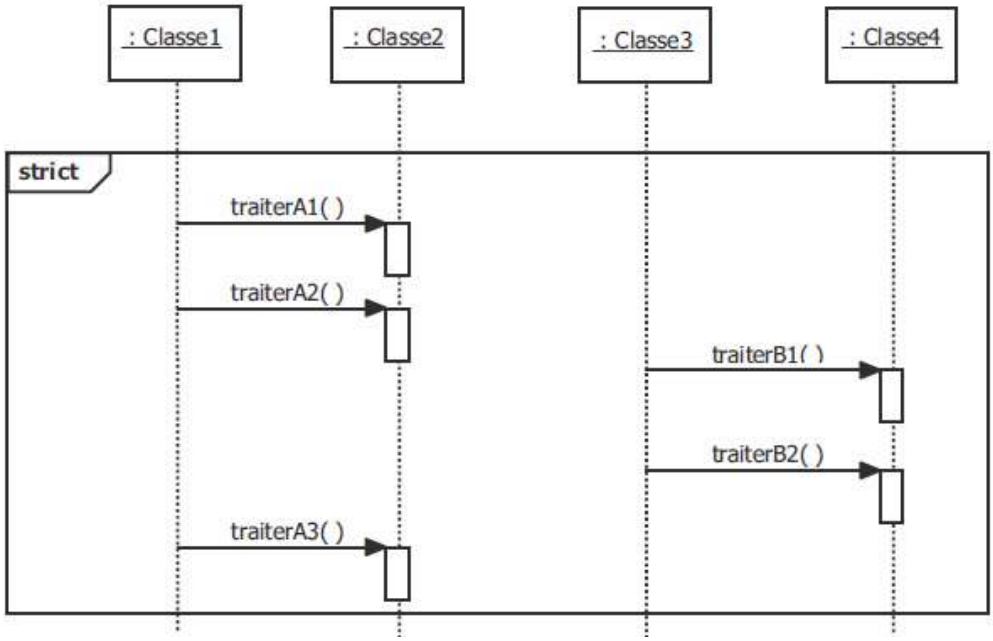


Figure 12: Exemple de fragment d'interaction avec l'opérateur strict

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( ).

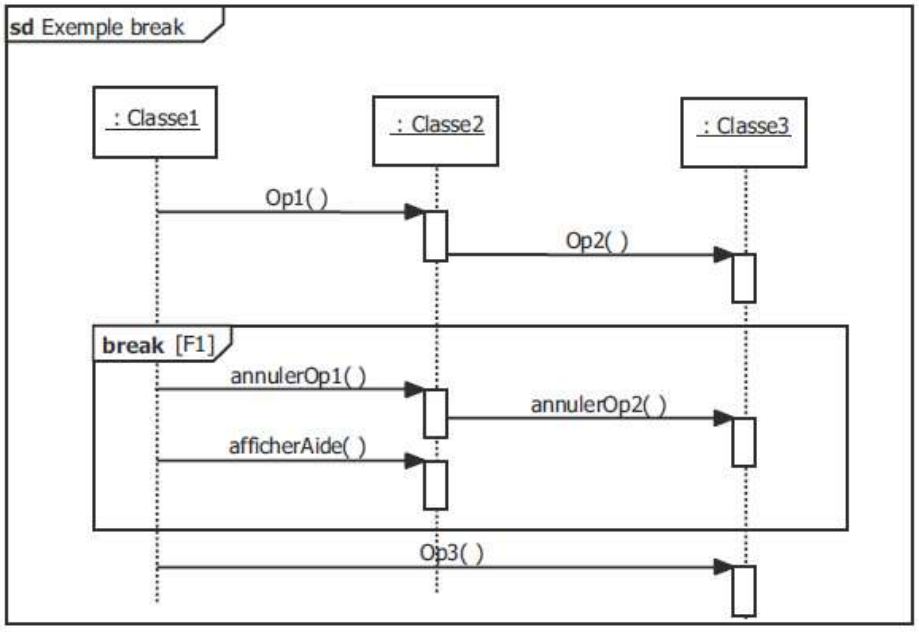


Figure 13: Exemple de fragment d'interaction avec l'opérateur break

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.

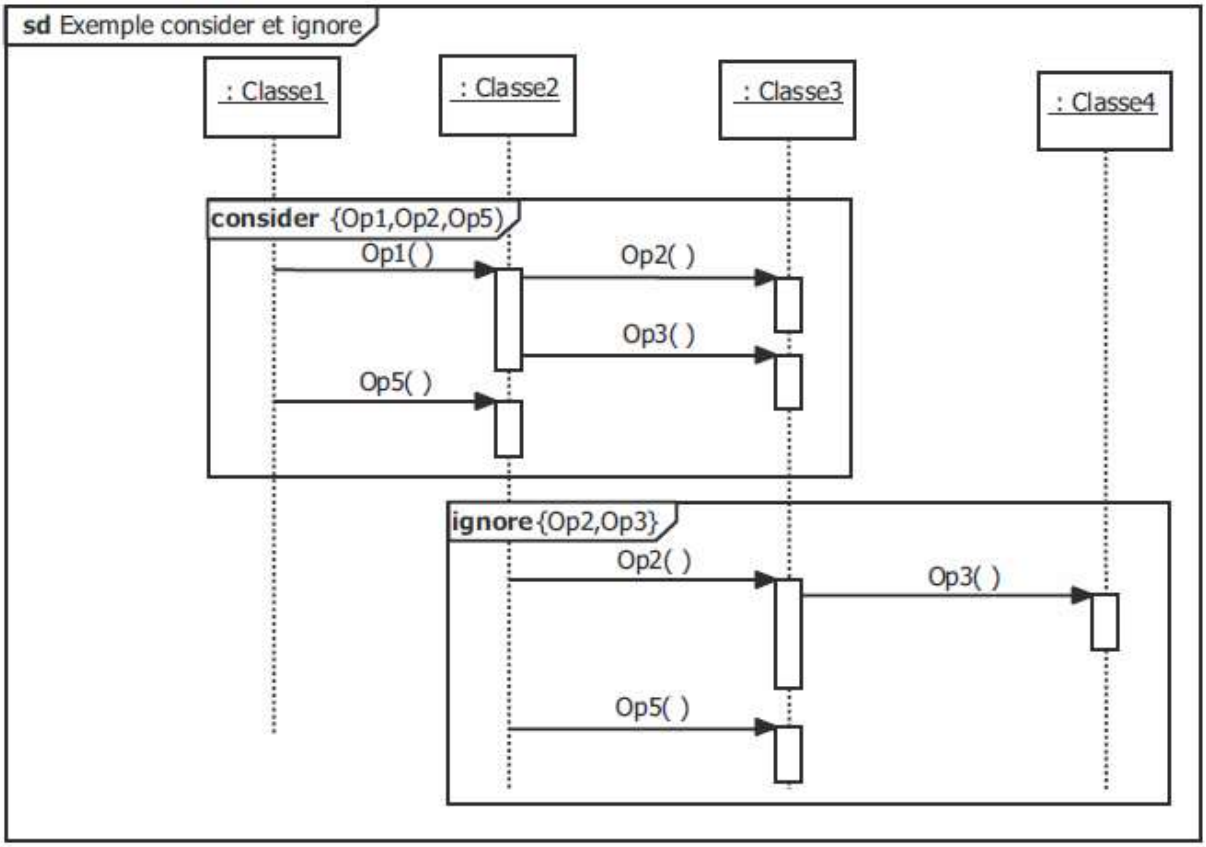


Figure 14: Exemple de fragment d'interaction

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**.

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.

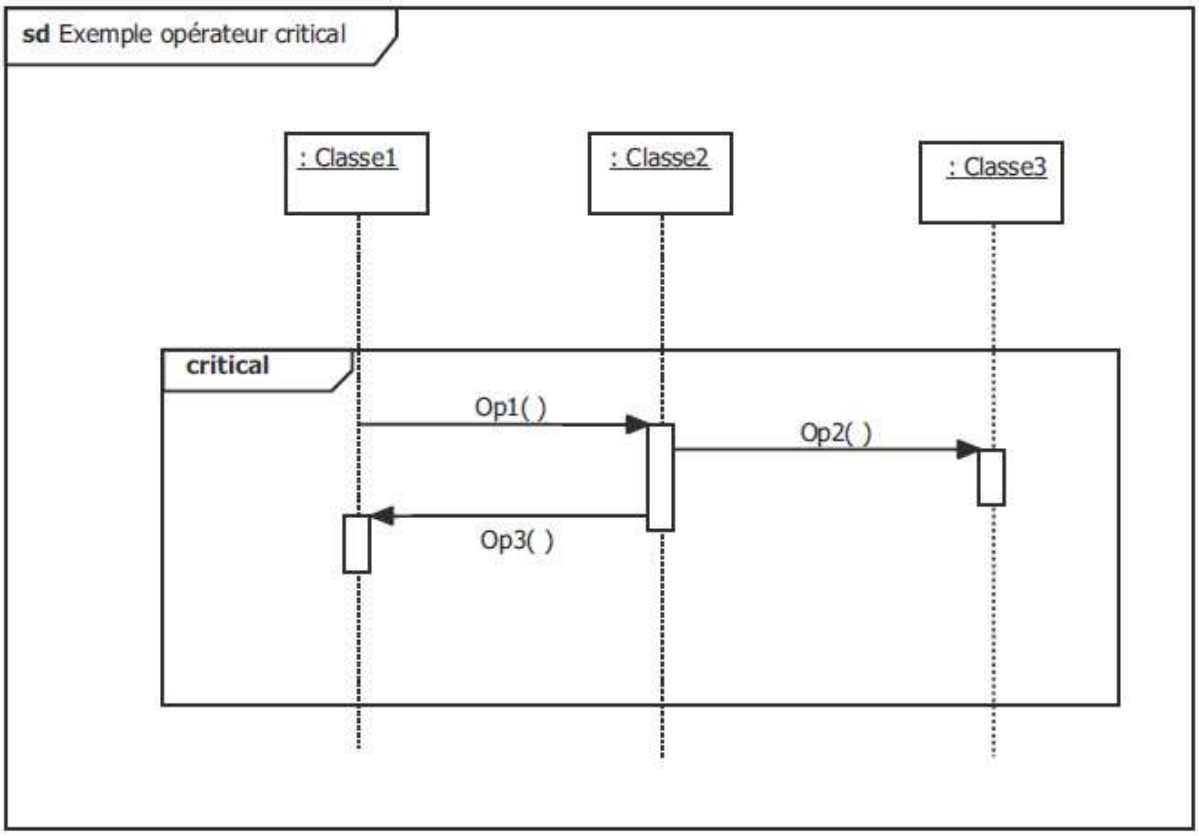


Figure 15: Exemple de fragment d'interaction avec l'opérateur critical

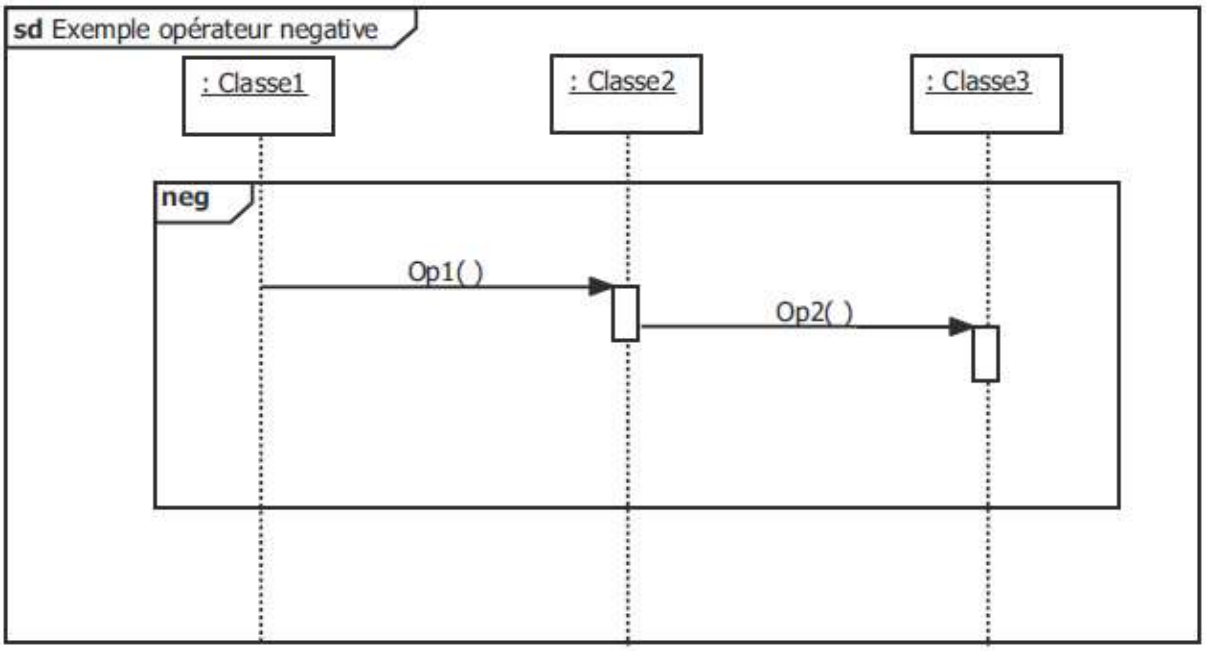


Figure 16: Exemple de fragment d'interaction avec l'opérateur neg

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.

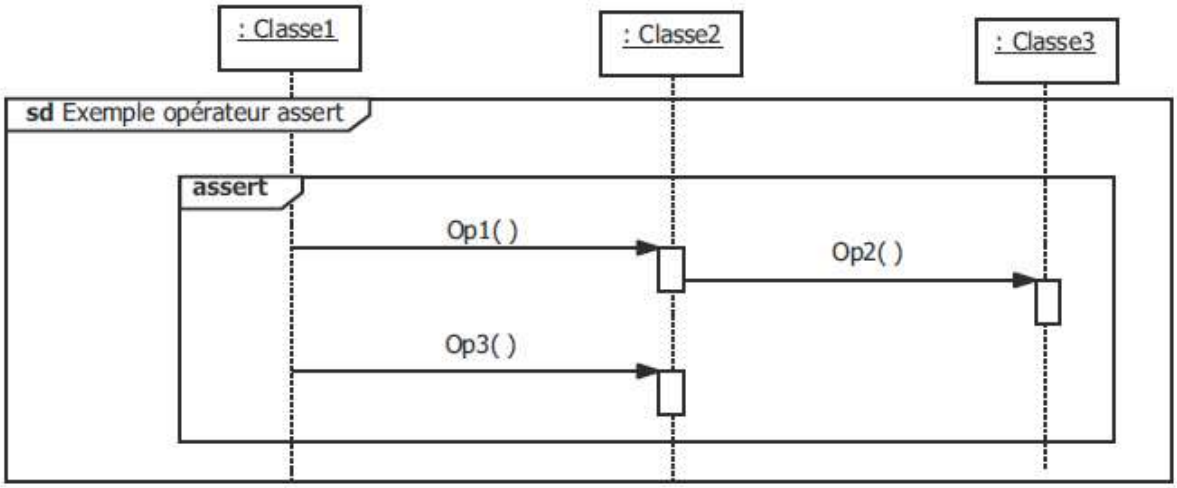


Figure 17: Exemple de fragment d'interaction avec l'opérateur assert

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.

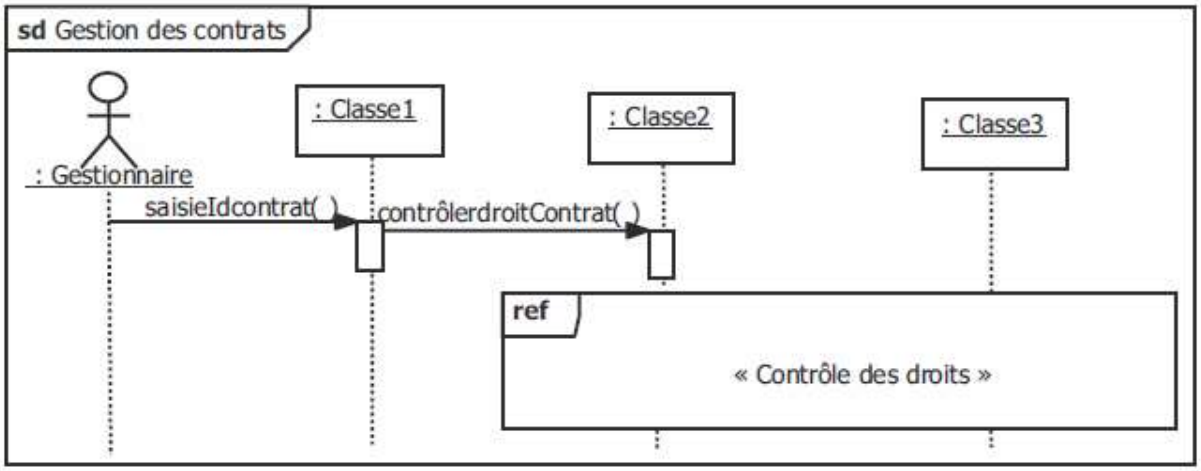


Figure 18: Exemple de fragment d'interaction avec l'opérateur ref

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.

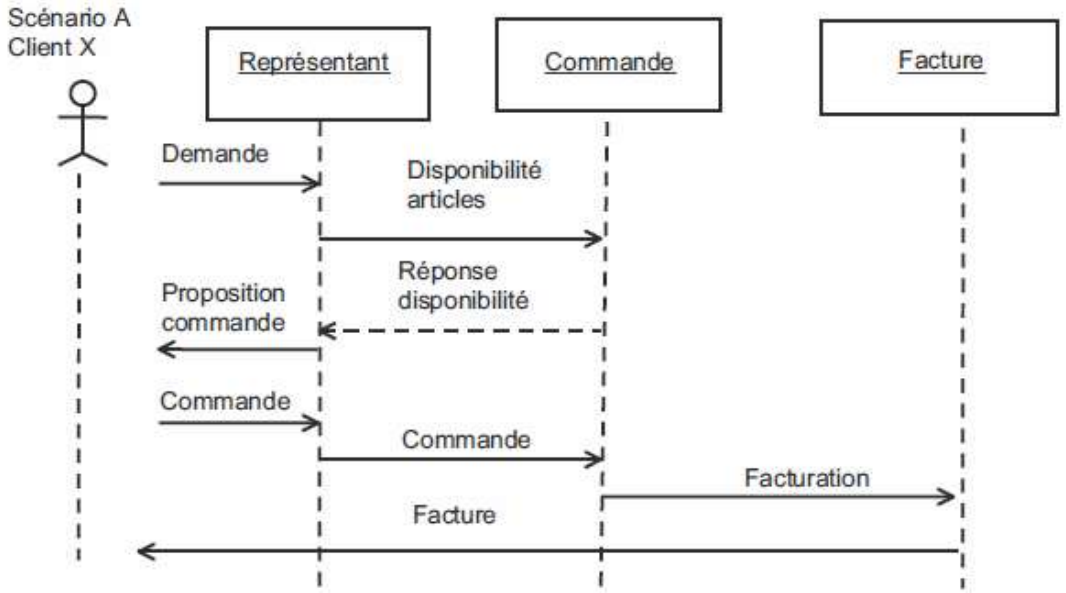
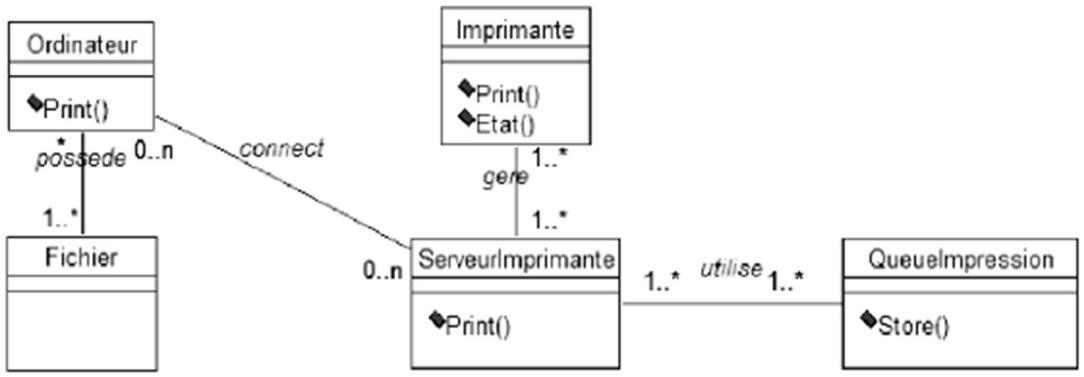


Figure 19: Exemple de diagramme de séquence associé à un cas d'utilisation

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.