Version expérimentale
En cours de validation

RÉSUMÉ THÉORIQUE – DÉVELOPPEMENT DIGITAL OPTION WEB FULL STACK

M109 – PRÉPARER UN PROJET WEB

Elaboré par :

Kawtar HARMOUCHI
Formatrice au CFMOTI – CASABLANCA

60 heures

Equipe de rédaction et de lecture

1.Equipe de conception:

Mme Kawtar HARMOUCHI : Formatrice en Développement Digital

2.Equipe de lecture:

Mme Soukaina LAOUIJA : Formatrice Animatrice en CDC Digital & IA

Mme Ghizlane EL KHATTABI : Formatrice Animatrice en CDC Digital & IA

SOMMAIRE

MODALITÉS PÉDAGOGIQUES

1

LE GUIDE DE SOUTIEN

Il contient le résumé théorique et le manuel des travaux pratiques

2

LA VERSION PDF

Une version PDF est mise en ligne sur l'espace apprenant et formateur de la plateforme WebForce Life

3

DES CONTENUS TÉLÉCHARGEABLES

Les fiches de résumés ou des exercices sont téléchargeables sur WebForce Life

4

DU CONTENU INTERACTIF

Vous disposez de contenus interactifs sous forme d'exercices et de cours à utiliser sur WebForce Life

5

DES RESSOURCES EN LIGNES

Les ressources sont consultables en synchrone et en asynchrone pour s'adapter au rythme de l'apprentissage

PARTIE 1

Modéliser un projet web

Dans ce module, vous allez :

- Appréhender le cycle de vie d'un projet web
- Modéliser les besoins client par un diagramme de cas d'utilisation
- Modéliser les données du projet par un diagramme de classes
- Modéliser les interactions d'objets d'un système à l'aide d'un diagramme de séquence
- Elaborer des diagrammes UML à l'aide d'un outil de modélisation

30 heures

CHAPITRE 1

Appréhender le cycle de vie d’un projet web

Ce que vous allez apprendre dans ce chapitre :

- Cycle de vie d’un projet (Définition, étapes,…)
- Analyse des besoins
- Conception
- Développement
- Tests et déploiement
- Maintenance et évolutivité

5 heures

❑ **La conception d'applications web** est devenue incontournable à l'équivalent des projets « classiques ». Cette conception vise à répondre à certaines questions :

- **Que vient faire l'internaute sur le site ? Quelles informations s'attend-il à trouver ?**
- **Comment ces informations sont-elles structurées, reliées entre elles, mises à jour ?**
- **Comment garantir que les choix de réalisation de l'application web sont bien adaptés aux objectifs de l'utilisateur ?**

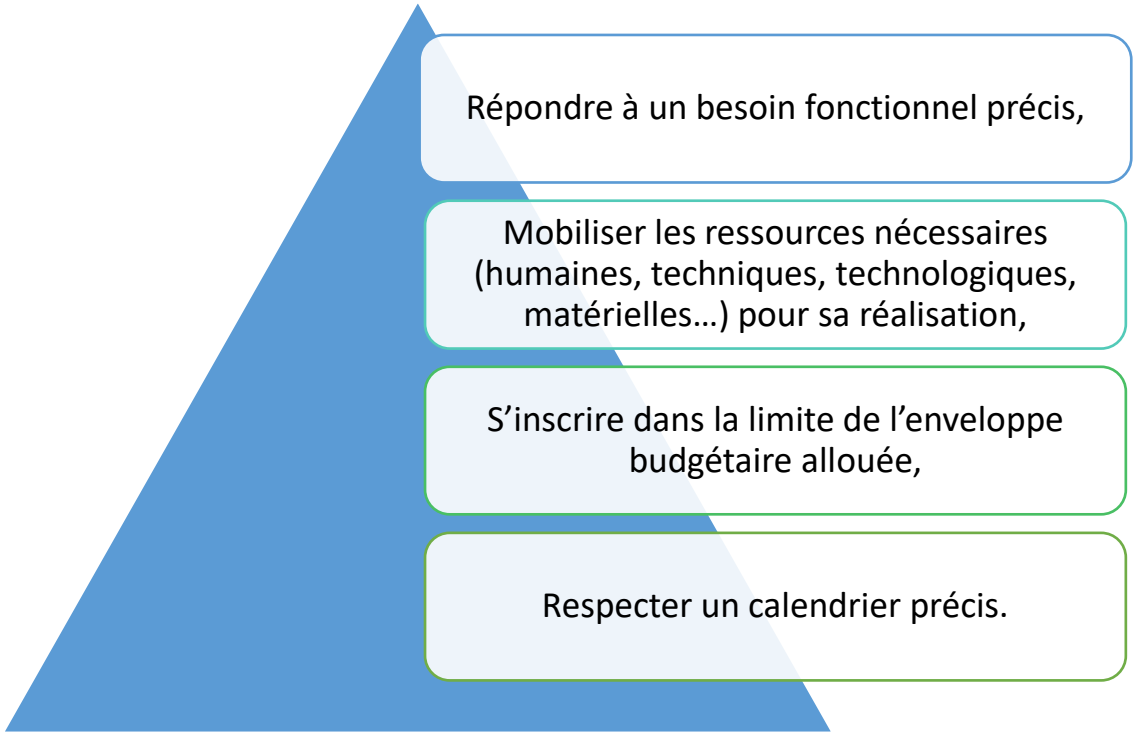
La réponse tient en un seul mot : **modéliser**

❑ Depuis quelques années, **la modélisation objet avec le langage UML** est devenue incontournable sur la plupart des projets informatiques. Alors pourquoi ne pas appliquer aux projets web ce qui marche pour les projets « classiques »?

❑ Contrairement à une idée répandue, les applications web sont justement, de par leur complexité croissante, des candidates idéales à la modélisation graphique et à l'application d'un processus de développement formalisé.

Projet web

- Un projet web est un projet informatique dont les livrables (résultats concrets attendus) ont pour destination finale, le réseau Internet.
- Le projet web doit :



Objectifs d'un projet web

Avant de démarrer un projet web, il est impératif pour l'entreprise de définir clairement ce que l'on attend concrètement de son projet web. Voici quelques objectifs fondamentaux :

La notoriété, la crédibilité, l'e-réputation

- l'image de la marque,

La visibilité, la popularité

- la repérabilité,

La mobilité, le Responsive Web Design

- l'offre multi-écran,

La qualité et la pérennité

- l'état de l'art,

La rentabilité et la productivité

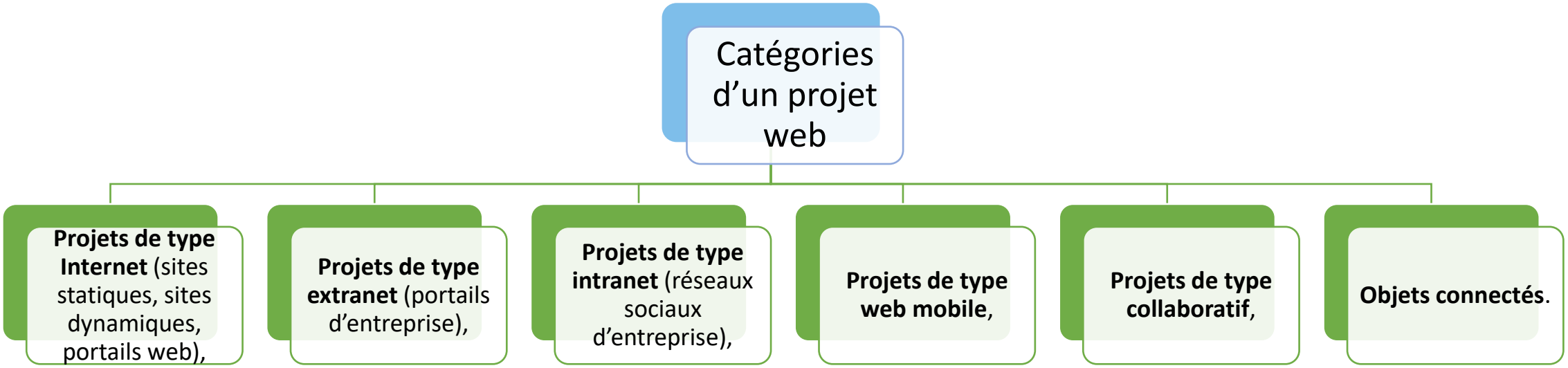
- le retour sur investissement,

La simplicité,

L'utilisabilité.

PARTIE 1

Catégories d'un projet web



CHAPITRE 1

Appréhender le cycle de vie d’un projet web

1. **Cycle de vie d’un projet (Définition, étapes,…)**
2. Analyse des besoins
3. Conception
4. Développement
5. Tests et déploiement
6. Maintenance et évolutivité

Qu'est-ce que le cycle de vie d'un projet ?

Le cycle de vie d'un projet décrit le **processus à suivre** pour mener à bien un projet, de la date de démarrage jusqu'à la clôture du projet. Il contient généralement 4 phases: la phase de cadrage, la phase de planification, la phase d'exécution et la phase de clôture.

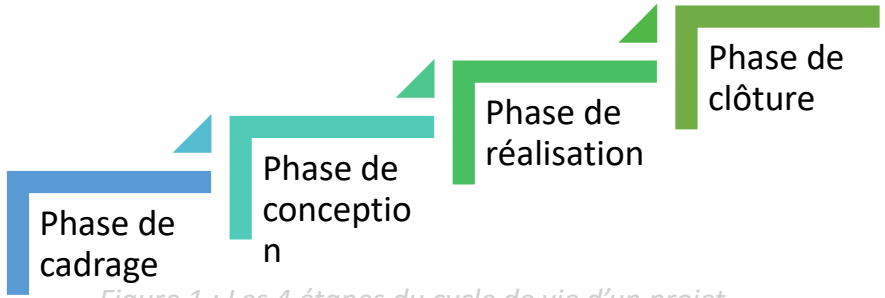


Figure 1 : Les 4 étapes du cycle de vie d'un projet

- **Phase de cadrage** : elle a pour objectif de cadrer le projet, identifier le besoin à l'origine du projet, ses enjeux et ses objectifs, le contexte du projet, le périmètre et ses limites, ...
- **Phase de conception** : nommée aussi phase de planification ou phase de préparation. Lors de cette étape, le chef de projet, accompagné de l'équipe projet, définit qui fait quoi, quand et comment.
- **Phase de réalisation** : C'est lors de cette phase que les actions sont réalisées, dans le respect du planning, du budget et des échéances fixées avec le client ou le commanditaire du projet.
- **Phase de clôture** : phase de finalisation ou de conclusion du projet.

PARTIE 1

Modèles du cycle de vie

Selon le type de projet, il existe des modèles de cycle de de vie.

- **Modèle en cascade** : est une approche linéaire et séquentielle des activités d'un projet. Les étapes s'exécutent en séquence. Pour passer à l'étape suivante, l'étape précédente doit être finalisée. Une fois que c'est le cas, on ne peut plus revenir en arrière

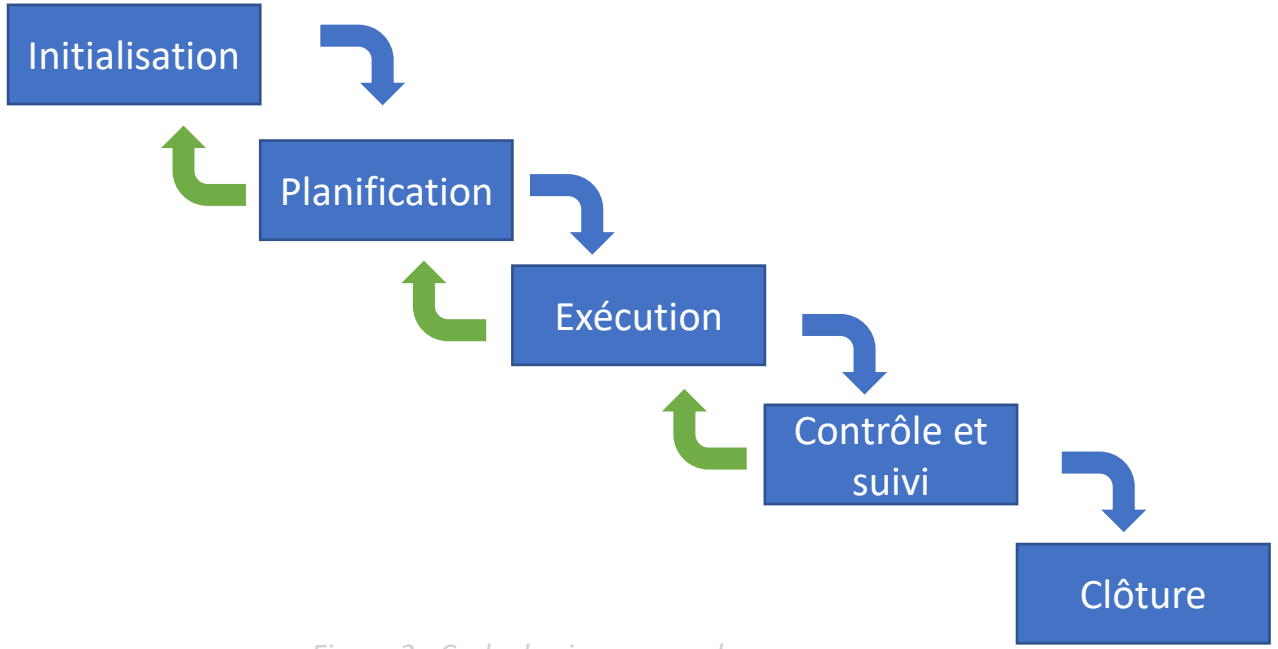


Figure 2 : Cycle de vie en cascade

PARTIE 1

Modèles du cycle de vie

Selon le type de projet, il existe des modèles de cycle de de vie.

- **Modèle en V** : adaptée pour le développement de projets informatiques. Cette méthode comprend une phase descendante, suivie d'une phase ascendante, illustrées par les deux branches de la lettre V :
  - La phase descendante correspond aux actions de conception et de développement du système.
  - La phase ascendante correspond aux actions de contrôle des exigences et de la qualité.
  - **Particularité** : il impose la création des dossiers de tests lors de chaque phase projet

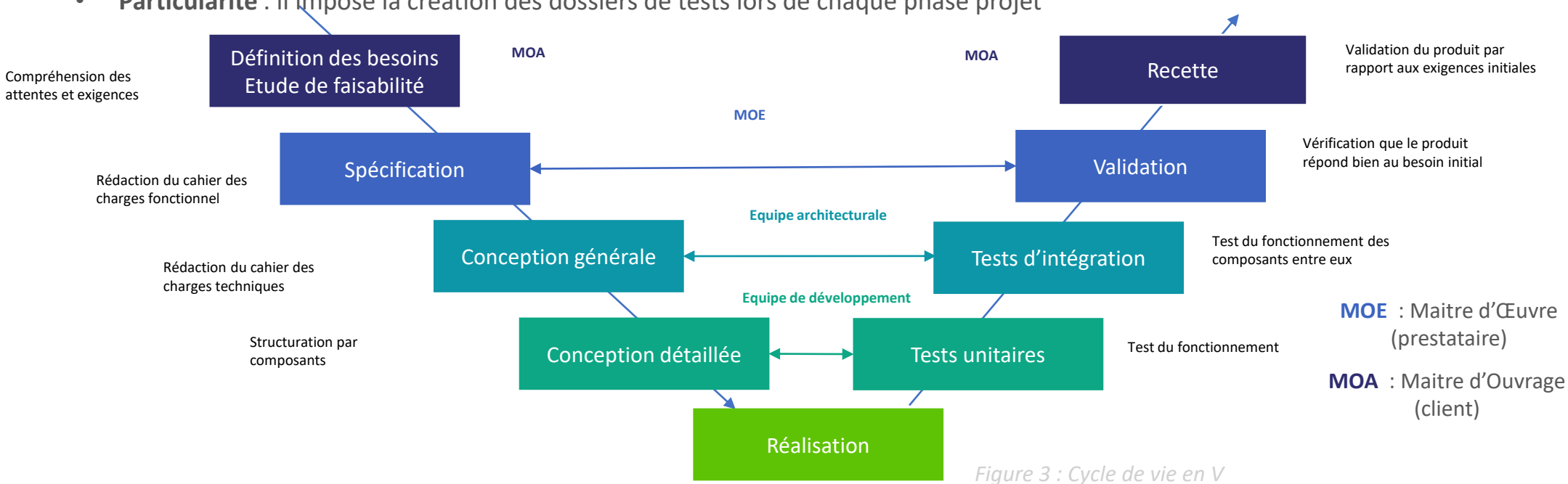


Figure 3 : Cycle de vie en V

Etapes principales du cycle de vie d'un projet web

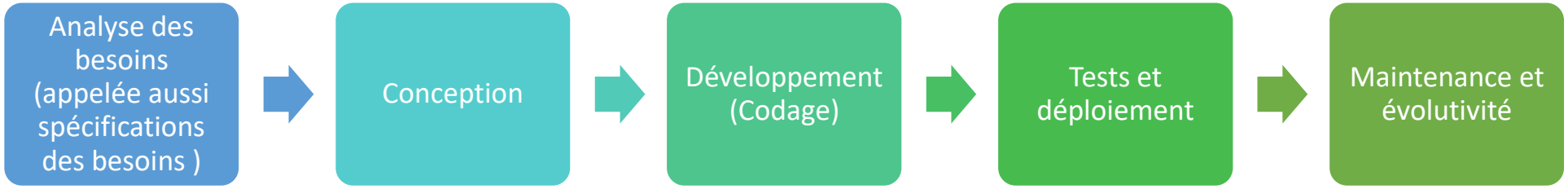


Figure 4 : Cycle de vie d'un projet web

CHAPITRE 1

Appréhender le cycle de vie d’un projet web

1. Cycle de vie d’un projet (Définition, étapes,…)
2. **Analyse des besoins**
3. Conception
4. Développement
5. Tests et déploiement
6. Maintenance et évolutivité

Définition des objectifs et besoins dans un cahier de charges web

- Avant de se lancer dans la conception et la réalisation d'un site web, il est recommandé d'établir clairement les objectifs, la liste des besoins et les résultats attendus dans **un cahier de charge web**.
- Avoir une liste des spécifications permet de hiérarchiser les développements en fonction de leur importance et d'établir ainsi des priorités :

Les spécifications opérationnelles

qui définissent les objectifs et les bénéfices attendus en termes marketing et métier (besoins métiers).

Les spécifications utilisateurs

qui décrivent les attentes des utilisateurs et comment ils vont interagir avec les fonctionnalités proposées (besoins utilisateur).

Les spécifications fonctionnelles

qui donnent des détails sur le comportement des services, des fonctionnalités et des contenus à développer.

Les spécifications qualitatives

qui indiquent les caractéristiques des services et des fonctionnalités et les implications éventuelles qu'ils peuvent avoir sur les processus existants.

PARTIE 1

Eléments clés d'un cahier de charges

- Le cahier des charges web est peut-être le document le plus important du projet web,
- Le cahier des charges est **contractuel** et sert de document de référence en cas de litige,
- Le cahier des charges est rédigé par la maîtrise d'ouvrage (MOA) dans un langage non technique et est à destination de la maîtrise d'œuvre-MOE (celle-ci peut être interne ou externe à l'entreprise).,
- La rédaction du cahier de charges est fonction du projet web et de l'entreprise (MOA), il doit contenir :

- Présentation de l'entreprise et de son organisation
- Contexte
- Objectifs du projet web
- Eléments fonctionnels et techniques
- Planning, budget, qualité et organisation du projet
- Eléments juridiques

- **L'analyse des besoins**, appelée aussi **spécifications** des besoins,
- L'analyse des besoins du projet web c'est réaliser **trois interventions** complémentaires :

Analyse des besoins métier (besoins projet)

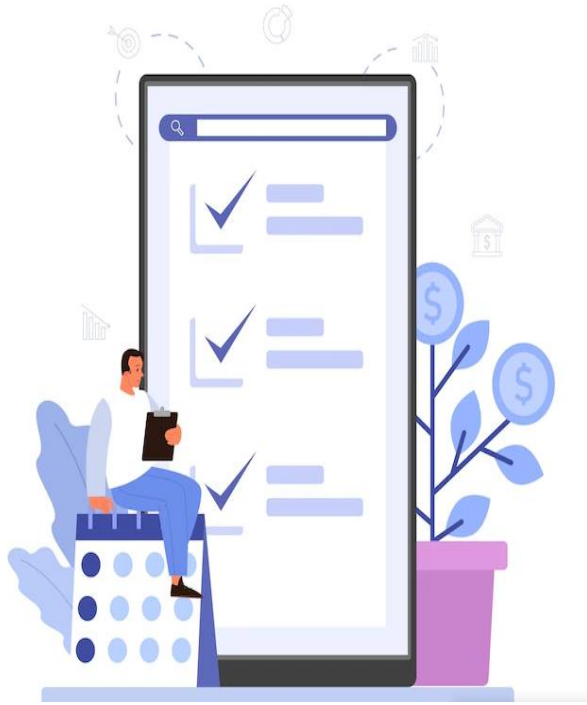
L'analyse de besoins métier est davantage orientée back-office (ce dont les administrateurs ont besoin) mais elle considère naturellement ce que doit voir l'utilisateur en front office. Elle consiste à :

Définir la feuille de route du projet web (objectifs, enjeux) :

- Cette feuille de route du projet décrit les **spécificités**, les **contraintes**, les **attentes** et les **besoins de chaque cellule de l'entreprise et de chaque corps de métier.** On peut répondre à certaines questions comme : Quelle audience viser ? Qui sont les concurrents pour le projet ? Comment se démarquer de la concurrence? Quels moyens à mettre en œuvre? Etc.

Fixer le périmètre du projet :

- pour chaque besoin, on affecte des **contenus** et des **fonctionnalités**. Il se base sur les entretiens et les réunions et prend forme au sein d'un tableau, tout simplement : besoin des équipes projet par service, et besoins identifiés de la cible. On peut répondre à des questions comme : **De quoi a-t-on besoin pour ? Comment peut-on faire pour ?**



Analyse des besoins métier (besoins projet)

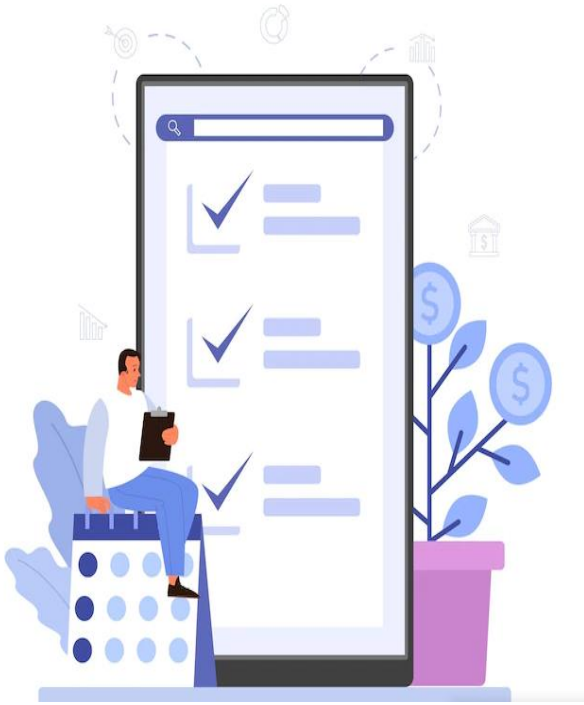
Par exemple:

Besoin des équipes SAV E-Commerce :

- Désengorger les appels au service SAV.
- Fournir des réponses aux questions les plus fréquentes.
- Valoriser le support des produits les plus vendus.

Contenu à créer :

- FAQ générale du SAV,
- Support produit (notice, FAQ, forums) pour les best-sellers.



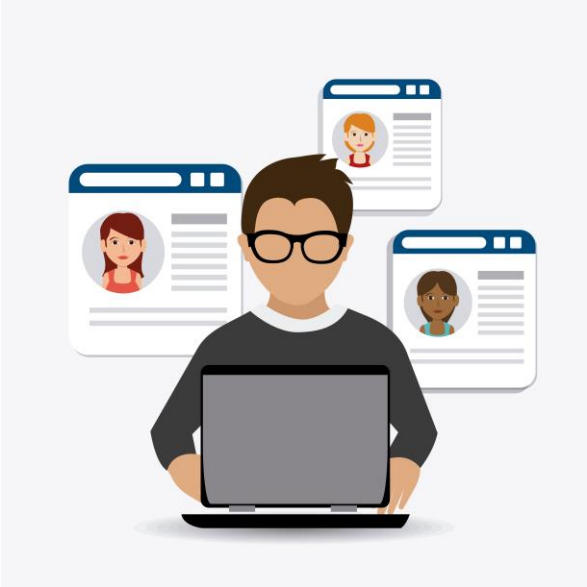
Analyse des besoins utilisateurs

Elle est davantage orientée **front-office** (ce que l'on voit côté utilisateur en matière d'interface). Il s'agit d'une extension, tout simplement, du périmètre projet, basé sur les inputs des utilisateurs.

Consiste à **identifier**, **affiner** et **prioriser** les besoins et attentes formulées par les utilisateurs par rapport à un sujet défini comme la réalisation d'un service digitalisé par exemple

Implique des **rencontres avec les utilisateurs finaux, les usagers, les clients**. Cela implique de bien connaître l'audience du dispositif digital (rencontrer 3 à 8 profils d'utilisateurs représentatifs de l'audience)

Repose sur les techniques fondamentales du marketing. On cherche à **enrichir les typologies de besoins en contenus et en fonctionnalités** en amont de la phase de conception.



Analyse fonctionnelle

L'analyse fonctionnelle est orientée sur les solutions : les fonctionnalités, les contenus.

On peut construire l'analyse fonctionnelle sur la base d'un tableau considérant deux données fondamentales :

Les données d'Entrée (Inputs) : les informations recueillis au cours de l'étude des besoins sur le terrain

Les données Sortie (Outputs) : les réponses que le dispositif digital pourra potentiellement apporter en réponse aux besoins.

CHAPITRE 1

Appréhender le cycle de vie d’un projet web

1. Cycle de vie d’un projet (Définition, étapes,…)
2. Analyse des besoins
3. **Conception**
4. Développement
5. Tests et déploiement
6. Maintenance et évolutivité

Définition

- La conception utilise les spécifications pour décider des solutions proposées. Elle peut contenir la description des fonctionnalités de l'application précisées lors de la spécification des besoins.
- La conception implique les facettes suivantes :

Conception fonctionnelle

Conception graphique

Conception technique

Conception fonctionnelle

Lors de cette phase, les acteurs du projet web vont imaginer et décrire le site web. Les étapes obligatoires sont :

- La structuration du contenu
- La navigation du site
- Les spécifications fonctionnelles détaillées du site

Conception fonctionnelle – Structuration contenu

Il convient **d'organiser tous les contenus du site web** de manière optimale.

Un site web doit être conçu selon une architecture cohérente facilitant la navigation des visiteurs dans vos pages

Il faut les **catégoriser** : les regrouper dans une logique attendue par les internautes

puis les **structurer** afin de faire ressortir les éléments les plus intéressants pour les visiteurs.

Il existe 3 modèles de structures de contenu :

a. **Structure hiérarchique :** adapté aux structures complexes

Les sites web sont généralement organisés autour d'une même page d'accueil celle ci est reliée aux autres rubriques et au menu des sous rubriques.

Eviter les structures trop profondes.

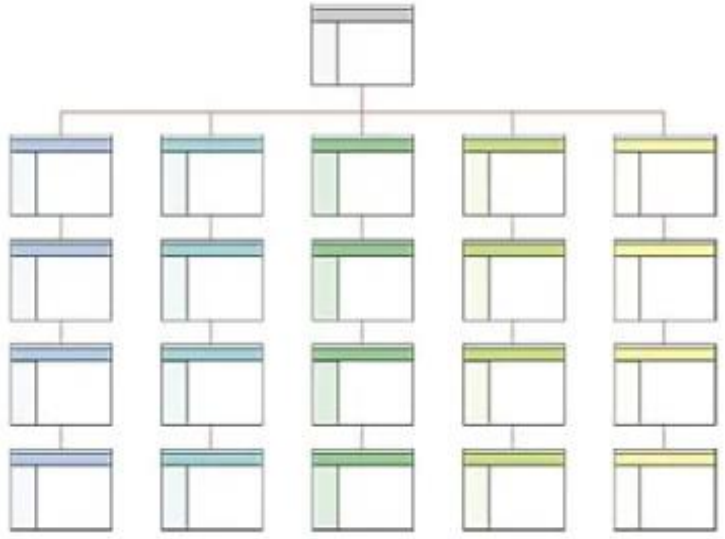


Figure 4 : Structure hiérarchique à plusieurs niveaux

PARTIE 1

Conception fonctionnelle – Structuration contenu

b. Structure séquentielle :

l'organisation des pages est linéaire selon un ordre chronologique (étape 1, étape 2, etc.),
un peu comme dans un livre.
On retrouve cette organisation sur les sites de formation ou les tutoriels en ligne.

Figure 5 : Structure séquentielle

c. Structure en réseau :

imiter la pensée associative et la libre circulation des idées ;
Souvent réalisé pour des utilisateurs expérimentés.
Permettre de trouver toute l'information dans un domaine bien précis
plus pratique pour les internautes ; Car il est toujours difficile de prédire
comment va naviguer l'internaute

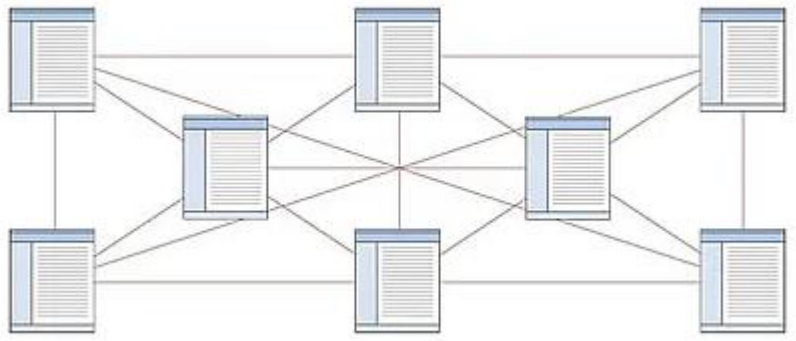


Figure 6 : Structure en réseau

PARTIE 1

Conception fonctionnelle – Structuration contenu

Une fois les bases de la navigation maîtrisées, on doit procéder au :

Rubriquage

- Choisir **les rubriques et leurs intitulés des menus de navigation**. Ceci aidera à constituer l'arborescence du site web. Dans un menu, une rubrique peut être composée d'une ou plusieurs pages ou d'un lien externe.

Plan du site

- C'est une représentation de l'architecture d'un site Internet qui liste les ressources proposées, en général sous forme hiérarchique. Il s'agit le plus souvent d'une page web qui permet à l'internaute d'accéder rapidement à l'ensemble des pages proposées à la lecture, et facilite le travail des robots d'indexation (utilisé pour le référencement du site).

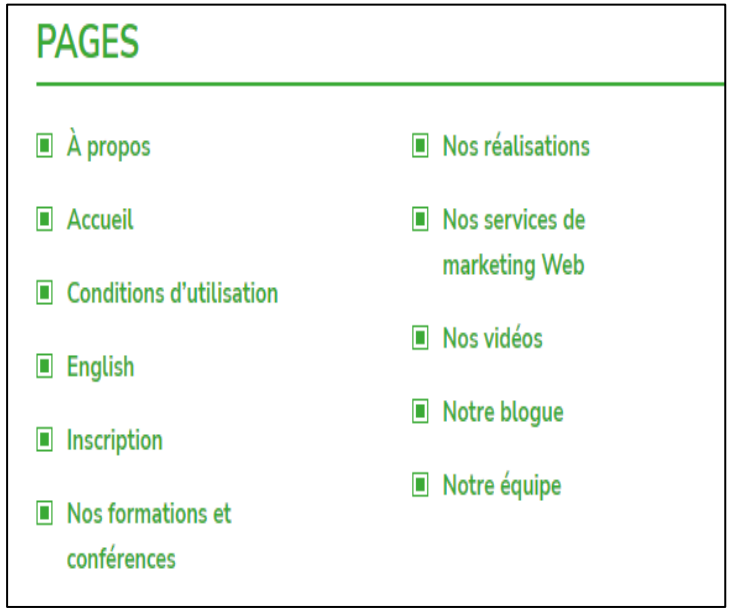


Figure 7 : Exemple d'un plan de site

PARTIE 1

PARTIE 1

Conception fonctionnelle – navigation

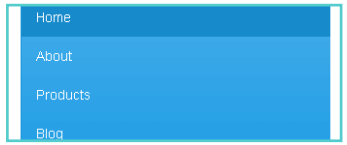
La navigation est un élément clé de tout site Web: c'est ainsi que l'utilisateur navigue d'une section à l'autre et de votre contenu. En plus de créer quelque chose d'unique, il existe plusieurs options de navigation dans la conception de site qui sont assez courantes :

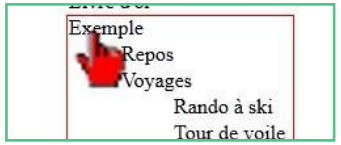
Texte horizontal

Une liste horizontale des sections du site (style le plus répandu en ligne)

Texte vertical

Également courante; souvent utilisée pour les sites nécessitant une liste plus longue d'éléments de la barre de boutons

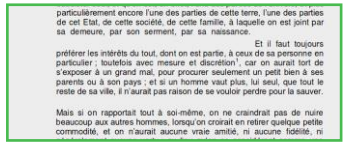


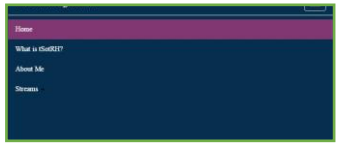


Sous-menus

Présenter à l'utilisateur la profondeur des informations dès le départ, même sans menu déroulant.

Texte avec des descriptions





Menus déroulants

Conception fonctionnelle – Spécifications fonctionnelles détaillées (SFD)

Elles sont élaborées par la maîtrise d'œuvre, c'est-à-dire le prestataire.

Elles précisent le comportement des fonctionnalités et les sous-fonctions du produit web, pour confirmer la prise en compte des besoins du client et obtenir la validation de ce dernier

Conception graphique

- **Objectif** : définir les éléments graphiques qui conduiront à construire l'identité visuelle du site.
- Les acteurs en charge des réalisations graphiques vont proposer à la maîtrise d'ouvrage(Client) des pistes graphiques qui aboutiront à l'élaboration de la charte graphique après validation.
- **Eléments graphiques :**

Définition du Gabarit des pages web (template, layout) qui sert de modèle pour la structure des pages d'un site

Conception des maquettes HTML : des maquettes fonctionnelles en noir et blanc(wireframe, ergolayout) et des maquettes graphiques intégrant l'identité visuelle et les couleurs

Conception technique

- La phase de conception du projet web est le plus souvent réalisée en étroite **collaboration entre le développeur, le webdesigner, l'intégrateur et l'hébergeur**.
- **La conception technique** consiste à faire des choix techniques et technologiques adaptés aux fonctionnalités du site web.

Figure 10: Architecture n-tiers d'un site web

- Elle peut contenir les aspects suivants

L'architecture logicielle (choix du langage de programmation, conception des bases de données, choix de l'architecture de l'application web : MVC, n-tiers, micro-services, échange d'applications externes, choix de frameworks, …)

L'architecture matérielle à mettre en place

Système de gestion de contenu (CMS) éventuel qui répondra le mieux à l'ensemble des besoins exprimés dans le cahier des charges.

L'architecture sécurisée (firewall, filtrage applicatif, logiciels de protection de site web, …)

Les spécifications techniques qui sont regroupées dans un document de référence (livrable).

PARTIE 1

CHAPITRE 1

Appréhender le cycle de vie d’un projet web

1. Cycle de vie d’un projet (Définition, étapes,…)
2. Analyse des besoins
3. Conception
4. **Développement**
5. Tests et déploiement
6. Maintenance et évolutivité

Phase de réalisation

- Cette phase repose sur les spécifications techniques livrées par la phase de conception. Elle est composée des étapes suivantes (menées en parallèle par l'équipe de projet):

Réalisation technique

Réalisation graphique

Réalisation éditoriale

Phase I : Réalisation technique

Elle comprend :

- **L'infrastructure de réalisation** (Mise en place de l'environnement de développement et frameworks, environnement serveurs et réseau)
- **Mise en place et alimentation des bases de données**. En cas de données telles que photos, publications sur réseaux sociaux, on se base sur le cloud.
- **Développement et codage**

Découpage en couches

Développement Front-End

- Mise en page,
- Navigation,
- Graphiques,
- Interaction avec l'utilisateur,
- Animation,
- Traitement des données(inputs)

Développement des scripts back-end

- Mis en réseau,
- Stockage des données,
- Gestion des utilisateurs.

Scripts trackage des visiteurs du site

- **Intégration de l'existant** : évolution d'un site web existant (reprise, amélioration, migration)

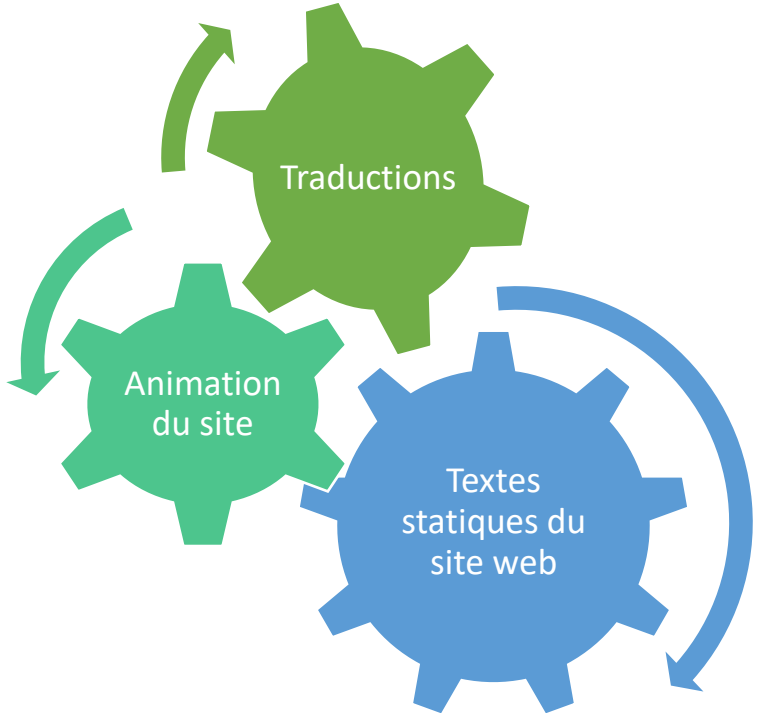
Phase II : Réalisation graphique

Elle comprend :

- Les éléments visuels (contenu visuel) :
  - Sélection d'images avec pertinence, simplicité et uniformité
  - Positionnement des éléments : bien placer les éléments et parvenir à une composition équilibrée
  - Choix d'une bonne palette de couleurs
  - Représentation visuelle des données statistiques par exemple
- Les bibliothèques d'images pour illustrer les contenus web, avec des résolutions adaptées à une navigation via réseau, mobile
- Bien choisir ses visuels : susciter l'action, mélanger texte et image, optimiser la taille des images
- Optimiser les visuels pour le référencement naturel, et ainsi une meilleure visibilité du contenu dans les moteurs de recherche (enrichissement textuel des contenus visuels : attribut, alt, titre, description, …)

Phase III : Réalisation éditoriale

Elle comprend :



CHAPITRE 1

Appréhender le cycle de vie d’un projet web

1. Cycle de vie d’un projet (Définition, étapes,…)
2. Analyse des besoins
3. Conception
4. Développement
5. **Tests et déploiement**
6. Maintenance et évolutivité

Tests

La phase de test du projet web favorise la détection de tous les bugs pour assurer la qualité de l'application. Il permet de rationaliser les coûts de développement du site web grâce à la maîtrise et à la correction en amont des défauts fonctionnels.

Elle garantit également l'acceptabilité du projet à la livraison lors de phase de recette chez le MOA.

Il existe **4 niveaux de tests** :

Tests unitaires / tests de composants	Tests d'intégration	Tests système	Tests de recette
Initiés par le développeur lui-même afin de vérifier son code au niveau du composant qu'il doit réaliser.	Exécutés par un testeur interne ou externe à la MOE, afin de s'assurer que plusieurs composantes du projet web interagissent conformément aux cahiers des charges et délivrent les résultats attendus	On exécute plusieurs scénarios complets qui constituent les cas d'utilisation du site web. Cela permet de s'assurer de la fonctionnalité globale du logiciel et de son comportement sur les terminaux d'utilisation	Réalisés par le client final afin de valider l'adéquation du logiciel aux spécifications du client exprimées dans le cahier de charges

Tests

Les tests peuvent être classés également en **famille de tests suivant leur nature**. Les deux familles principales :

- **Tests fonctionnels :**

Tests fonctionnels
On teste le comportement du site vis-à-vis des fonctionnalités souhaitées et attendues par le client

Tests unitaires	Tests d'intégration	Tests de système	Tests de recette

- **Tests non fonctionnels**. On peut citer :

Tests non fonctionnels

Tests de robustesse tester le comportement du site web dans des cas "extrêmes" (forte activité, disponibilité …)	**Tests de performance** tester par exemple : consommation CPU, exploitation de mémoire RAM, volume de commandes lancées à la seconde ou encore mouvement d'entrée et de sortie des utilisateurs sur le site	**Tests de montée en charge** tester sa capacité à supporter de plus en plus d'internautes tout en maintenant une expérience utilisateurs optimale et un fonctionnement correspondant aux cahier des charges	**Tests de compatibilité de plateforme** vérifier le bon fonctionnement du logiciel sur des terminaux ciblés notamment les systèmes d'exploitation après leur installation, les navigateurs clients	**Tests d'ergonomie** évaluer l'expérience utilisateur (UX) côté design, esthétique, visuel, etc.	**Tests d'interface graphique** s'assurer que la présentation graphique est suffisamment attrayante pour être accepté	**Tests de sécurité** définir les niveaux de sécurité et prévoir les tests de sécurité associés

PARTIE 1

Tests

Les tests peuvent être classés également en **famille de tests suivant leur nature**. Les deux familles principales :

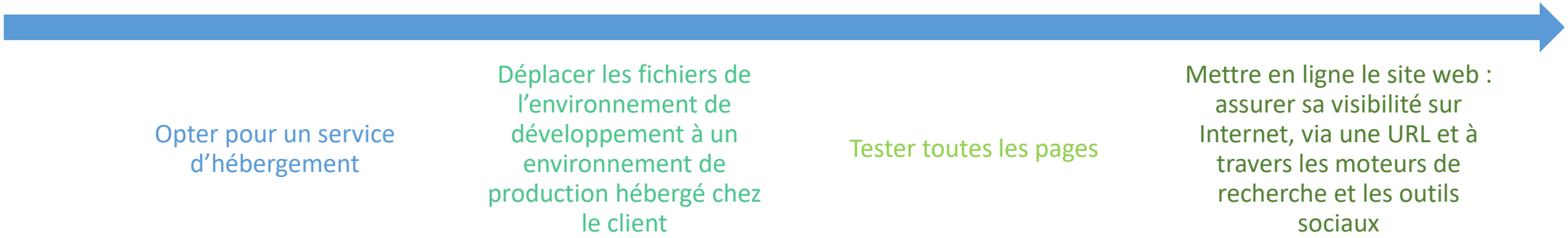
- **Tests non fonctionnels**. On peut citer :

Tests non fonctionnels

Tests de robustesse tester le comportement du site web dans des cas "extrêmes" (forte activité, disponibilité …)	**Tests de performance** tester par exemple : consommation CPU, exploitation de mémoire RAM, volume de commandes lancées à la seconde ou encore mouvement d'entrée et de sortie des utilisateurs sur le site	**Tests de montée en charge** tester sa capacité à supporter de plus en plus d'internautes tout en maintenant une expérience utilisateurs optimale et un fonctionnement correspondant aux cahier des charges	**Tests de compatibilité de plateforme** vérifier le bon fonctionnement du logiciel sur des terminaux ciblés notamment les systèmes d'exploitation après leur installation, les navigateurs clients	**Tests d'ergonomie** évaluer l'expérience utilisateur (UX) côté design, esthétique, visuel, etc.	**Tests d'interface graphique** s'assurer que la présentation graphique est suffisamment attrayante pour être accepté	**Tests de sécurité** définir les niveaux de sécurité et prévoir les tests de sécurité associés

Déploiement

- Déployer une application ou un site web signifie **appliquer un procédé permettant d'installer ou mettre à jour le site web** sur un environnement donné.
- **Le processus de déploiement** :



- La mise en ligne est l'aboutissement du projet web. Tout ce qui a été pensé et réalisé va se retrouver en ligne, soumis à l'appréciation d'une population internaute que l'on a ciblée et pour laquelle on a essayé d'anticiper les attentes
- Actuellement, il existe de nombreux outils pour automatiser le déploiement des sites web

CHAPITRE 1

Appréhender le cycle de vie d’un projet web

1. Cycle de vie d’un projet (Définition, étapes,…)
2. Analyse des besoins
3. Conception
4. Développement
5. Tests et déploiement
6. **Maintenance et évolutivité**

Les maintenances

- Un site internet a besoin d'un **entretien régulier** pour garantir tout son potentiel technique. Le web étant un environnement technologique où tout évolue très rapidement, il est d'autant plus important de faire de la maintenance **un processus continu** pour bénéficier d'un outil performant et générateur de trafic après plusieurs années.
- Un site mal entretenu et contenant beaucoup d'erreur projette une image peu soignée de l'entreprise, avec un impact négatif sur le référencement naturel : baisse du nombre de pages indexées par les moteurs de recherche, chute des positions dans les pages de résultats de recherche…

- La maintenance d'un site internet repose de manière générale sur deux types d'intervention :

Maintenance technique du site

- Corrections des bugs,
- Optimisation du code,
- Mises à jour des plugins, …

Maintenance du contenu

- Ajouts de nouveaux articles d'actualités ou de blog,
- Optimisation sémantique d'une page en vue d'améliorer son référencement naturel,
- Ajouts d'images ou de contenus multimédia, ...

PARTIE 1

Maintenance technique

- Appelée également maintenance **corrective** et **évolutive**, est applicable aux sites internet après leur livraison, et formalisée sous forme de contrat entre le propriétaire du site et le prestataire (le plus souvent une agence web).

- Ces contrats de maintenance permettent de bénéficier des actions suivantes :

Vérification et **mise à jour des technologies employées** pour le fonctionnement du site internet

- Mise à jour de plugins,
- Mise à jour de la base de données…

Détection et corrections des erreurs (bugs)

- Liées à la mise en forme et à l'affiche du site web

Maintenance du contenu du site

- Tout comme la maintenance technique, la mise à jour des contenus du site web est primordiale si on veut rester visible auprès des internautes/clients.

- L'actualisation des contenus est en effet importante pour optimiser le référencement naturel du site web. Plus on met à jour les pages du site et on ajoute des nouveaux articles, et plus les moteurs de recherche auront tendance à valoriser le site dans les pages des résultats de recherche.

CHAPITRE 2

Modéliser les besoins client par un diagramme de cas d'utilisation

Ce que vous allez apprendre dans ce chapitre :

- Introduction au langage de modélisation UML
- Définition du diagramme des cas d'utilisation
- Acteurs
- Cas d'utilisation
- Relation entre acteurs et cas d'utilisation
- Relations entre cas d'utilisation
- Relation de généralisation ou de spécialisation
- Description textuelle des cas d'utilisation

5 heures

CHAPITRE 2

Modéliser les besoins client par un diagramme de cas d'utilisation

1. **Introduction au langage de modélisation UML**
2. Définition du diagramme des cas d'utilisation
3. Acteurs
4. Cas d'utilisation
5. Relation entre acteurs et cas d'utilisation
6. Relations entre cas d'utilisation
7. Relation de généralisation ou de spécialisation
8. Description textuelle des cas d'utilisation

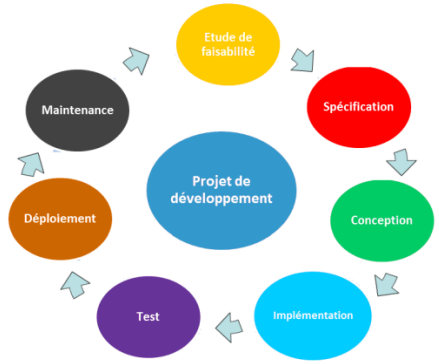
Modélisation

- Un **modèle** : **une représentation abstraite** d'un système destiné à en faciliter l'étude et à le documenter.
- C'est un **outil majeur de communication entre les différents intervenants** au sein d'un projet web.
- Chaque membre de l'équipe, depuis l'utilisateur jusqu'au développeur, utilise et enrichit le modèle différemment.
- C'est un outil pour maîtriser les systèmes devenant de plus en plus complexes.
- Il permet de **faciliter la traçabilité du système**, à savoir la possibilité de partir d'un de ses éléments et de **suivre ses interactions et liens avec d'autres parties du modèle**.

PARTIE 1

Modélisation

Il accompagne le processus de développement du projet web (cycle de vie)



* Le système est comme une **boîte noire** à part entière.

* Réaliser un modèle de niveau contexte

* Tracer précisément les **frontières fonctionnelles du système**

Activités de spécification des exigences

* **Le modèle** représente le système vu de l'intérieur

* Il se compose **d'objets** représentant une abstraction des concepts manipulés par les utilisateurs

* Le modèle comprend deux points de vue : **la structure statique** et le **comportement dynamique**

Activités d'analyse

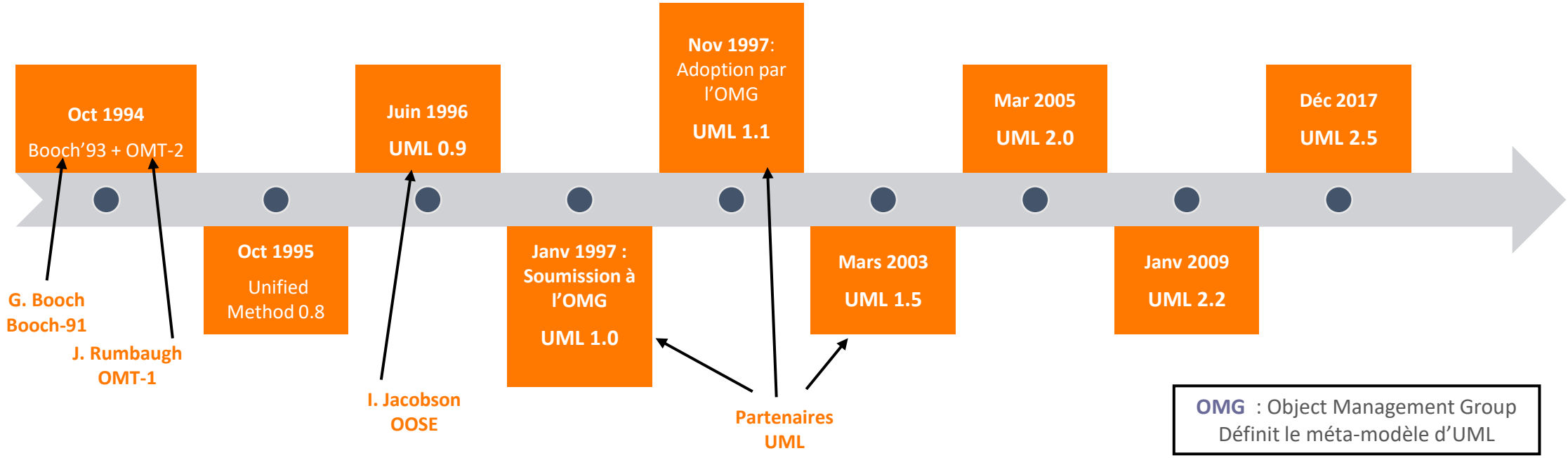
* Le modèle correspond aux concepts informatiques qui sont utilisés par les outils, les langages ou les plates-formes de développement

* Il sert ici à **étudier**, **documenter**, communiquer et **anticiper** une solution

Activités de conception

Modélisation avec UML

- Le modèle en tant qu'abstraction d'un système s'accorde parfaitement bien avec les concepts orientés objet.
- Cela explique le succès de la modélisation objet dans la programmation orienté objet,
- Le standard industriel de modélisation objet est **UML (Unified Modeling Language)**
- **UML** est une synthèse de langages de modélisation objet antérieurs : **Booch**, **OMT**, **OOSE**. Principalement issu des travaux de **Grady Booch**, **James Rumbaugh** et **Ivar Jacobson**



PARTIE 1

UML

- **UML** est un langage visuel dédié à la spécification, la construction et la documentation d'un système d'information
- **UML** comporte treize types de diagrammes représentant autant de vues distinctes pour représenter des concepts particuliers du logiciel, **réparties en 3 niveaux :**

Niveau Fonctionnel	Niveau Structurel	Niveau comportemental
• **Diagramme Use Case** (Cas d'utilisation)	• **Diagramme de classes** • Diagramme d'objets • Diagramme de composants • Diagramme de déploiement • Diagramme de paquetages • Diagramme de structures composites	• **Diagramme d'activités** • **Diagramme d'états-transitions** • Diagrammes d'interaction • **Diagramme de séquence** • Diagramme de communication • Diagramme global d'interaction • Diagramme de temps

- Dans ce chapitre, on va étudier les diagrammes mis **en gras foncé** dans le tableau

CHAPITRE 2

Modéliser les besoins client par un diagramme de cas d'utilisation

Diagramme Cas d'Utilisation (DCU)

- Le **diagramme de cas d'utilisation** (**DCU**) est utilisé dans l'activité de spécification des besoins. Il montre les interactions fonctionnelles entre **les acteurs** et **le système à l'étude**

- Le **diagramme des cas d'utilisation** permet de représenter la vision détaillée de l'application du point de vue de l'utilisateur

- **Il répond à la question : A qui et pour quoi faire?**

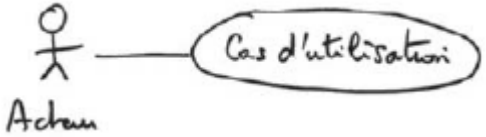
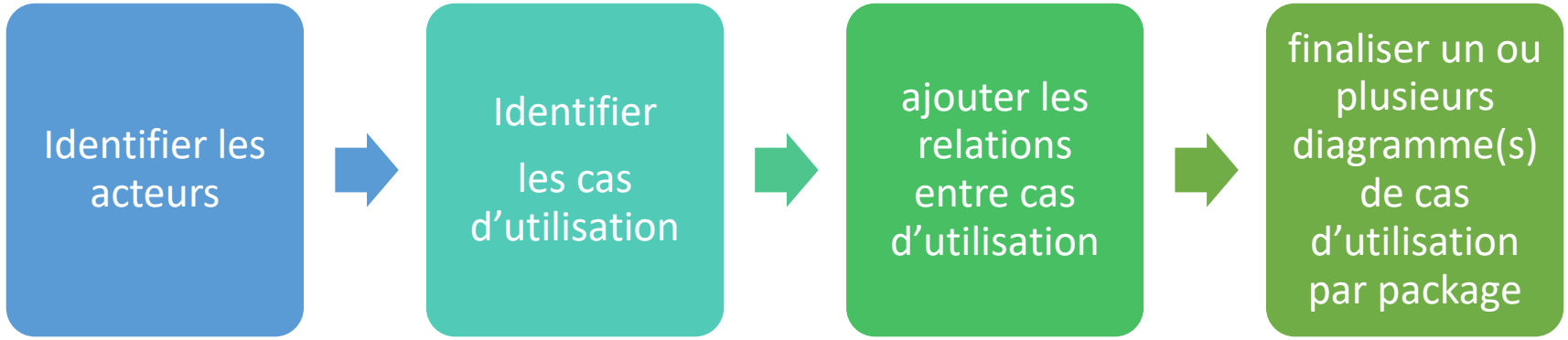


Figure 11 : Diagramme Cas d'utilisation

DCU - Démarche

- Voici les différentes étapes afin d'aboutir au modèle des cas d'utilisation



CHAPITRE 2

Modéliser les besoins client par un diagramme de cas d'utilisation

1. Introduction au langage de modélisation UML
2. Définition du diagramme des cas d'utilisation
3. **Acteurs**
4. Cas d'utilisation
5. Relation entre acteurs et cas d'utilisation
6. Relations entre cas d'utilisation
7. Relation de généralisation ou de spécialisation
8. Description textuelle des cas d'utilisation

Acteurs - Définition

- Un acteur représente **un rôle joué par une entité externe** qui interagit directement avec le système étudié.
- Il peut être **un utilisateur humain, un dispositif matériel ou un autre système.**
- L'acteur est celui qui bénéficie de l'utilisation du système.

Exemples d'interaction avec le système :

- **Consulter** et/ou **modifier** directement l'état du système.
- **Emettre** et/ou en **recevoir** des messages susceptibles d'être porteurs de données

Remarques

- **Ne pas confondre rôle et personne physique**. Une même personne peut jouer successivement différents rôles par rapport au système étudié, et être modélisée par plusieurs acteurs.
- un même rôle peut être tenu simultanément par plusieurs personnes, qui seront alors modélisées par le même acteur.
- **Exemple :** Une seule personne physique peut jouer successivement les rôles de manager et Webmaster vis à vis du site web, il s'agit bien de deux acteurs distincts, de deux profils différents.

Acteur principal vs Acteur secondaire

- Tous les acteurs n'utilisent pas forcément le système.
- **Un acteur principal** est celui pour qui le cas d'utilisation produit un résultat observable.
- **Un acteur secondaire** est un participant au cas d'utilisation. Il est souvent sollicité pour des informations complémentaires; il peut uniquement consulter ou informer le système lors de l'exécution du cas d'utilisation.

Bonne pratique

- Faire figurer **les acteurs principaux à gauche** des cas d'utilisation et **les acteurs secondaires à droite**.
- Utiliser **la forme graphique du stick man** pour les acteurs humains
- Utiliser **une représentation rectangulaire avec le mot-clé <<actor>>** pour les systèmes connectés

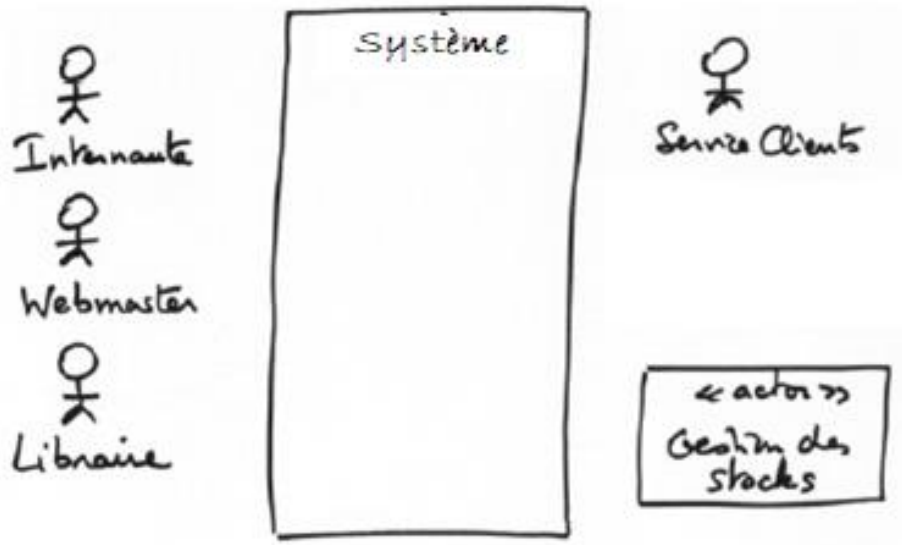


Figure 12 : Exemple Acteurs d'un site Librairie en Ligne

PARTIE 1

CHAPITRE 2

Modéliser les besoins client par un diagramme de cas d'utilisation

Définition

- **Un cas d'utilisation (use case)** représente **un ensemble de séquences d'actions qui sont réalisées par le système** et qui produisent un résultat observable intéressant pour un acteur particulier.
- **Un cas d'utilisation** modélise **un service rendu** par le système. Il exprime les interactions acteurs/système et apporte une valeur ajoutée « **notable** » à l'acteur concerné.
- Pour chaque acteur identifié précédemment, on doit **rechercher les différentes intentions (objectif métier)** d'utilisation du système
- **Un cas d'utilisation** peut faire participer plusieurs acteurs et qu'un acteur peut participer à plusieurs cas d'utilisation

Remarques

Cas d'utilisation = ensemble d'actions concrétisant une intention de l'acteur

- **Erreur fréquente A éviter** : Le cas d'utilisation se réduit systématiquement à une seule action
- **Bonne pratique :** Nommer les cas d'utilisation par un verbe à l'infinitif suivi d'un complément, du point de vue de l'acteur

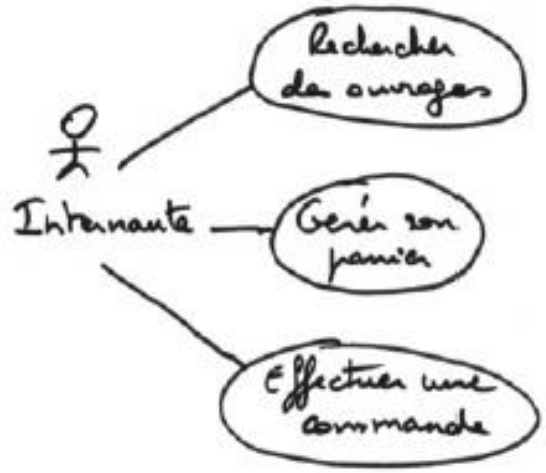


Figure 13 : Cas d'utilisation d'un internaute dans un site Librairie en Ligne

PARTIE 1

Comment découvrir les cas d'utilisation?

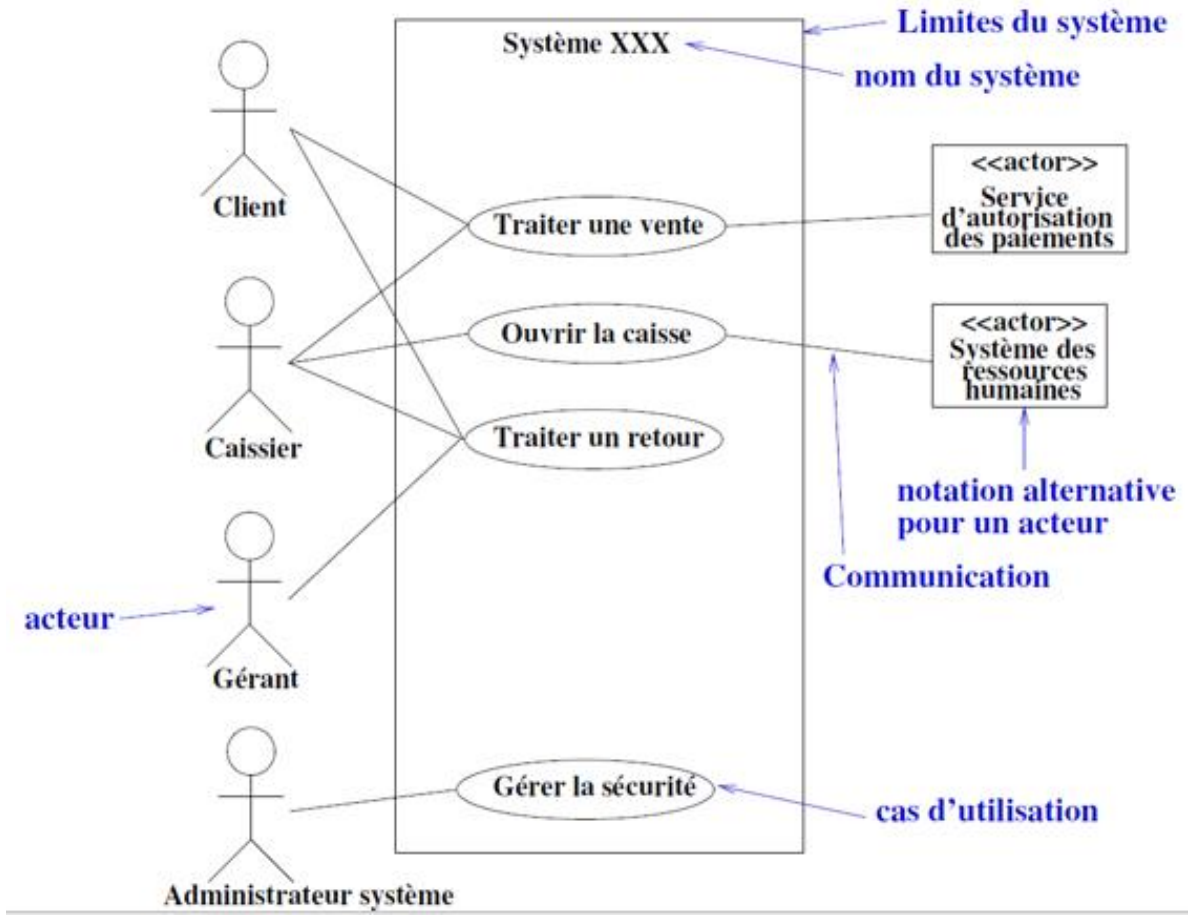
Système
- Délimiter le périmètre du système

Acteurs secondaires
- Qui utilisent le système
- Qui fournissent un service au système

Acteurs principaux
- Qui utilisent le système pour atteindre un but

Cas d'utilisation
- Définir les cas d'utilisation correspondants à ces buts

Diagramme des Cas d'utilisation - Exemple



PARTIE 1

CHAPITRE 2

Modéliser les besoins client par un diagramme de cas d'utilisation

1. Introduction au langage de modélisation UML
2. Définition du diagramme des cas d'utilisation
3. Acteurs
4. Cas d'utilisation
5. **Relation entre acteurs et cas d'utilisation**
6. Relations entre cas d'utilisation
7. Relation de généralisation ou de spécialisation
8. Description textuelle des cas d'utilisation

PARTIE 1

Relation d'association

- **Une association est une relation** entre éléments UML qui décrit un ensemble de liens.
- Elle est utilisée dans le cadre du diagramme de cas d'utilisation pour relier les acteurs et les cas d'utilisation par une relation qui signifie simplement : « **participe à** ».
- Une association peut être **sans flèches** ou **à flèches (sens du ou vers le système)**

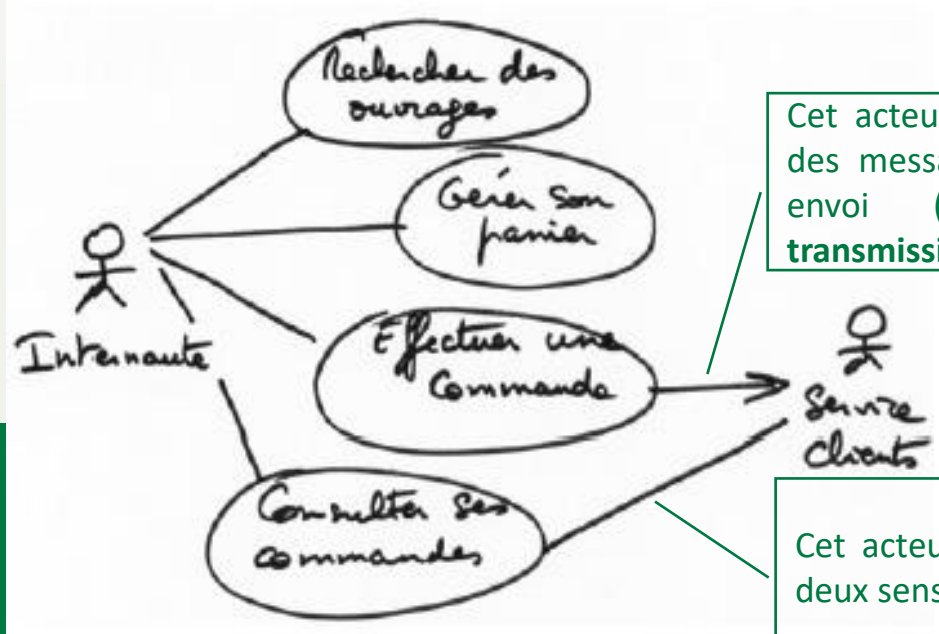


Figure 14 : Cas d'utilisation d'un internaute

Cet acteur ne fait que **recevoir** des messages du système sans envoi **(Sens unique de transmission d'information)**

Cet acteur va interagir dans les deux sens avec le système

Les acteurs non humains ne font **qu'envoyer** des messages au système, sans recevoir

Figure 15 : Cas d'utilisation d'un employé

Multiplicité

- Lorsqu'un acteur peut interagir plusieurs fois avec un cas d'utilisation, il est possible d'ajouter une multiplicité sur l'association du côté du cas d'utilisation :
  - **plusieurs :** avec le symbole *,
  - **exactement n :** s'écrit tout simplement ***n***,
  - **entre n et m :** s'écrit ***n..m***

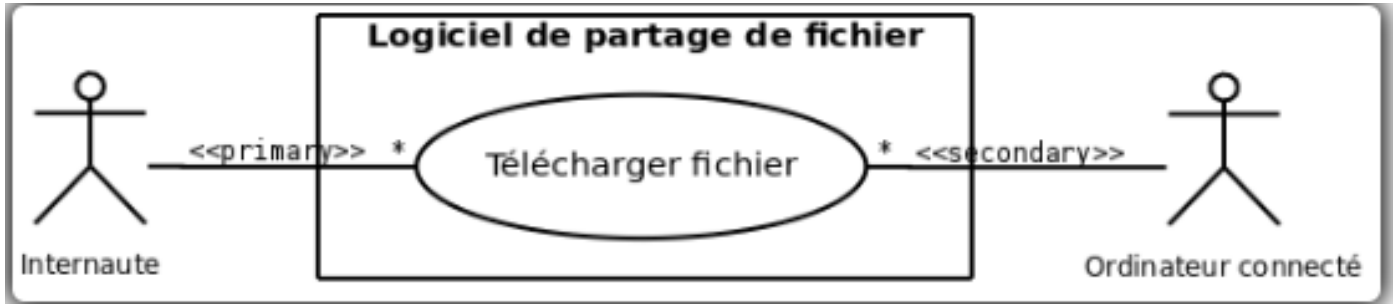


Figure 16 : DCU d'un logiciel de partage de fichiers

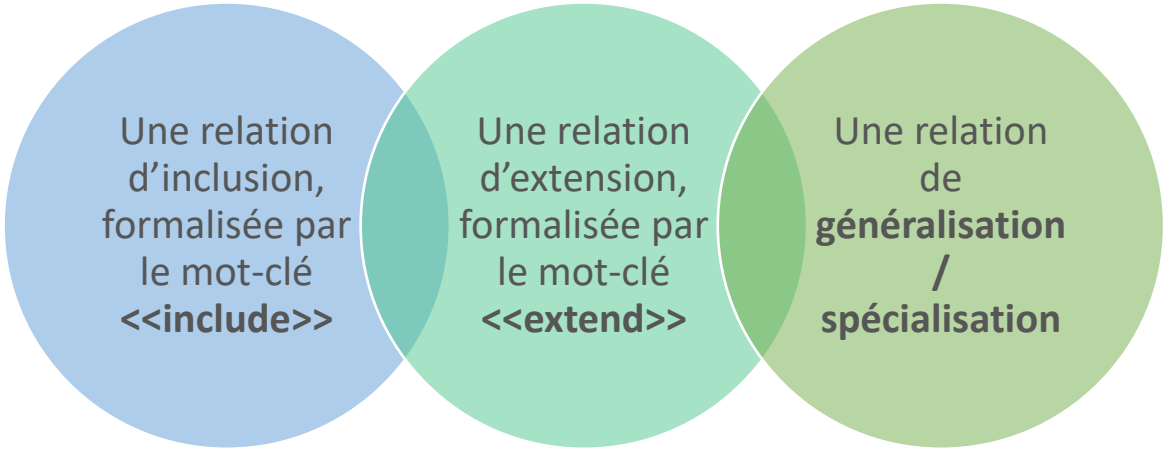
CHAPITRE 2

Modéliser les besoins client par un diagramme de cas d'utilisation

1. Introduction au langage de modélisation UML
2. Définition du diagramme des cas d'utilisation
3. Acteurs
4. Cas d'utilisation
5. Relation entre acteurs et cas d'utilisation
6. **Relations entre cas d'utilisation**
7. Relation de généralisation ou de spécialisation
8. Description textuelle des cas d'utilisation

Relation entre cas d'utilisation

- Pour affiner le diagramme de cas d'utilisation, UML définit trois types de relations standardisées entre cas d'utilisation :



Relation d'inclusion `- - - - - «include» - - - - - >`

- Notée ***<<include>>*** (on trouve aussi ***<<uses>>),***
- Le cas d'utilisation de base **incorpore** explicitement un autre, **de façon obligatoire,**
- Une relation ***include*** montre une fonctionnalité commune à plusieurs cas d'utilisation.

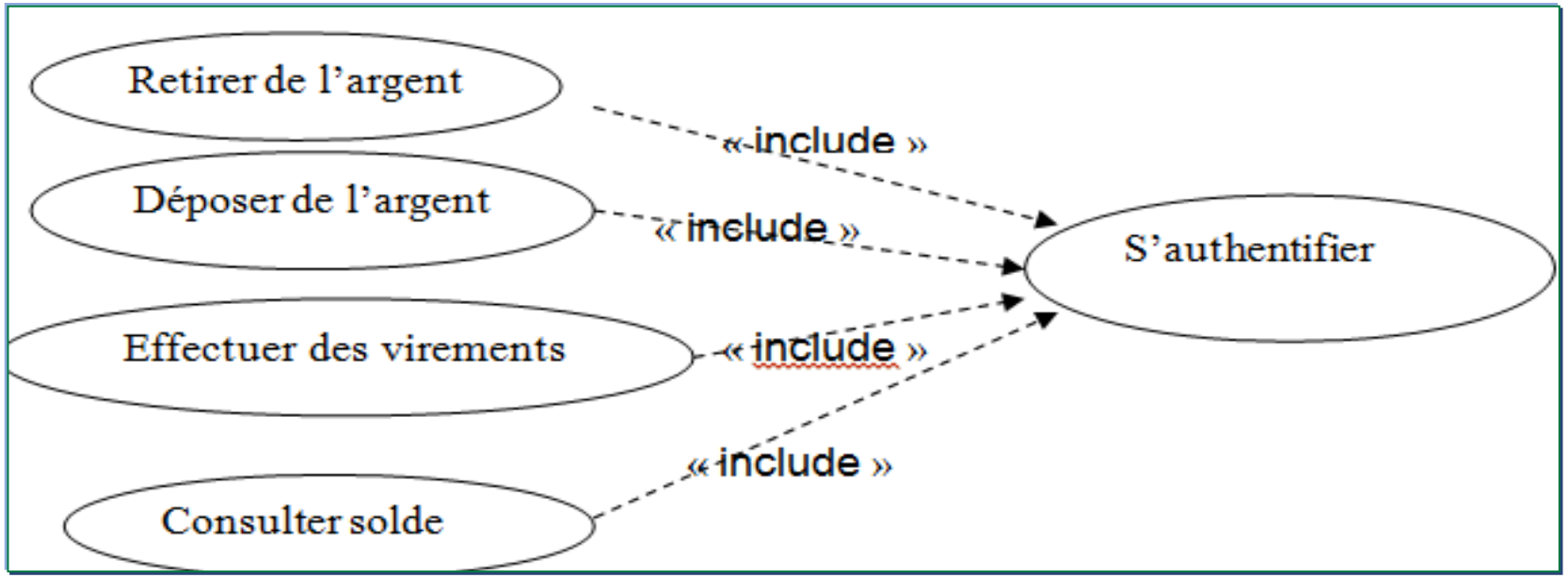
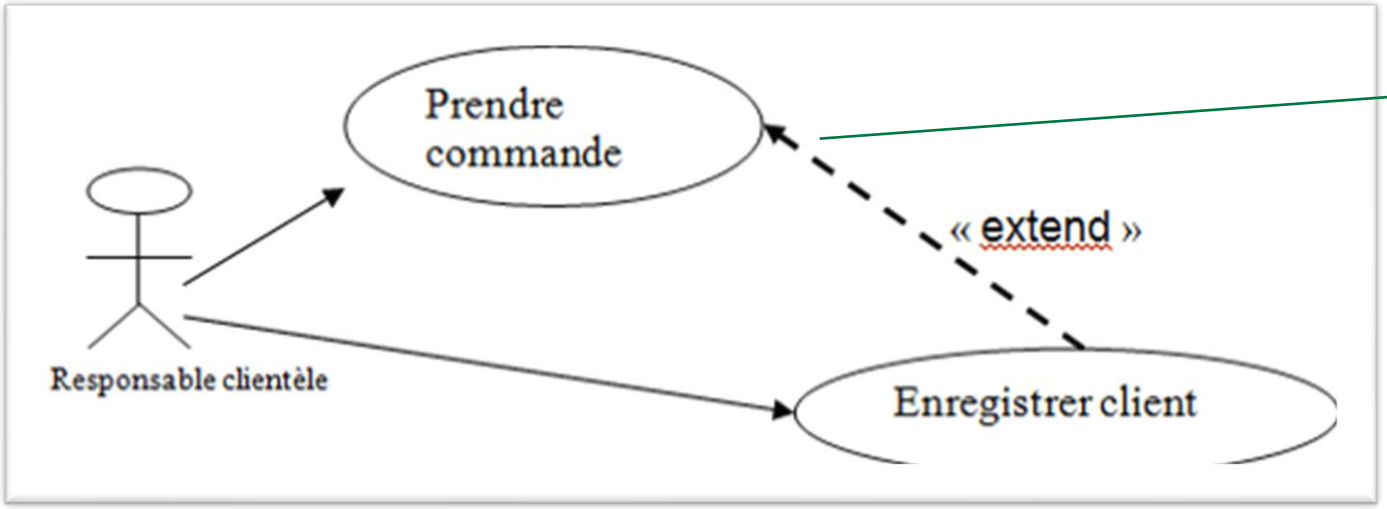


Figure 17 : Exemple DCU avec relation <<include>>

Relation d'extension `- - - - - «extend» - - - - - >`

- Notée ***<<extends>> ,***
- Le cas de base peut fonctionner tout seul, mais **il peut également être complété par un autre** (**optionnel** avec une condition préciser)..



Le cas d'utilisation X **(Enregistrer Client**) étend le cas **Y (Prendre Commande**) lorsque X peut être appelé au cours de l'exécution du cas d'utilisation Y

Figure 18 : Exemple DCU avec relation <<extend>>

Point d'extension

- L'extension peut intervenir à un point précis du cas étendu (cas de base). Ce point s'appelle **le point d'extension**.
- Il porte un nom, qui figure dans un compartiment du cas étendu sous la rubrique *points d'extension (extension points en Anglais)*, et est éventuellement associé à une contrainte *{…}* indiquant le moment où l'extension intervient.
- **Une extension est souvent soumise à condition**. Graphiquement, la condition est exprimée sous la forme d'une note.
- **Exemple** : Dans un système e-banking, la vérification du solde du compte n'intervient que si la demande du montant dépasse 200 Dhs.

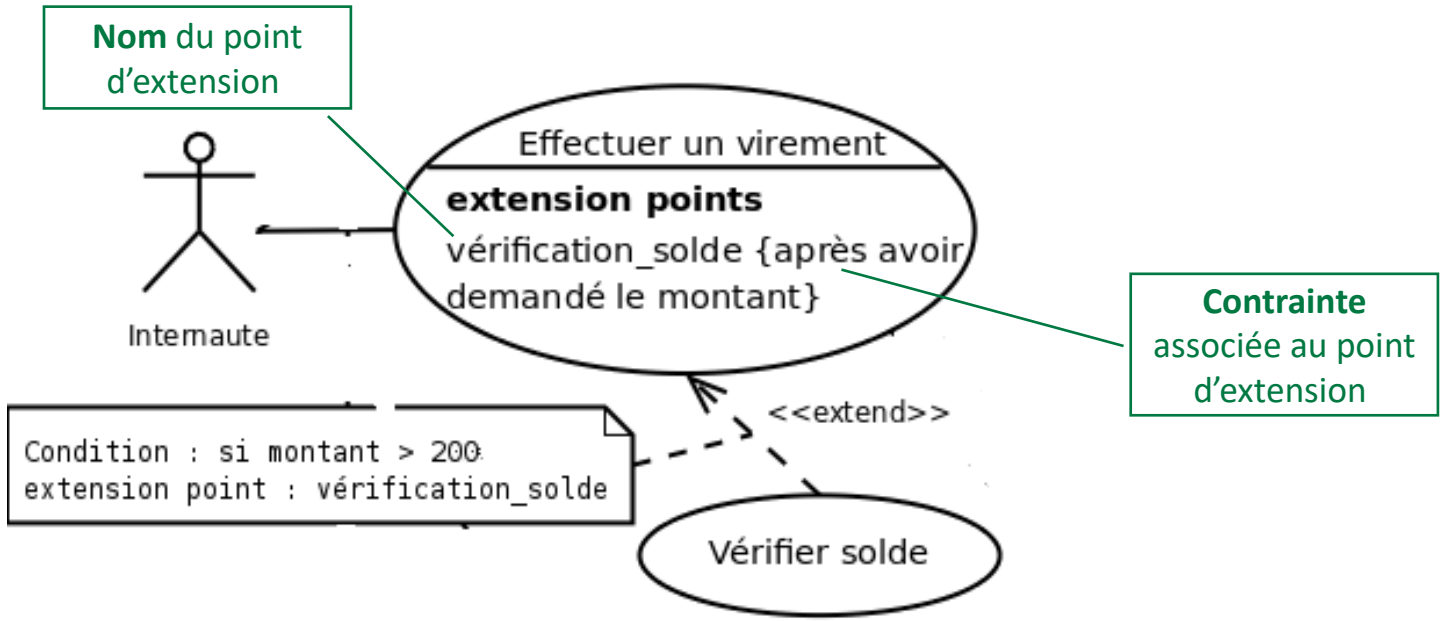


Figure 19 : Exemple Point d'extension sur un cas d'utilisation

CHAPITRE 2

Modéliser les besoins client par un diagramme de cas d'utilisation

PARTIE 1

Relation de généralisation/spécialisation entre acteurs

- **Un acteur A est une généralisation d'un acteur B si l'acteur A peut être remplacé par l'acteur B.** Dans ce cas, tous les cas d'utilisation accessibles à A le sont aussi à B, mais l'inverse n'est pas vrai.

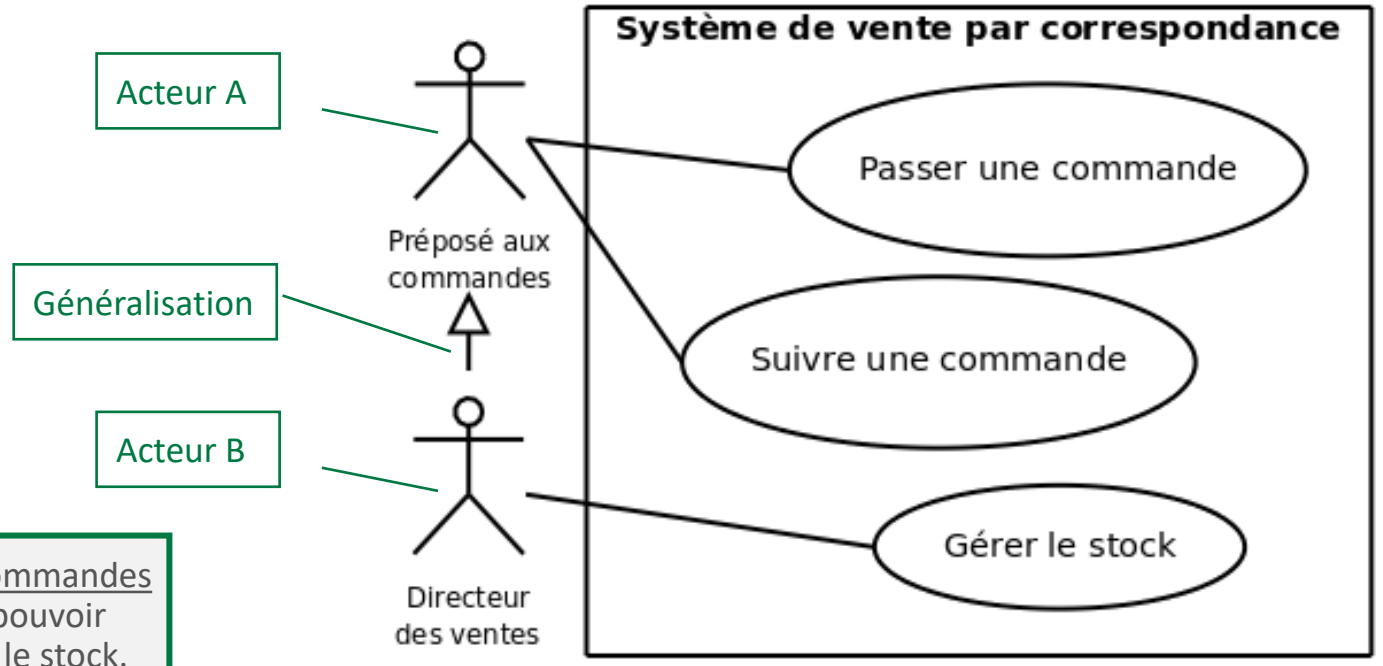


Figure 20 : Relation de généralisation entre acteurs concrets

Exemple

Le directeur des ventes **est un** préposé aux commandes avec un pouvoir supplémentaire : en plus de pouvoir passer et suivre une commande, il peut gérer le stock. Par contre, le préposé aux commandes ne peut pas gérer le stock.

Relation de généralisation/spécialisation entre acteurs

- Dans certaines situations, il arrive que deux acteurs, ou plus, présentent des similitudes dans leurs relations aux cas d'utilisation.
- On peut l'exprimer en **créant un acteur généralisé, éventuellement abstrait,** qui modélise les aspects communs aux différents acteurs concrets. Cette relation se traduit par le concept d'héritage dans les langages orientés objet.

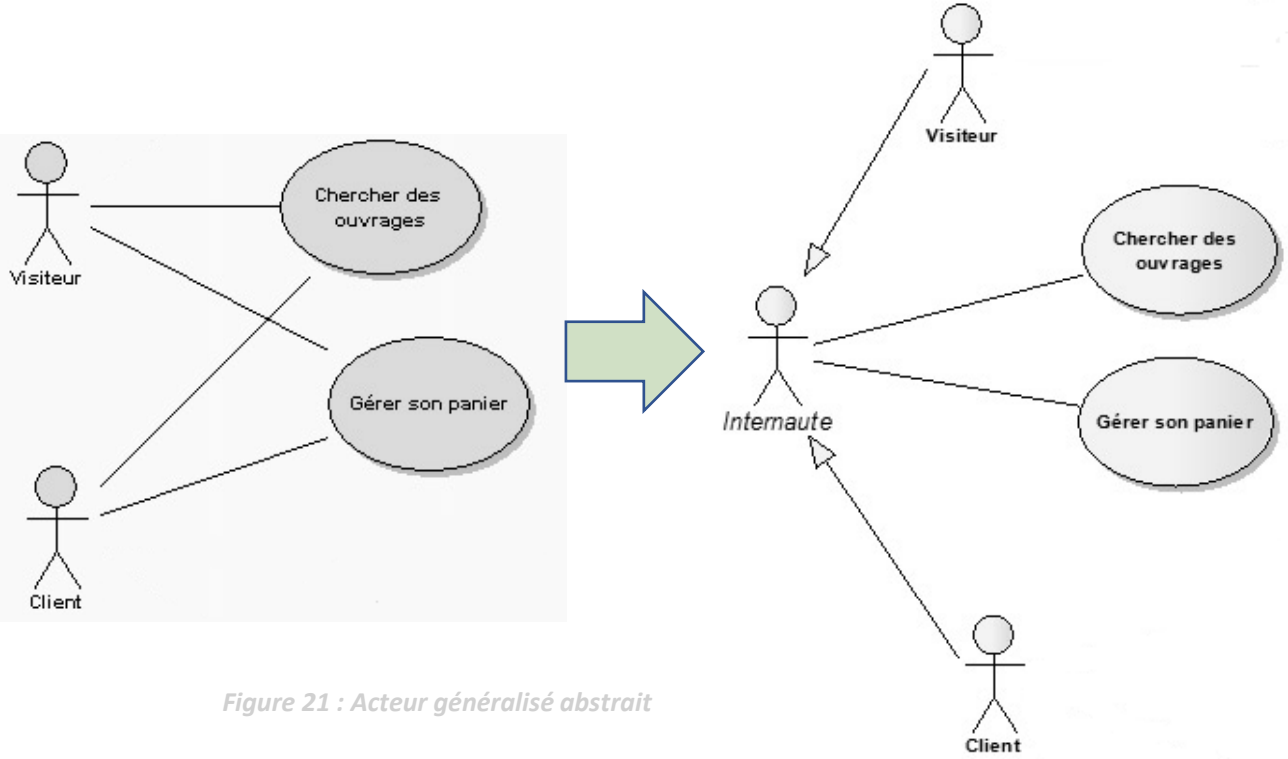


Figure 21 : Acteur généralisé abstrait

Bonne Pratique

- Les deux acteurs principaux Client et Visiteur partagent deux cas d'utilisation ***Chercher des ouvrages*** et ***Gérer son panier***
- Dans l'exemple, l'acteur **Internaute** est la **généralisation abstraite** des rôles Visiteur et Client.

PARTIE 1

Relation de généralisation/spécialisation entre cas d'utilisation

- Un cas d'utilisation A est une généralisation d'un cas d'utilisation B **si B est un cas particulier de A**.
- **Exemple** : « ***Faire un virement par Internet*** » est un cas particulier de « Faire un virement ».

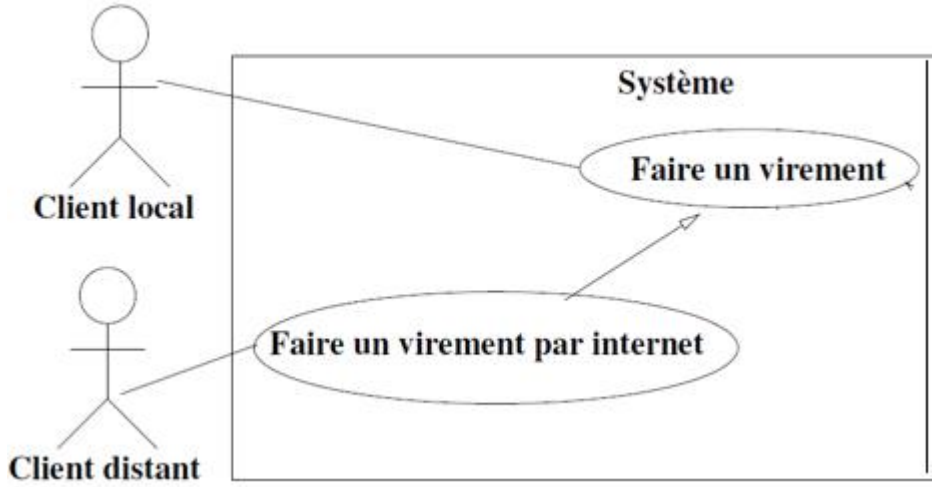


Figure 22 : Généralisation entre cas d'utilisation

PARTIE 1

PARTIE 1

Exemple complet avec tous les types de relation

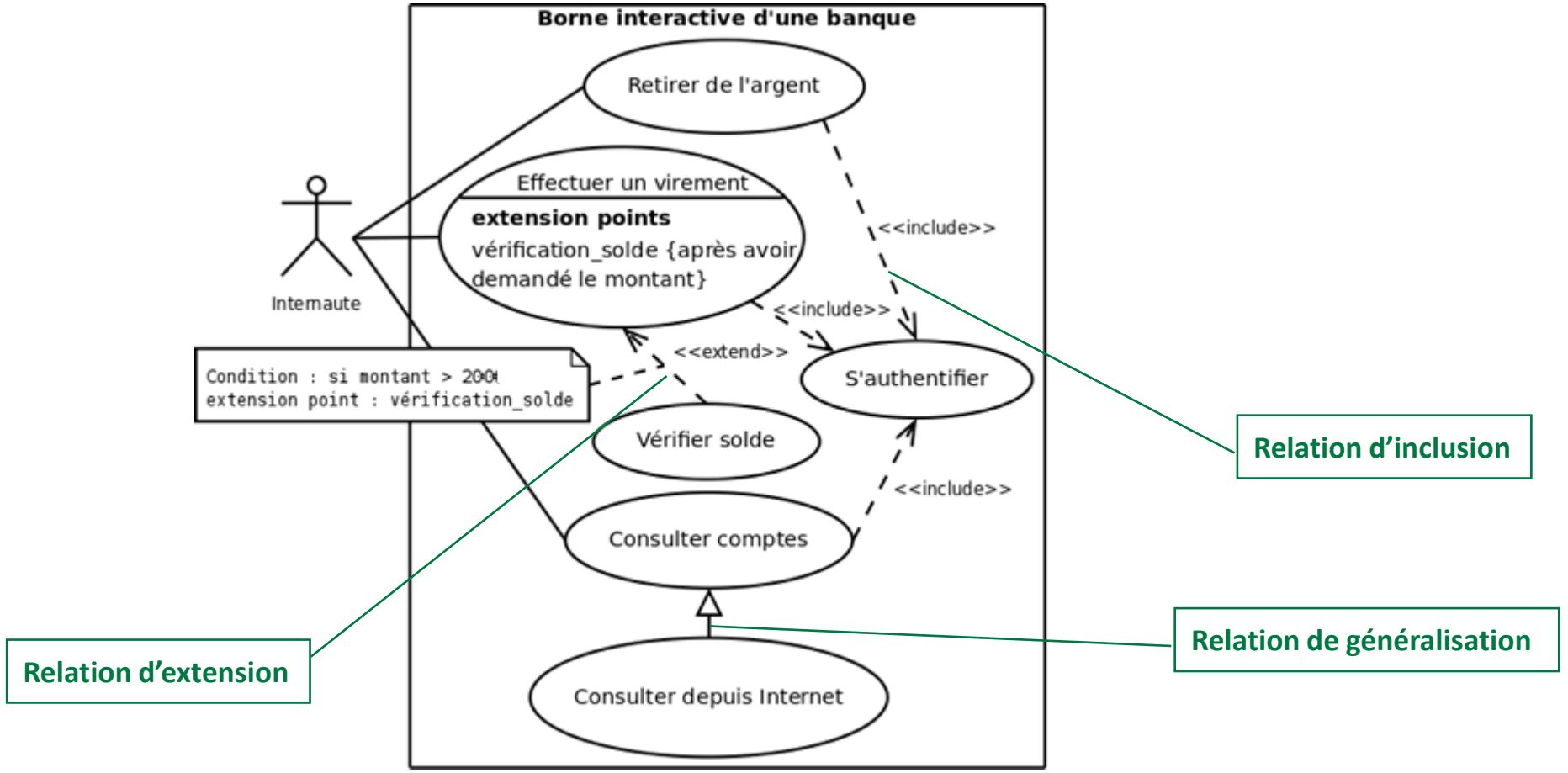


Figure 23 : DCU d'un système de borne en ligne d'une banque (e-banking)

CHAPITRE 2

Modéliser les besoins client par un diagramme de cas d'utilisation

1. Introduction au langage de modélisation UML
2. Définition du diagramme des cas d'utilisation
3. Acteurs
4. Cas d'utilisation
5. Relation entre acteurs et cas d'utilisation
6. Relations entre cas d'utilisation
7. Relation de généralisation ou de spécialisation
8. **Description textuelle des cas d'utilisation**

Documentation des cas d'utilisation

- Le diagramme de cas d'utilisation décrit les grandes fonctions d'un système du point de vue des acteurs, mais n'expose pas de façon détaillée le dialogue entre les acteurs et les cas d'utilisation. **Il est recommandé de rédiger une description textuelle de chaque cas d'utilisation**
- **Une description textuelle du cas d'utilisation se compose de trois parties :**

Identification du cas d'utilisation

Identifier un cas d'utilisation par les informations suivantes :

- **Nom :** utiliser un verbe à l'infinitif (ex. : Consulter le solde de compte)
- **Objectif :** une description résumée permettant de comprendre l'intention principale du cas d'utilisation.
- **Acteurs principaux :** ceux qui vont réaliser le cas d'utilisation.
- **Acteurs secondaires :** ceux qui ne font que recevoir des informations à l'issue de la réalisation du cas d'utilisation.
- **Dates :** les dates de création et de mise à jour de la description courante.
- **Responsable :** le nom des responsables.
- **Version :** le numéro de version.

Description des scénarios

Décrire un cas d'utilisation :

- **Pré-conditions :** définissent ce qui doit être vrai en amont du cas d'utilisation pour que celui-ci puisse démarrer. Elles ne sont pas testées à l'intérieur du cas d'utilisation, mais sont tenues pour acquises.

- **Des scénarii :** ces scénarii sont décrits sous forme d'échanges de messages entre l'acteur et le système. On distingue **le scénario nominal**, qui se déroule quand il n'y a pas d'erreur, **des scénarii alternatifs** qui sont les variantes du scénario nominal et enfin **les scénarii d'exception** qui décrivent les cas d'erreurs.

- **Post-conditions :** définissent ce qui doit être vrai lorsque le cas d'utilisation se termine avec succès, soit pour le scénario nominal ou pour le scénario alternatif

Exigences non fonctionnelles

- C'est une rubrique optionnelle.
- Elle se rapporte spécifiquement à un cas d'utilisation plutôt qu'au système dans sa totalité
- Elle contient généralement des spécifications non fonctionnelles (spécifications techniques,…).
- Typiquement, pour un site web, il s'agira de **performance, de sécurité ou d'ergonomie**.
- On complètera par exemple la description des scénarios par des copies d'écran de la maquette

PARTIE 1

Exemple : documenter le cas d'utilisation « S'authentifier »

- **Cas d'utilisation** : S'authentifier
- **Acteurs** : Administrateur et autres utilisateurs.
- **Objectif** : Il permet à l'acteur de s'identifier en saisissant son login et mot de passe.
- **Pré-condition** : La connexion avec le système est opérationnelle.
- **Post-condition** :
  - Acteur authentifié.
  - La page d'accueil s'affiche.
- **Scénario nominal** :
  1. L'acteur ouvre l'application,
  2. Le système affiche la page d'authentification,
  3. L'acteur saisit le login et le mot de passe,
  4. Le système vérifie l'existence des données,
  5. Le système affiche la page d'accueil.
- **Scénario alternatif** :
  - **4.a.** Erreur d'authentification : login ou mot de passe non valide.
    5. Le système affiche un message d'erreur.
    Le scénario reprend au point 2.
  - **4.b.** Champs obligatoires vides.
    Le scénario reprend au point 2.
- **Exigences non fonctionnelles** : Au bout de 3 tentatives échouées d'authentification, le système affiche un message de verrouillage d'accès à durée limitée.

CHAPITRE 3

Modéliser les données du projet par un diagramme de classes

Ce que vous allez apprendre dans ce chapitre :

- Définition du diagramme de classes
- Visibilité, encapsulation des méthodes et attributs
- Attributs et Méthodes d'instance
- Attributs et méthodes de classe
- Contraintes
- Attributs calculés (dérivé)
- Multiplicité (cardinalité)
- Constructeur et destructeurs
- Modèles de classe
- Relations entre classes
- Classes concrètes et abstraites
- Relation de réalisation et notion d'interfaces

10 heures

CHAPITRE 3

Modéliser les données du projet par un diagramme de classes

PARTIE 1

Diagramme de classes

- La modélisation par les cas d'utilisation (comme expliqué au chapitre 2) donne des descriptions fonctionnelles du système futur du point de vue des acteurs, avec une description dynamique des scénarios d'exécution.
- **La conception objet** demande principalement une description **structurelle**, **statique**, du système à réaliser sous forme d'un ensemble de classes logicielles, éventuellement regroupées en packages.
- **Le diagramme de classes** est considéré comme le plus important de la modélisation orientée objet, il est le seul obligatoire lors d'une telle modélisation.
- Il permet de fournir une **représentation abstraite des objets du système qui vont interagir pour réaliser les cas d'utilisation**. Il est important de noter qu'un même objet peut très bien intervenir dans la réalisation de plusieurs cas d'utilisation.
- C'est **une vue statique**, car on ne tient pas compte du facteur temporel dans le comportement du système.
- Le diagramme de classes permet de **modéliser les classes du système et leurs relations** indépendamment d'un langage de programmation particulier.

PARTIE 1

Modéliser la structure interne du système

Diagramme de classes d'Analyse (DCA)

- représenter les **classes métiers (modèle)** issues du Cahier de Charges
- **Exemple d'un système de gestion de commandes** : classes **Client**, **Commande**, **Produit**

Niveau Analyse

Diagramme de classes de Conception (DCC)

- Compléter par les **classes de Vue, de Contrôle, etc.**
- Exemple : classes pour les interfaces graphiques de l'application, classe Contrôleur pour contrôler la saisie des données, la sécurité, etc.

Niveau Conception

Structuration orienté objet – Qu'est-ce qu'un objet?

Objet = Etat + Comportement + Identité

Etat d'un objet

- Ensemble de valeurs décrivant l'objet
- Chaque valeur est associée à un attribut (propriété)
- Les valeurs sont également des objets (→liens entre objets)

Comportement d'un objet

- Ensemble d'opérations que l'objet peut effectuer
- Chaque opération est déclenchée par l'envoi d'un message (Appel d'une méthode)

Identité d'un objet

- Permet de distinguer les objets indépendamment de leur état
- Attribuée automatiquement à la création de l'objet
- L'identité d'un objet ne peut être modifiée
- Identité <> nom de la variable qui référence l'objet

Structuration orienté objet – Qu'est-ce qu'une classe d'objet?

- Une classe est une **représentation abstraite d'un d'ensemble d'objets**
- Elle contient **les informations nécessaires à la construction de l'objet** (c'est-à-dire **la définition des attributs et des méthodes**).
- La classe peut donc être considérée comme le **modèle**, le moule ou la notice qui va permettre la construction d'un objet. On peut parler de **type** (comme pour une donnée).
- On dit également qu'un **objet est l'instance d'une classe** (la concrétisation d'une classe).

Classe avec instance

Classe concrète = regroupement d'objets (instances) possédant les mêmes caractéristiques

Toutes les instances d'une classe ont les mêmes attributs et opérations

Classes sans instance

Classes abstraites = certaines opérations peuvent être non définies

Interfaces : pas d'attribut et toutes les opérations sont non définies

PARTIE 1

Représentation graphique d'une classe avec UML

- Une classe est représentée par un rectangle (**appelé aussi classeur**) divisé en 3 compartiments :
  - Le 1er compartiment contient le **nom de la classe** qui représente le type d'objet instancié (1ère lettre en majuscules). **Exemple** : *Voiture*
  - Le 2ème compartiment contient les attributs : ce sont **les données encapsulées dans les objets** de cette classe. **Exemple** : *vitesse courante*, *numéro d'immatriculation, etc.*
  - Le 3ème compartiment contient les opérations (méthodes) : les fonctionnalités/services assurées par les objets de la classe.

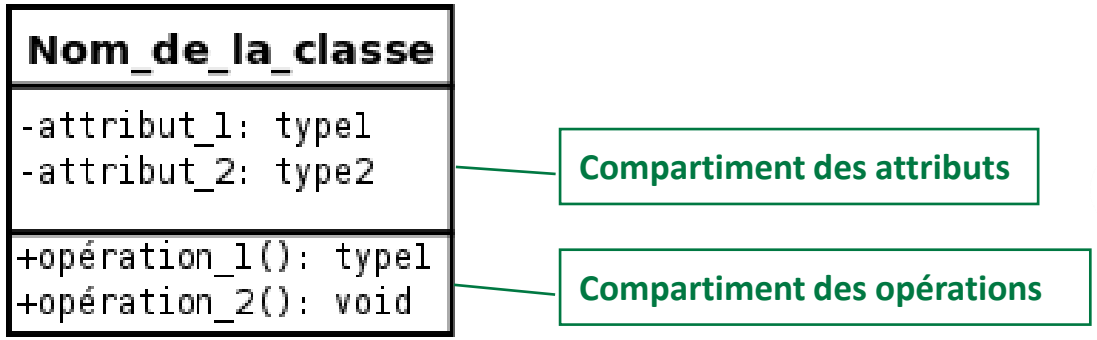


Figure 24 : Représentation UML d'une classe

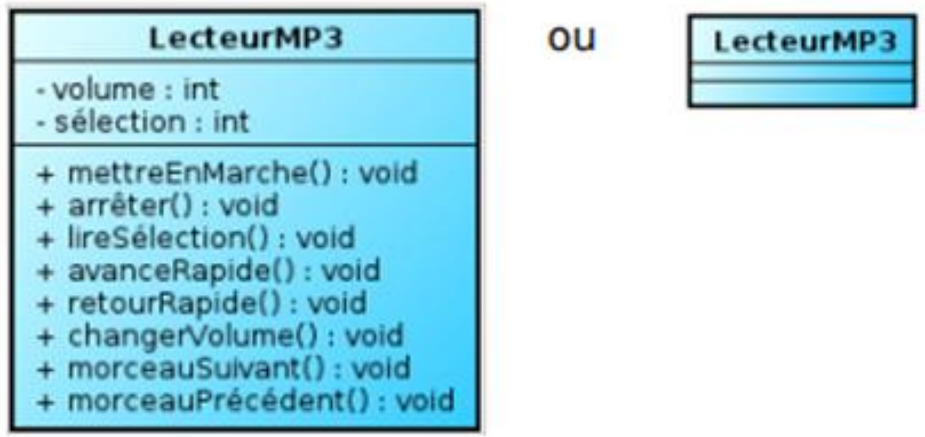
Remarque

- Si la modélisation ne s'intéresse qu'aux relations entre les différentes classes du système (et pas au contenu des classes), **on peut laisser le 2ème et 3ème compartiment vide.**

PARTIE 1

Représentation graphique d'une classe avec UML

- **Exemple** : Lecteur MP3



PARTIE 1

CHAPITRE 3

Modéliser les données du projet par un diagramme de classes

Encapsulation

- L'encapsulation est un mécanisme consistant à **rassembler les données et les méthodes au sein d'une structure (vue interne de l'objet)** en **empêchant l'accès aux données par un autre moyen que les services proposés**.
- Ces services accessibles aux utilisateurs de l'objet définissent ce que l'on appelle l'interface de l'objet (**vue externe de l'objet**).
- L'encapsulation permet de **garantir l'intégrité des données** contenues dans l'objet.

Visibilité

- L'encapsulation permet **de définir des niveaux de visibilité** des éléments d'un conteneur.
- La **visibilité** déclare la possibilité pour un élément de modélisation de référencer un élément qui se trouve dans un espace de noms différent de celui de l'élément qui établit la référence.
- Elle fait partie de la relation entre un élément et le conteneur qui l'héberge
- Le conteneur peut être :
  - un **paquetage**,
  - une **classe**
  - ou **un autre espace de noms**.
- Il existe quatre visibilités prédéfinies :

public +	• élément non encapsulé visible par tous.
protected #	• élément encapsulé visible seulement dans la classe.
private -	• élément encapsulé visible dans la classe et dans les sous-classes .
package ~	• élément encapsulé visible dans les classes du même paquetage.

PARTIE 1

Visibilité des attributs

- Dans la pratique, lorsque des attributs doivent être accessibles de l'extérieur, il est préférable que cet accès ne soit pas direct, mais se fasse par l'intermédiaire d'opérations (**Accesseurs getter/setter**)

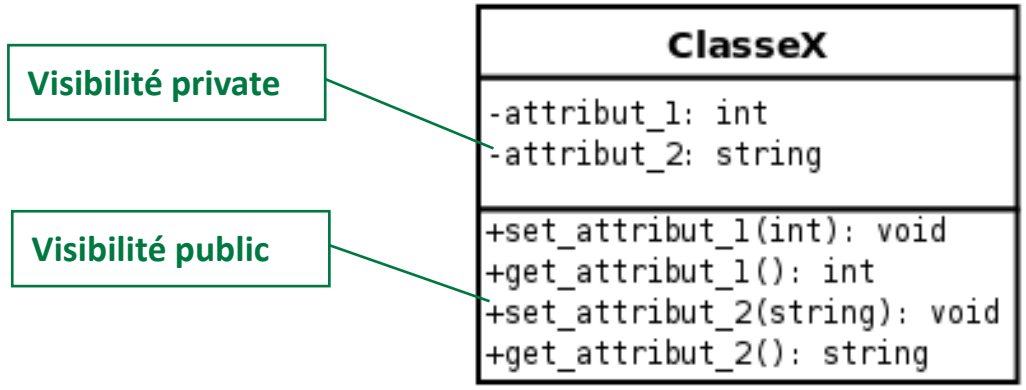


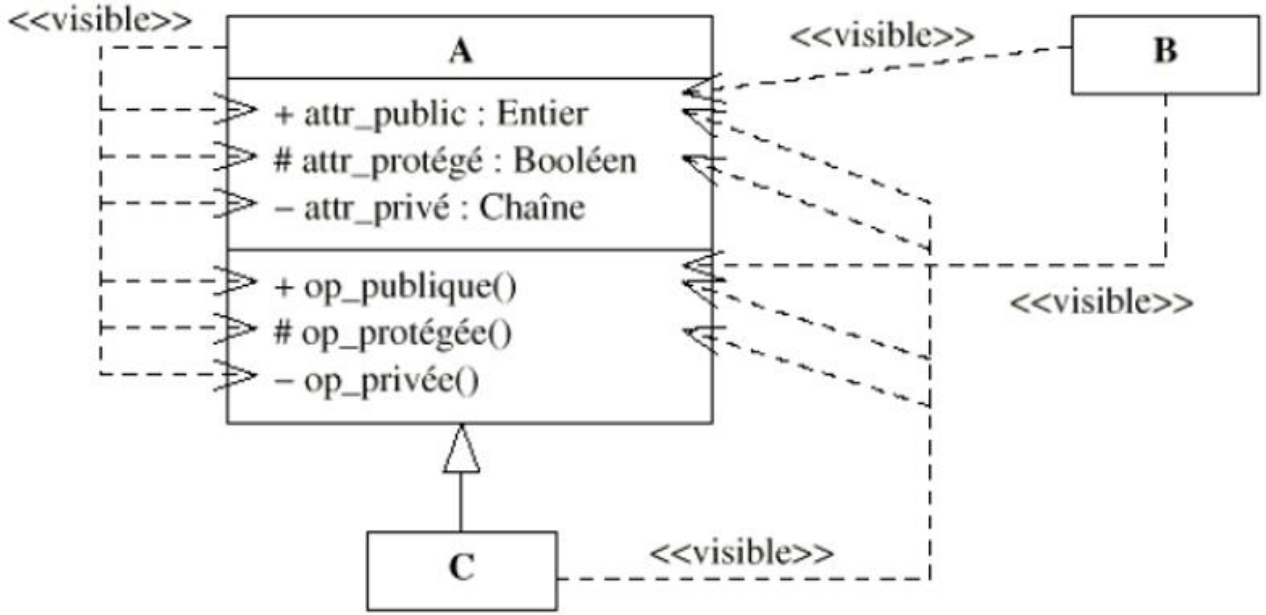
Figure 25 : Bonnes pratiques pour la manipulation des attributs

Remarque

- Dans une classe, le marqueur de visibilité se situe au niveau de chacune de ses caractéristiques (attributs et opérations). Il permet d'indiquer si une autre classe peut y accéder.

PARTIE 1

Démonstration de la visibilité



PARTIE 1

CHAPITRE 3

Modéliser les données du projet par un diagramme de classes

Attributs d'instance

- **Syntaxe de déclaration d'un attribut** :

```
<visibilité> [/] <nom_attribut> :
<type> [ '['<multiplicité>']' [{<contrainte>}] ] [ = <valeur_par_défaut> ]
```

Métalangage Syntaxe

- [ ] : partie optionnelle
- < > : partie plus ou moins libre
- ‘ : caractère d'échappement

Définir un attribut d'instance par :

- **Un nom :** doit être unique dans la classe
- Un **type de données :** *(<type>)* peut être un nom de classe, un nom d'interface ou un type de donnée prédéfini
- Une **visibilité**
- Une **valeur initiale** (*<valeur_par_défaut>*)
- La **multiplicité** *(<multiplicité>)* d'un attribut précise le nombre de valeurs que l'attribut peut contenir. On la définit entre []. Par défaut, la multiplicité est de 1.
- Une **contrainte** : lorsque la multiplicité est > 1, il est possible d'ajouter une contrainte (***{***<contrainte>***}***) pour préciser si les valeurs sont ordonnées (*{ordered}) ou non ({list}*).

PARTIE 1

Multiplicité d'un attribut

- **Syntaxe de déclaration d'un attribut** :

```
<visibilité> [/] <nom_attribut> :
<type> [ '['<multiplicité>']' [{<contrainte>}] ] [ = <valeur_par_défaut> ]
```

- **Exemple** : L'attribut ***sous-titre*** est optionnel : tous les livres n'ont pas de sous-titre, alors qu'ils ont un titre, une langue, etc. UML indique ce caractère optionnel en ajoutant **une multiplicité [0..1]** derrière l'attribut

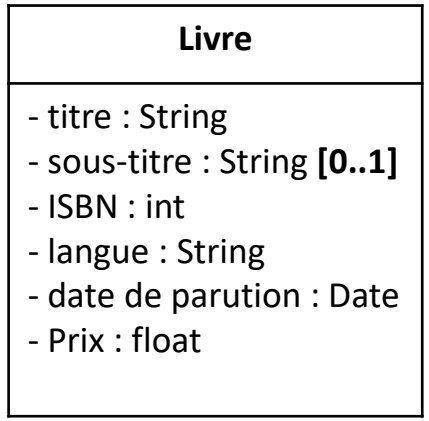


Figure 26 :
Attributs d'une classe Livre avec multiplicité

Méthodes d'instance

- Dans une classe, une opération (même nom et mêmes types de paramètres) doit être unique.
- Quand le nom d'une opération **apparaît plusieurs fois avec des paramètres différents**, on dit que l'opération est **surchargée**
- **Syntaxe de déclaration d'une méthode**:

```
<visibilité> <nom_méthode> ([<paramètre_1>, ... , <paramètre_N>]) :
[<type_renvoyé>] [{<propriétés>}]
```

Définir une méthode d'instance par:

- **Un nom de méthode**
- Les **paramètres** avec leur type
- Une **visibilité**
- Le **type de la valeur de retour** : peut être un nom de classe, un nom d'interface ou un type de données prédéfini
- Les **propriétés** *(<propriétés>)* correspondent à des **contraintes** ou à des informations complémentaires comme les exceptions, les pré-conditions, les post-conditions ou encore l'indication qu'une méthode est abstraite (abstract), etc.

Signature d'une méthode

- La signature d'une méthode indique le nombre de paramètres en entrée ainsi que le type des paramètres (string, integer, array, object...).

Direction des paramètres des méthodes

- La syntaxe de définition d'un paramètre *(<paramètre>)* est la suivante :

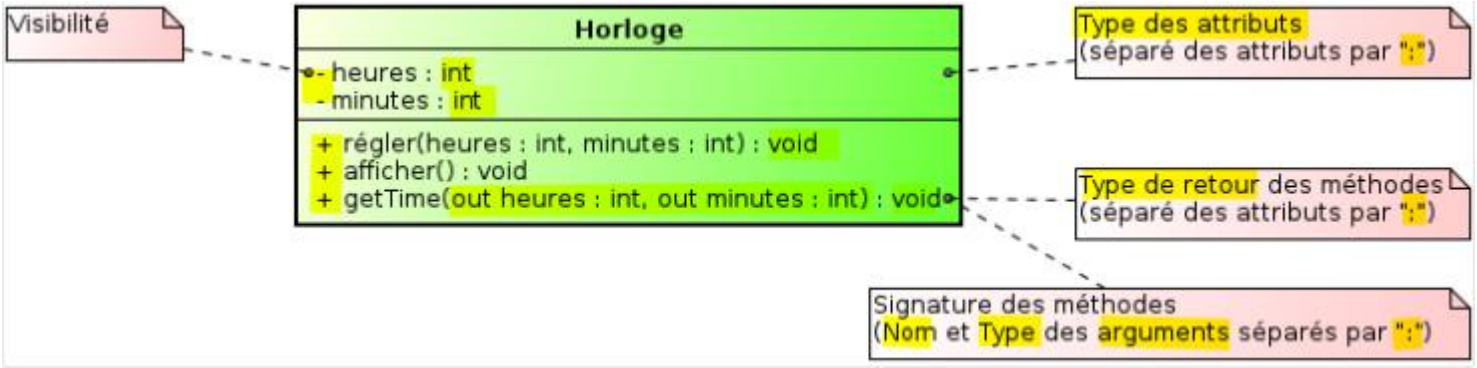
```
[<direction>] <nom_paramètre>:<type> ['['<multiplicité>']'] [=<valeur_par_défaut>]
```

La direction de transmission d'un paramètre peut être:

- **in** : paramètre d'entrée passé par valeur. Les modifications du paramètre ne sont pas disponibles pour l'appelant. C'est le comportement par défaut.
- **out** : paramètre de sortie uniquement. Il n'y a pas de valeur d'entrée et la valeur finale est disponible pour l'appelant.
- **inout** : paramètre d'entrée/sortie. La valeur finale est disponible pour l'appelant.

Exemple : la classe Horloge

- La méthode `régler(heures : int, minutes : int) : void` → L'appelant de la méthode veut affecter les attributs heures et minutes avec des valeurs qu'il va donner.
- La direction des paramètres sera alors **in** (par défaut) : `régler(in heures : int, in minutes : int) : void`

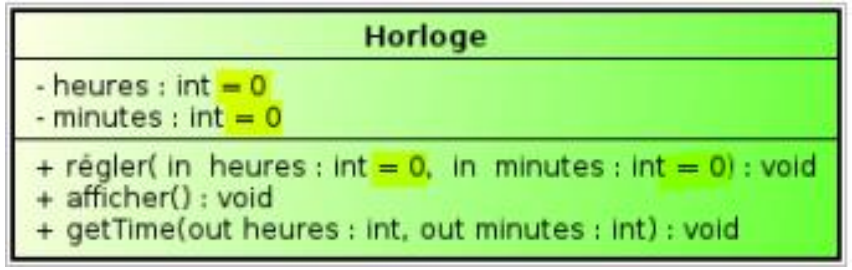


- La méthode `getTime(heures : int, minutes : int) : void` → L'appelant de la méthode veut récupérer les valeurs des attributs heures et minutes.
- La direction des paramètres sera alors out : `getTime(out heures : int, out minutes : int) : void`

PARTIE 1

Valeurs par défauts des attributs et des paramètres des méthodes

- On indique les valeurs par défaut des attributs **lors de leur construction**
- et les valeurs par défauts des paramètres des méthodes **s'ils ne sont pas clairement spécifiés lors de l'appel**.



CHAPITRE 3

Modéliser les données du projet par un diagramme de classes

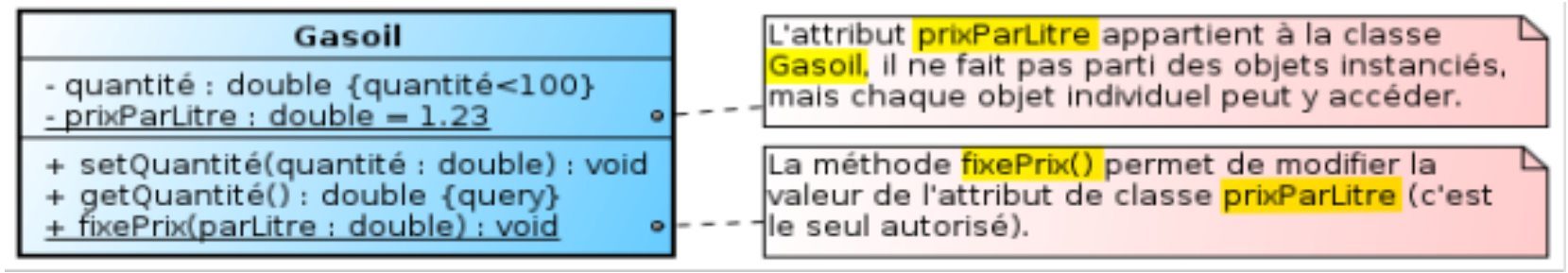
Attributs et méthodes statiques

- Une classe peut contenir des attributs et des méthodes qui lui sont **propres** et auxquels on peut **accéder sans nécessairement instancier des objets**.

- **Un attribut de classe** n'appartient pas à un objet en particulier mais à toute la classe (il n'est pas instancié avec l'objet).
- Il **garde une valeur unique et partagée par toutes les instances de la classe.**

- **Une méthode de classe ne peut manipuler que des attributs statiques et ses propres paramètres**

- Graphiquement, un attribut ou une méthode de classe est représenté par un nom souligné.



PARTIE 1

CHAPITRE 3

Modéliser les données du projet par un diagramme de classes

1. Définition du diagramme de classes
2. Visibilité, encapsulation des méthodes et attributs
3. Attributs et Méthodes d'instance
4. Attributs et méthodes de classe
5. **Contraintes**
6. Attributs calculés (dérivé)
7. Multiplicité (cardinalité)
8. Constructeur et destructeurs
9. Modèles de classe
10. Relations entre classes
11. Classes concrètes et abstraites
12. Relation de réalisation et notion d'interfaces

Contraintes

- Une **contrainte est une condition** écrite entre 2 accolades **{}** elle peut être exprimée dans :
  - Un langage naturel (description textuelle)
  - Un langage formel (C++, java, …).
- Les contraintes **sont appliquées sur les attributs** ou les **méthodes**
- En voici quelques unes qui sont utilisées :

{readOnly} : la valeur de l'attribut ne peut plus être modifiée une fois initialisée

{ordered} : pour les attributs de type collection. C'est pour préciser si les valeurs sont ordonnées

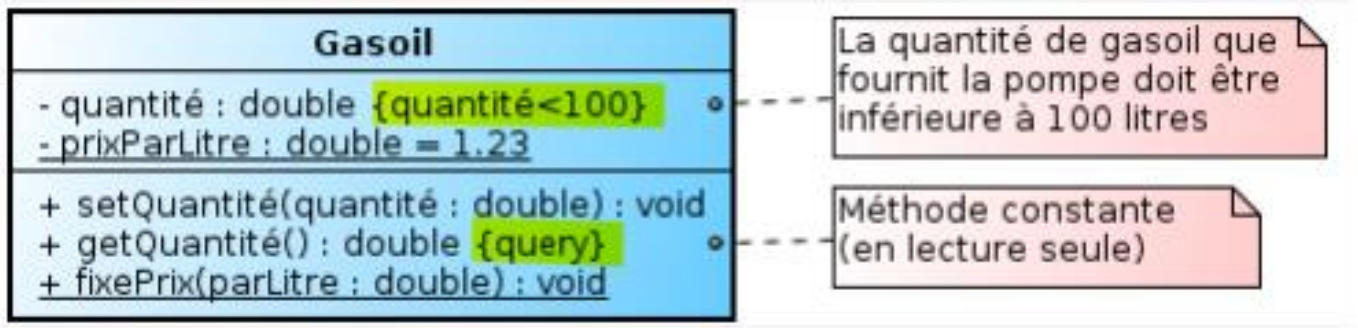
{list} : similaire à {ordered} mais pour préciser si les valeurs ne sont pas ordonnées (par défaut)

{unique} : aucun doublon dans les valeurs de la collection

{not null} : L'attribut doit à tout prix être initialisé

{query} : appliqué sur une méthode si son code ne modifie nullement l'état de l'objet (aucun de ses attributs)

Contraintes - Exemple

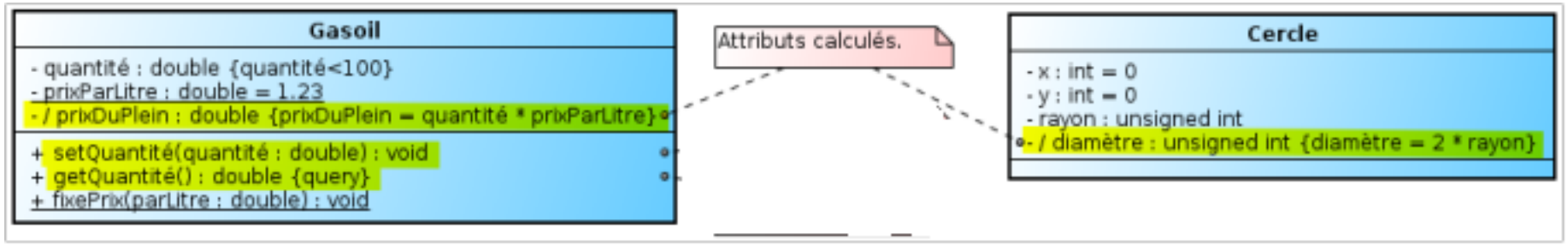


CHAPITRE 3

Modéliser les données du projet par un diagramme de classes

Attributs dérivés

- **Un attribut dérivé** est un attribut dont **la valeur peut être calculée à partir d'autres informations disponibles dans le modèle**, par exemple d'autres attributs de la même classe, ou de classes en association.
- Cet attribut, qui pourrait donc être considéré comme redondant, est néanmoins gardé par l'analyste s'il correspond à un concept important aux yeux de l'expert métier.
- **Il est noté en UML avec un « / » avant son nom** et **suivi d'une contrainte** permettant de le calculer



PARTIE 1

CHAPITRE 3

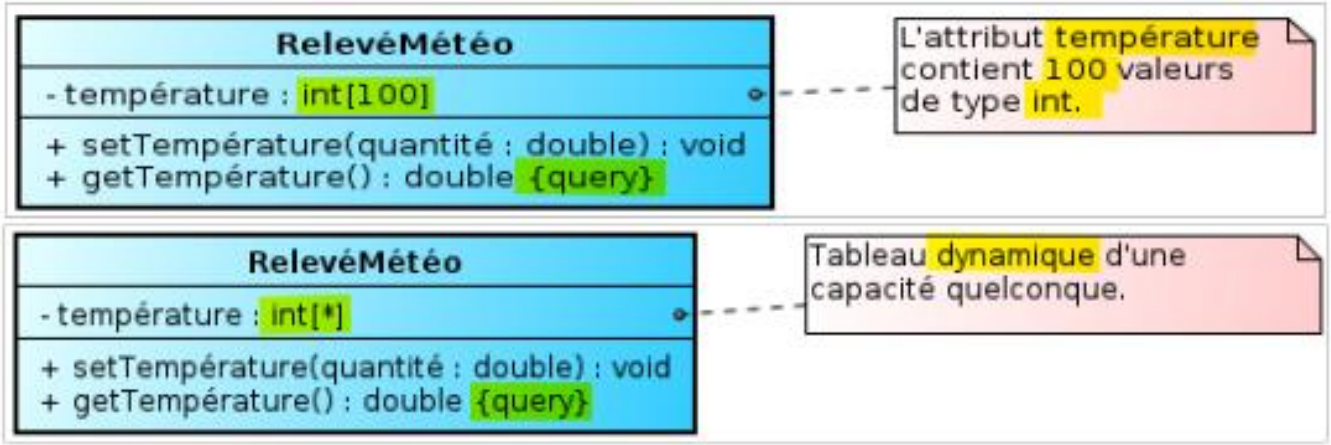
Modéliser les données du projet par un diagramme de classes

PARTIE 1

Cardinalité

- La multiplicité indique **le nombre de valeur que l'attribut peut contenir**. L'attribut est souvent un tableau de valeurs statique ou dynamique (collection).
- La multiplicité se note entre crochets **[]** après le type de valeur que contient l'attribut.
- On peut préciser la multiplicité **également pour un paramètre de méthode** au niveau de la signature

- **Exemple** : Une station météo doit relever la température à intervalle de temps régulier.
Elle doit pouvoir stocker 100 relevés.



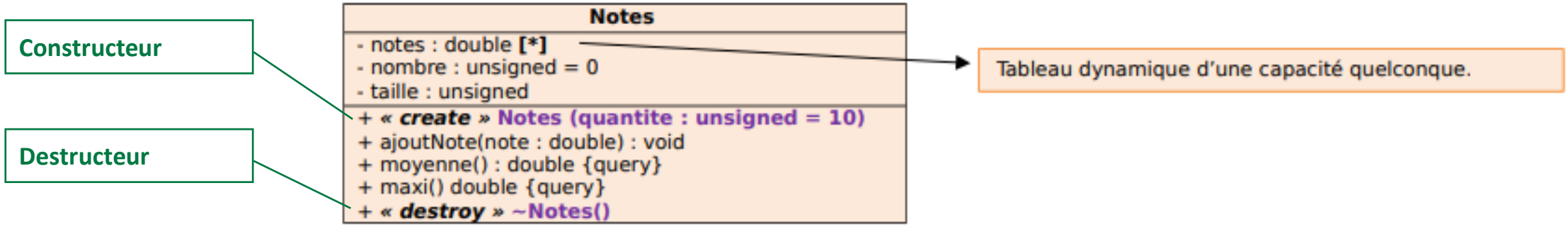
Cardinalité	Signification
0..1	Zéro ou une fois
1..1	Une et une seule fois
0..* (ou *)	De zéro à plusieurs fois
1..*	De une à plusieurs fois
m..n	Entre m et n fois
n..n (ou n)	n fois

CHAPITRE 3

Modéliser les données du projet par un diagramme de classes

Constructeurs et destructeurs

- Les stéréotypes peuvent être utilisés pour identifier des opérations particulières comme :
  - **les constructeurs** : stéréotype **« create »**
  - et **le destructeur** : stéréotype **« destroy ».**
- Dans UML, les constructeurs et les destructeurs sont des méthodes spéciales qui ont le même nom que la classe (Formalisme Java)



Remarque

Un **stéréotype** est **un mot-clé inventé par les utilisateurs**.

- Il existe des stéréotypes définis comme <<include>>, <<create>>, etc.
- Mais on peut créer notre propre mot-clé

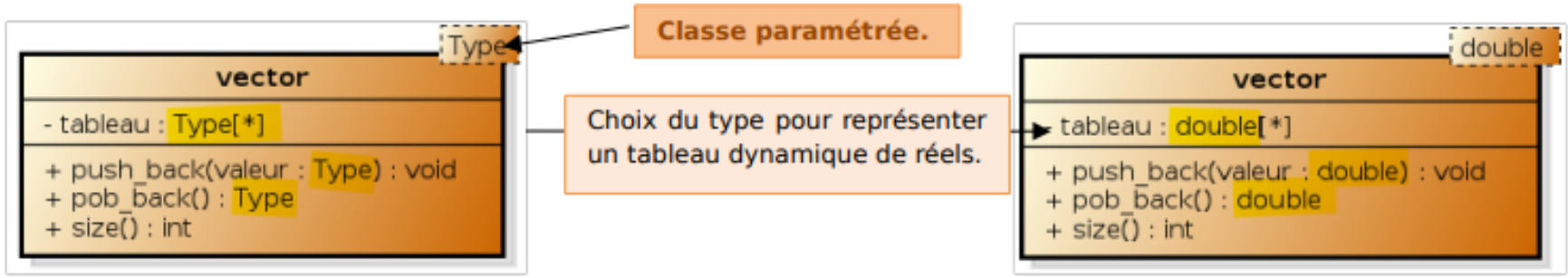
PARTIE 1

CHAPITRE 3

Modéliser les données du projet par un diagramme de classes

Modèles de classe

- Les modèles (**templates**) sont une fonctionnalité avancée de l'orientée objet.
- Un modèle est une classe paramétrée qui permet ainsi de choisir le type des attributs au besoin suivant le paramètre précisé, dans le coin supérieur droit dans un rectangle dont les côtés sont en pointillés.
- Ces modèles de classes sont particulièrement utiles pour toutes les collections qui stockent des valeurs d'un même type, soit sous forme de tableaux dynamiques ou de listes.
- **Exemple** : La classe **vector**



CHAPITRE 3

Modéliser les données du projet par un diagramme de classes

Relation de dépendance

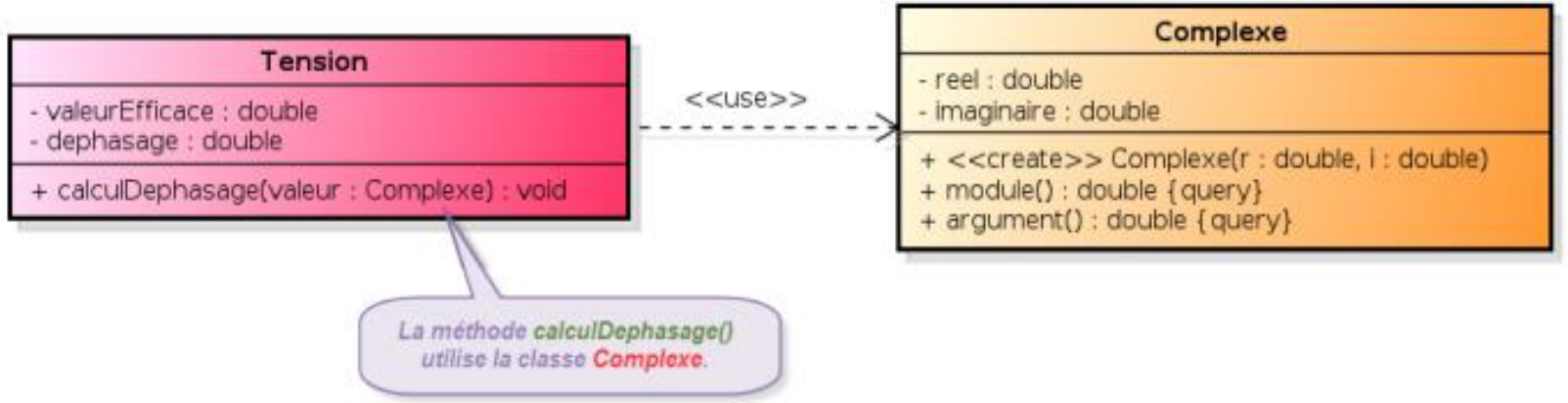
- La dépendance est **la forme la plus faible** de relation entre classes.
- Une dépendance entre deux classes **autorise simplement une classe à utiliser des objets d'une autre classe**
- Il s'agit d'une relation transitoire, au sens où la première interagit brièvement avec la seconde sans conserver à terme de relation avec elle (**liaison ponctuelle**).
- La dépendance **Notation UML** : représentée par un trait discontinu orienté, reliant les deux classes. La dépendance est souvent stéréotypée par ***<<use>>*** (<<utilise un>>) <<use>>

Une relation de dépendance est habituellement utilisée si :

- **Une classe utilise un objet d'une autre classe comme argument** dans la signature d'une méthode
- **Un objet de l'autre classe est créé à l'intérieur de la méthode**.
- Dans les deux cas la durée de vie de l'objet est très courte, elle correspond à la durée d'exécution de la méthode

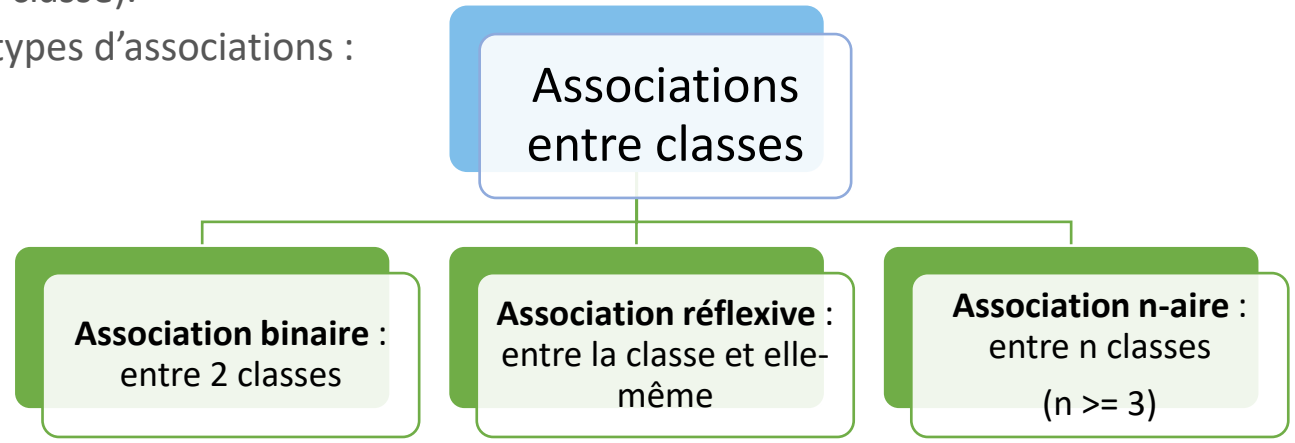
PARTIE 1

Relation de dépendance - Exemple



Associations entre classes

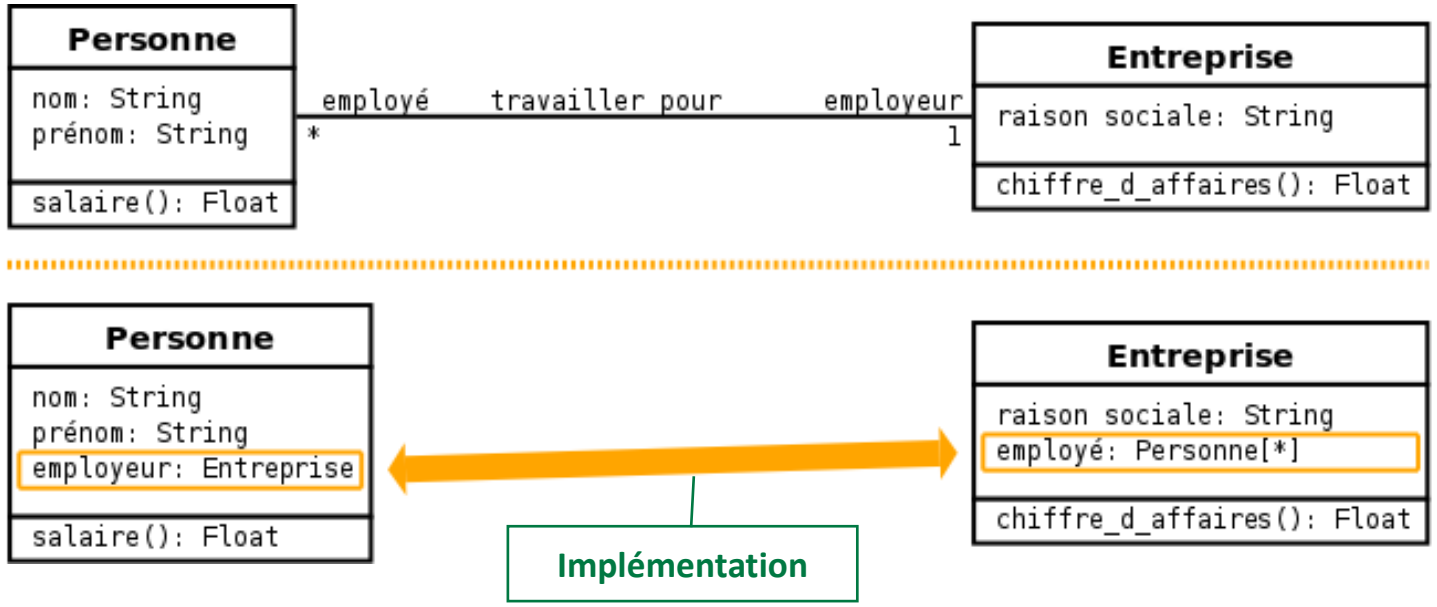
- Par contre, **l'association** est une relation **plus forte**. Elle indique **qu'une classe est en relation avec une autre pendant un certain laps de temps**.
- La ligne de vie des deux objets concernés ne sont cependant pas associés étroitement (un objet peut être détruit sans que l'autre le soit nécessairement).
- **Notation UML :** L'association est représentée par un **simple trait continu**, reliant les deux classes. Le fait que deux instances soient ainsi liées permet la navigation d'une instance vers l'autre, et vice versa (chaque classe possède un attribut qui fait référence à l'autre classe).
- Il existe plusieurs types d'associations :



PARTIE 1

Associations binaire - Implémentation

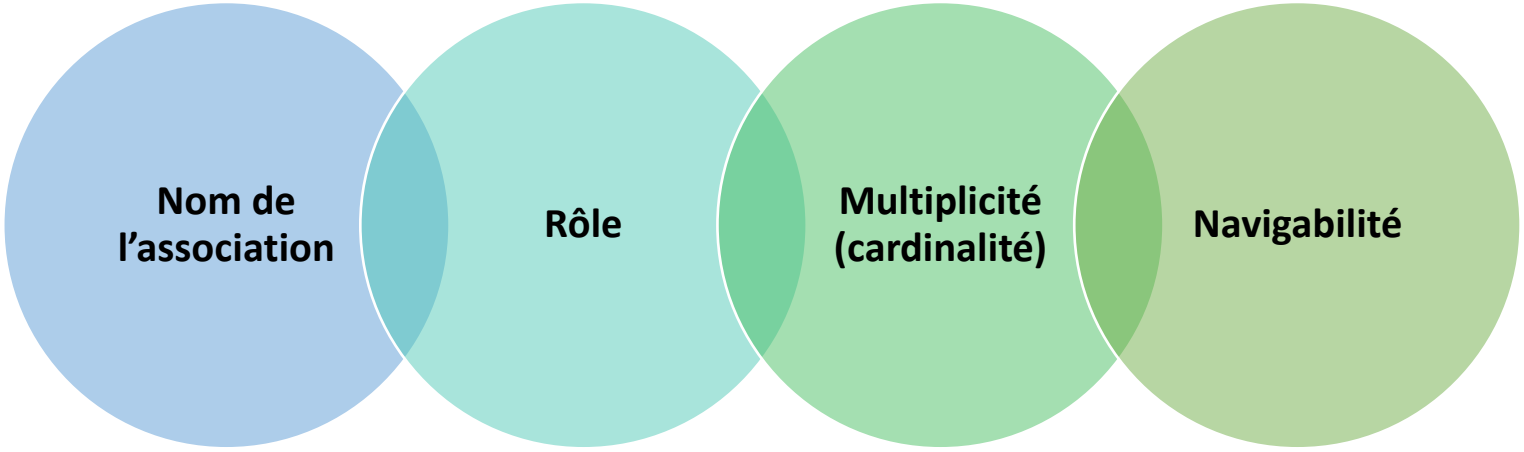
- Une association **binaire** est matérialisée par un trait plein entre les classes,
- Si on veut implémenter l'association dans un langage orienté objet, **chaque classe de l'association contiendra une référence (ou un pointeur) de l'objet de la classe associée sous la forme d'un attribut.**
- **Exemple**



PARTIE 1

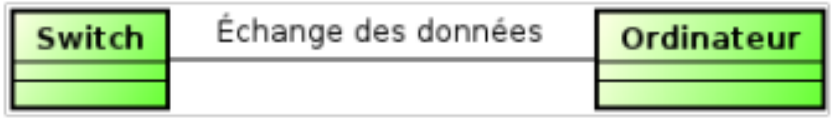
Associations entre classes

- Nous pouvons détailler l'association en indiquant :



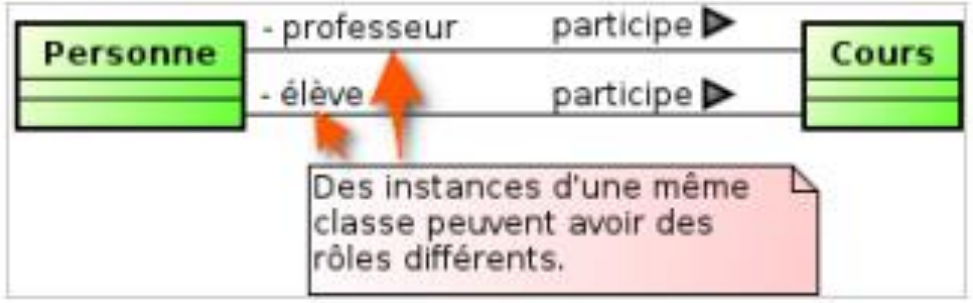
Associations entre classes – Nom de l'association

- L'association peut être orné d'un texte, avec un éventuel sens de lecture, qui permet de nous **informer de l'intérêt de cette relation**.
- Ce nom n'est absolument pas exploité dans le code.
- Le nom d'une association doit respecter les conventions de nommage des classeurs : **commencer par une lettre majuscule.**



Associations entre classes – Rôle des extrémités de l'association

- **Chaque extrémité d'une association peut être nommée**.
- Ce nom est appelé **rôle** et indique la **manière dont l'objet est vu de l'autre coté de l'association**.
- **Lorsqu'un objet A est lié à un autre objet B par une association, cela se traduit souvent par un attribut supplémentaire dans A qui portera le nom du rôle B. (et inversement).**



PARTIE 1

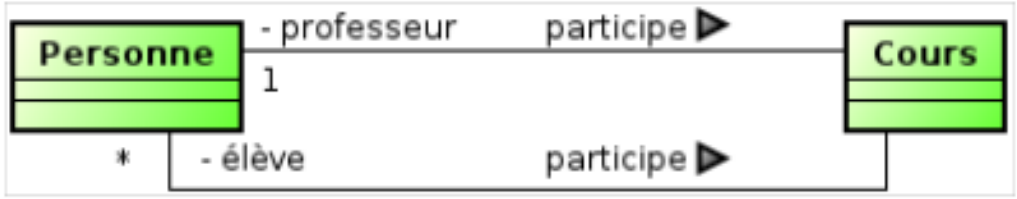
PARTIE 1

Associations entre classes – Multiplicité de l'association

- La multiplicité (cardinalité) indique **le nombre d'instances de classe étant en relation avec la classe situé à l'autre extrémité de l'association**.
- Par défaut, la cardinalité est égale à 1.

Cardinalité	Signification
0..1	Zéro ou une fois
1..1	Une et une seule fois
0..* (ou *)	De zéro à plusieurs fois
1..*	De une à plusieurs fois
m..n	Entre m et n fois
n..n (ou n)	n fois

- **Exemple** :
  - dans un cours, il y a 1 professeur et plusieurs élèves

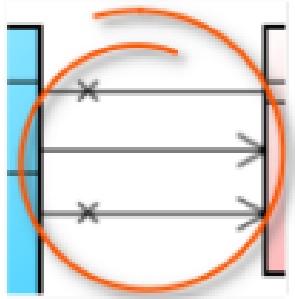


  - Une personne travaille pour **1** entreprise et une entreprise emploie **1 ou plusieurs** personnes

Associations entre classes – Navigabilité

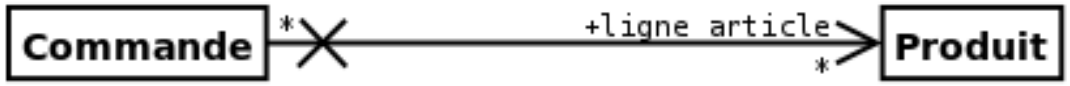
Navigabilité bidirectionnelle	Navigabilité unidirectionnelle

• par défaut
• chaque classe possède un attribut qui fait référence à l'autre classe en association.
• plus complexe à réaliser
• A éviter dans la mesure du possible
	• une seule classe possède un attribut qui fait référence à l'autre classe
	• plus fréquente
	• la première classe peut solliciter une deuxième et que l'inverse est impossible
	• peut se représenter de 3 façons différentes :
	  • Une **croix** du coté de l'objet qui ne peut pas être sollicité
	  • Une **flèche** du coté de l'objet qui peut être sollicité
	  • **Les 2** représentations précédentes à la fois



Associations entre classes – Navigabilité

- **Exemple :**
  - L'extrémité du côté de la classe Commande **n'est pas navigable** : les instances de la classe Produit ne stockent pas de liste d'objets du type Commande.

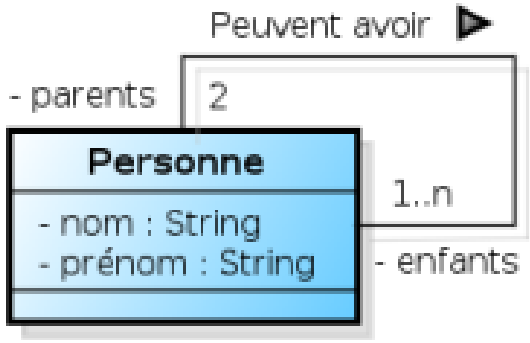


  - L'extrémité du côté de la classe Produit est **navigable** : chaque objet commande contient une liste de produits.

PARTIE 1

Associations entre classes - Association réflexive

- Une association **réflexive** (ou **récursive)** : **lie une classe avec elle-même**



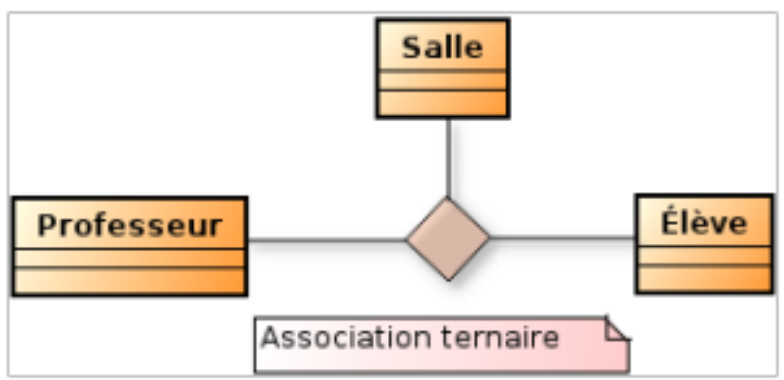
PARTIE 1

PARTIE 1

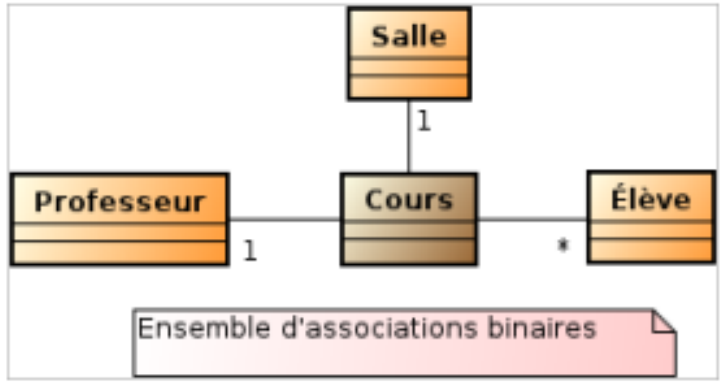
Associations reliant plusieurs classes - Association n-aire

- **L'association n-aire :** est une association qui lie plus de 2 classes entre elles, est une. (**n >=3**)
- **Notation** : L'association n-aire se représente par un losange d'où part un trait allant à chaque classe.
- L'association n-aire est imprécise, difficile à interpréter et souvent source d'erreur, elle est donc **très peu utilisée**.
- La plupart du temps, on se sert pour la modélisation au début du projet, puis elle est vite remplacée par un ensemble d'associations binaires afin de lever toute ambiguïté.

- **Exemple** : Un cours peut correspondre à l'association ternaire de 3 classes.



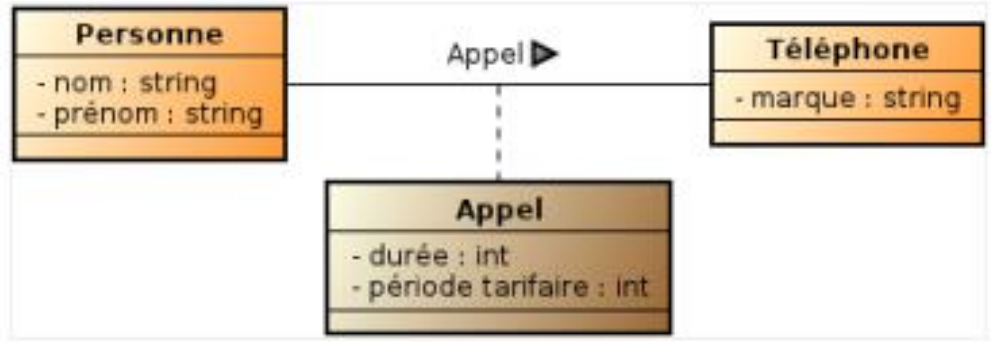
À remplacer par :



PARTIE 1

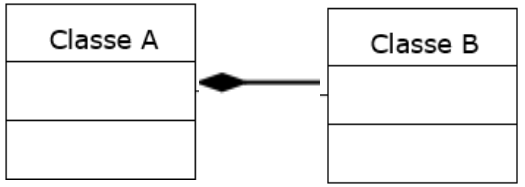
Associations reliant plusieurs classes - Classe association

- Une association peut apporter de nouvelles informations (**attributs et méthodes**) qui n'appartiennent à aucune des deux classes qu'elle relie et qui sont spécifiques à l'association.
- Ces nouvelles informations peuvent être représentées par **une nouvelle classe attachée à l'association via un trait en pointillés.**
- **Exemple :**
  - Lorsqu'une personne utilise un téléphone, il faut pouvoir mesurer **la durée de l'appel** et savoir **à quel moment** il a lieu afin de le tarifier.
  - On va ajouter donc deux attributs **durée** et **période tarifaire** qui n'appartiennent ni à la classe Personne ni à la classe Téléphone.
  - Ces deux attributs **sont mis dans une nouvelle classe (la classe Appel)** qui est attachée à l'association.



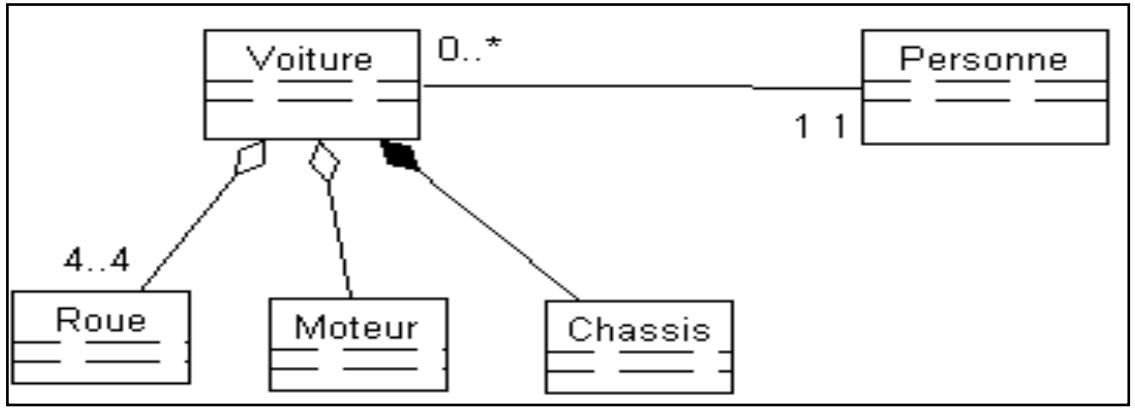
Associations particulières - Composition

- La **composition** indique qu'un **objet A (appelé conteneur) est constitué d'un autre objet B**.
- Cet objet A n'appartient qu'a l'objet B et ne peut pas être partagé avec un autre objet
- **C'est une relation très forte**, si l'objet A disparaît, alors l'objet B disparaît aussi.
- Elle se représente par **un losange plein** du coté de l'objet conteneur



- **Exemple** :

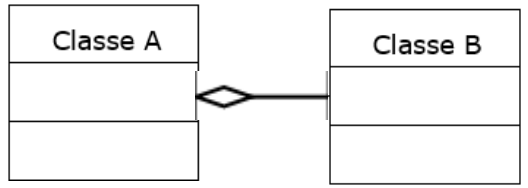
  Une voiture **est composée d'un châssis**. Il ne peut pas être utilisé pour une autre voiture



PARTIE 1

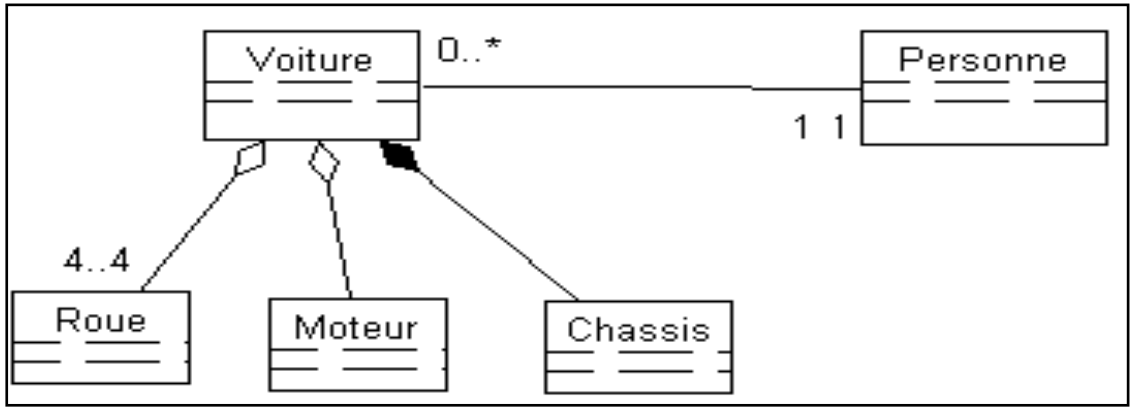
Associations particulières - Agrégation

- **L'agrégation** indique **qu'un objet A possède un autre objet B**, mais contrairement à la composition, l'objet B peut exister indépendamment de l'objet A.
- La suppression de l'objet A n'entraîne pas la suppression de l'objet B.
- Elle se représente par **un losange vide** du coté de l'objet conteneur



- **Exemple** :

Une voiture **est composée de 4 roues et un moteur**. Ils peuvent être utilisés pour une autre voiture

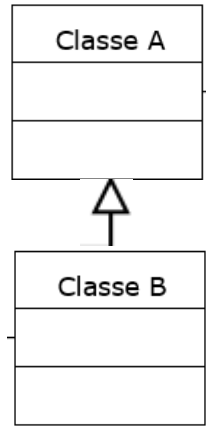


PARTIE 1

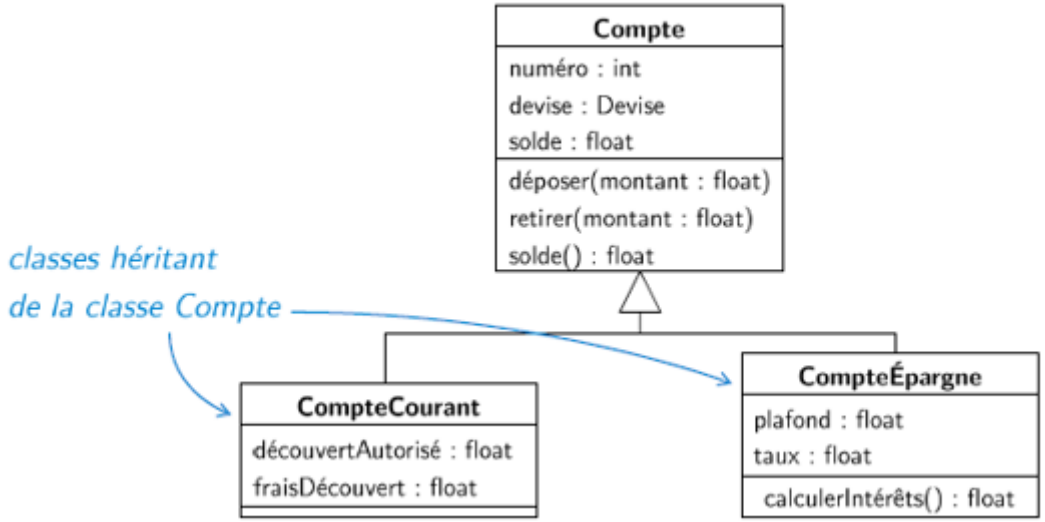
PARTIE 1

Relation d'héritage

- Le mécanisme **d'héritage** permet de **mettre en relation des classes ayant des caractéristiques communes** (attributs et méthodes) en respectant une certaine filiation.
- L'héritage indique **qu'une classe B est une spécialisation d'une classe A**. La classe B (appelé classe fille, classe dérivée ou sous classe) hérite des attributs et des méthodes de la classe A (appelée classe mère, classe de base ou super classe).
- **Notation** : Il se représente par **un triangle vide** afin d'indiquer le sens de la généralisation (inverse de la spécialisation).



- **Exemple** :



CHAPITRE 3

Modéliser les données du projet par un diagramme de classes

Classes concrètes et abstraites

Classe concrète

- Possède des instances.
- Elle constitue un modèle complet d'objet
- Tous les attributs et méthodes sont complètements définis

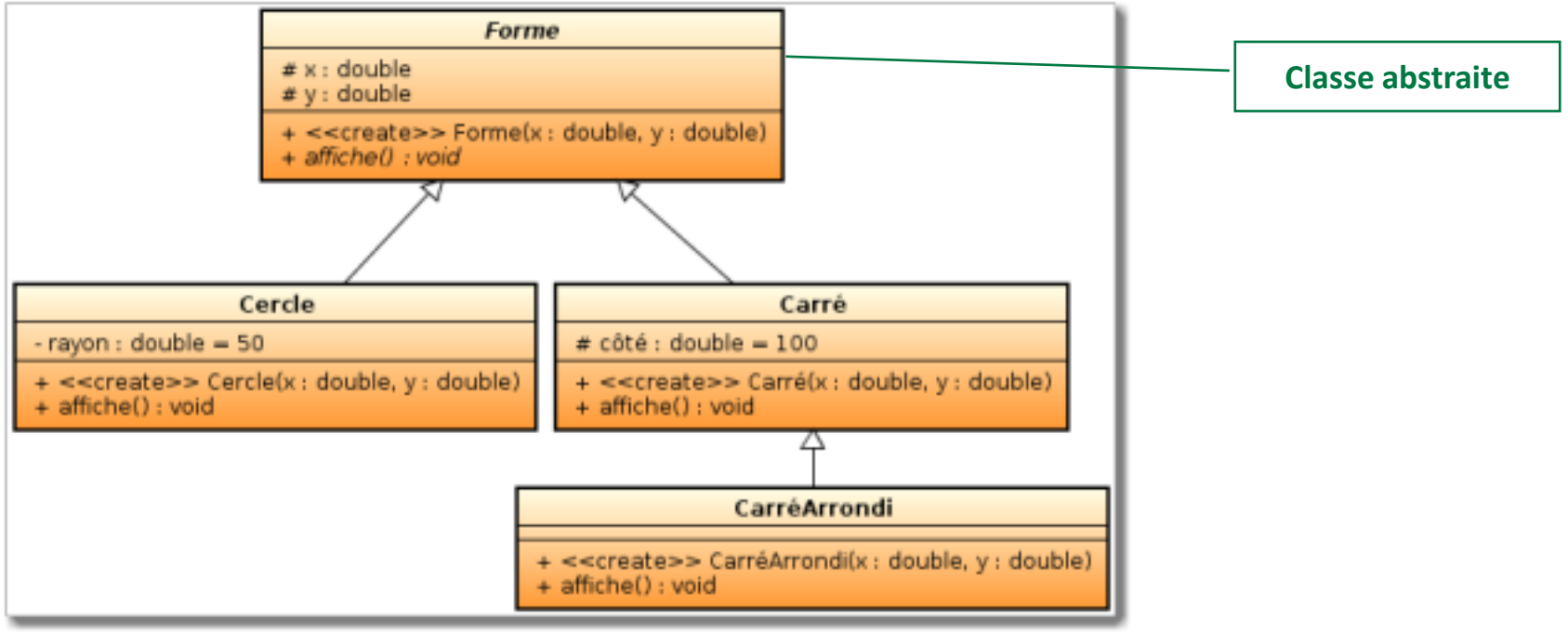
Classes abstraites

- Ne peut pas posséder d'instance directe
- Elle ne fournit pas une description complète
- Elle force le passage par l'héritage pour avoir des sous-classes concrètes
- Elle sert à factoriser des attributs et des méthodes à ses sous-classes
- Une classe abstraite possède généralement des méthodes communes non définies

Classes concrètes et abstraites

- En UML, une classe ou une méthode abstraite sont représentées avec une ***mise en italique*** du nom de la classe ou de la méthode.
- **On peut aussi mettre le stéréotype <<abstract>>**

- **Exemple** :



CHAPITRE 3

Modéliser les données du projet par un diagramme de classes

Notion d'interfaces

- **L'interface est une classe particulière** qui ne définit que des éléments d'interface.
- Il peut s'agir de l'interface complète d'une entité, ou très généralement d'une partie d'interface qui sera commune à plusieurs entités.
- Le rôle de cette interface, stéréotypé ***<<interface>>,*** est de **regrouper un ensemble d'opérations assurant un service cohérent**.
- **Objectif** : diminuer fortement le couplage entre deux classes.
- Une interface **ne dispose que de méthodes publiques abstraites** (ou dit autrement, non définies).
- La généralisation peut être utilisée entre interfaces,
- par contre, une interface ne peut être instanciée (comme une classe abstraite).

PARTIE 1

Notion d'interfaces

- On a déjà parlé du principe **d'encapsulation** qui consiste, entre autre, de protéger au maximum les attributs d'un objet.
- Dans ce cas de figure (qui est normalement le cas courant) la modification des attributs passe obligatoirement par des méthodes adaptées.

- Il peut être intéressant de **protéger encore plus un objet afin qu'il ne soit accessible que par un intermédiaire** : **C'est l'interface qui joue ce rôle. C'est une forme d'encapsulation**
- **Seules quelques méthodes sont autorisées à être utilisées.** Ce sont les méthodes désignées par l'interface.
- Comme son nom l'indique, l'interface sert d'intermédiaire entre l'application et l'objet sollicité.

Relation de réalisation

- Comme l'interface n'a qu'un objectif de spécification, au moins un élément d'implémentation (une classe) doit lui être associée.
- **La relation de réalisation** permet de **mettre en relation une classe avec son implémentation.**
- Graphiquement, la réalisation se représente par un trait discontinu terminé par une flèche triangulaire parfois, mais pas nécessairement, stéréotypé ***<<realize>>***

- <u>Attention</u> : **Une classe qui réalise (implémente) une interface est dans l'obligation de définir les méthodes décrites par l'interface (notion de contrat à respecter) comme lorsqu'une classe concrète redéfinie les méthodes abstraites héritées de sa classe parente abstraite.**

PARTIE 1

CHAPITRE 4

Modéliser les interactions d'objets d'un système à l'aide d'un diagramme de séquence

Ce que vous allez apprendre dans ce chapitre :

- Rôle du diagramme de séquences
- Distinguer le diagramme de séquence boîte noire du diagramme de séquence boîte blanche
- Diagramme de séquence comme boîte blanche
- Délimitation du diagramme de séquence
- Types de messages
- Fragments d'interactions combinés

5 heures

CHAPITRE 4

Modéliser les interactions d'objets d'un système à l'aide d'un diagramme de séquence

1. **Rôle du diagramme de séquences**
2. Distinguer le diagramme de séquence boîte noire du diagramme de séquence boîte blanche
3. Diagramme de séquence comme boîte blanche
4. Délimitation du diagramme de séquence
5. Types de messages
6. Fragments d'interactions combinés

Diagramme de séquence (DS)

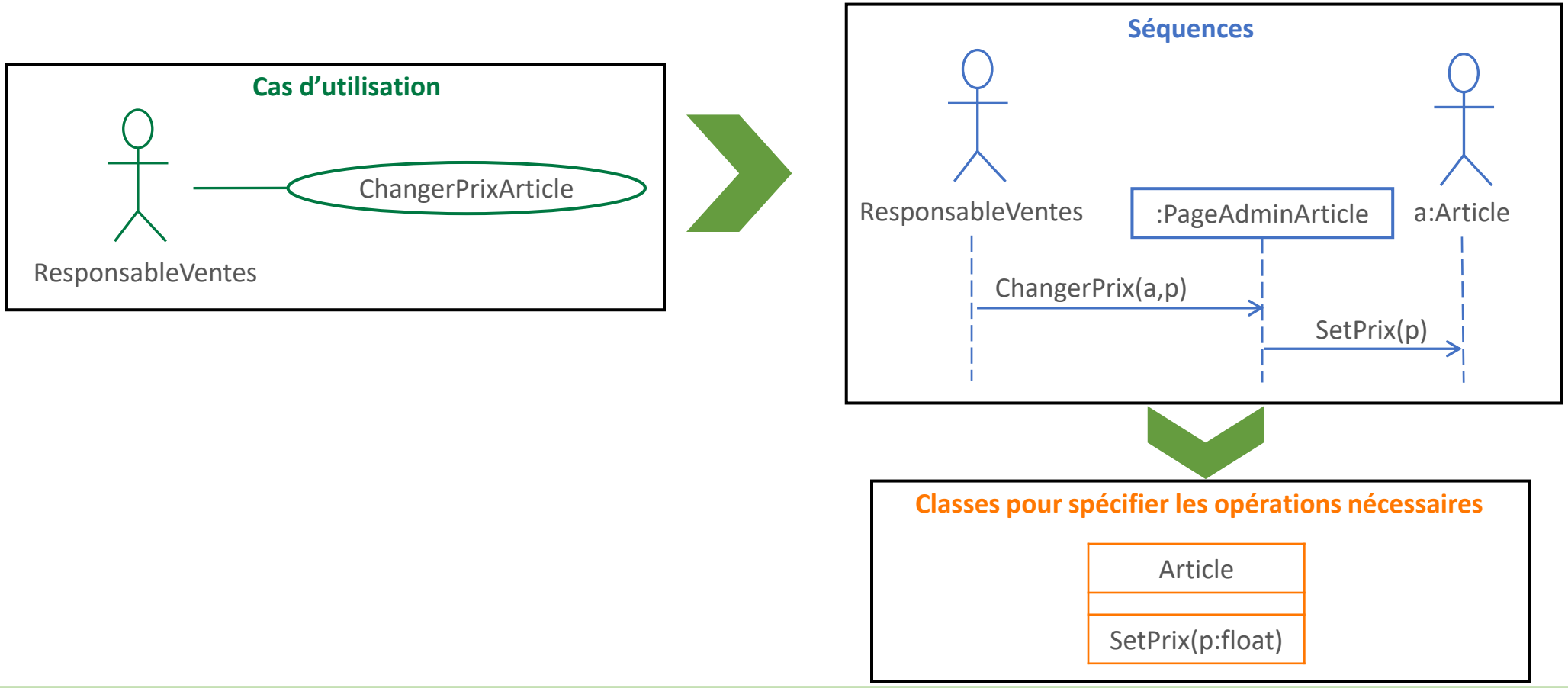
- Les diagrammes de cas d'utilisation modélisent à **QUOI** sert le système, en organisant les interactions possibles avec les acteurs.

- Les diagrammes de classes permettent de spécifier la structure et les liens entre les objets dont le système est composé : ils spécifient **QUI** sera à l'œuvre dans le système pour réaliser les fonctionnalités décrites par les diagrammes de cas d'utilisation.

- **Les diagrammes de séquences** permettent de décrire **COMMENT** les éléments du système interagissent entre eux et avec les acteurs :
  - Les objets au cœur d'un système interagissent **en s'échangeant des messages**.
  - Les acteurs interagissent avec le système au **moyen d'IHM** (Interfaces Homme-Machine).

Modéliser les interactions

- Pour être complètement spécifiée, une interaction doit être décrite dans plusieurs diagrammes UML :



PARTIE 1

Point de vue temporel sur les interactions

Diagramme de Séquence Boîte Noire (BN)

- Appelé aussi Diagramme Séquence Système (DSS)
- **Modélise les interactions entre acteurs et Système**
- Fait pendant la capture des besoins
- Permet de décrire les scénarios des cas d'utilisation

Niveau Analyse

Diagramme de séquence Boîte Blanche (BB)

- **Modélise les interactions entre objets** à l'intérieur du système
- Permet de réfléchir à l'affectation de **responsabilités aux objets** :
  - ✓ Qui crée les objets?
  - ✓ Qui permet d'accéder à un objet?
  - ✓ Quel objet reçoit un message provenant de l'interface?
  - ✓ …

Niveau Conception

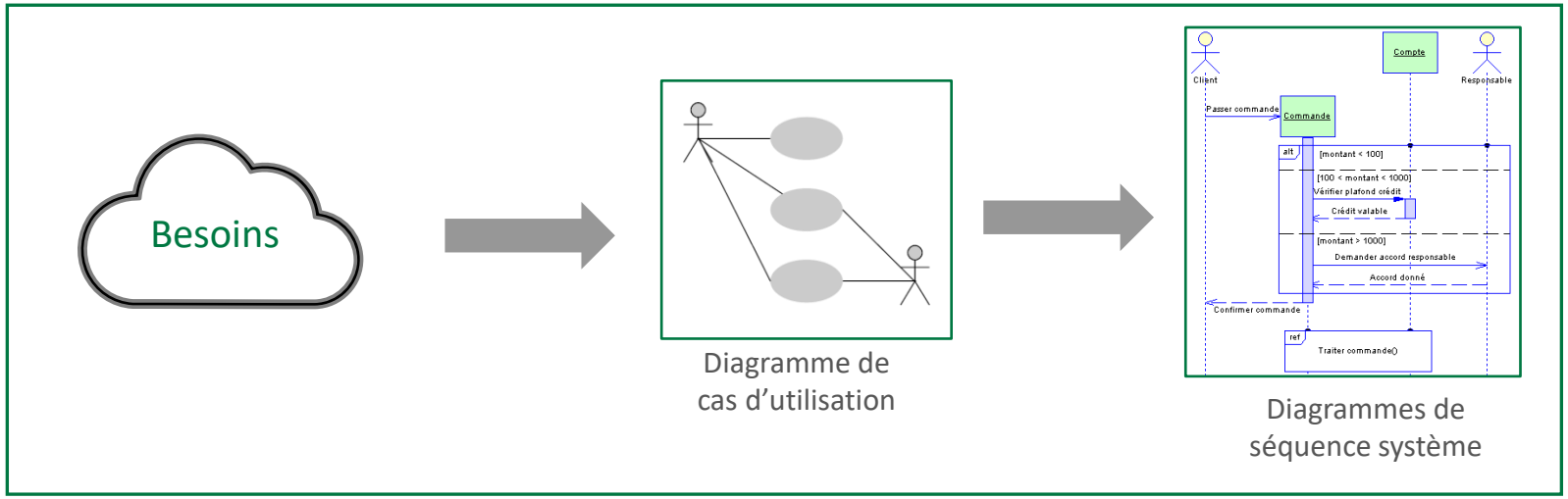
CHAPITRE 4

Modéliser les interactions d'objets d'un système à l'aide d'un diagramme de séquence

1. Rôle du diagramme de séquences
2. **Distinguer le diagramme de séquence boîte noire du diagramme de séquence boîte blanche**
3. Diagramme de séquence comme boîte blanche
4. Délimitation du diagramme de séquence
5. Types de messages
6. Fragments d'interactions combinés

DS Boîte Noire

- Appelé aussi **Diagramme de Séquence « Système »** (**DSS**) ;
- Le système informatique à développer est considéré comme une boîte noire ;
- Le comportement du système est décrit vu de l'extérieur, sans modéliser le comment de sa réalisation en interne ;
- On repart de la description textuelle d'un cas d'utilisation et transformer chaque étape en une flèche représentant un message.

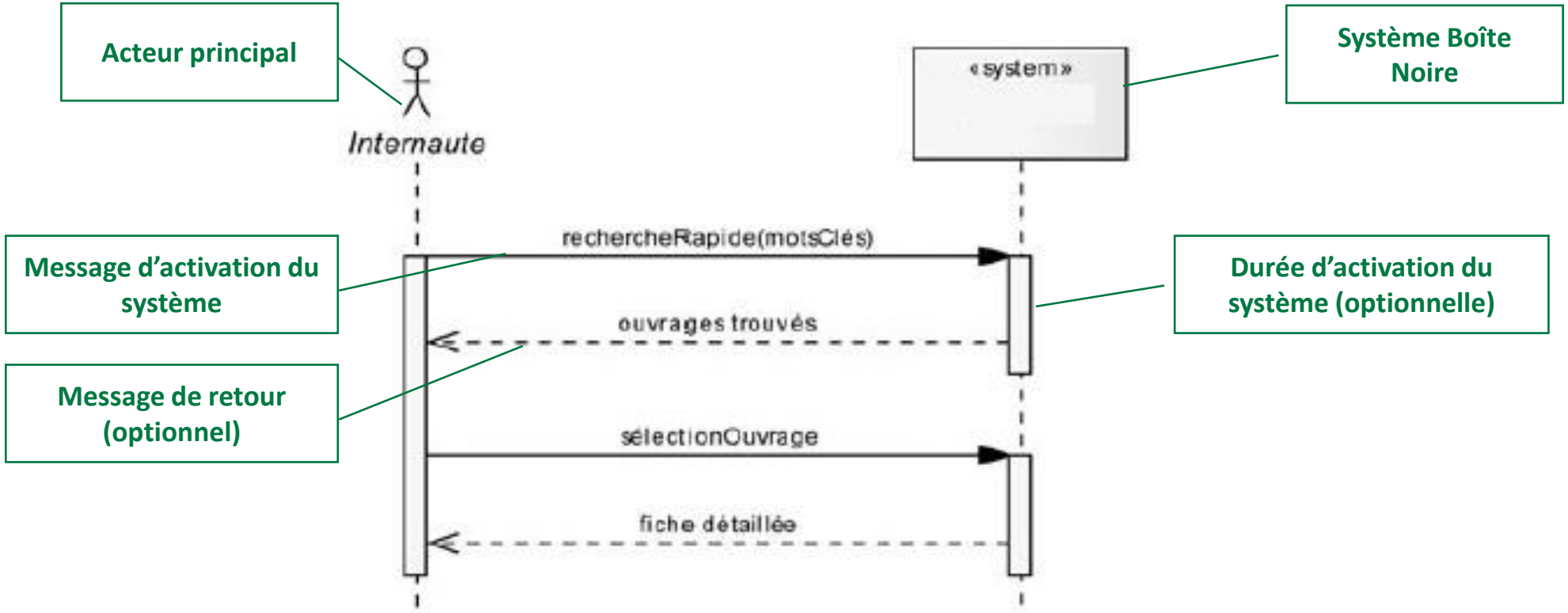


DS Boîte Noire

- **Le système est considéré comme un tout** et est représenté par **une ligne de vie ;**
- Chaque acteur est également associé à une ligne de vie ;
- **L'ordre chronologique se déroule vers le bas ;**
- L'ordre des messages doit suivre la séquence décrite dans le cas d'utilisation.

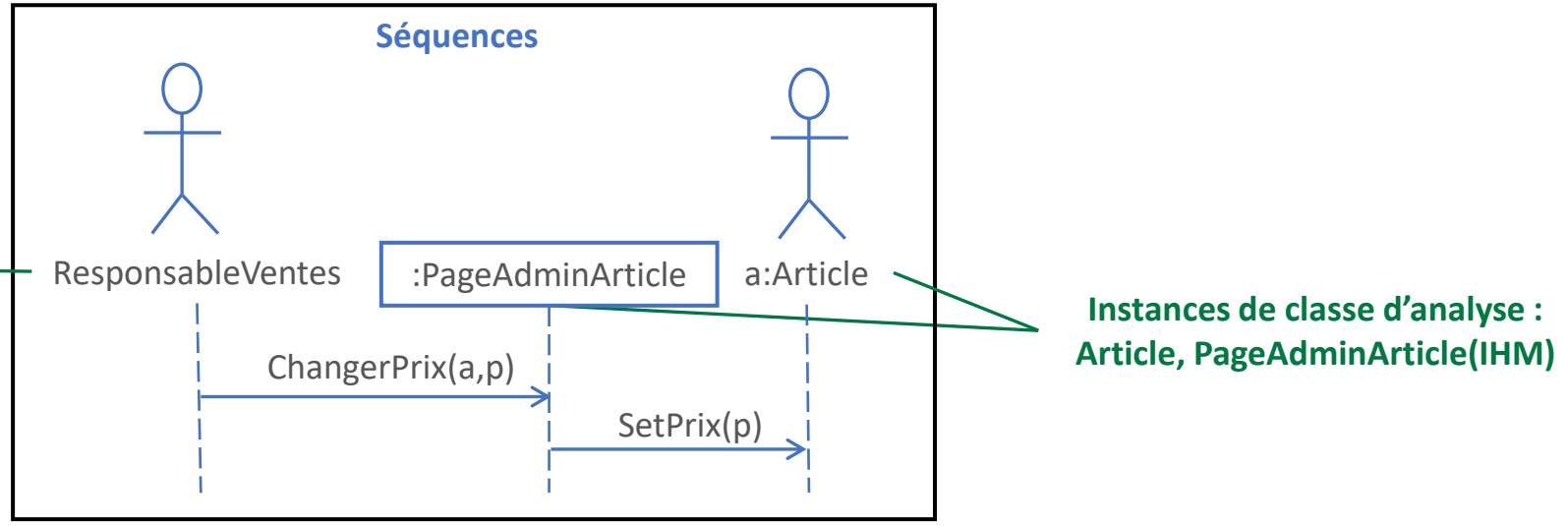
DS Boîte Noire - Exemple

- On considère un système de Librairie en ligne. On veut modéliser le DSS du cas d'utilisation « Chercher des ouvrages » déclenché par l'acteur principal : l'Internaute



DS Boîte Blanche

- Appelé aussi **Diagramme d'interaction ;**
- Permet d'attribuer précisément **les responsabilités de comportement aux classes d'analyse** du diagramme de classes ;
- Permet de **corriger le diagramme de classes d'analyse ;**
- Permet **de construire une première *version* de diagramme de classe de conception ;**
- **Il est centré sur une collection d'objets (instances) qui collaborent (échanges des messages).**



PARTIE 1

CHAPITRE 4

Modéliser les interactions d'objets d'un système à l'aide d'un diagramme de séquence

1. Rôle du diagramme de séquences
2. Distinguer le diagramme de séquence boîte noire du diagramme de séquence boîte blanche
3. **Diagramme de séquence comme boîte blanche**
4. Délimitation du diagramme de séquence
5. Types de messages
6. Fragments d'interactions combinés

DS Boîte Blanche (Diagramme d'interactions)

- Le diagramme de séquence Boîte Blanche est un **diagramme d'interaction mettant l'accent sur la chronologie de l'envoi des messages**.
- Il permet d'établir un lien entre les diagrammes de cas d'utilisation et les diagrammes de classes : il montre comment des objets (i.e. des instances de classes) communiquent pour réaliser une certaine fonctionnalité.
- Il apporte ainsi un aspect dynamique à la modélisation du système.

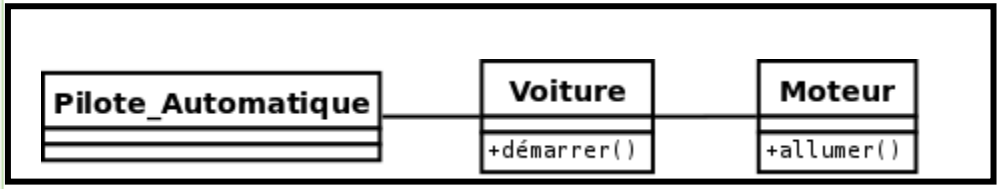


Figure 27 : Diagramme de classes d'un système de pilotage

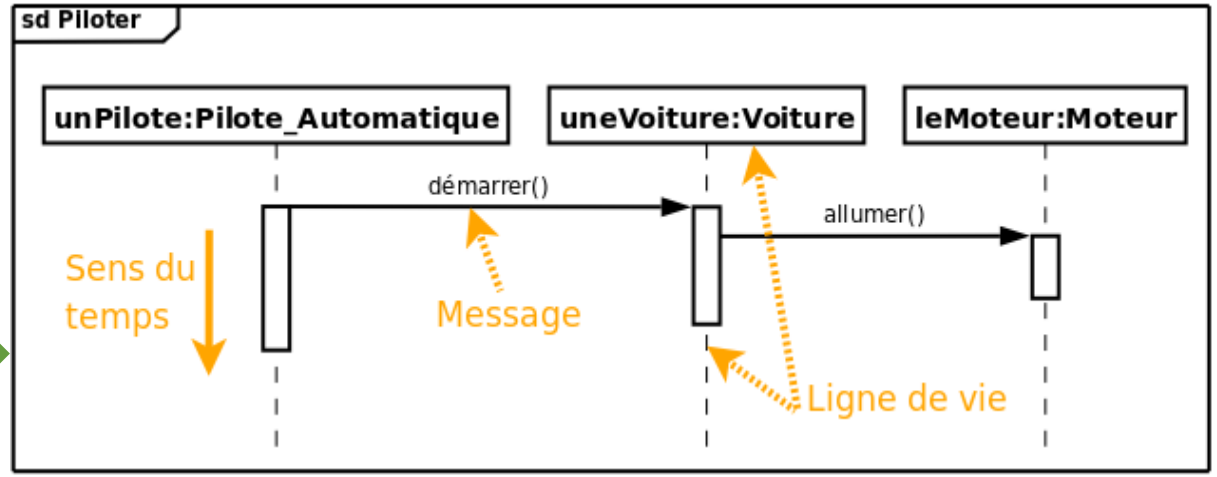
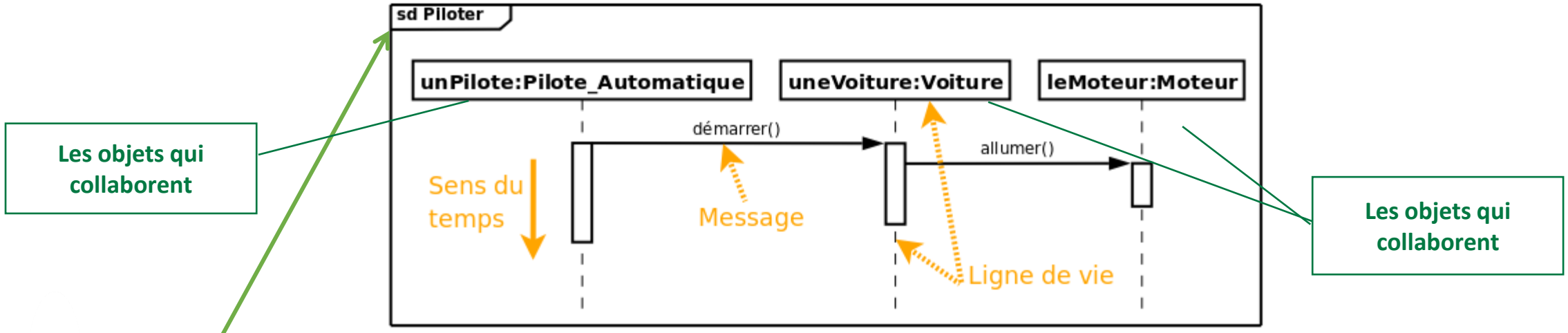


Figure 28 : Diagramme de séquence d'un système de pilotage

PARTIE 1

DS Boîte Blanche (Diagramme d'interactions)

- Pour produire un DS Boîte Blanche, il faut se focaliser sur un sous-ensemble d'éléments du système (objets, classes) et étudier leur façon d'interagir (collaborer) pour décrire un comportement particulier.
- L'interaction décrit **l'activité interne des objets d'une classe** (appelés lignes de vie), **par les messages qu'ils échangent.**



Remarque

- Un diagramme de séquence se représente par un rectangle contenant, dans le coin supérieur gauche, un pentagone accompagné du mot-clef **sd suivi du nom de l'interaction (Exemple : « sd Piloter »)**

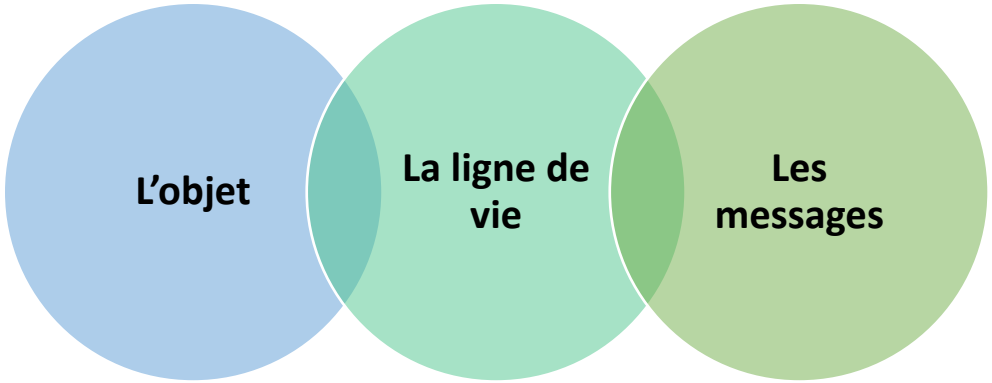
PARTIE 1

CHAPITRE 4

Modéliser les interactions d'objets d'un système à l'aide d'un diagramme de séquence

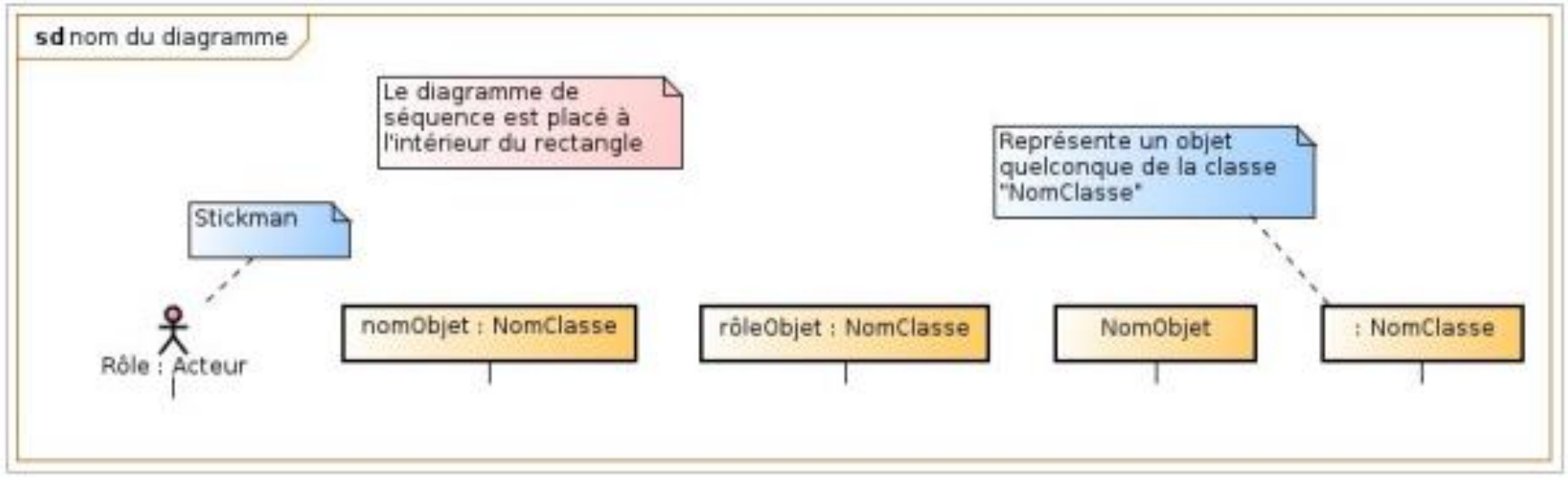
1. Rôle du diagramme de séquences
2. Distinguer le diagramme de séquence boîte noire du diagramme de séquence boîte blanche
3. Diagramme de séquence comme boîte blanche
4. **Délimitation du diagramme de séquence**
5. Types de messages
6. Fragments d'interactions combinés

Délimitation du DS



L'objet

- Dans un diagramme de séquence, l'objet à la même représentation que dans le diagramme des objets. C'est-à-dire un rectangle dans lequel figure le nom de l'objet.
- Le nom de l'objet est généralement souligné et peut prendre l'une des **quatre formes suivantes** :



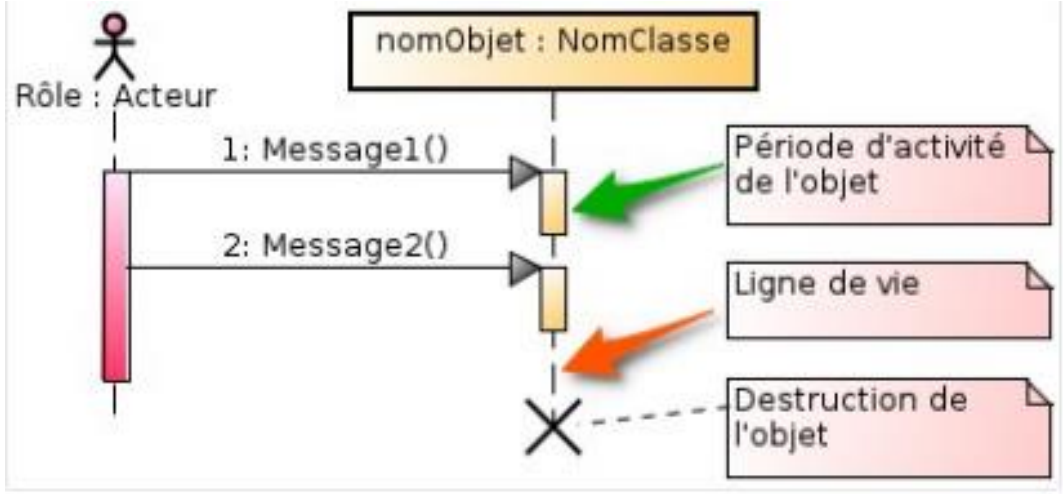
- Les diagrammes de séquences représentant **les échanges entre les objets mais aussi les échanges avec les acteurs**, on trouve aussi la représentation du **stickman** (qui peut être considéré comme un objet).

PARTIE 1

La ligne de vie

Comme il représente la dynamique du système, le diagramme de séquence fait entrer en action les instances de classes intervenant dans la réalisation d'un cas d'utilisation particulier.

- **A chaque objet est associé une ligne de vie** (en trait pointillés à la verticale de l'objet) qui peut être considéré **comme un axe temporel** (le temps s'écoule du haut vers le bas).
- La ligne de **vie indique les périodes d'activité de l'objet** (généralement, les moments ou l'objet exécute une de ces méthodes).
- Lorsque l'objet est détruit, la ligne de vie s'achève par une croix.



Les messages

- Un message définit une communication particulière entre des lignes de vie.
- Un message est une **communication d'un objet vers un autre objet**.
- La réception d'un message est considérée par l'objet récepteur comme un événement qu'il faut traiter (ou pas).

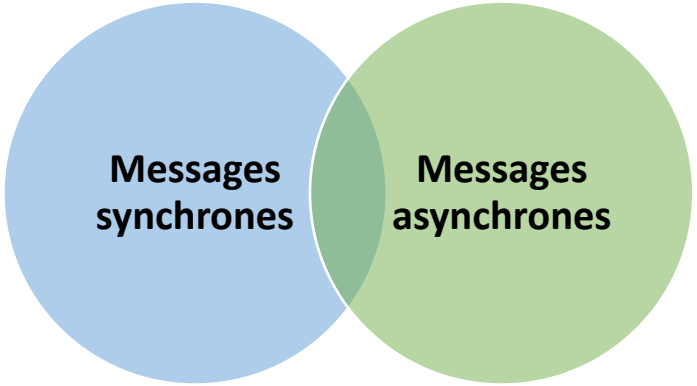
- Plusieurs types de messages existent, les plus communs sont :
  - **L'invocation d'une opération** : **message synchrone** (appel d'une méthode de l'objet cible).
  - **L'envoi d'un signal** : **message asynchrone** (typiquement utilisé pour la gestion événementielle).
  - La création ou la destruction d'une instance de classe au cours du cycle principal.

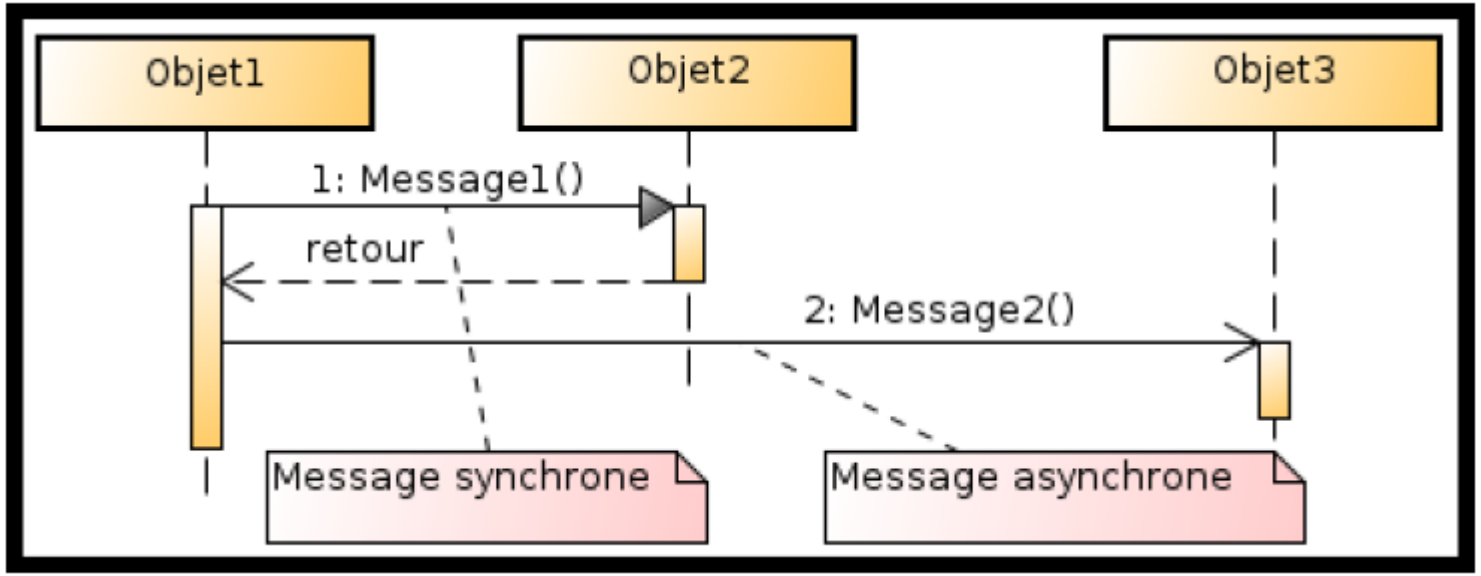
CHAPITRE 4

Modéliser les interactions d'objets d'un système à l'aide d'un diagramme de séquence

1. Rôle du diagramme de séquences
2. Distinguer le diagramme de séquence boîte noire du diagramme de séquence boîte blanche
3. Diagramme de séquence comme boîte blanche
4. Délimitation du diagramme de séquence
5. **Types de messages**
6. Fragments d'interactions combinés

Types de messages





Messages asynchrones

- Dans le cas d'un message asynchrone, **l'expéditeur n'attend pas la fin de l'activation de la méthode invoquée chez le destinataire**.
- Un message asynchrone peut être :
  - o **Un appel de méthode** :

    Fréquent dans un système **multi-threads** (multi-tâche). Ainsi, l'objet expéditeur n'étant pas bloqué pendant l'exécution de la méthode, il peut continuer ainsi à envoyer d'autres messages.

  - o **Un signal (cas le plus fréquent)** :

    L'objet expéditeur transmet juste une information à l'objet destinataire. Souvent, ce sont les acteurs ou les périphériques qui envoient des signaux, **typiquement utilisé dans la gestion événementielle d'une IHM graphique.**

- Graphiquement, un message asynchrone **se représente par une simple flèche en traits pleins** partant de la ligne de vie d'un objet expéditeur et allant vers celle de l'objet cible

Figure 29 : Représentation d'un message asynchrone

Messages synchrones

- La réception d'un message synchrone doit provoquer chez le destinataire le lancement d'une de ses méthodes (qui souvent porte le même nom que le message).
- **L'expéditeur du message reste bloqué pendant toute l'exécution de la méthode et attend donc la fin de celle-ci avant de pouvoir lancer un nouveau message.**
- **C'est le message le plus fréquemment utilisé.**

- Graphiquement, un message synchrone se représente par **une flèche en traits pleins et à l'extrémité pleine** partant de la ligne de vie d'un objet expéditeur et allant vers celle de l'objet cible. Ce message peut être suivi d'une réponse (message de retour) qui se représente par une flèche en pointillé.

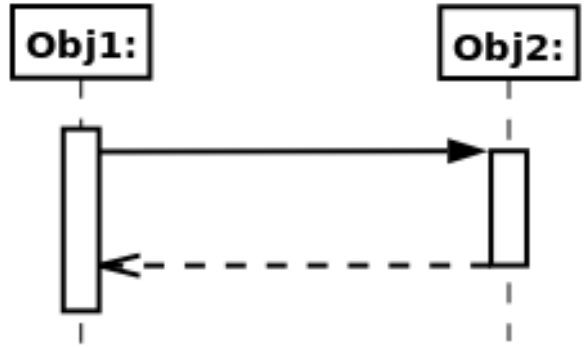
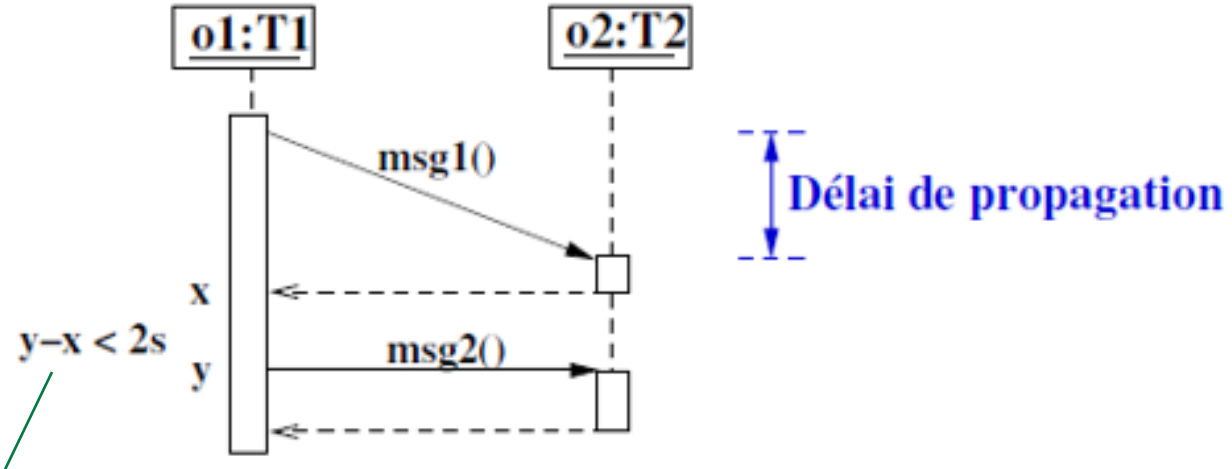


Figure 30 : Représentation d'un message synchrone

Messages synchrones – Contraintes temporelles



Messages synchrones – Syntaxe des messages et des réponses

- Dans la plupart des cas, la réception d'un message synchrone est suivie **de l'exécution d'une méthode d'une classe.** Cette méthode peut recevoir des arguments
- La syntaxe des messages permet de transmettre ces arguments : `message (p1,p2,…)`

- La syntaxe de réponse à un message est la suivante :

```
[<attribut> = ] message [ : <valeur_de_retour>]
```

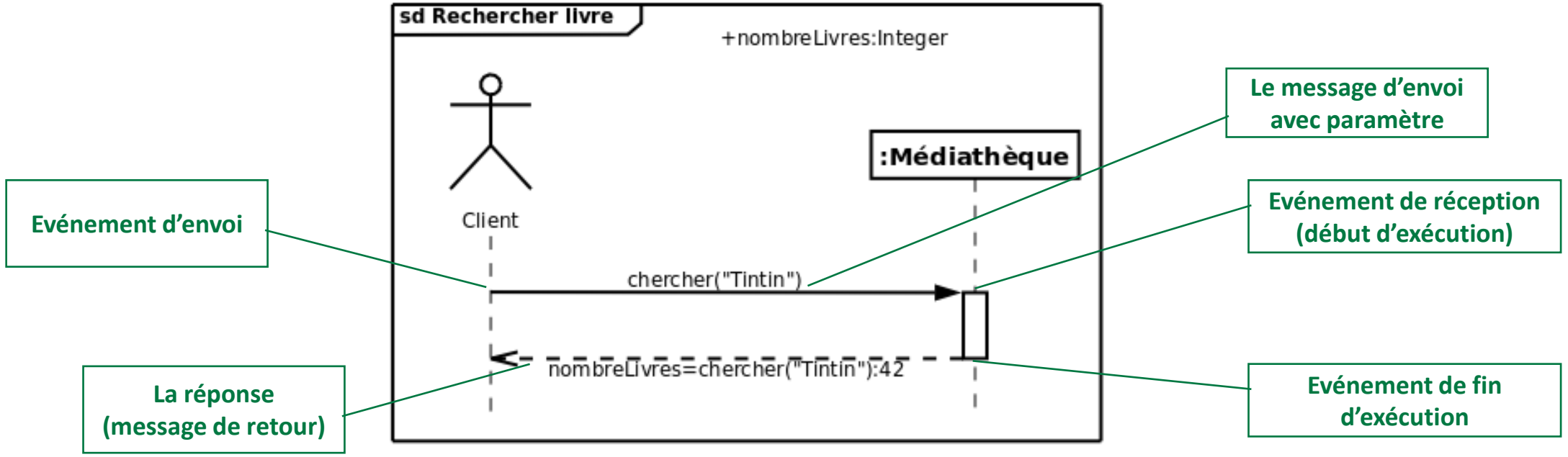


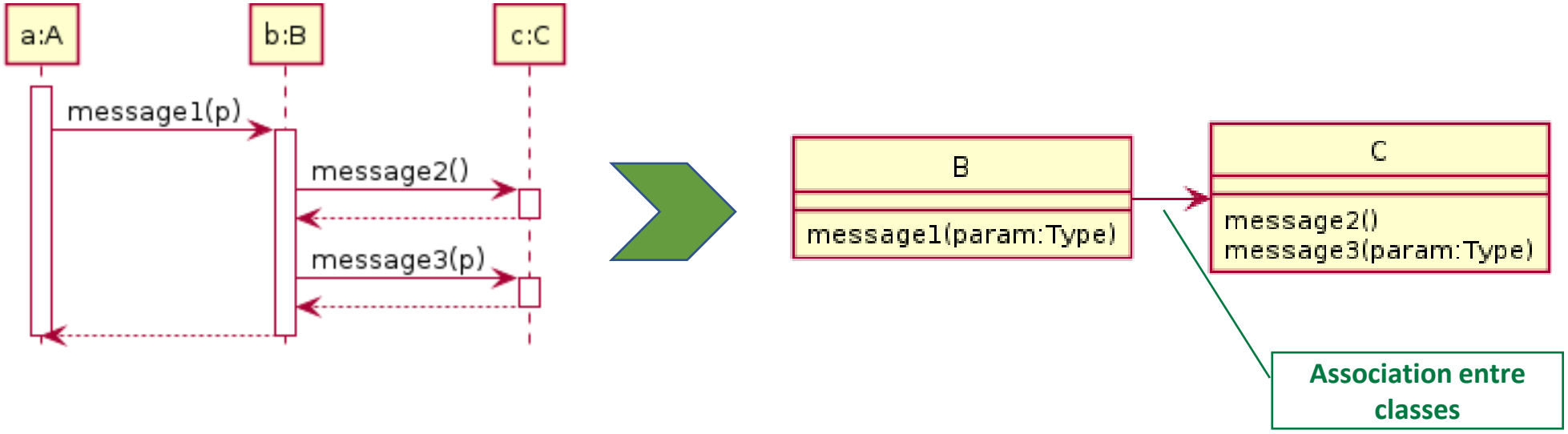
Figure 31 : Exemple d'exécution d'une méthode avec une réponse

PARTIE 1

Messages synchrones et diagramme de classe

- Les méthodes correspondant aux messages synchrones doivent être définies dans un diagramme de classes.

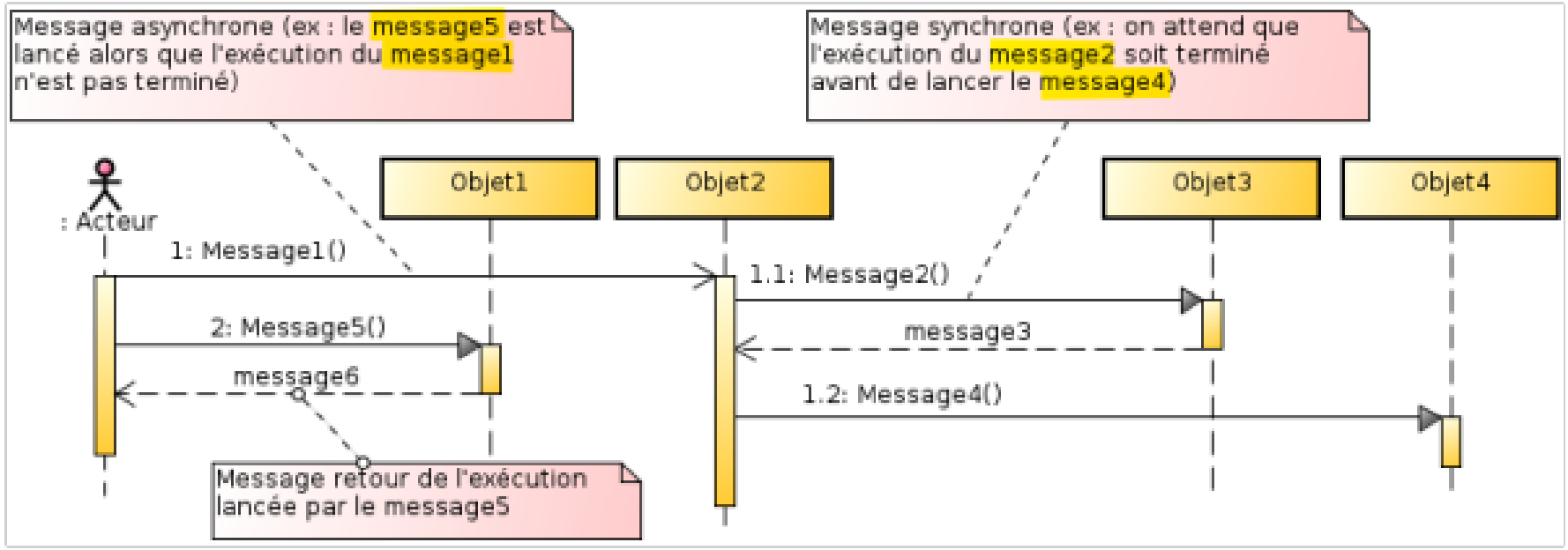
- **Les méthodes sont définies dans la classe du récepteur**, et pas de l'émetteur du message.



PARTIE 1

PARTIE 1

Messages synchrones et asynchrones – Exemple



Autres types – Message perdu et trouvé

- **Le message perdu** est un message dont on connait l'émetteur mais pas le récepteur. Il est représenté par une flèche partant de la ligne de vie d'un élément vers un disque noir.
- Le message perdu pouvant être, à l'origine, synchrone ou asynchrone,
- **Cette sorte de message permet de modéliser, par exemple, les scenarii de pertes de message sur un réseau.**

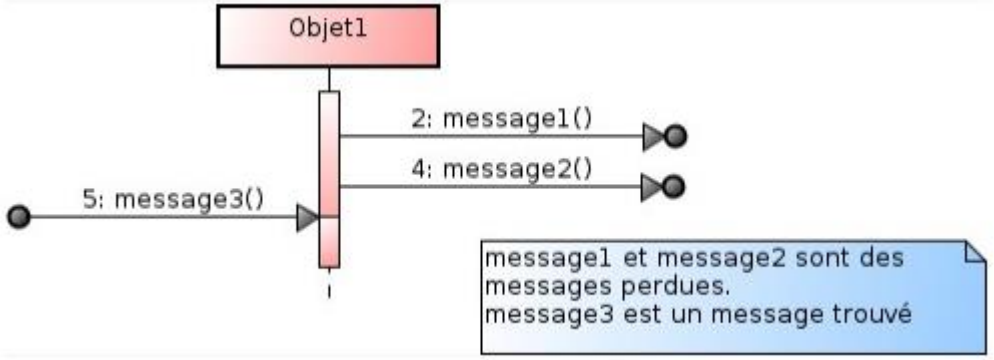


Figure 32 : Représentation d'un message trouvé et d'un message perdu

- **Le message trouvé** est un message dont on connait le destinataire mais pas l'émetteur. Il est représenté par une flèche partant d'un disque noir vers la ligne de vie d'un élément. **Ce message peut être utilisé pour modéliser le comportement d'un élément suite à la réception d'un message d'exception.**

Autres types – Messages de création et destruction d'instance

- Une séquence peut aussi contenir la **création** ou la **destruction** d'un objet :

  o La création d'un objet est matérialisée par un message spécifique, appel d'un **constructeur**, généralement accompagné du stéréotype `« create »` qui pointe sur le début (le sommet) de la ligne de vie de l'objet créé (Le rectangle de l'instance de la classe est alors surbaissée).

  o La destruction d'un objet est représentée par une croix à la fin de sa ligne de vie. Souvent l'objet est détruit suite à la réception d'un message mais ce n'est pas obligatoire. Dans ce cas là, il porte le stéréotype `« destroy »`.

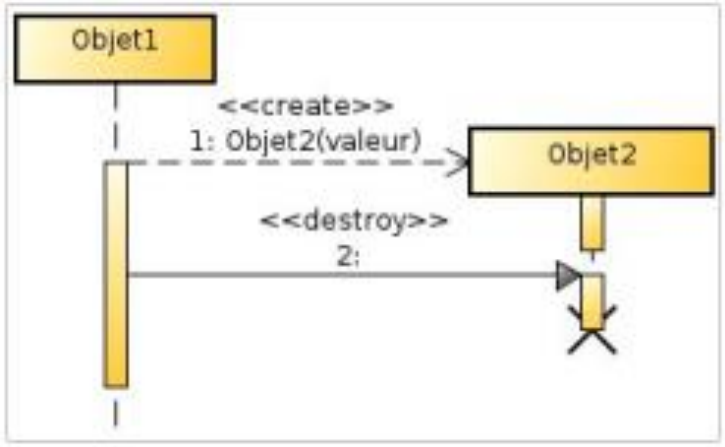


Figure 33 : Représentation d'un message de création et destruction d'instance

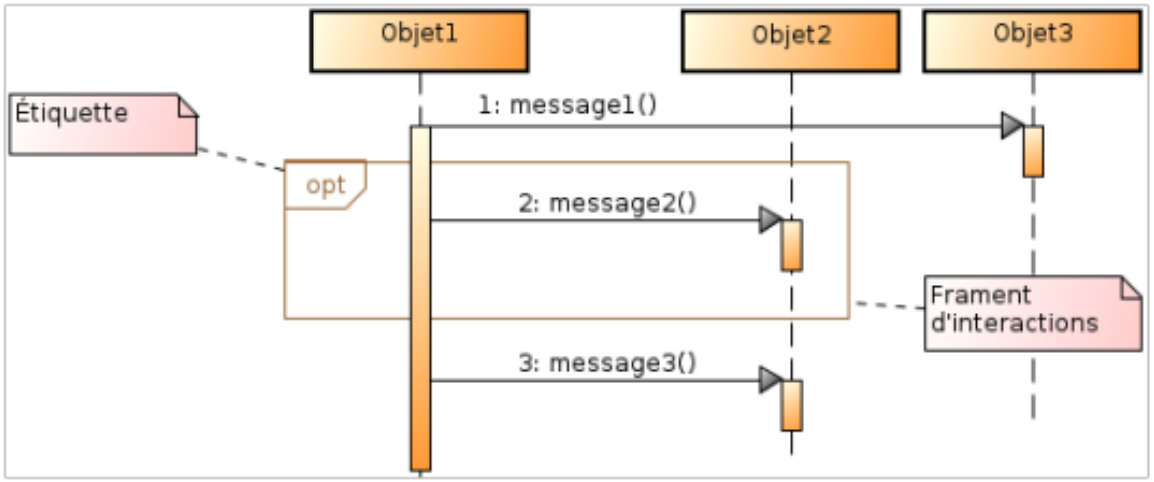
CHAPITRE 4

Modéliser les interactions d'objets d'un système à l'aide d'un diagramme de séquence

1. Rôle du diagramme de séquences
2. Distinguer le diagramme de séquence boîte noire du diagramme de séquence boîte blanche
3. Diagramme de séquence comme boîte blanche
4. Délimitation du diagramme de séquence
5. Types de messages
6. **Fragments d'interactions combinés**

Fragment combiné

- Un fragment d'interactions est une partie du diagramme de séquence (délimitée par un rectangle) associée à une étiquette (dans le coin supérieur gauche).
- L'étiquette **contient un opérateur d'interaction** qui permet de décrire des **modalités d'exécution des messages** à l'intérieur du cadre.



- Les **opérandes** d'un **opérateur d'interaction** sont séparés par une ligne pointillée.
- Les conditions de choix des **opérandes** (**éventuels**) sont données par des **expressions booléennes** entre crochets ([ ]).

PARTIE 1

Fragment combiné

- Dans l'étiquette du fragment, figure le **type de la combinaison (appelé opérateur d'interaction).**
- Voici une liste des opérateurs (non complète) :

Opérateurs de choix et de boucle	Opérateurs contrôlant l'envoi en parallèle de messages	Opérateurs contrôlant l'envoi de messages
• **Opérateur d'option « opt »** • **Opérateur d'alternative « alt »** • **Opérateur de boucle « loop »** • **Opérateur d'interruption « break »**	• **Opérateur Parallèle « par »**	• **Opérateur d'omission « ignore »** • **Opérateur d'affirmation « assert »**

Fragment d'interaction avec l'opérateur d'option « opt »

- L'opérateur *option*, ou ***opt***, **comporte une opérande et une condition de garde associée**.
- Le sous-fragment s'exécute si la condition de garde est vraie et ne s'exécute pas dans le cas contraire.

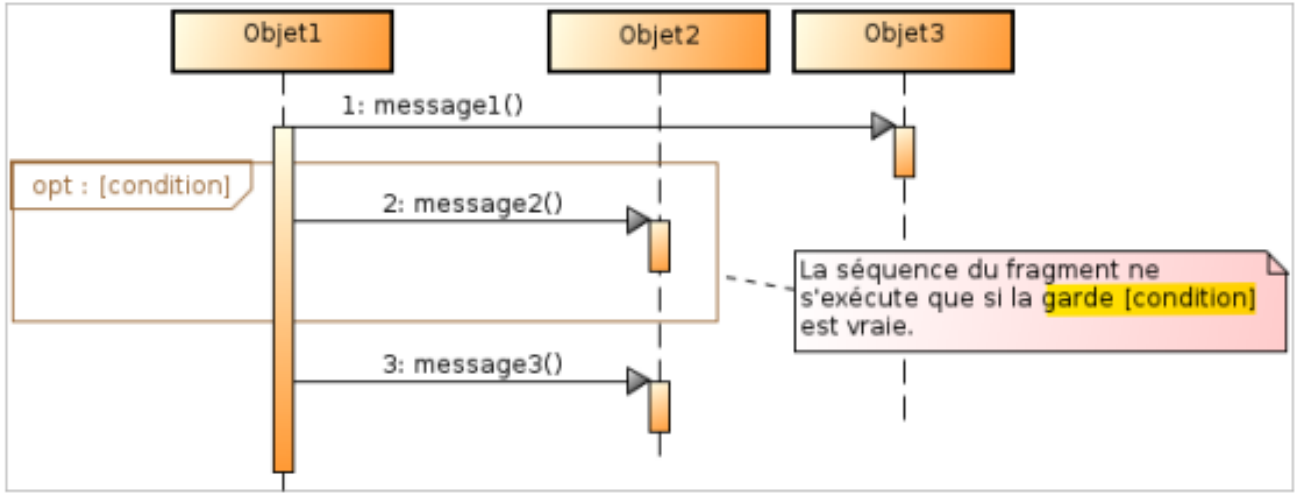


Figure 34 : Représentation d'un fragment avec l'opérateur opt

Fragment d'interaction avec l'opérateur d'alternative « alt »

- L'opérateur *alternative*, ou *alt*, est un **opérateur conditionnel possédant plusieurs opérandes** séparés par des pointillés.
- **Equivalent à une exécution à choix multiple (condition *switch*).**
- **Chaque opérande détient une condition de garde**.
- Seul le sous-fragment dont la condition est vraie est exécuté.
- La condition ***else*** est exécutée que si aucune autre condition n'est valide.

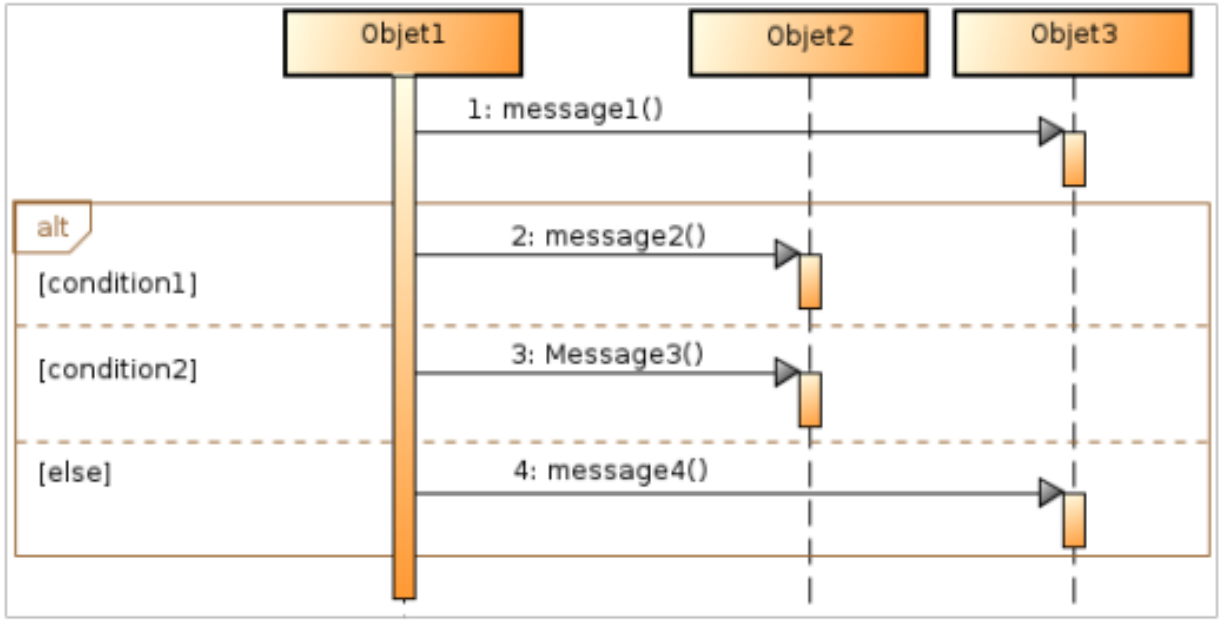


Figure 35 : Représentation d'un fragment avec l'opérateur alt

Fragment d'interaction avec l'opérateur de boucle « loop »

- L'opérateur de **boucle** (***loop***) exécute la séquence contenue dans le fragment **tant que** la garde qui lui est associée est vraie.
- La garde s'écrit de la façon suivante :

```
loop [min, max, condition]
```

- Chaque paramètre (min, max et condition) est **optionnel**.
- Le contenu du fragment est exécuté **min** fois, puis continue à s'exécuter tant que la condition et que le nombre d'exécution de la boucle ne dépasse pas **max** fois.
- **Exemple** de gardes :
  - `loop[3]` →La séquence s'exécute **3** fois.
  - `Loop[1, 3, code=faux]` → La séquence s'exécute **1** fois puis un maximum de **2** autres fois si **code=faux**.

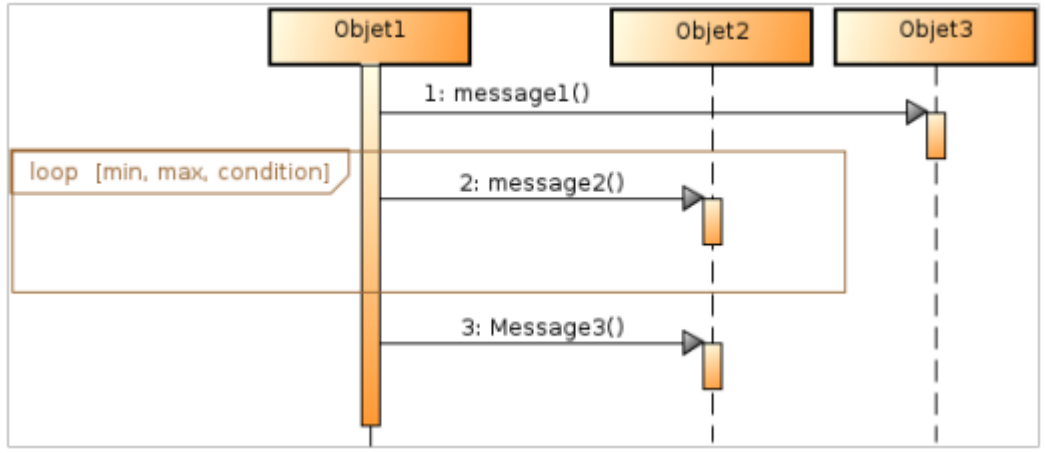


Figure 36 : Représentation d'un fragment avec l'opérateur loop

PARTIE 1

Fragment d'interaction avec l'opérateur Parallèle « par »

- Un fragment d'interaction avec l'opérateur de **traitements parallèles** (***par***) contient au moins deux sous fragments (**opérandes**) séparés par des pointillés qui s'exécutent simultanément (**traitements concurrents**).

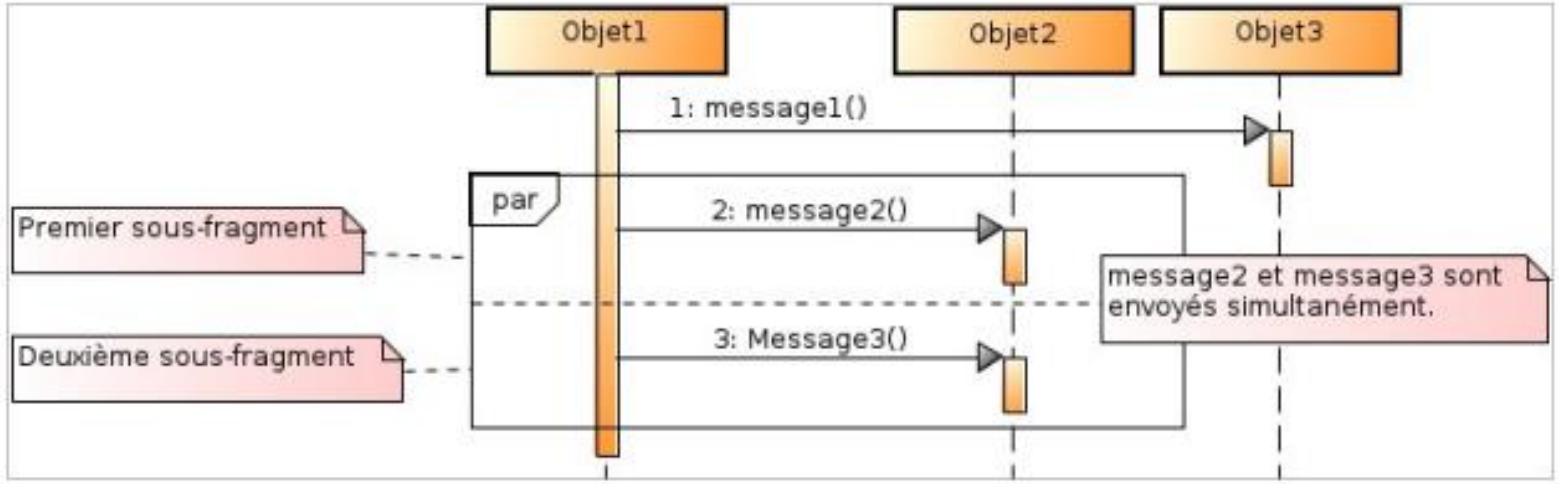


Figure 37 : Représentation d'un fragment avec l'opérateur par

PARTIE 1

Fragment d'interaction avec l'opérateur d'interruption « break »

- L'opérateur « ***break*** » est utilisé dans les fragments combinés qui représentent **des scénarios d'exception** en quelque sorte. Les interactions de ce fragment seront exécutées à la place des interactions décrites en dessous.
- Il y a donc une notion d'interruption du flot « normal » des interactions.

- **Exemple** : Lorsque le distributeur lui demande son code, l'utilisateur peut choisir de rentrer son code ou de consulter l'aide. S'il choisit de consulter l'aide (en appuyant sur la touche F1), **le flot d'interaction relatif à la saisie du code est interrompu**. **Les interactions de l'opérateur break sont « exécutées ».**

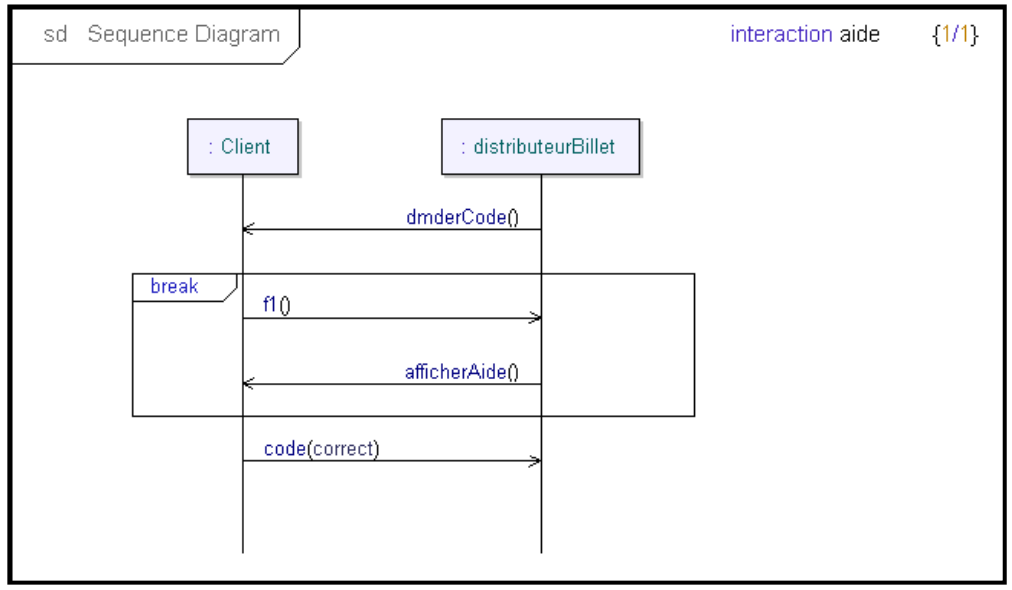


Figure 38 : Exemple de représentation d'un fragment avec l'opérateur break

Fragment d'interaction avec l'opérateur d'omission « ignore »

- L'opérateur « ***Ignore*** » (ignorer) indique qu'il existe des **messages qui peuvent ne pas être présents** dans le fragment combiné. **Ce sont des messages facultatifs**.

- **Exemple** : **Le message connexionEtablie est spécifié comme ignoré**. On considère que la séquence est tout de même correcte si jamais lors de l'exécution ce message n'apparaissait pas.

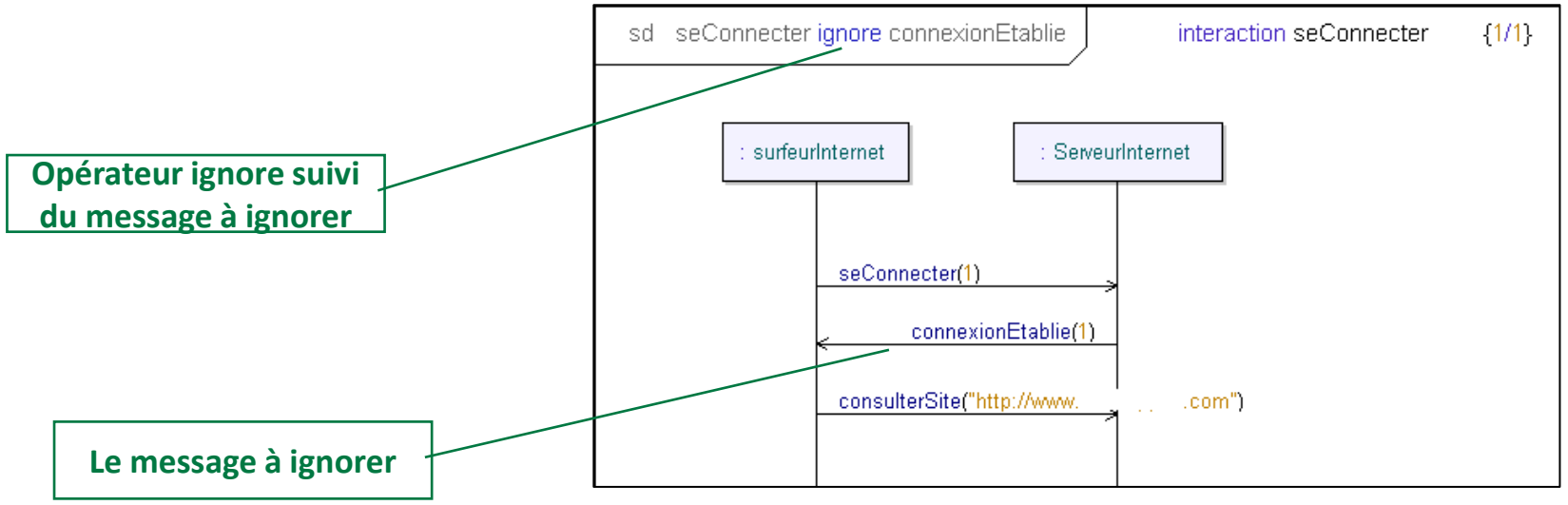


Figure 39 : Exemple de représentation d'un fragment avec l'opérateur ignore

Fragment d'interaction avec l'opérateur d'affirmation « assert »

- L'opérateur « *Assertion* » est noté « *assert* ».
- La séquence décrite dans l'opérande désigne **l'unique séquence possible**.
- **Toutes les autres séquences possibles sont des séquences invalides**.

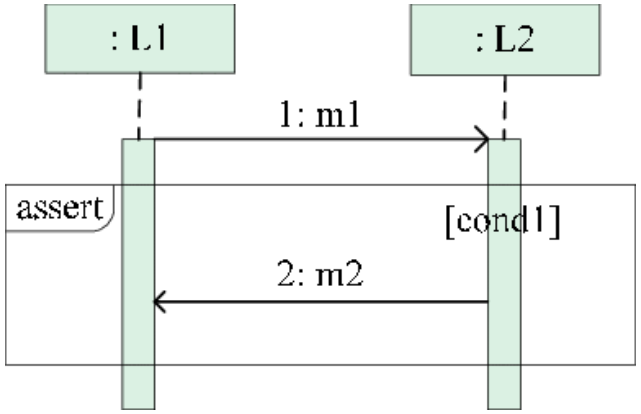
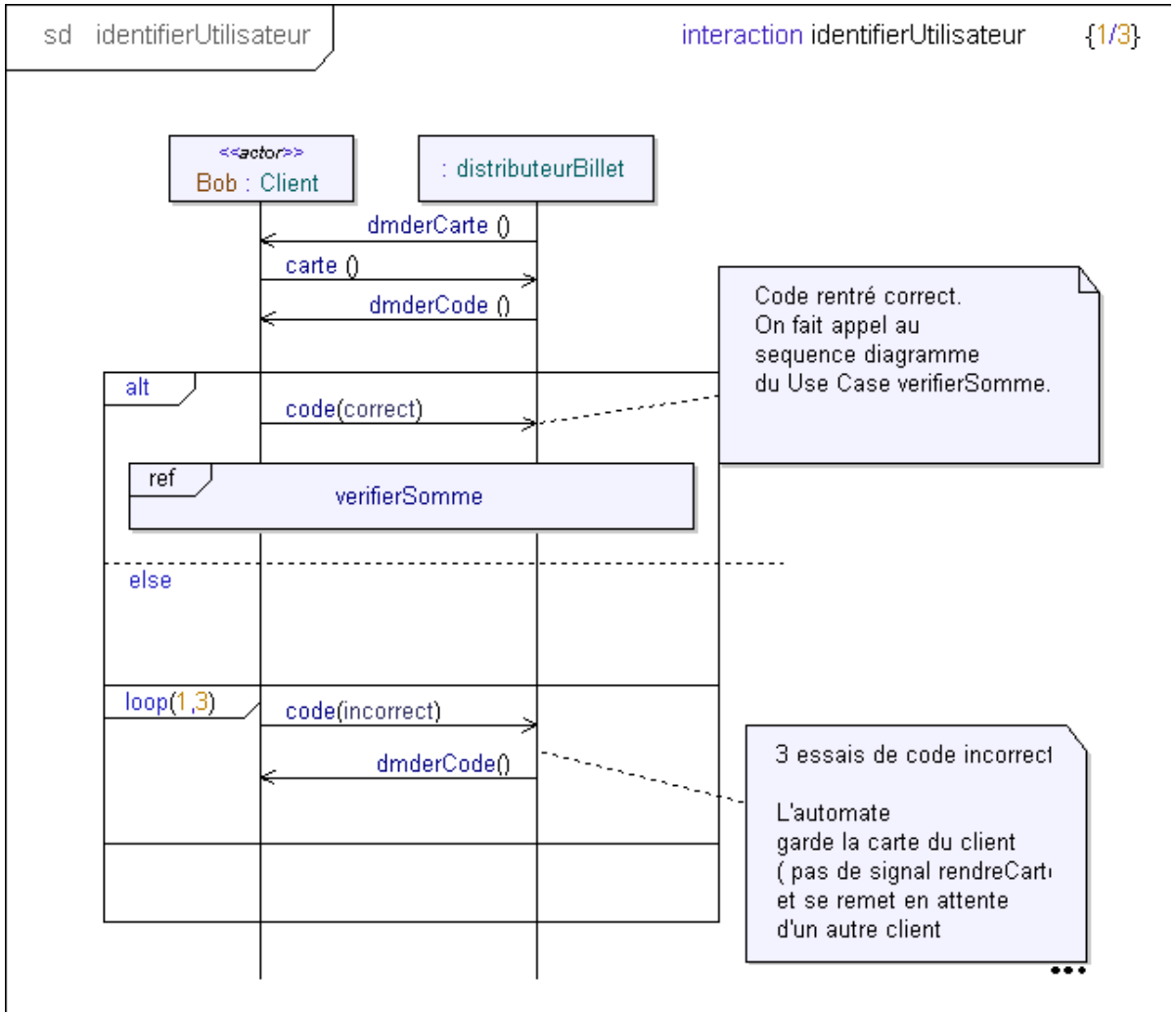


Figure 40 : Exemple de représentation d'un fragment avec l'opérateur assert

Combiner les opérateurs

- Les fragments combinés et leurs opérateurs peuvent être **combinés/mixés en vue de décrire des comportements complexes.**

- **Exemple** : Lorsque l'utilisateur se trompe trois fois de code, la carte est gardée et le distributeur se remet en mode d'attente d'une carte.



PARTIE 1

CHAPITRE 5

Elaborer des diagrammes UML à l'aide d'un outil de modélisation

Ce que vous allez apprendre dans ce chapitre :

- Environnement de travail et fonctionnalités
- Création et édition d'un diagramme de classe
- Sauvegarde et impression
- Génération du code source
- Création et édition d'un diagramme des cas d'utilisation
- Création et édition d'un diagramme de séquences

5 heures

CHAPITRE 5

Elaborer des diagrammes UML à l'aide d'un outil de modélisation

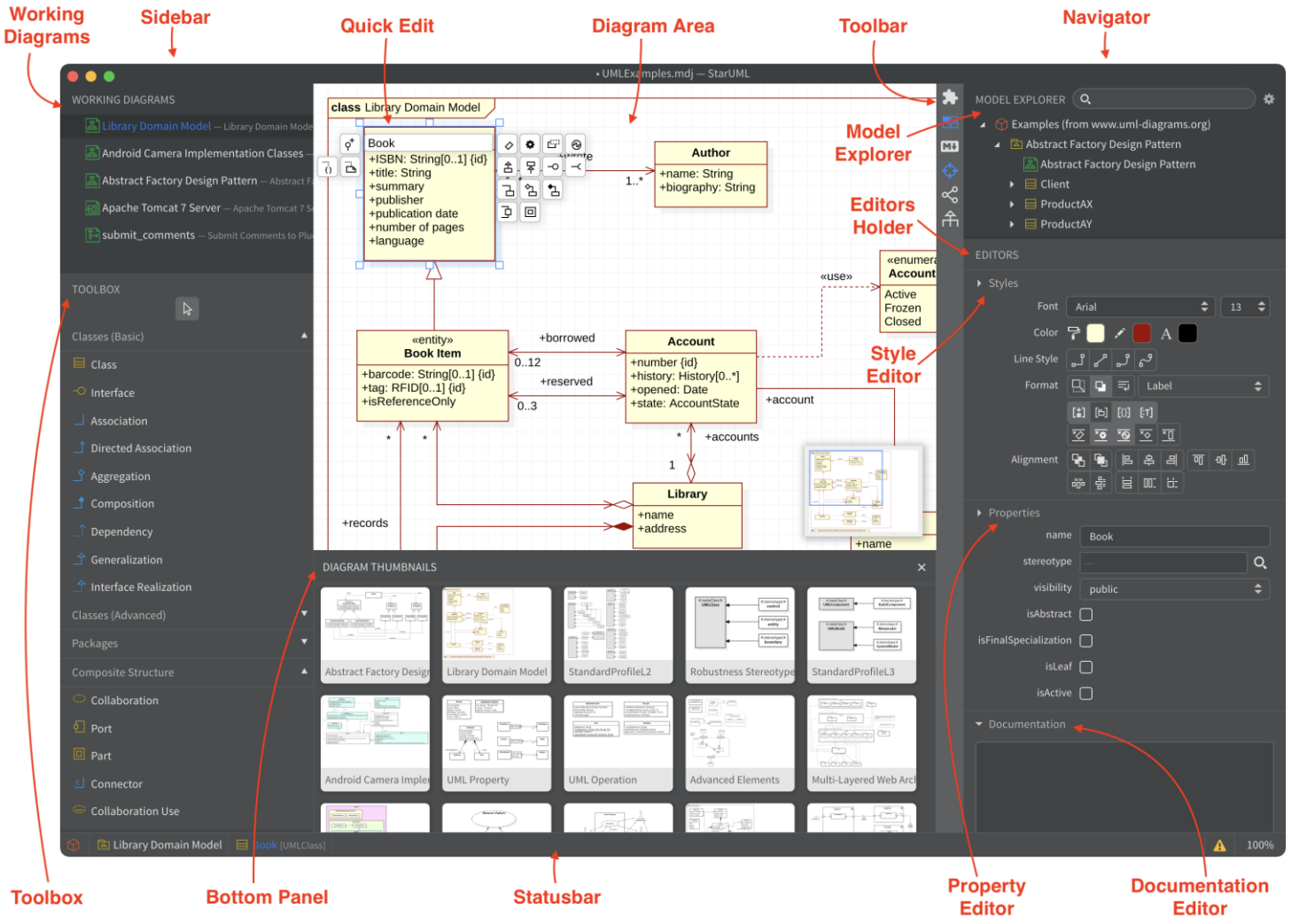
1. **Environnement de travail et fonctionnalités**
2. Création et édition d'un diagramme de classe
3. Sauvegarde et impression
4. Génération du code source
5. Création et édition d'un diagramme des cas d'utilisation
6. Création et édition d'un diagramme de séquences

Environnement de travail : StarUML

- StarUML est un **logiciel de modélisation UML** qui est entré récemment dans le monde de l'OpenSource.
- Ecrit en Delphi, il est modulaire et propose plusieurs générateurs de code.
- **Editeur depuis 2014 : L**a société coréenne *MKLabs Co. Ldt* – http://staruml.io
- La dernière version (5.0.2 datée de Juin 2022) gère **l'ensemble des diagrammes définis par UML 2**, ainsi que plusieurs diagrammes **SysML** (Diagramme de besoins, …), les **organigrammes**, les **diagrammes de flux de données**, et les **diagrammes entité-association**.

PARTIE 1

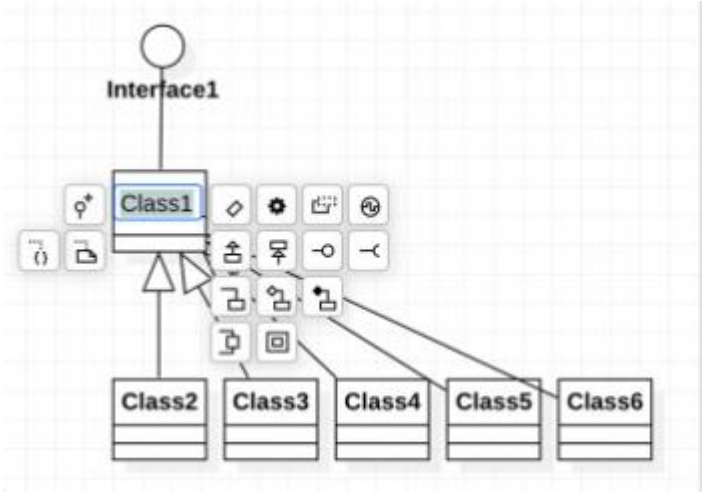
StarUML



StarUML – Fonctionnalités clés

- **Compatible avec le méta-modèle et les diagrammes standards UML 2** : **Classes**, Objets, **Cas d'utilisation**, Composants, Déploiemet, Structure Composite, **Séquence**, Communication, Etat-Transitions, Activité, etc.

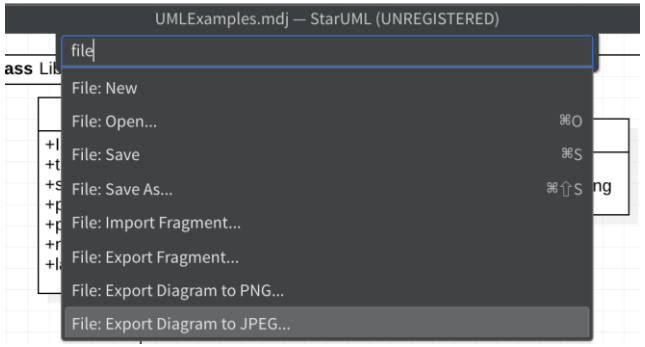
- Supporte la modélisation avec d'autres types de diagrammes (SysML, Entités-Relations, etc.)
- Personnaliser les profils utilisateurs, thèmes
- Multi-plateformes : inclus MacOS, Windows et Linux
- Les mises à jour sont installés automatiquement par le logiciel (MacOS, Windows)
- Il permet également d'exporter des fragments de modèles dans des fichiers distincts avec l'extension (*mdf, PNG, JPEG, etc.)* et d'importer ceux-ci par la suite.
- permet une **modélisation rapide** grâce à des raccourcis (*Quick Edit*)



PARTIE 1

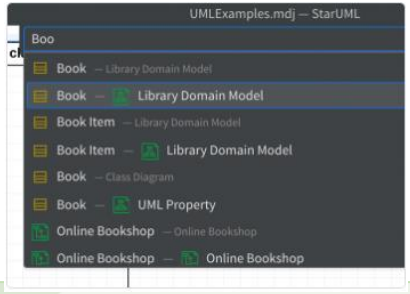
StarUML – Fonctionnalités clés

- **La palette de commandes** permet de rechercher et d'exécuter des commandes dans StarUML ainsi que des extensions installées.



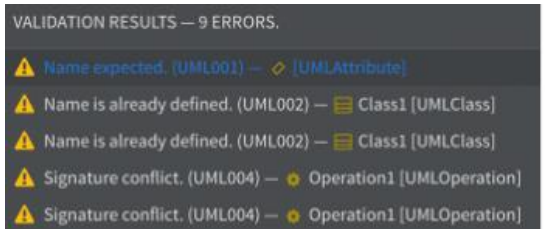
- supporte **une génération de code** pour différents langages de programmation (PHP, Java, C#, C++, Python) via des extensions open-source

- Possibilité de **la recherche rapide** pour trouver des modèles, des vues et des diagrammes et de sélectionner rapidement l'élément.

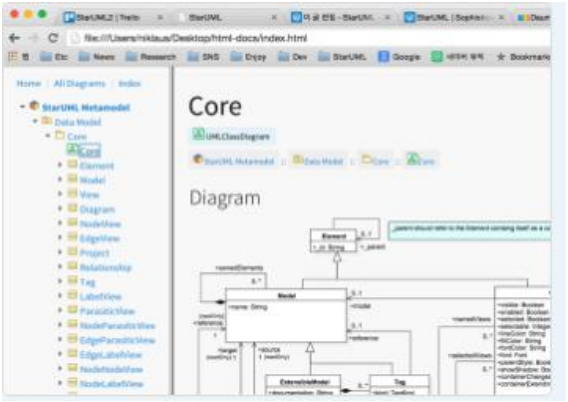


StarUML – Fonctionnalités clés

- De nombreuses **règles de validation de modèle** sont définies et vérifiées de manière asynchrone à l'enregistrement et à l'ouverture de fichiers de modèle.



- Possibilité de **publier les modèles dans des documents HTML, PDF**



CHAPITRE 5

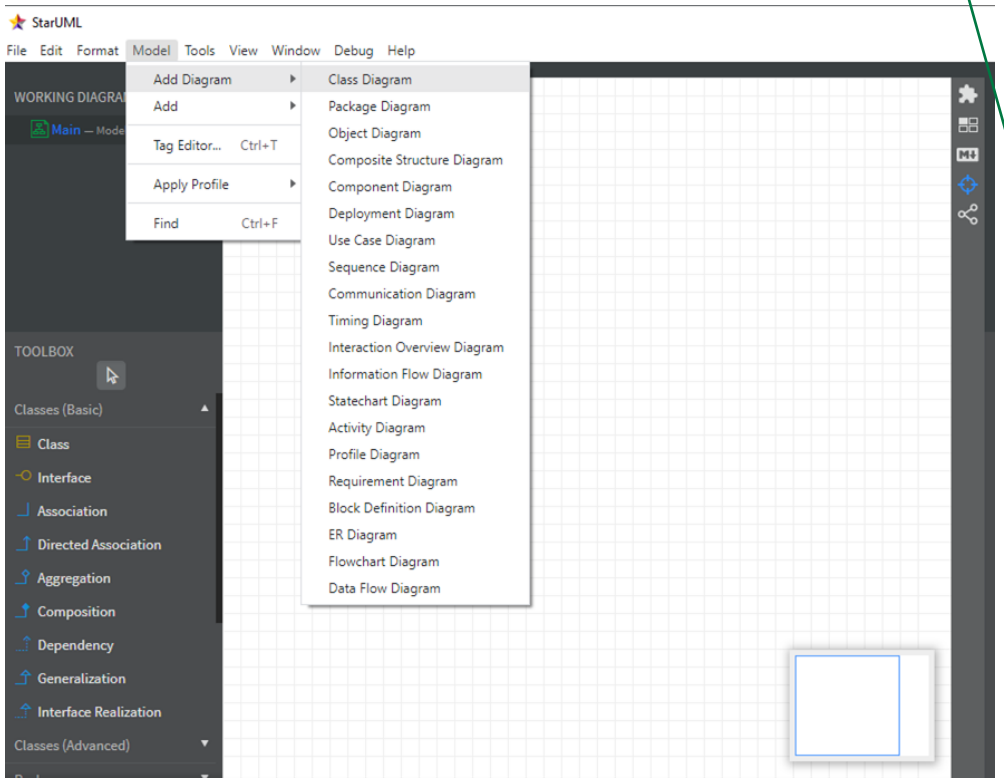
Elaborer des diagrammes UML à l'aide d'un outil de modélisation

1. Environnement de travail et fonctionnalités
2. **Création et édition d'un diagramme de classe**
3. Sauvegarde et impression
4. Génération du code source
5. Création et édition d'un diagramme des cas d'utilisation
6. Création et édition d'un diagramme de séquences

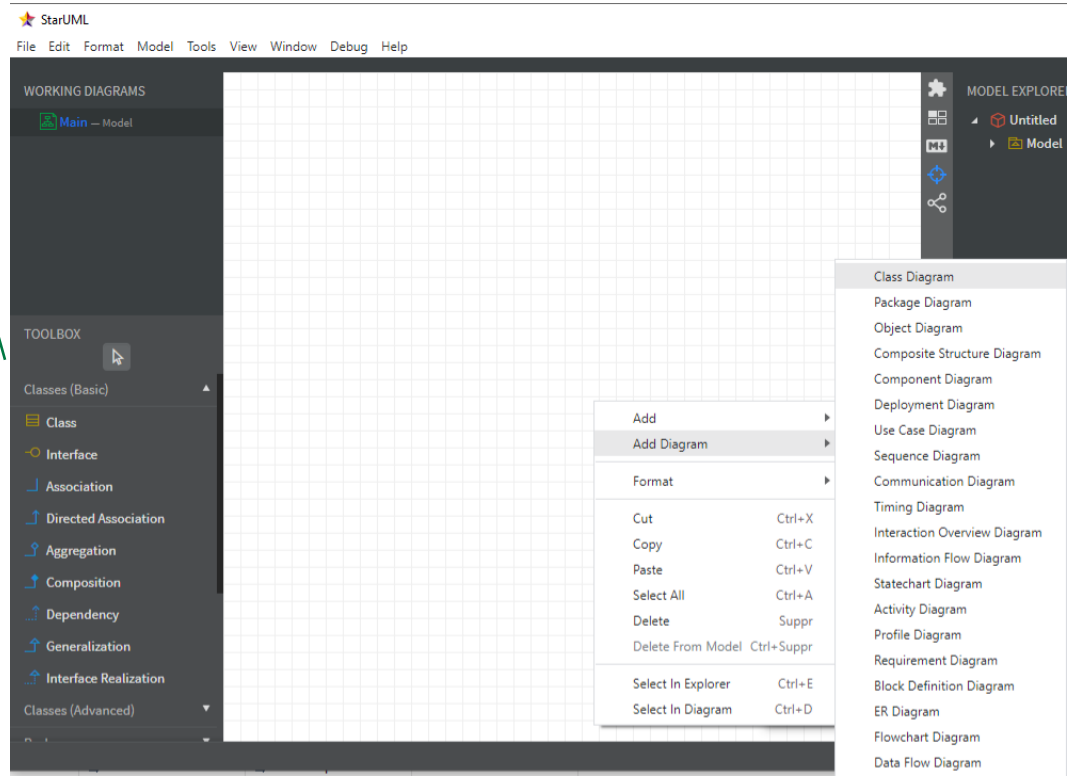
PARTIE 1

StarUML – Diagramme de classe

Boîte à Outils



Barre de menus



Menu Contextuel

StarUML – Diagramme de classe - Eléments

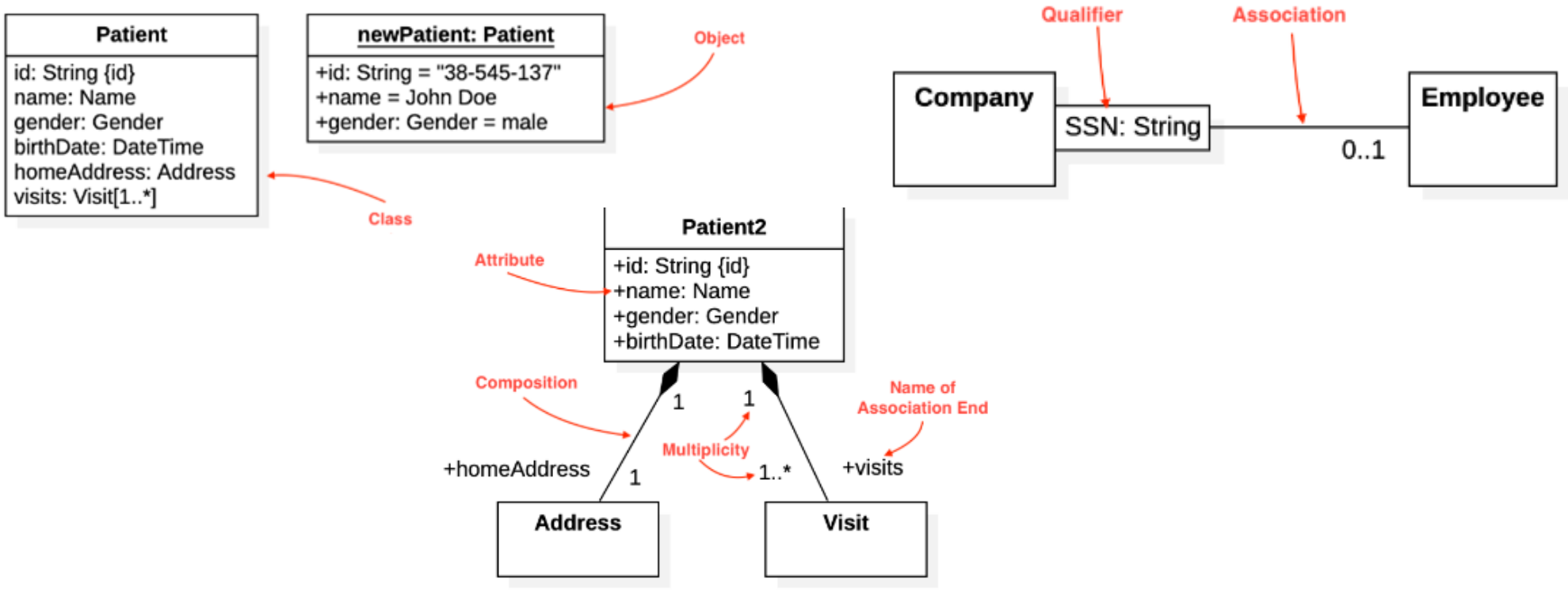


Figure 41 : Exemples d'éléments à représenter dans le diagramme de classes

PARTIE 1

CHAPITRE 5

Elaborer des diagrammes UML à l'aide d'un outil de modélisation

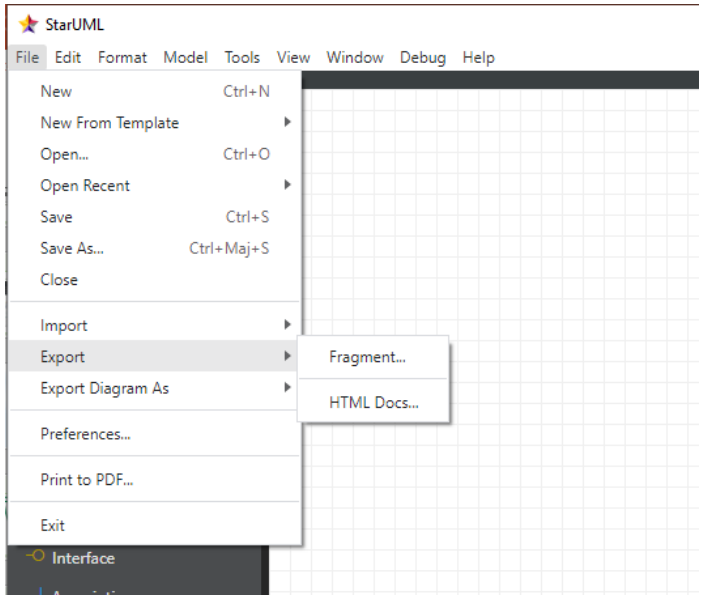
1. Environnement de travail et fonctionnalités
2. Création et édition d'un diagramme de classe
3. **Sauvegarde et impression**
4. Génération du code source
5. Création et édition d'un diagramme des cas d'utilisation
6. Création et édition d'un diagramme de séquences

PARTIE 1

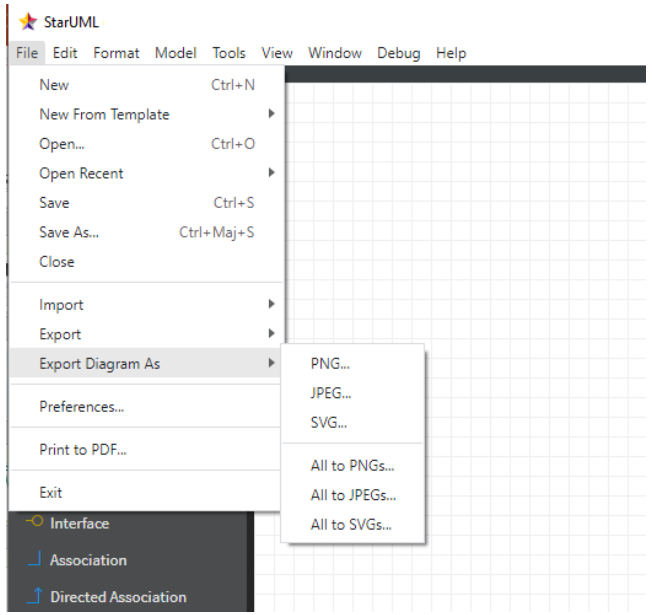
Sauvegarde

- StarUML gère les modèles dans des fichiers de projets ayant pour extension ***.mdj***. Il s'agit de fichiers texte au format JSON.

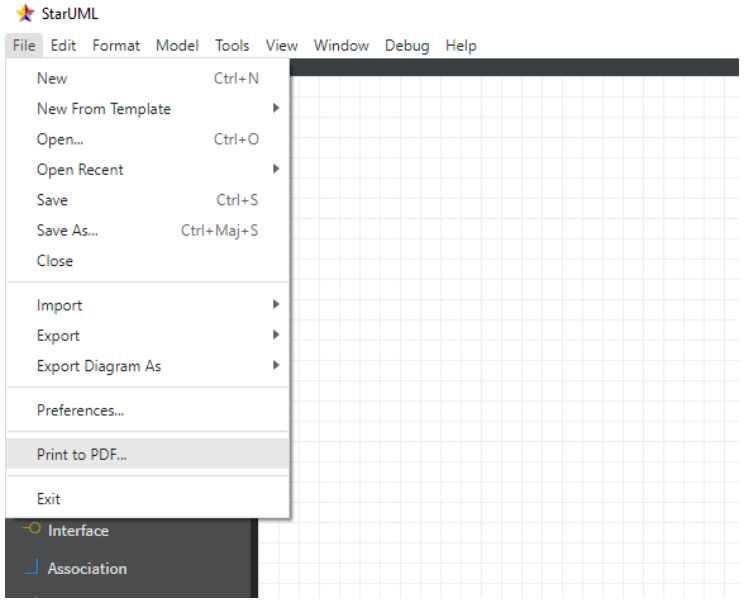
Impression



Export en fragment, HTML



Export en images



Impression en PDF

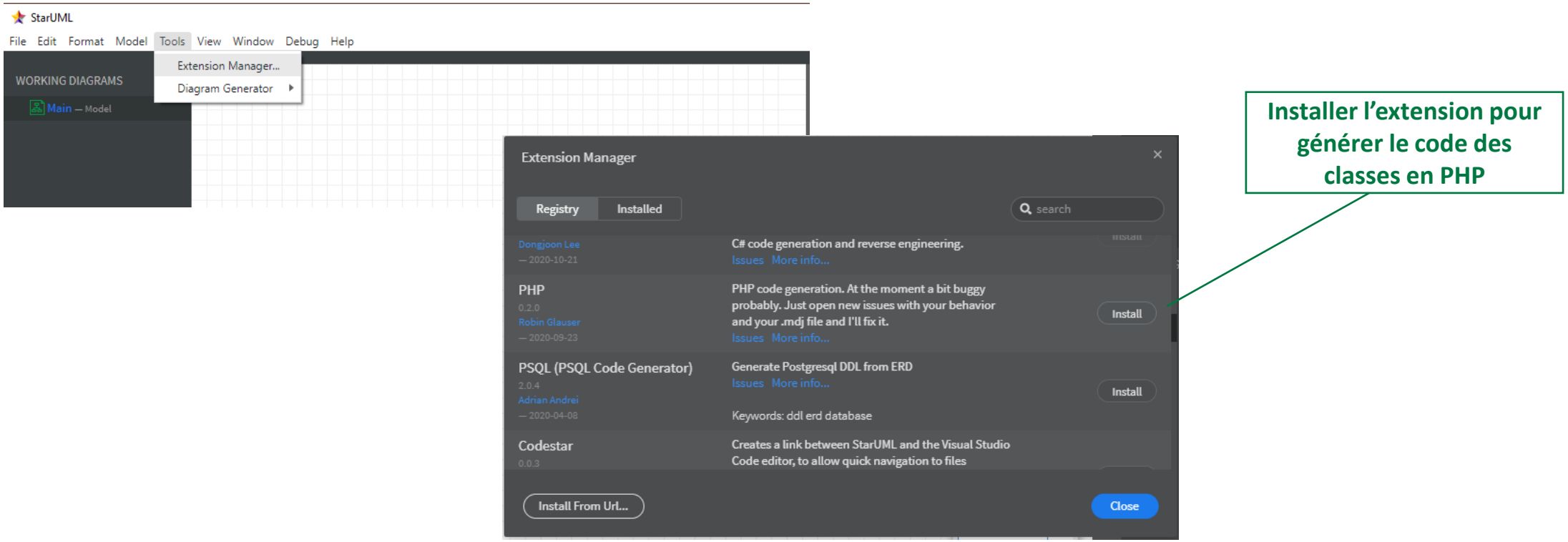
CHAPITRE 5

Elaborer des diagrammes UML à l'aide d'un outil de modélisation

1. Environnement de travail et fonctionnalités
2. Création et édition d'un diagramme de classe
3. Sauvegarde et impression
4. **Génération du code source**
5. Création et édition d'un diagramme des cas d'utilisation
6. Création et édition d'un diagramme de séquences

Génération du code Source

- StarUML supporte **une génération de code** pour différents langages de programmation (PHP, Java, C#, C++, Python) **via des extensions open-source**



Installer l'extension pour générer le code des classes en PHP

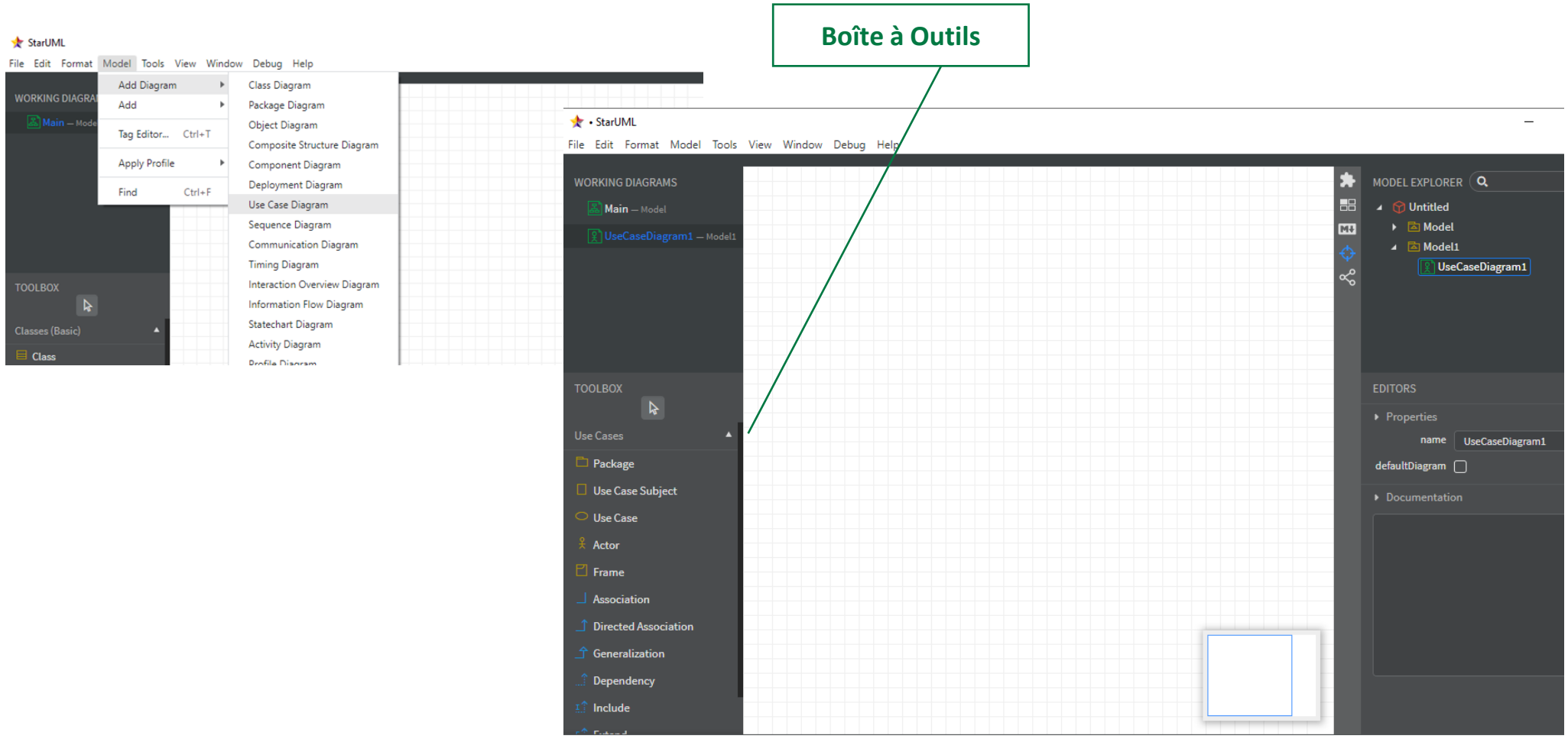
PARTIE 1

CHAPITRE 5

Elaborer des diagrammes UML à l'aide d'un outil de modélisation

1. Environnement de travail et fonctionnalités
2. Création et édition d'un diagramme de classe
3. Sauvegarde et impression
4. Génération du code source
5. **Création et édition d'un diagramme des cas d'utilisation**
6. Création et édition d'un diagramme de séquences

StarUML – Diagramme des cas d'utilisation



PARTIE 1

StarUML – Diagramme des cas d'utilisation - Eléments

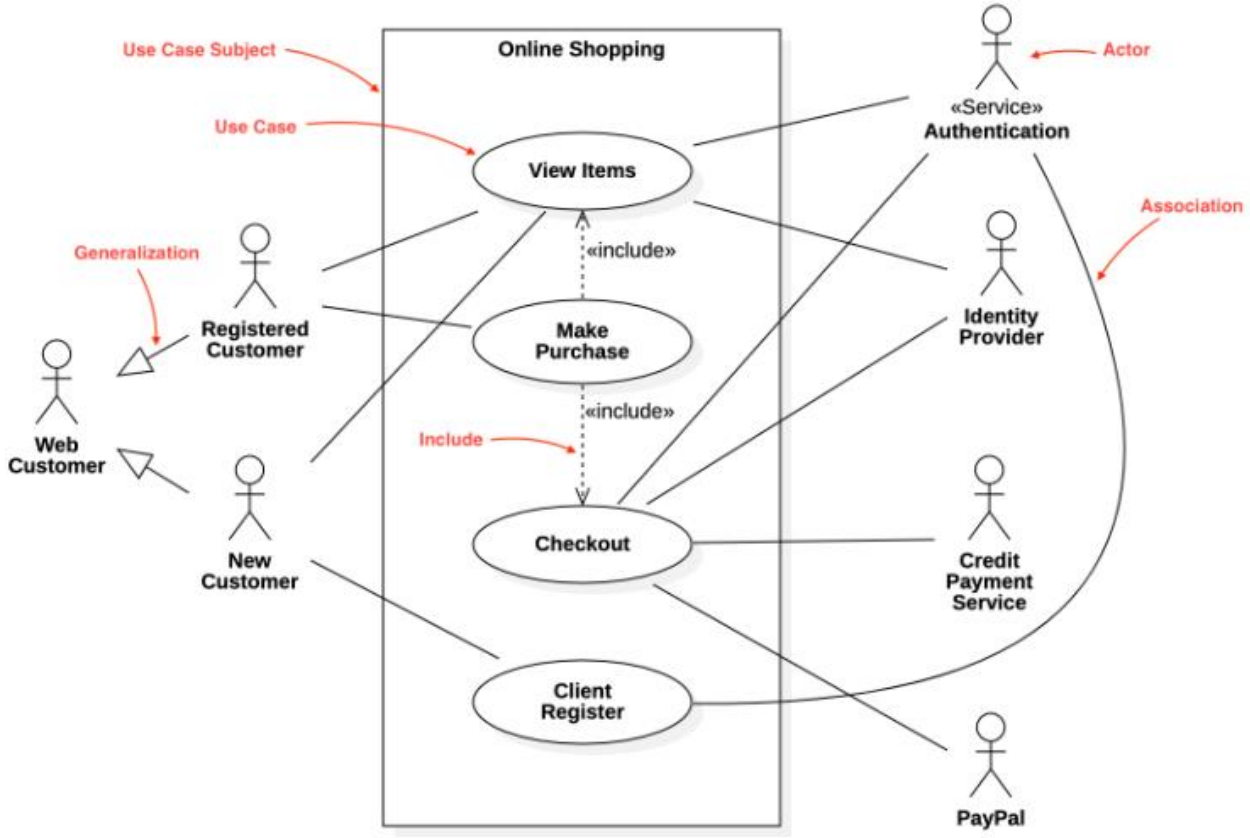


Figure 42 : Exemples d'éléments à représenter dans le diagramme des cas d'utilisation

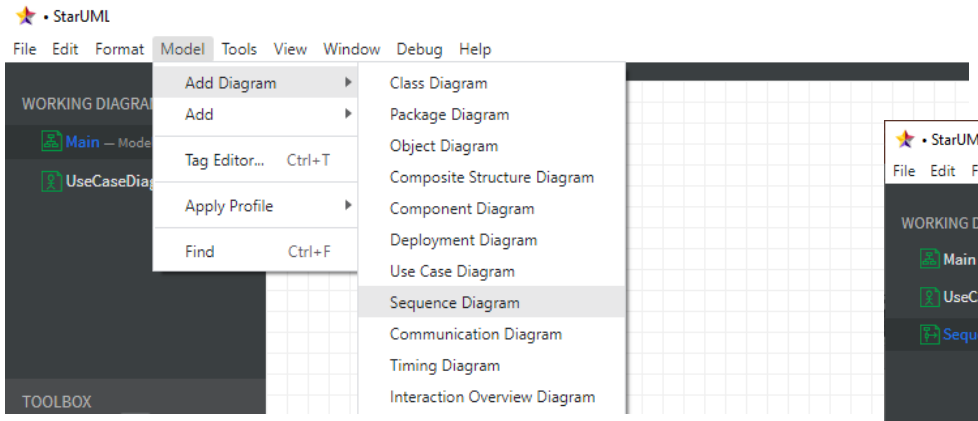
PARTIE 1

CHAPITRE 5

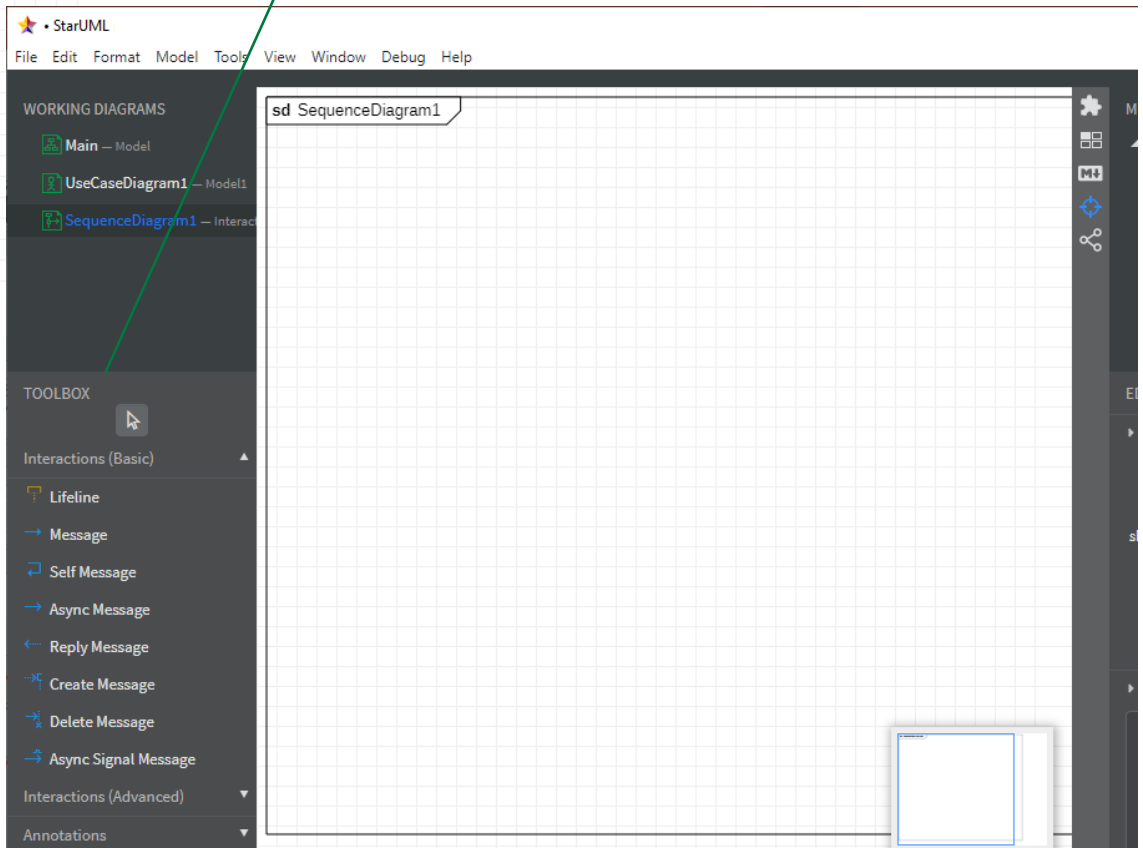
Elaborer des diagrammes UML à l'aide d'un outil de modélisation

1. Environnement de travail et fonctionnalités
2. Création et édition d'un diagramme de classe
3. Sauvegarde et impression
4. Génération du code source
5. Création et édition d'un diagramme des cas d'utilisation
6. **Création et édition d'un diagramme de séquences**

StarUML – Diagramme de séquences



Boîte à Outils



PARTIE 1

PARTIE 1

StarUML – Diagramme de séquences - Eléments

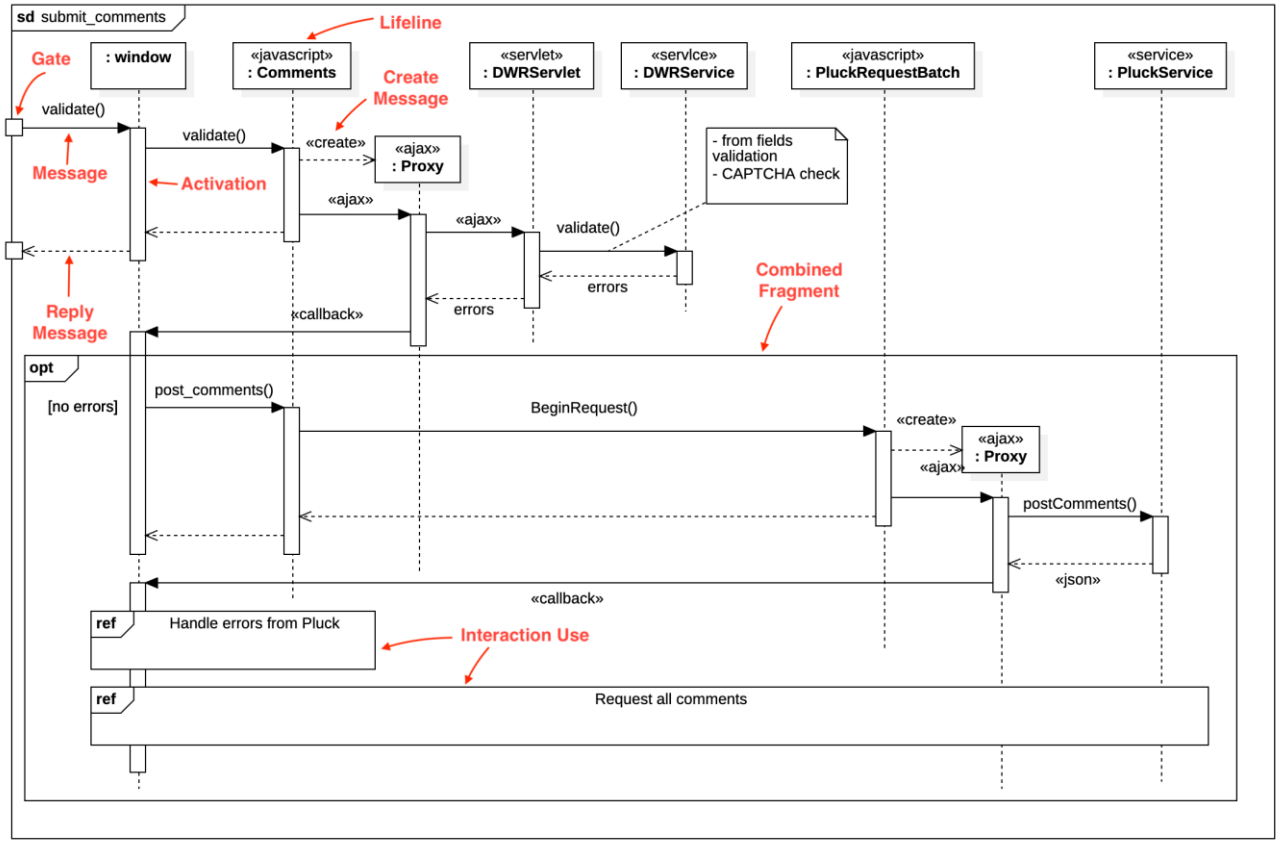


Figure 43 : Exemples d'éléments à représenter dans le diagramme de séquences

PARTIE 2

Représenter la vue dynamique d'un système

Dans ce module, vous allez :

- Décrire les changements d'états d'objets à l'aide d'un diagramme d'état transition
- Décrire le comportement d'un système à l'aide d'un diagramme d'activités

10 heures

CHAPITRE 1

Décrire les changements d'états d'objets à l'aide d'un diagramme d'état transition

Ce que vous allez apprendre dans ce chapitre :

- Rôle du diagramme états-transitions
- Etat
- Evénement
- Transition externe
- Transition interne
- Action et activité
- Point de choix
- État composite
- État historique

05 heures

CHAPITRE 1

Décrire les changements d'états d'objets à l'aide d'un diagramme d'état transition

1. **Rôle du diagramme états-transitions**
2. Etat
3. Evénement
4. Transition externe
5. Transition interne
6. Action et activité
7. Point de choix
8. État composite
9. État historique

Diagramme états-transitions (State Machine ou StateChart)

- Il fait parti des diagrammes comportementaux.
- Rôle : **Décrire le fonctionnement d'une machine (ou d'un objet) ayant un comportement séquentiel.**
- Il représente les différents états (situations) dans lesquels peut se trouver la machine (ou l'objet) ainsi que la façon dont cette machine (ou l'objet) passe d'un état à l'autre en réponse à des événements.

- On s'intéresse au **cycle de vie d'un objet générique d'une classe particulière** au fil de ses interactions, dans tous les cas possibles.
- Cette vue locale d'un objet, qui décrit comment il **réagit à des événements** en fonction de son état courant et comment il passe dans un nouvel état, est représentée graphiquement sous la forme **d'un diagramme d'états.**

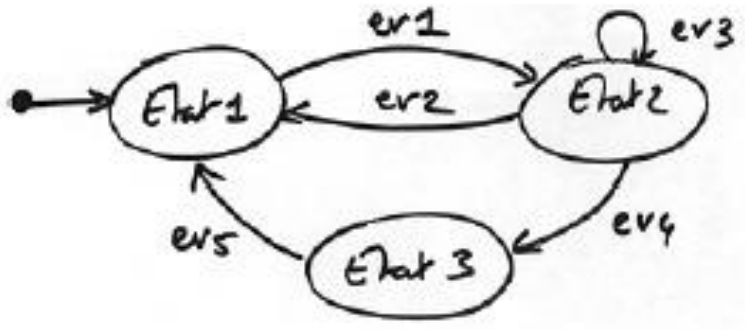


Figure 44 : Premier exemple simple de diagramme d'états

Diagramme états-transitions (State Machine ou StateChart)

- Heureusement, toutes les classes du modèle ne requièrent pas nécessairement une machine à états.
- Il s'agit donc de **trouver les classes qui ont un comportement dynamique complexe** nécessitant une description plus poussée.
- Cela correspond à l'un des deux cas suivants :

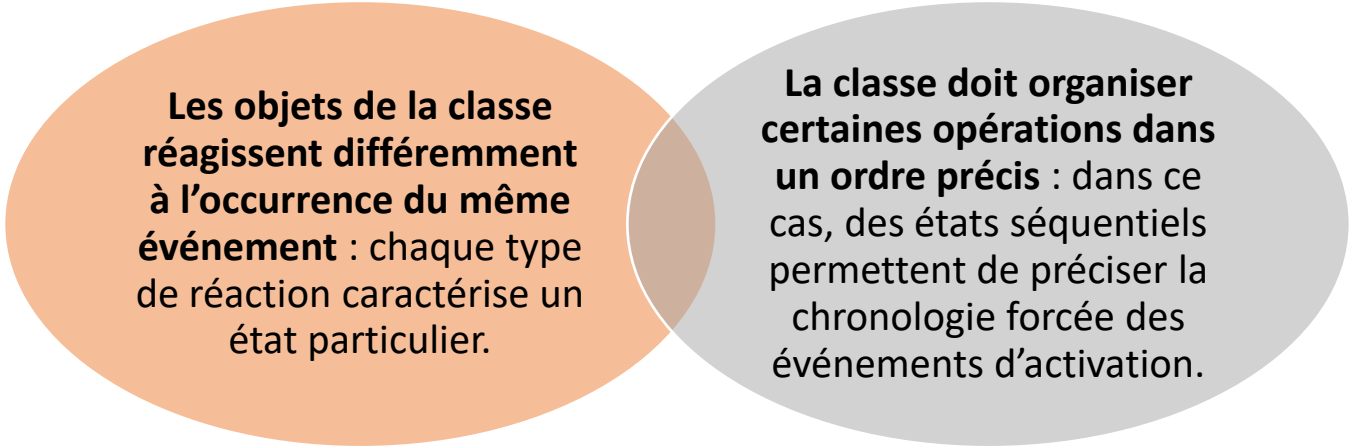


Diagramme états-transitions – Notation de base

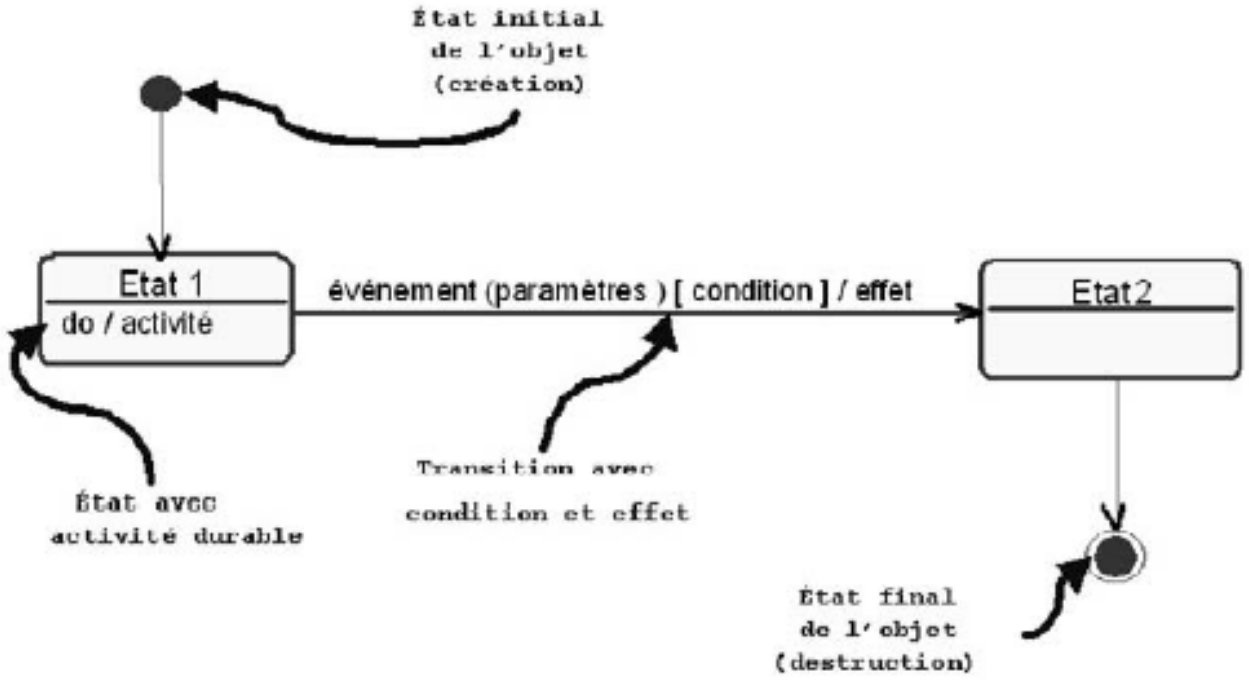


Figure 45 : Notation de base du diagramme d'états

CHAPITRE 1

Décrire les changements d'états d'objets à l'aide d'un diagramme d'état transition

1. Rôle du diagramme états-transitions
2. **Etat**
3. Evénement
4. Transition externe
5. Transition interne
6. Action et activité
7. Point de choix
8. État composite
9. État historique

Etat

- Un état représente une **situation stable durant la vie d'un objet** pendant laquelle :
  - il satisfait une certaine condition,
  - il exécute une certaine activité,
  - ou bien il attend un certain événement.

- Un objet passe par une **succession d'états durant son existence.**
- Un état a une **durée finie, variable selon la vie de l'objet**, en particulier en fonction des événements qui lui arrivent.

- Un état est représenté par un rectangle arrondi contenant son nom.

Etat initial et état final

- En plus de la succession d'états « normaux » correspondant au cycle de vie d'un objet, le diagramme d'états comprend également deux pseudo-états :
  - **L'état initial** du diagramme d'états correspond à **la création de l'instance**. Il est représenté par un point noir 
  - **L'état final** du diagramme d'états correspond à **la destruction de l'instance**. Il se représente par un point entouré d'un cercle 
- Tous les objets n'ont pas d'état final. C'est notamment le cas des objets permanents dans le système.

CHAPITRE 1

Décrire les changements d'états d'objets à l'aide d'un diagramme d'état transition

1. Rôle du diagramme états-transitions
2. Etat
3. **Evénement**
4. Transition externe
5. Transition interne
6. Action et activité
7. Point de choix
8. État composite
9. État historique

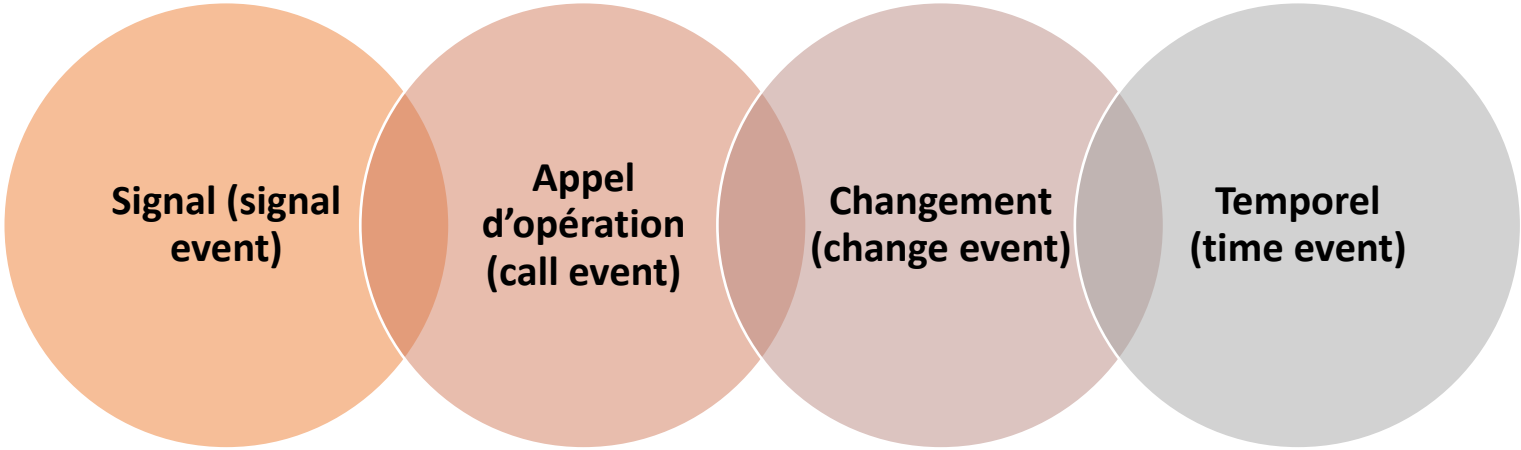
Evénement

- C'est un **fait qui déclenche le changement d'état**
- Il fait passer un objet d'un état à un autre état (**désactivation d'un état et activation de l'état suivant**).
- Il est lié à la réception d'un signal (d'un message) par l'objet, demandé généralement par un autre objet.

- **Un événement se produit à un instant précis et est dépourvu de durée**.
- Quand un événement est reçu, une transition peut être déclenchée et faire basculer l'objet dans un nouvel état.

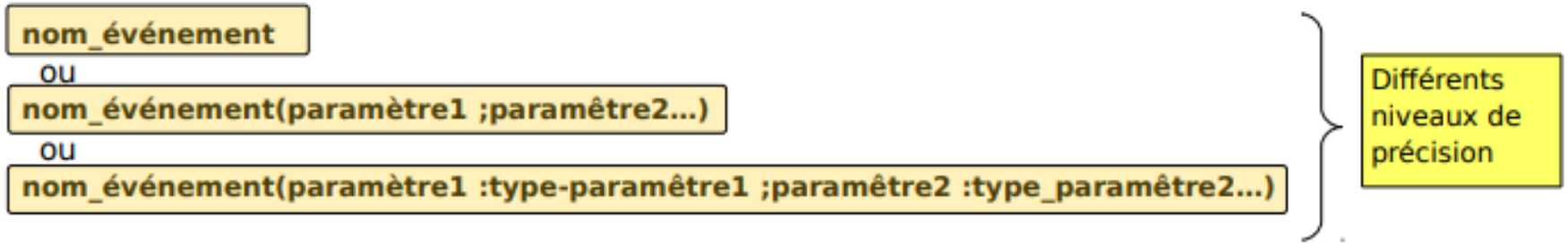
Types d'événement

- Il existe 4 types d'événements :



Evénement de type signal (signal event)

- Correspond à la réception d'un signal **asynchrone** émis par un autre objet ou par un acteur.
- Le signal est très souvent associé à la gestion événementielle d'une structure complexe comme une interface IHM.
- Un événement de type signal peut être porteur de paramètres et s'écrit comme suit :



- Les signaux peuvent être déclarés dans le diagramme de classes de la même manière qu'une classe mais en ajoutant le **stéréotype <<signal>>** au dessus du nom. Un signal peut aussi être déclaré de façon textuelle.

PARTIE 2

Evénement appel opération (call event)

- **Appel d'une méthode de l'objet courant** par un autre objet ou par un acteur.
- Il possède la même syntaxe que l'événement de type signal.
- La méthode est aussi déclarée dans le diagramme de classe, **mais à l'intérieur de la classe**

Evénement de changement (change event)

- Il est **généré par la satisfaction (vrai ou faux) d'une expression booléenne sur des valeurs d'attributs**.
- Il s'agit d'une manière déclarative d'attendre qu'une condition particulière soit satisfaite.
- L'expression est testée en permanence.

- La syntaxe est la suivante : `when(expression booléenne)`

Evénement temporel (time event)

- Les événements temporels sont générés par l'écoulement du temps.
- Par défaut, le temps commence à s'écouler dès l'entrée dans l'état courant
- Ils sont spécifiés :
  - Soit **de manière absolue :** déclenchement à une date précise

Syntaxe : `when(date= « expression précise d'une date »)` ex : **when(date=17/12/2010)**

  - Soit **de manière relative :** déclenchement après une certaine durée passée dans l'état actuel.

Syntaxe : `after(« expression d'une durée »)` ex : **after(10secondes)**

CHAPITRE 1

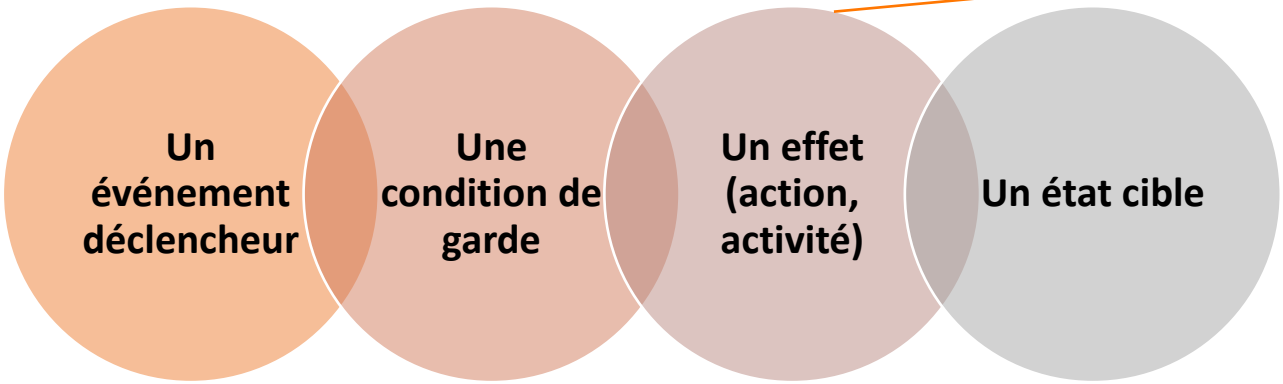
Décrire les changements d'états d'objets à l'aide d'un diagramme d'état transition

1. Rôle du diagramme états-transitions
2. Etat
3. Evénement
4. **Transition externe**
5. Transition interne
6. Action et activité
7. Point de choix
8. État composite
9. État historique

Transitions

- Une transition décrit **la réaction d'un objet lorsqu'un événement se produit** (généralement l'objet change d'état, mais pas forcément).

- En règle générale, une transition possède :



- **Mise à jour d'un attribut,**
- **Appel d'opération,**
- **Création ou Destruction d'un autre objet,**
- **Envoi d'un signal à un autre objet.**

Exemples d'action

- Une transition indique qu'un objet qui se trouve dans un état peut « transiter » vers un autre état (état cible) en exécutant éventuellement certaines activités (effet), si un événement déclencheur se produit et qu'une condition de garde est vérifiée.

Transition externe

- Une transition externe est une transition **qui modifie l'état actif**.
- Il s'agit du type de transition le plus répandu.
- Une transition entre deux états est représentée par un trait droit fléché allant de l'état source vers l'état cible.

- L'événement qui détermine le franchissement de la transition est indiqué sous forme de texte.
- **Si la transition est automatique**, aucun événement n'est spécifié.

- Une transition n'a pas de durée. On dit qu'elle est déclenchée

Transition externe

Transition externe

- **Syntaxe de l'événement** qui correspond aux détails de la transition : ***déclencheur [garde] / effet***

Déclencheur

nom de l'événement qui provoque la transition

Garde

Condition de garde entre []

Une expression booléenne (sur les attributs de l'objet ou les paramètres de l'événement déclencheur) qui **doit être vraie** pour que la transition soit déclenchée.

Testée **uniquement** lorsque l'événement déclencheur se produit

Effet

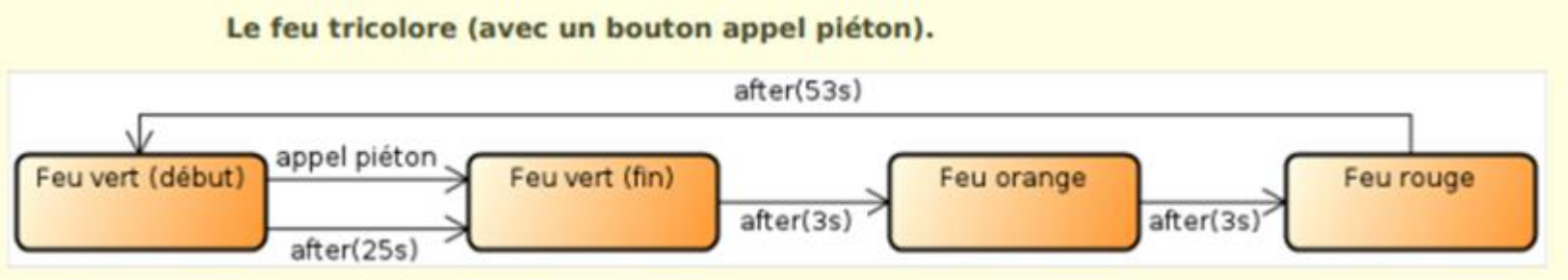
Une **activité (série d'actions)** à effectuer lors du déclenchement (nom de l'activité précédé de /)

Les activités durables (**do-activity**) ont une certaine durée, sont interruptibles et sont associées aux états.

Une fois terminé, **l'état cible de la transition devient actif**

PARTIE 2

Transition externe - Exemple



CHAPITRE 1

Décrire les changements d'états d'objets à l'aide d'un diagramme d'état transition

1. Rôle du diagramme états-transitions
2. Etat
3. Evénement
4. Transition externe
5. **Transition interne**
6. Action et activité
7. Point de choix
8. État composite
9. État historique

Transition interne

- Les états déjà vus n'effectuaient qu'une seule activité.
- On peut avoir **des états qui effectuent plusieurs activités successivement ou en parallèle**.
- L'enchaînement de ces activités à l'intérieur d'un même état peut être spécifié grâce à des transitions internes.

- La **syntaxe** d'une transition interne est la même que celle d'une transition externe : ***déclencheur [garde] / effet***

- Les transitions internes **s'écrivent à l'intérieur de l'état**, séparé du nom de l'état par un trait.

- Les transitions internes ont des événements particuliers :

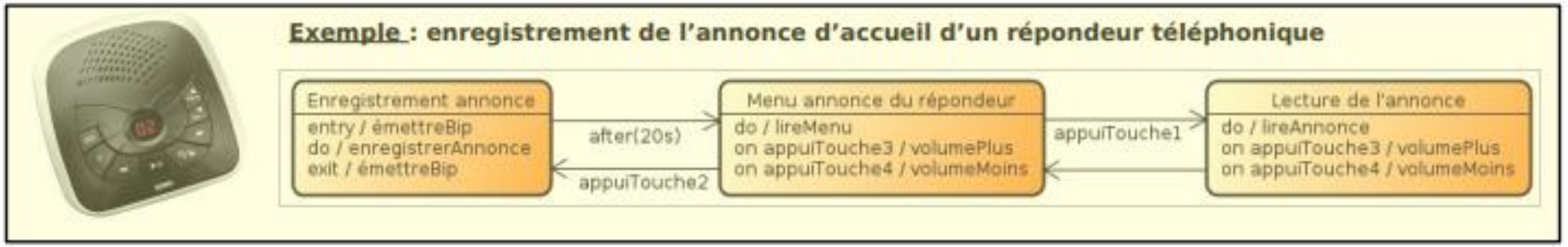
entry — Définit l'activité à exécuter lors de l'entrée dans l'état

exit — Définit l'activité à exécuter lors de la sortie de l'état

do — Définit l'activité à exécuter dès que celle définie par entry est terminée

On — (**Optionnel**) définit l'activité à exécuter à chaque fois qu'on a un événement particulier

Transition interne - Exemple



CHAPITRE 1

Décrire les changements d'états d'objets à l'aide d'un diagramme d'état transition

1. Rôle du diagramme états-transitions
2. Etat
3. Evénement
4. Transition externe
5. Transition interne
6. **Action et activité**
7. Point de choix
8. État composite
9. État historique

Action et Activité

- Une **action** consiste à envoyer un signal, à faire appel à une méthode, à affecter une valeur à un attribut...
- Une **activité** est une série d'actions

- Il existe **deux types d'activité** :

Activités atomiques

- Ce sont des activités qui **n'ont pas de durée**. Elles ne peuvent pas être interrompues
- Elles peuvent être **associées à des états ou à des transitions**.

Activités continues (durables)

- Ce sont des activités qui ont **une certaine durée**, elles peuvent être par conséquent interrompues.
- Elles ne peuvent être **associées qu'à des états**.

Exemple

Affecter une valeur à un attribut

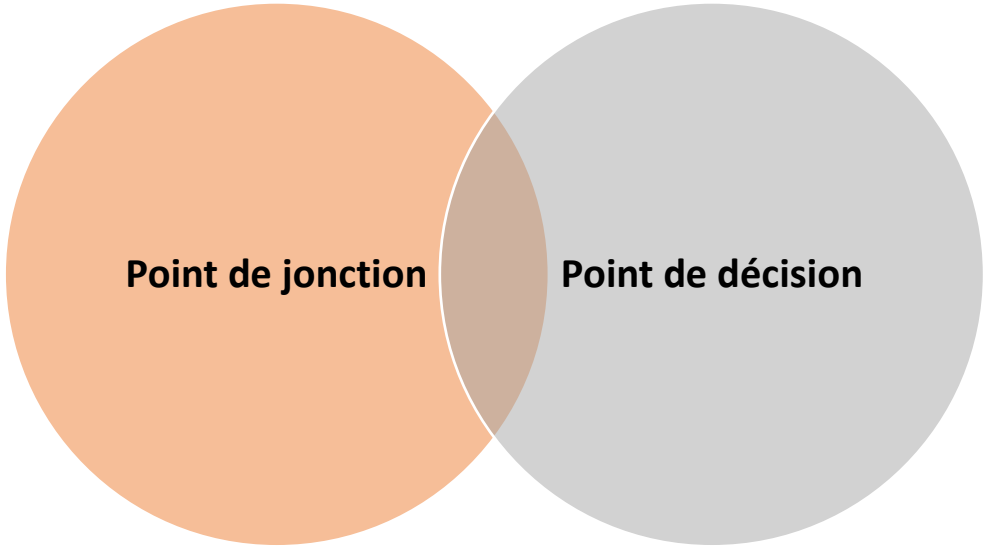
CHAPITRE 1

Décrire les changements d'états d'objets à l'aide d'un diagramme d'état transition

1. Rôle du diagramme états-transitions
2. Etat
3. Evénement
4. Transition externe
5. Transition interne
6. Action et activité
7. **Point de choix**
8. État composite
9. État historique

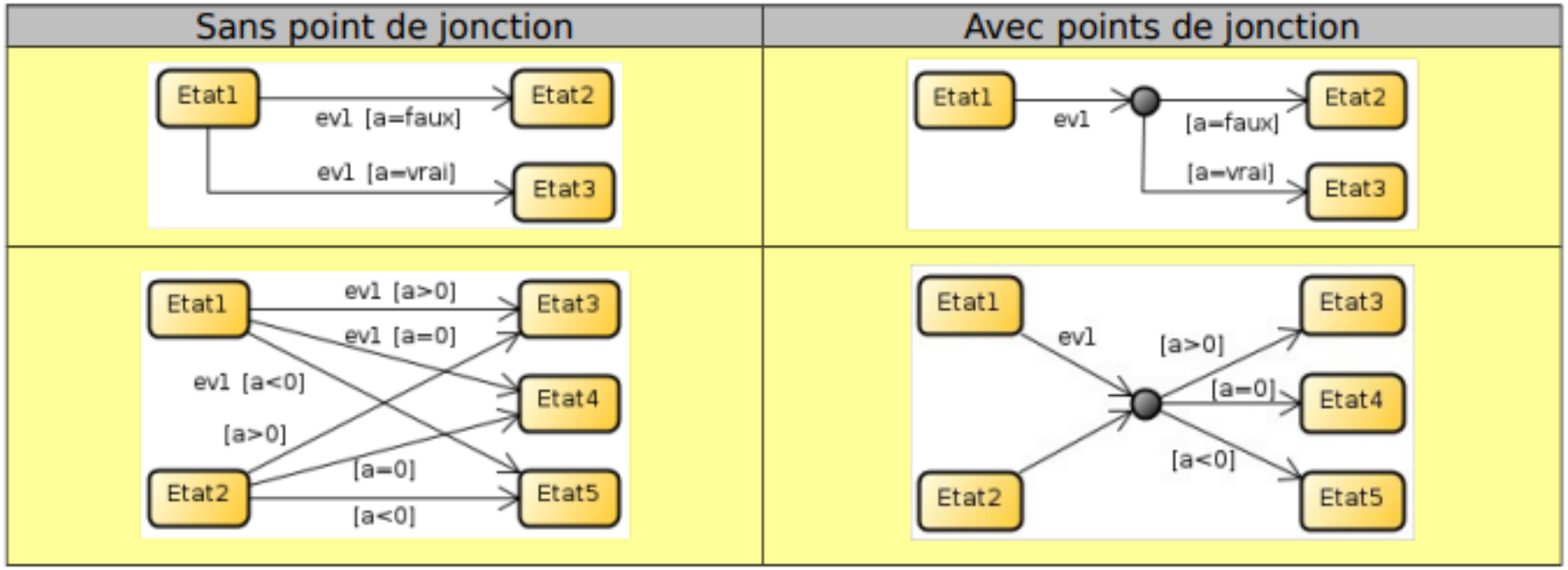
Point de choix

- Il est possible de représenter des alternatives **pour le franchissement d'une transition en utilisant des pseudo-états** particuliers :



Point de jonction

- Les points de jonction sont représentés par un cercle plein : ●
- Ils permettent à **plusieurs transitions d'avoir une partie commune** en partageant des segments de transition.
- L'utilisation de points de jonction a pour but de rendre la notation des transitions alternatives plus lisible.

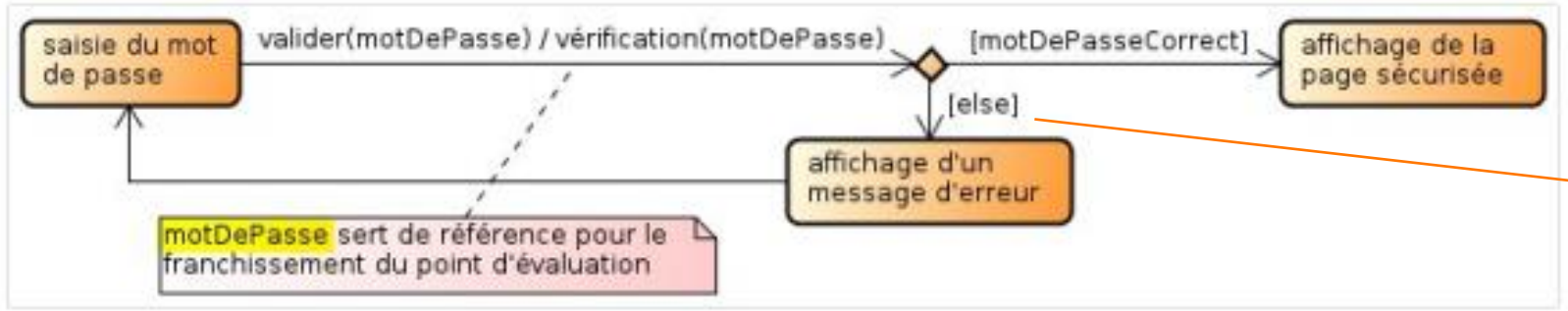


PARTIE 2

Point de décision

- Les points de décision (point d'évaluation) sont représentés  des losanges :
- Ils ont un fonctionnement similaire à celui du point de jonction mise à part qu'on peut sortir de l'état d'origine dès que le segment avant le point de décision est franchissable (même si aucun des segments après le point de décision n'est franchissable).
- **Le choix du segment à franchir derrière le point de décision se fait au moment de l'arrivée sur le point de décision**. Cela permet aux **segments** qui sont derrières le point de décision **d'avoir une condition de garde** qui dépend d'éléments qui sont définis par une activité effectuée lors du franchissement du segment de transition avant le point de décision.

- **Un segment de transition derrière un point de décision ne peut pas comporter d'événement.**



- **Il faut qu'un des segments de transition qui suit le point de décision, soit franchissable.**
- **Pour éviter ce problème on peut ajouter un segment avec la garde [else] qui est automatiquement franchie si aucune des gardes des autres segments n'est vraie.**

PARTIE 2

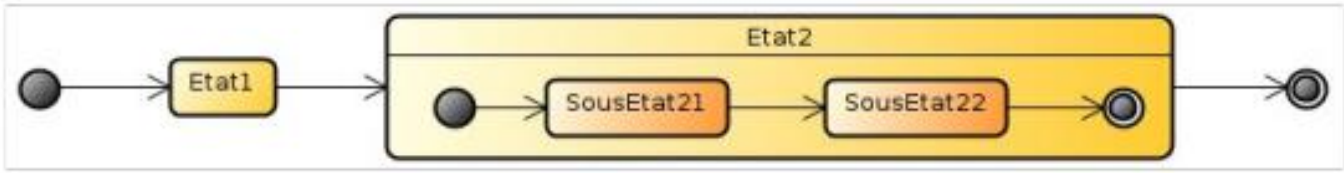
CHAPITRE 1

Décrire les changements d'états d'objets à l'aide d'un diagramme d'état transition

1. Rôle du diagramme états-transitions
2. Etat
3. Evénement
4. Transition externe
5. Transition interne
6. Action et activité
7. Point de choix
8. **État composite**
9. État historique

Etat composite

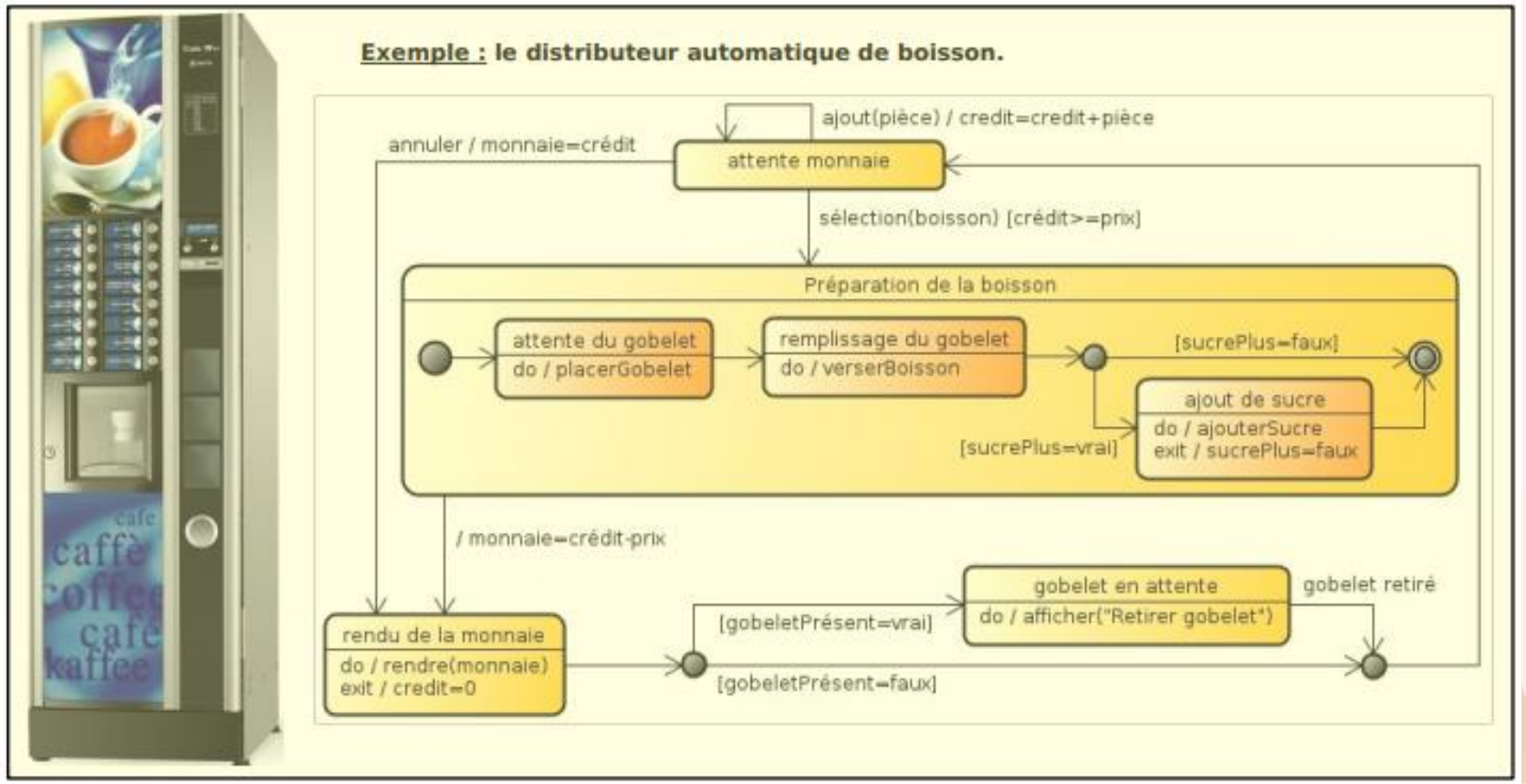
- Certains états sont **complexes** et correspondent à la réalisation de plusieurs activités (séquentielles ou simultanées) qui ne pourront pas être définis par des transitions internes.
- **Il peut alors être intéressant de le décomposer en sous-états**.
- On parle alors **d'état composite**.



- Dès qu'un état composite est actif, il active son sous état initial.
- Si un état composite est raccordé à une transition automatique, elle est franchie lorsqu'on atteint le sous état final de l'état composite.
- Si une des transitions externes attenantes à l'état composite est franchissable alors elle est franchie et tous les sous-états deviennent inactifs.

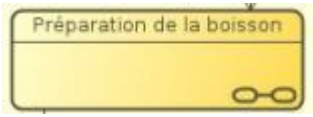
Etat composite

Etat composite - Exemple



Remarque

Afin d'éviter de surcharger le diagramme d'états-transition, il est possible de placer le symbole ⚯ à l'intérieur de l'état composite et de **spécifier ensuite son contenu ailleurs dans une autre page.**



PARTIE 2

CHAPITRE 1

Décrire les changements d'états d'objets à l'aide d'un diagramme d'état transition

1. Rôle du diagramme états-transitions
2. Etat
3. Evénement
4. Transition externe
5. Transition interne
6. Action et activité
7. Point de choix
8. État composite
9. **État historique**

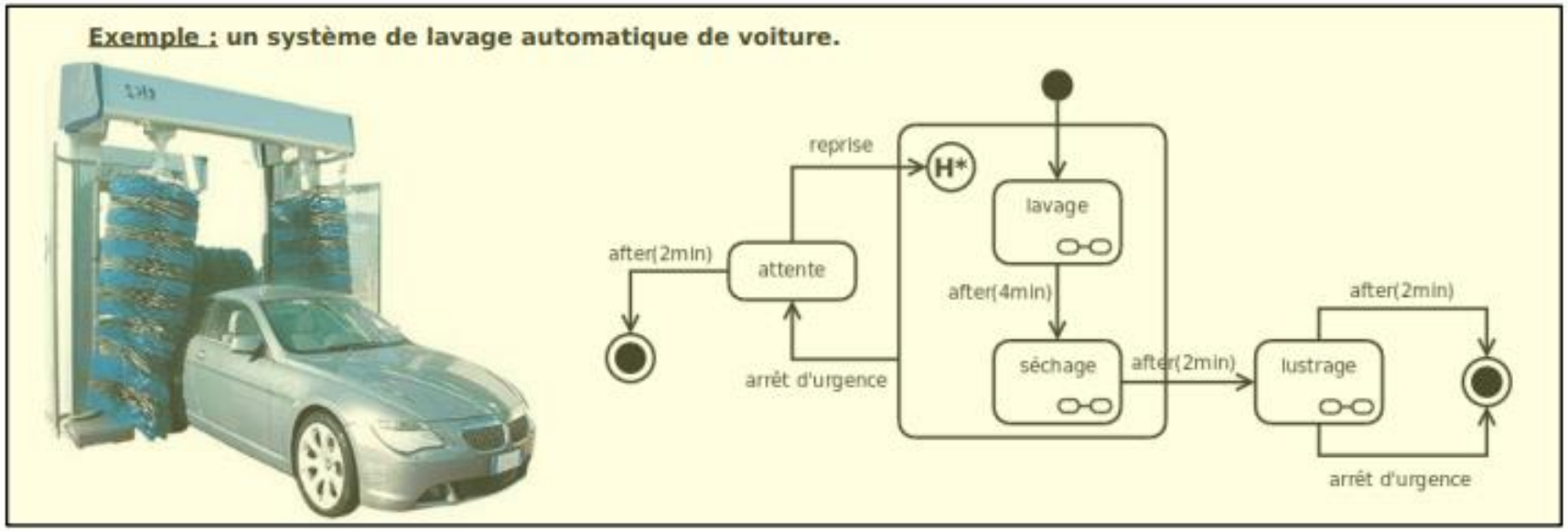
Etat historique

- On a vu que lorsqu'une transition raccordée à un état composite est franchissable alors **elle est franchie même si les activités en cours (sous-états) ne sont pas terminées**.
- Si on veut qu'un état composite reprenne sont activité à l'endroit ou il s'était interrompu lors de sa précédente activation, il faut lui **définir un nouveau sous état d'entrée appelé état historique et** désigné par :  ou 

pour reprendre au début du sous-état **du plus haut niveau** dans lequel on était arrêté

pour reprendre au début du sous état dans lequel on était arrêté, **quelque soit son niveau d'imbrication** (historique profond)

Etat historique - Exemple



PARTIE 2