Les systèmes d'information avec l'UML

1 Introduction

1.1 C'est quoi un SI ?

Un système d'information (SI) est un ensemble organisé de moyens (outils, logiciels, données) qui servent à collecter, stocker, traiter et partager des informations pour aider une organisation à fonctionner et à prendre des décisions. Il mobilise également des acteurs, internes ou externes, qui interagissent avec le système.

- **Outils/matériels** : ordinateurs, terminaux, serveurs, réseaux.
- **Logiciels** : applications, systèmes de gestion, interfaces.
- **Données** : informations manipulées et stockées.
- **Acteurs internes** : personnes appartenant à l'organisation et utilisant le système (par exemple, les employés).
- **Acteurs externes** : personnes extérieures qui interagissent avec le système (par exemple, les clients).

Exemple : SI bancaire

- **Ressources** :
  - Outils : ordinateurs, distributeurs automatiques (DAB), terminaux de paiement.
  - Logiciels : applications de gestion des comptes, logiciels de transactions, applications mobiles bancaires.
  - Données : informations sur les clients (nom, adresse, revenus), données des comptes (soldes, historiques des opérations), contrats de crédit, taux d'intérêt.
- **Acteurs internes** : conseillers bancaires, guichetiers, gestionnaires de comptes.
- **Acteurs externes** : clients, entreprises informatiques (développement, sécurité du SI).

1.2 Approches pour la conception des SI

1) Approche traditionnelle : L'approche traditionnelle repose sur une séparation stricte entre les données et les traitements. Les données sont organisées dans des structures (tables ou fichiers), tandis que les traitements sont implémentés dans des procédures ou programmes distincts.

Exemple : gestion d'un étudiant dans un système universitaire

```
Table Étudiant
----------------
Numéro
Nom
Prénom
Filière
```

- Les opérations comme l'ajout d'un étudiant, le calcul de la moyenne ou l'inscription à un cours sont réalisées par des traitements séparés de la structure de données.

2) Approche orientée objet : L'approche orientée objet regroupe les données et les traitements au sein d'entités appelées objets. Cette approche favorise la réutilisabilité, la modularité et une meilleure correspondance avec le monde réel grâce aux concepts de classes, d'attributs et de méthodes.
Exemple :

```
Classe Étudiant {
    Attributs : Numéro, Nom, Prénom, DateNaissance, Filière
    Méthodes : inscrireCours(), calculerMoyenne(), afficherInfos()
}
```

Comparaison : Dans l'approche traditionnelle, les données et les traitements sont définis séparément. Dans l'approche orientée objet, ils sont regroupés dans une même entité, ce qui reflète plus fidèlement la logique du monde réel.

1.3 L'approche orientée objet (Rappel)

Concept d'objet : Un objet représente une entité du monde réel (ou virtuelle, pour les objets immatériels). Il se caractérise par un ensemble de propriétés (attributs), des états significatifs et un comportement (opérations).

Concept de classe : Une classe est l'abstraction d'un ensemble d'objets qui partagent une même structure (liste d'attributs) et un même comportement (liste d'opérations). Un objet est une instance d'une seule classe. Une classe abstraite est une classe qui ne peut pas être instanciée. Les concepts de classe et d'objet sont donc étroitement liés.

Encapsulation : L'approche objet regroupe dans une même classe les attributs et les opérations, ce qui constitue l'*encapsulation*. Les données ne sont accessibles qu'au travers des opérations définies dans la classe. L'ensemble des opérations rendues visibles aux autres classes constitue l'*interface*.

Association et agrégation :

- **L'association** représente une relation entre plusieurs classes. Elle correspond à l'abstraction des liens existant entre objets dans le monde réel. Les multiplicités (cardinalités) et les rôles des objets complètent cette description.
- **L'agrégation** est une forme particulière d'association. Elle exprime qu'une classe est composée d'une ou plusieurs autres classes.

Généralisation et spécialisation : La généralisation consiste à regrouper, dans une classe appelée *super-classe*, les attributs et/ou opérations communs à plusieurs classes. La spécialisation est l'opération inverse : elle consiste à créer, à partir d'une classe, plusieurs classes dérivées appelées *sous-classes*. Une sous-classe hérite des attributs et opérations de sa super-classe et peut ajouter des éléments spécifiques.

Polymorphisme : Le polymorphisme désigne la capacité d'une même méthode à adopter des comportements différents selon la classe dans laquelle elle est utilisée. Une méthode définie dans une super-classe peut ainsi être redéfinie dans une sous-classe avec un traitement spécifique, tout en réutilisant le traitement commun de la super-classe.

Exemple : Soit la classe *Animal* et ses deux sous-classes *Chien* et *Chat*.

```
Classe Animal {
    Attributs : nom, age
    Méthodes : crier()
}

Classe Chien hérite de Animal {
    Attributs : race
    Méthodes : crier()  // "Aboyer"
}

Classe Chat hérite de Animal {
    Attributs : couleurYeux
    Méthodes : crier()  // "Miauler"
}
```

Lorsqu'on appelle la méthode crier() sur un objet de type Animal, c'est la version adaptée (*Chien* ou *Chat*) qui est exécutée, selon la nature réelle de l'objet.

1.4 Modéliser pour analyser

Pourquoi modéliser ? Modéliser consiste à représenter de manière visuelle et graphique les besoins et les solutions fonctionnelles et techniques d'un projet logiciel. L'utilisation de schémas rend les systèmes complexes plus faciles à comprendre qu'une longue description textuelle. Dans ce cadre, UML est un langage de modélisation qui permet de visualiser le futur logiciel de manière claire et partagée.

Domaine métier et modélisation d'une solution informatique : La modélisation des processus métier est au cœur de l'analyse des organisations. Elle permet de formaliser le fonctionnement d'une organisation à l'aide d'un langage standard et compréhensible par tous. Cette représentation commune favorise l'amélioration continue et la réorganisation éventuelle des processus. La qualité de la modélisation conditionne la pertinence de la vision partagée des principes de fonctionnement de l'organisation et contribue à éviter les erreurs d'interprétation.

Analyse et conception d'une solution informatique : Une méthode d'analyse et de conception vise à formaliser les étapes préliminaires du développement afin d'assurer une meilleure adéquation avec les besoins du client. L'analyse s'appuie sur :

- les besoins exprimés par le client et les utilisateurs,
- l'étude de l'existant, c'est-à-dire des processus déjà en place.

La phase d'analyse permet d'identifier les fonctionnalités attendues ainsi que des critères de performance, de robustesse, de maintenance, de sécurité et d'évolutivité. La phase de conception décrit de façon non ambiguë, en utilisant un langage de modélisation, le fonctionnement futur du système afin d'en faciliter la réalisation.

Évolution vers l'analyse et la conception orientées objet : Un modèle est une abstraction simplifiée de la réalité. Le processus de modélisation permet de passer d'un cahier des charges exprimé en langage naturel à une mise en œuvre logicielle. Ce passage s'effectue par étapes, en affinant progressivement les modèles pour aboutir au code source.

L'approche orientée objet consiste à décomposer un problème en sous-problèmes représentés par des objets et leurs interactions. Par exemple, pour modéliser une banque, on identifie les objets suivants : clients, comptes bancaires, conseillers, opérations de transfert, placements, emprunts. On étudie ensuite leurs interactions, comme la création d'un compte bancaire qui met en relation le client et le conseiller. Cette approche permet de mieux représenter la réalité et de structurer le système à concevoir.

2 Unified Modeling Language (UML)

Le langage de modélisation unifié (UML, pour *Unified Modeling Language*) est un langage de modélisation graphique basé sur des diagrammes normalisés. Il a été conçu pour fournir une méthode standard permettant de représenter visuellement la conception d'un système. UML est largement utilisé dans le développement logiciel et la conception orientée objet, car il facilite la communication entre les différents acteurs d'un projet et réduit les risques d'ambiguïté dans la compréhension des besoins et des solutions.

2.1 Les différents types de diagrammes

Les diagrammes UML se complètent et permettent de modéliser un projet tout au long de son cycle de vie. Ils sont classés en deux grandes catégories : les diagrammes de comportement et les diagrammes de structure.

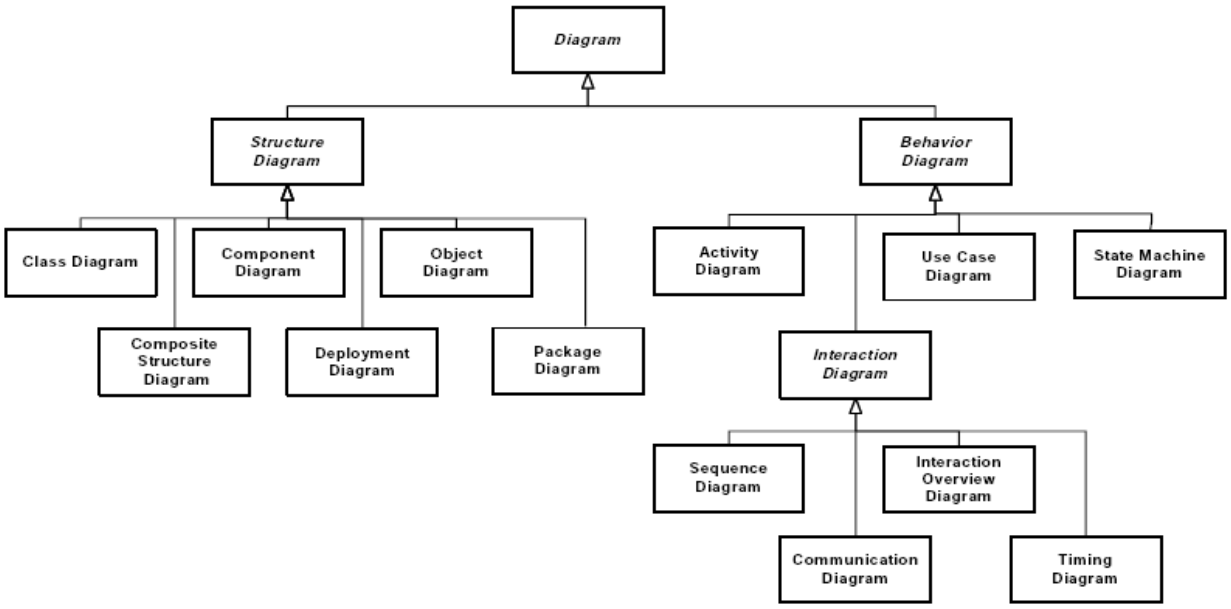

FIGURE 1 – Classification générale des types de diagrammes UML

2.1.1 Diagrammes de comportement

Les diagrammes de comportement (*behavior diagrams*) décrivent la dynamique d'un système. On distingue notamment :

- **Diagramme des cas d'utilisation (use-case diagram)** : représente les interactions possibles entre le système et ses acteurs, c'est-à-dire les fonctionnalités attendues.
- **Diagramme états-transitions (state machine diagram)** : modélise le comportement d'un système ou d'un composant sous forme de machine à états finis.

- **Diagramme d'activité (activity diagram)** : illustre l'enchaînement des activités ou des flux de contrôle dans le système.
- **Diagrammes d'interaction (interaction diagrams)** : également appelés diagrammes dynamiques, ils représentent les échanges entre les objets du système. Parmi eux :
  - **Diagramme de séquence (sequence diagram)** : décrit l'ordre temporel des interactions entre objets et/ou acteurs.
  - **Diagramme de communication (communication diagram)** : version simplifiée du diagramme de séquence, centrée sur les messages échangés (introduit avec UML 2.x).
  - **Diagramme global d'interaction (interaction overview diagram)** : décrit les enchaînements entre différents scénarios représentés par des diagrammes de séquence (variante du diagramme d'activité, UML 2.x).
  - **Diagramme de temps (timing diagram)** : illustre l'évolution des valeurs ou des états d'un objet au cours du temps (introduit avec UML 2.3).

2.1.2 Diagrammes de structure

Les diagrammes de structure (*structure diagrams*), aussi appelés diagrammes statiques, décrivent l'architecture d'un système. Ils comprennent :

- **Diagramme de classes (class diagram)** : décrit les classes du système et leurs relations.
- **Diagramme d'objets (object diagram)** : illustre les instances de classes (objets) utilisées dans le système.
- **Diagramme de composants (component diagram)** : représente les composants logiciels du système (fichiers, bibliothèques, bases de données).
- **Diagramme de déploiement (deployment diagram)** : décrit les éléments matériels (ordinateurs, réseaux, périphériques) et la répartition des composants logiciels sur ces éléments.
- **Diagramme des paquets (package diagram)** : montre les dépendances entre paquets, utilisés pour organiser logiquement les éléments du modèle.
- **Diagramme de structure composite (composite structure diagram)** : décrit les relations internes entre les composants d'une classe (introduit avec UML 2.x).
- **Diagramme de profils (profile diagram)** : permet de spécialiser et de personnaliser UML pour un domaine particulier (introduit avec UML 2.2).

3 Diagramme des cas d'utilisation

Un diagramme de cas d'utilisation permet de décrire, de manière visuelle et synthétique, les interactions entre les acteurs (utilisateurs ou systèmes externes) et le système étudié. Il est construit du point de vue de l'utilisateur, en se concentrant sur ce que fait le système et non sur la manière dont il est implémenté.

Un diagramme de cas d'utilisation comporte quatre éléments principaux : le système, les acteurs, les cas d'utilisation et les relations.

Système : Le système correspond à l'application ou au processus étudié (site web, application mobile, logiciel de gestion, etc.). Il est représenté par un rectangle dans lequel sont placés les cas d'utilisation.

Le rectangle définit la frontière du système :

- Ce qui est **à l'intérieur** du rectangle fait partie du système.
- Ce qui est **à l'extérieur** n'appartient pas au système mais peut interagir avec lui.

Exemple : une agence de voyages en ligne (eDreams).

Acteur : Un acteur est une entité externe qui interagit avec le système pour atteindre un objectif. Il peut s'agir d'une personne, d'une organisation, d'un autre système ou d'un dispositif matériel. Les acteurs sont représentés par des pictogrammes (stickman) placés **à l'extérieur** du rectangle du système.

On distingue :

- Les **acteurs primaires**, qui initient une interaction avec le système (exemple : un client qui utilise une application de voyages en ligne). Ils sont placés à gauche du système.
- Les **acteurs secondaires**, qui réagissent à une action de l'acteur primaire (exemple : une compagnie aérienne qui fournit les vols et valide les réservations). Ils sont placés à droite du système.

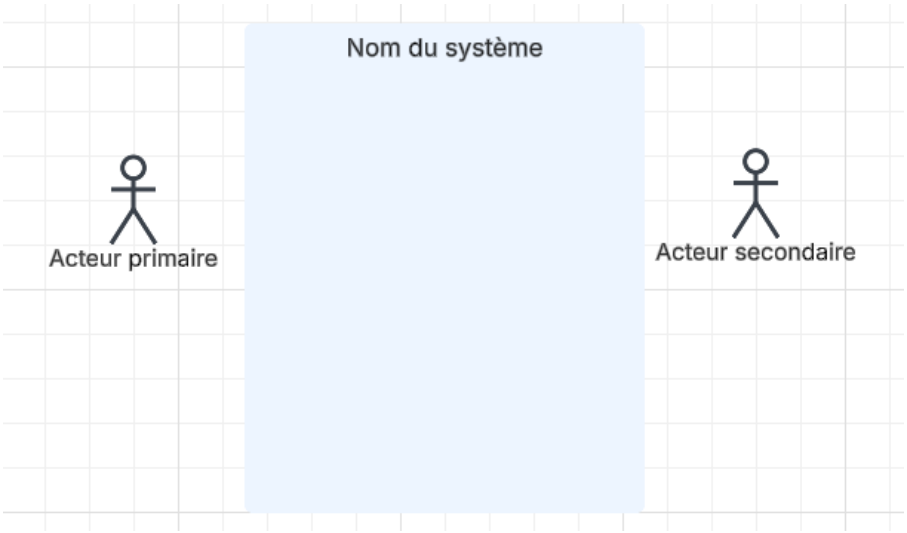

Figure 2 – Acteurs primaires et secondaires

Cas d'utilisation : Un cas d'utilisation représente une fonctionnalité du système répondant à un besoin d'un acteur. Il est représenté par un ovale portant un nom clair, généralement formulé comme une action (verbe à l'infinitif). **Exemples :** *Rechercher un vol*, *Réserver un billet*, *Annuler une réservation*, *S'enregistrer en ligne*.

FIGURE 3 – Cas d'utilisation

Relations : Les relations relient les acteurs aux cas d'utilisation, les cas d'utilisation entre eux, et parfois les acteurs entre eux par généralisation. On distingue :

1. **L'association (ou communication)** : relie un acteur à un cas d'utilisation pour représenter une interaction directe.

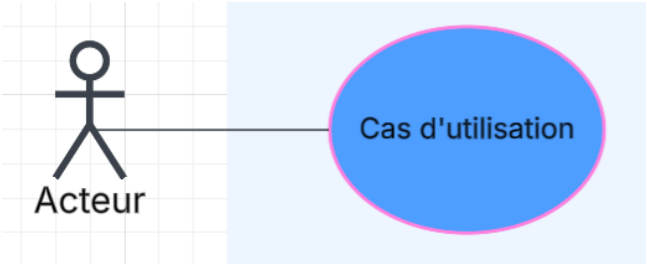

FIGURE 4 – Association entre un acteur et un cas d'utilisation

2. **La relation d'inclusion (*include*)** : un cas d'utilisation A inclut systématiquement le comportement d'un autre cas d'utilisation B. L'exécution de A entraîne toujours celle de B. Par exemple : le cas *Se connecter* inclut *Vérifier mot de passe*.

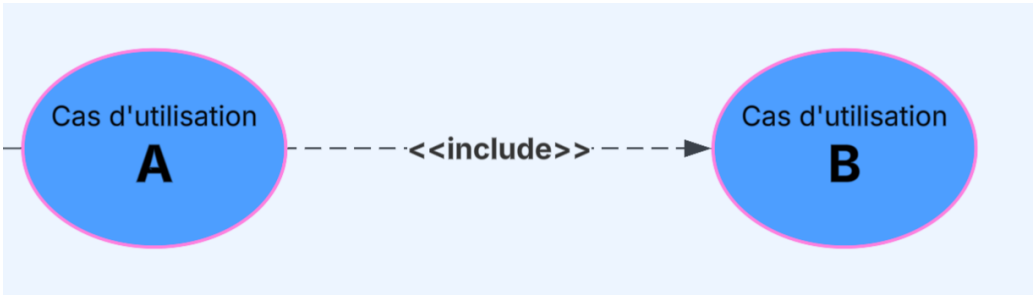

FIGURE 5 – Relation d'inclusion

3. **La relation d'extension (*extend*)** : un cas d'utilisation A peut être complété par un cas B dans certaines conditions seulement. Par exemple : *Se connecter* peut être étendu par *Afficher erreur de connexion* si le mot de passe est incorrect.

FIGURE 6 – Relation d'extension

Points d'extension : Il existe une notation particulière permettant de préciser les endroits où des comportements optionnels peuvent compléter un cas d'utilisation : ce sont les **points d'extension**. Ils représentent une version détaillée des relations d'extension.

Par exemple : le cas d'utilisation *Configurer profil* peut comporter deux points d'extension :

- *Accéder à l'aide du profil*,
- *Afficher les informations de confidentialité*.

Ces extensions ne sont exécutées que si certaines conditions sont remplies. Une note peut être ajoutée au diagramme pour expliciter ces conditions.

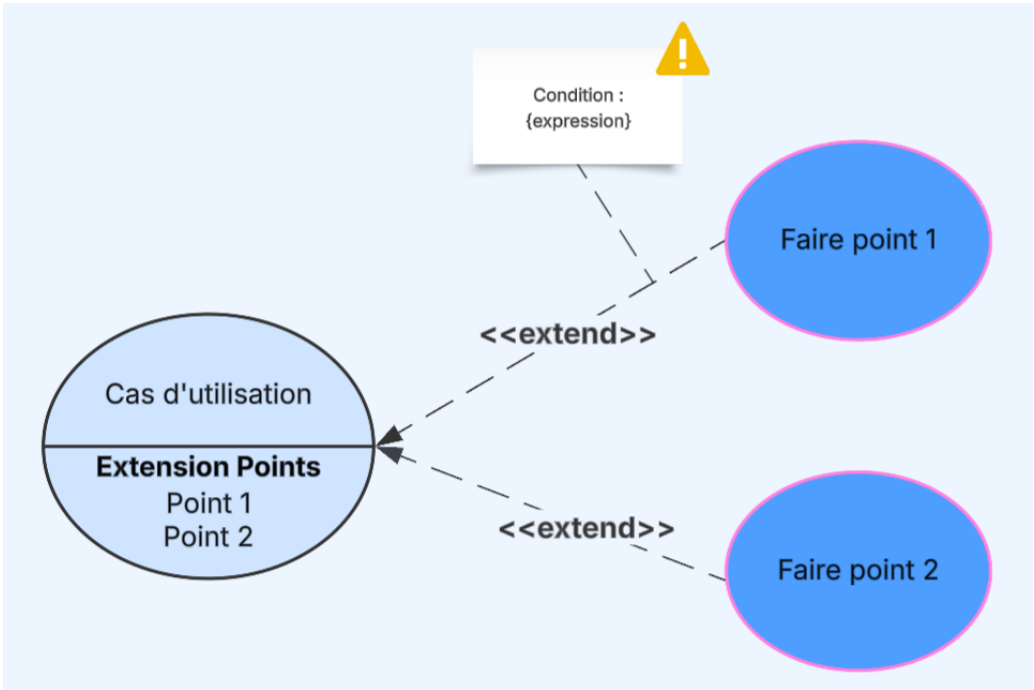

FIGURE 7 – Points d'extension

4. **La généralisation** : un cas d'utilisation peut être spécialisé en plusieurs variantes. Par exemple : le cas d'utilisation *Effectuer un paiement* peut être généralisé en *Payer depuis compte courant* et *Payer via PayPal*.

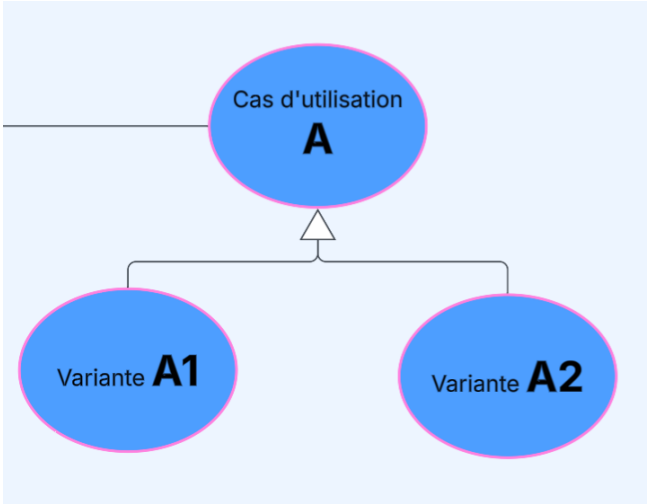

FIGURE 8 – Généralisation de cas d'utilisation

De même, un acteur peut être spécialisé en plusieurs variantes. Par exemple : l'acteur *Client* peut être spécialisé en *Nouveau client* et *Client fidèle*.

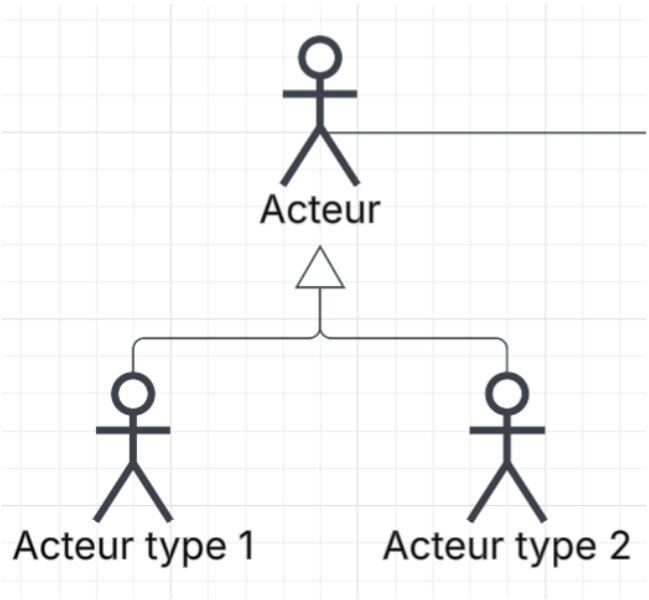

FIGURE 9 – Généralisation d'acteurs