

LA MODÉLISATION AVEC UML

Au sommaire de ce chapitre :

1. *Introduction.*
2. *Modélisation.*
3. *Langage de modélisation UML.*
4. *Historique d'UML.*
5. *L'utilisation d'UML.*
6. *Caractéristiques d'UML.*
7. *Les avantages et les inconvénients d'UML.*
8. *Les vues d'un système.*
9. *Les diagrammes d'UML.*
10. *Diagramme d'activité (DAC).*
11. *Conclusion.*

Chapitre 01 : La modélisation avec UML

1.1. Introduction

Au début des années 1990 la programmation par objet prend de l'importance, la nécessité d'une méthode qui lui soit adaptée devient évidente. Plus de cinquante méthodes apparaissent au milieu de cette année (BOOCH, CLASSE-RELATION, FUSION, HOOD, OMT, OOA, OOSE, etc.), mais aucune de ces méthodes ne parvient à s'imposer. Le consensus se fait autour de trois méthodes : OMT de James Rumbaugh, BOOCH de Grady Booch et OOSE d'Ivar Jacobson, ainsi la fusion de ces méthodes est l'origine de la création d'UML.

UML (Unified Modeling Language) est un langage de modélisation graphique (n'est pas une méthode) qui a été publié par l'OMG (Object Management Group). UML comporte 14 types de diagrammes. Il permet de représenter, modéliser et de communiquer les divers aspects d'un système d'information.

L'objectif de ce chapitre est d'expliquer les notions fondamentales d'UML en se focalisant sur le diagramme d'activité.

1.2. Modélisation

1.2.1. Définition

Modéliser c'est décrire de manière visuelle et graphique les besoins et les solutions fonctionnelles et techniques de votre projet logiciel [1].

D'autre part un modèle est synonyme de théorie, mais avec une connotation pratique : un modèle, c'est une théorie orientée vers l'action qu'elle doit servir.

1.2.2. But de modélisation

La modélisation a plusieurs buts tels que [2]:

- La modélisation permet de réduire la complexité de système en ignorant les détails qui n'influencent pas son comportement de manière significative.
- Modéliser un système avant sa réalisation permet de mieux comprendre le fonctionnement de ce système.
- Un modèle est un langage commun, précis qui est connu par tous les membres de l'équipe, donc c'est un vecteur privilégié pour communiquer.
- La modélisation permet de décomposer les tâches d'un système et de les automatiser, c'est également un facteur de minimisation et réduction des coûts et des délais.

1.2.3. Les types de modélisation

Categories de langages			
Langage informel		Langage semi-formel	Langage formel
<i>Simple</i>	<i>Standardisé</i>		
Langage qui n'a pas un ensemble complet de règles pour restreindre une construction.	Langage avec une structure, un format et des règles pour la composition d'une construction.	Langage qui a une syntaxe définie pour spécifier les conditions sur lesquelles les constructions sont permises.	Langage qui possède une syntaxe et une sémantique définies rigoureusement. Il existe un modèle théorique qui peut être utilisé pour valider une construction.
Exemples de langages ou méthodes			
Langage naturel.	Texte structuré en langage naturel.	Diagramme Entité-Relation, Diagramme à Objets.	Réseaux de pétri, Machines à états finis, VDM, Z.

Figure 1.1 : Les types de modélisation [3].

Les méthodes de spécification (conception) des systèmes informatiques :

- ✓ **Les méthodes informelles** : Une spécification est dite informelle si elle est exprimée au moyen d'un langage informel, qui rend la modélisation imprécise et parfois ambiguë, avec

un raisonnement humain utilisable pour l'analyse et la validation de la spécification qui conduit à des erreurs de compréhension et de vérification.

- ✓ **Les méthodes semi-formelles** : Le processus de modélisation semi-formelle utilise comme outil de modélisation un langage textuel ou graphique pour lequel une syntaxe précise est définie ainsi qu'une sémantique assez faible qui permet une certaine dose de contrôle et d'automatisation de quelques tâches.
- ✓ **Les méthodes formelles** : Les méthodes formelles sont des techniques permettant de raisonner rigoureusement, à l'aide de logique mathématique, afin de démontrer leur validité par rapport à une certaine spécification et d'éliminer les ambiguïtés, les malentendus et les mauvaises interprétations qui peuvent survenir dans la description en langage naturel.

1.3. Langage de modélisation UML

UML "Unified Modeling Language", est un langage de modélisation graphique et textuel basé sur l'approche par objet qui permet de visualiser, spécifier, construire et décrire les aspects et les artifices d'un système logiciel.

La naissance d'UML est le résultat de la fusion des trois méthodes OMT, BOOCH et OOSE, qui a été publiée par OMG en 1997.

Les créateurs d'UML insistent tout particulièrement sur le fait qu'UML est un langage de modélisation et non une méthode.

1.4. Historique d'UML

Il existe plusieurs méthodes de conception objet, et parmi ces méthodes on a trois méthodes importantes : la méthode BOOCH de Grady Booch (qui mettait l'accent sur le design et la construction des systèmes logiciels), OMT de James Rumbaugh (qui insistait sur l'analyse des systèmes logiciels) et la méthode OOSE de Ivan Jacobson (se consacrait au business engineering et à l'analyse des exigences). Ces trois analystes ont décidé de constituer un langage commun, en 1996 la version 0.9 d'UML a été présentée [4].

La figure ci-dessous montre l'évolution d'UML depuis sa création.

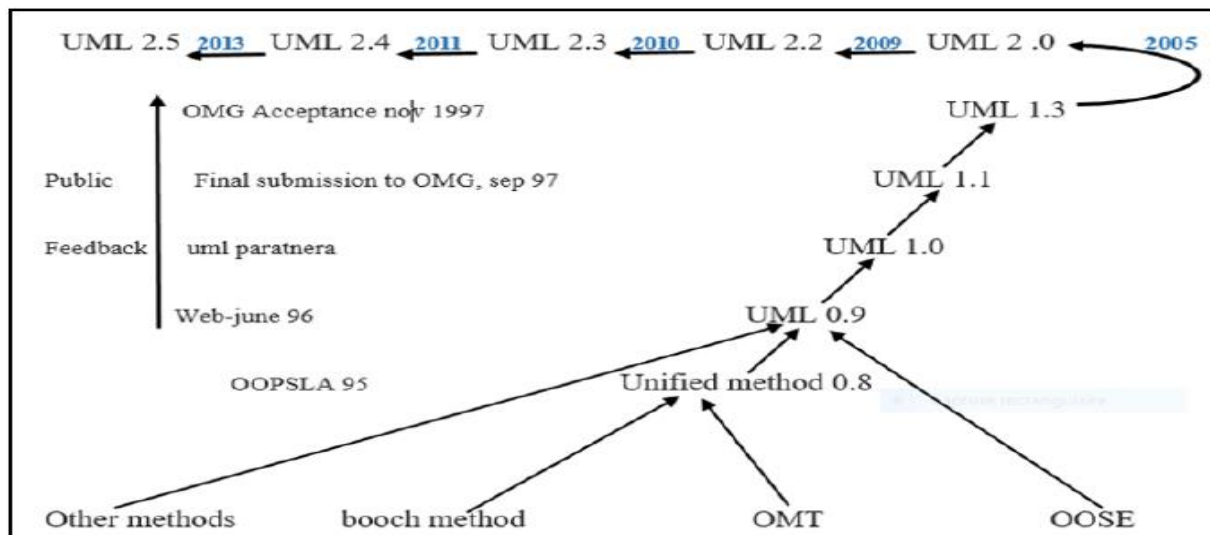


Figure 1.2 : L'histoire d'UML [5].

1.5. L'utilisation d'UML

UML est utilisé pour spécifier, visualiser, vulgariser, modifier, construire et décrire les documents nécessaires au bon développement d'un logiciel orienté objet. Il offre un standard de modélisation, pour représenter l'architecture logicielle [6].

Grâce aux outils de modélisation UML, il est également possible de générer automatiquement tout ou partie du code d'une application logicielle, par exemple en langage Java, à partir des divers documents réalisés.

L'OMG définit les objectifs de l'UML comme suit [7] :

- ✎ Fournir aux concepteurs de systèmes, ingénieurs logiciels et développeurs de logiciels des outils pour l'analyse, la modélisation, la conception et la mise en œuvre de systèmes logiciels.
- ✎ Faire progresser l'industrie en permettant l'interopérabilité des outils de modélisation visuelle orientés objet.

UML répond aux exigences suivantes:

- ✎ Fixer une définition formelle d'un méta-modèle basé sur une norme Meta-Object Facility (MOF) commune qui spécifie la syntaxe abstraite de l'UML.
- ✎ Fournir une explication détaillée de la sémantique de chaque concept de modélisation UML.

- ✎ Définir des moyens grâce auxquels les outils UML peuvent être mis en conformité avec cette spécification.

1.6. Caractéristiques d'UML

UML est caractérisé par [8]:

- UML est basé sur un méta-modèle :

Le modèle fourni par UML est valable pour n'importe quel langage de programmation car UML est un moyen d'exprimer des modèles objet en faisant abstraction de leur implémentation. C'est un langage qui s'appuie sur un méta-modèle, un modèle de plus haut niveau qui définit les éléments d'UML (les concepts utilisables) et leur sémantique (leur signification et leur mode d'utilisation).

- UML cadre l'analyse objet, en offrant :
 - différentes vues (perspectives) complémentaires d'un système grâce aux diagrammes, qui guide l'utilisation des concepts objets,
 - plusieurs niveaux d'abstraction, qui permettent de mieux contrôler la complexité dans l'expression des solutions objets.
- UML n'est qu'un ensemble de formalismes permettant d'appréhender un problème et de le modéliser.
- UML est un langage de modélisation, il définit les éléments de modélisation ainsi que leurs sémantiques.
- UML n'est pas un langage de programmation, mais il existe des outils qui peuvent être utilisés pour générer du code en plusieurs langages à partir de diagrammes UML. L'UML a une relation directe avec l'analyse et la conception orientées objet.
- UML n'est pas une méthode. Chacun est libre d'utiliser les types de diagramme qu'il souhaite, dans l'ordre qu'il veut. Il suffit que les diagrammes réalisés soient cohérents entre eux, avant de passer à la réalisation du logiciel.
- UML convient pour toutes les méthodes objet.
- UML laisse la liberté de " penser ".

1.7. Les avantages et les inconvénients d'UML

1.7.1. Les points forts

- 👉 UML est un langage semi-formel et normalisé, il est conceptuellement riche [9].
- 👉 UML facilite la compréhension de représentation abstraite complexes grâce à ces diagrammes, et UML cadre l'analyse [9].
- 👉 Le caractère polyvalent et sa souplesse en font un langage de description universel [9].
- 👉 UML considéré comme un médiateur entre langage mathématique et naturel puisque il est basé sur un méta-modèle [10].

1.7.2. Les points faibles

- 👉 UML nécessite un apprentissage et l'expérience.
- 👉 UML a une sémantique floue ou mal défini c.-à-d. manque de sémantique précise, pour certains types de diagrammes [9].
- 👉 Les notations sont parfois redondantes.
- 👉 Le processus (non couvert par UML) est une autre clé de la réussite d'un projet [9].
- 👉 L'intégration d'UML dans un processus n'est pas triviale et améliorer un processus est une tâche complexe et longue [9].
- 👉 UML est complexe et hétérogène car il est utilisé dans différents domaines [11].

1.8. Les vues d'un système : les 4+1 vues de Kruchten

UML est décomposé en plusieurs parties, parmi ces parties on a les vues. Les vues sont des observables du système. Elles décrivent le système d'un point de vue organisationnel, géographique, logique, architectural, etc. En combinant toutes ces vues pour retrouver un système complet.

La figure 1.3 schématise le modèle « 4+1 » vues (Le modèle de Philippe Kruchten). Ce modèle est composé de cinq vues :

- ✓ **Vue logique (logical view)** : c'est la définition du système vu de l'intérieur. Elle explique comment peuvent être satisfaits les besoins des acteurs.

- ✓ **Vue d'implémentation (implimentation view)** : définit les dépendances entre les modules (codes sources, exécutable, etc.) dans l'environnement de développement.
- ✓ **Vue des processus (precess view)** : représente la vue temporelle et technique qui manipulant les notions de : tâches concurrentes, synchronisation, etc.
- ✓ **Vue de déploiement (deployment view)** : décrit l'emplacement géographique et l'architecture physique de chaque élément de système.
- ✓ **Vue des cas d'utilisation (use-case view)** : c'est la description de modèle vu par les acteurs de système. Elle correspond aux besoins attendus par chaque acteur.
 - La vue des cas d'utilisation est construite en premier juste après la réalisation du cahier des charges, elle valide en quelque sorte les autres vues et assure la cohérence globale.

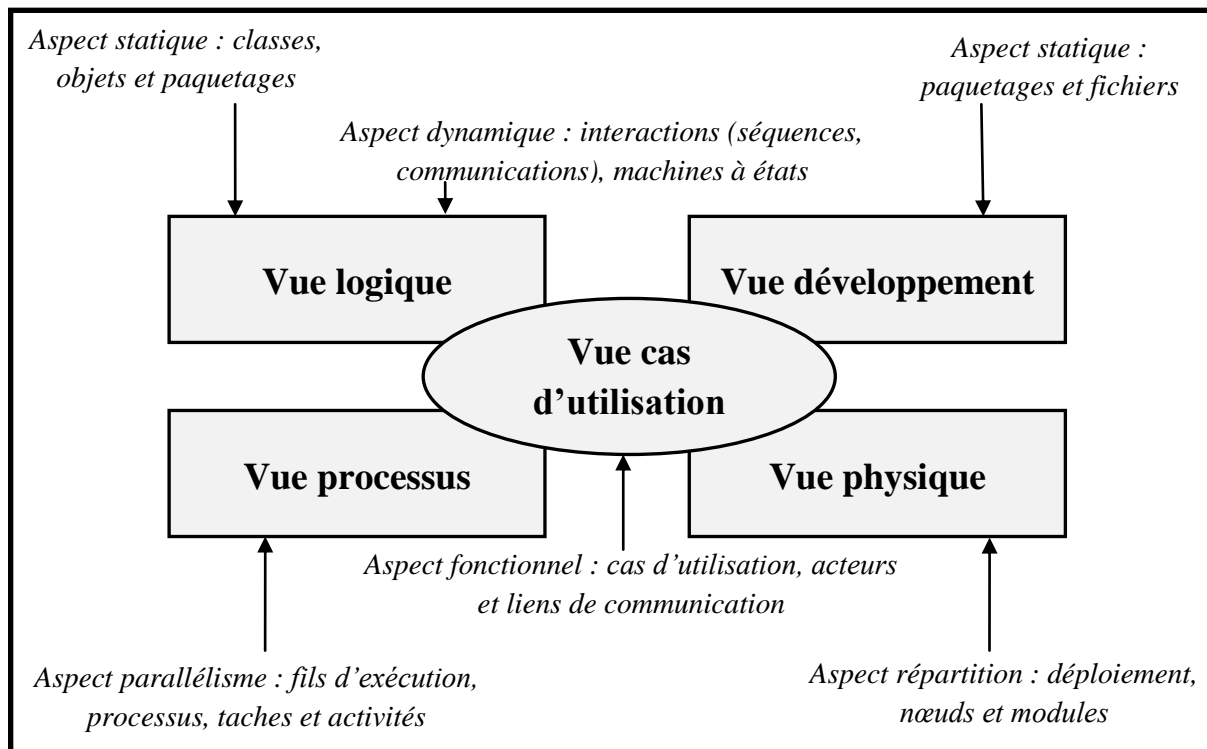


Figure 1.3 : Les vues d'un système [12].

1.9. Les diagrammes d'UML

Pour construire une maison, il nécessite des plans à différents niveaux (vision extérieure, plan des différents étages, plans techniques...), Aussi la réalisation d'une application informatique ou d'un ensemble d'applications est basée sur plusieurs diagrammes. Un diagramme donne à l'utilisateur un moyen de visualiser et de manipuler des éléments de modélisation.

Depuis UML 2.5 il existe quatorze (14) diagrammes représentant autant de vues distinctes pour représenter des concepts particuliers du système d'information. Ces diagrammes sont regroupés dans deux grands ensembles : structurels (ou statiques) et comportementaux (ou dynamiques).

La figure 1.4 représente les 14 diagrammes d'UML.

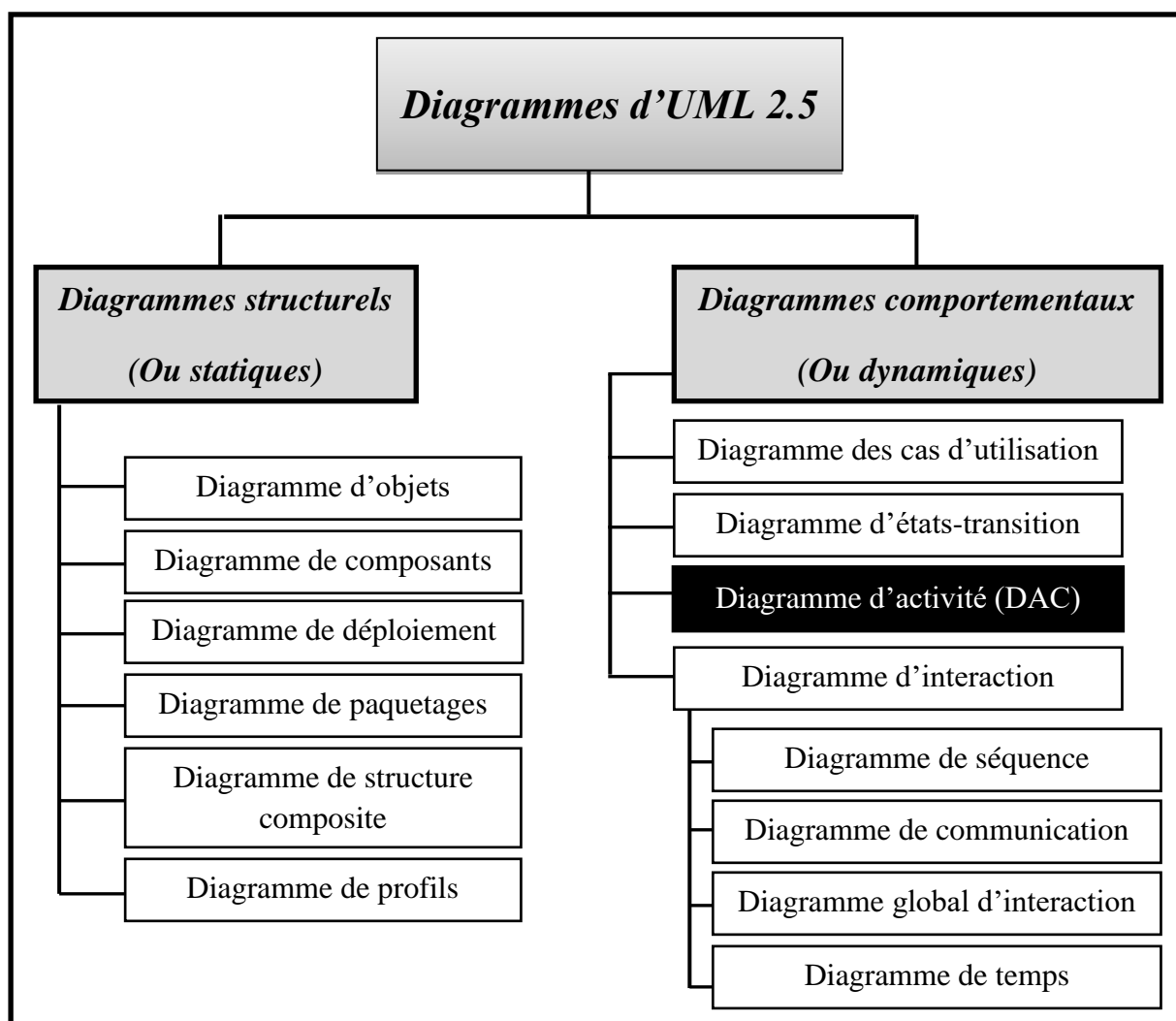


Figure 1.4 : Les 14 diagrammes d'UML.

1.9.1. Les diagrammes structurels (ou statiques)

Ces diagrammes au nombre de six, permettent de visualiser, spécifier, construire et documenter l'aspect statique d'un système informatisé (classes, objets, composants, etc.) [13] [14].

- ✓ **Diagramme de classes (Class diagram)** : Ce diagramme est considéré comme le plus important dans un développement orienté objet, permet d'exprimer l'architecture conceptuelle d'un système, en termes de classes que le système utilise et leurs liens, les attributs de ces classes, les opérations et les relations entre eux.
- ✓ **Diagramme d'objets (Object diagram)** : On dit aussi diagramme d'instance, permet la représentation d'instance des classes et les liens entre instances, il est utile pour vérifier l'adéquation d'un diagramme de classes à différents cas possibles.
- ✓ **Diagramme de composants (Component diagram)** : Permet de montrer les composants du système d'un point de vue physique, tels qu'ils sont mis en œuvre (fichiers, bibliothèques, bases de données...).
- ✓ **Diagramme de déploiement (Deployment diagram)** : Sert à représenter les éléments matériels (ordis, périphériques, réseau, système de stockage ...) et la manière dont les composants du système sont répartis sur ces éléments matériels et interagissent entre eux.
- ✓ **Diagramme de paquetage (Package diagram)** : Le paquetage représente un ensemble homogène d'élément de système (classe, composant, etc.) permettant de regrouper et d'organiser les éléments dans le modèle UML.
- ✓ **Diagramme de structure composite (Composite structure diagram)** : Depuis UML 2.x, permet de décrire sous forme de boîte blanche les relations entre composants d'une classe.
- ✓ **Diagramme de profils (Profil diagram)** : Depuis UML 2.2, permet de spécialiser, de personnaliser pour un domaine particulier un méta-modèle de référence d'UML.

1.9.2. Les diagrammes comportementaux (ou dynamiques)

Ces diagrammes au nombre de huit, permettent de modéliser les aspects dynamique du système réagissant aux événements et de produire les résultats attendus par les utilisateurs [15] [16].

- ✓ **Diagramme de cas d'utilisations (Use case diagram)** : C'est le premier diagramme du modèle UML, il est destiné à représenter les besoins des utilisateurs par rapports au système, il permet d'assurer la relation entre l'utilisateur et les objets que le système met en œuvre.

- ✓ **Diagramme d'activités (Activity diagram)** : C'est un diagramme graphique qui permet de montrer l'enchaînement des activités propres à une opération ou à un cas d'utilisation, comme il est une variante des diagrammes d'états-transitions.
- ✓ **Diagramme d'états-transitions (State machine diagram)** : Il représente le cycle de vie des instances d'une classe, il permet de décrire sous forme de machine à états finis le comportement du système ou de ses composants.
- ✓ **Diagramme de séquences (Sequence diagram)** : Représentation séquentielle du déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs. Il peut servir à illustrer un cas d'utilisation.
- ✓ **Diagramme de communication (Communication diagram)** : Représentation de façon simplifiée d'un diagramme de séquence, il se concentre sur les échanges des messages entre les objets (le diagramme de séquence et diagramme de communication sont deux vues différents mais logiquement équivalentes).
- ✓ **Diagramme global d'interactions (Interaction overview diagram)** : Permet de décrire les enchaînements possibles entre les scénarios préalablement identifiés sous forme de diagramme de séquence (variante du diagramme d'activité).
- ✓ **Diagramme de temps (Timing diagram)** : C'est une nouveauté apparue dans UML 2, il permet de décrire les variations d'une donnée au cours du temps.

1.10. Diagramme d'activité (DAC)

Un DAC est utilisé pour afficher la séquence des activités, pour la modélisation d'entreprise où il est utilisé pour détailler les processus impliqués dans des activités commerciales. Les DAC représentent le flux de travail (workflow) à partir d'un point de départ au point d'arrivée.

Le DAC est un diagramme comportemental d'UML, permettant de représenter le déclenchement d'événements en fonction des états du système et de modéliser des comportements parallélisables (multi-threads ou multiprocessus), dont la séquence des actions et leurs conditions d'exécution. Un DAC permet de grouper et de dissocier des actions. Si une action peut être divisée en plusieurs actions en séquence, vous pouvez créer une activité les représentant.

1.10.1. Présentation générale et concepts de base

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

- ✎ transition,
- ✎ ● nœud initial (état initial),
- ✎ ● nœud final (état final),
- ✎ ⊗ nœud de fin flot (état de sortie),
- ✎ ◇ nœud de décision (choix).

Le formalisme reste identique pour ces nœuds de contrôle. Les concepts spécifiques au diagramme d'activité sont :

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

1.10.2. Intérêts des diagrammes d'activités

Les intérêts des diagrammes d'activités sont [17] :

- ☞ Représenter graphiquement le comportement interne d'une méthode, d'un cas d'utilisation ou plus généralement d'un processus impliquant un ou plusieurs classificateurs (classes / cas d'utilisation / paquetages /...).
- ☞ Utiliser le mécanisme de synchronisation pour représenter les successions d'états synchrones, alors que les diagrammes d'états-transitions sont utilisés principalement pour représenter les suites d'états asynchrones.

- ☞ Modéliser un workflow (flux de travail) dans un cas d'utilisation, ou entre plusieurs cas d'utilisations.
- ☞ Représenter des processus métiers, les cas d'utilisation et le flux d'action.
- ☞ Démontrer la logique d'un algorithme et décrire la logique d'une opération.
- ☞ Simplifier et améliorer n'importe quel processus en clarifiant les cas d'utilisation complexes.
- ☞ Le DAC est le plus approprié pour modéliser la dynamique d'une tâche ou d'un cas d'utilisation, lorsque le diagramme de classe n'est pas encore stabilisé.

1.10.3. Représentation du diagramme d'activité

1.10.3.1. Les nœuds

- ☞ **L'activité** : Une activité représente le comportement d'une partie d'un système en termes d'actions et de transitions. Une activité regroupant des nœuds et des arcs est appelée un groupe d'activité [18].
- ☞ **Nœud d'activité** : C'est une classe abstraite permettant de représenter les étapes du flux d'une activité [13].

Il existe trois types de ces nœuds :

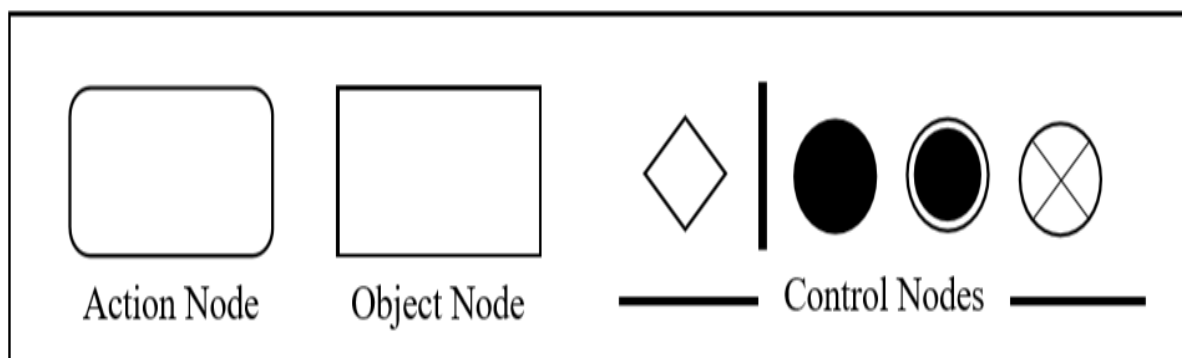


Figure 1.5 : Notation nœuds d'activités.

- **Nœud d'objet** : C'est une méta-classe abstraite permettant de définir les flux d'objets dans les diagrammes d'activités. Il représente l'existence d'un objet généré par une action dans une activité et utilisé par d'autres actions. La figure 1.6 donne deux représentations équivalentes de flux d'objets entre deux actions. La première utilise des pins et l'autre utilise la notation d'un nœud objet.

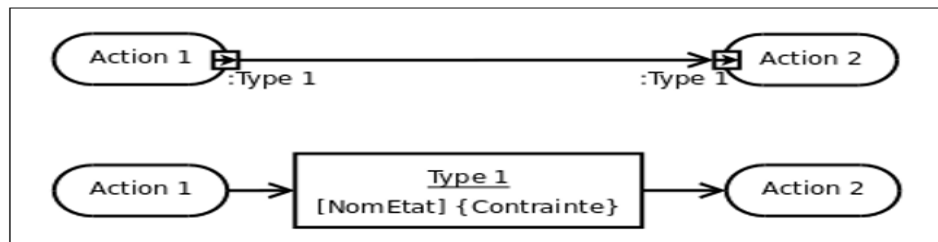


Figure 1.6 : Représentations d'un flux d'objet.

- **Nœud exécutable :** C'est une classe abstraite pour les nœuds d'activités qui peuvent être exécutés. Il possède un gestionnaire d'exception qui peut capturer les exceptions levées par le nœud ou par l'un des nœuds imbriqués.
- **Action :** C'est le plus petit traitement qui puisse être exprimé en UML. L'exécution d'une action peut être une transformation ou un calcul dans le système modélisé. Nous citons ci-dessous quelques types d'actions prédéfinis dans la notation UML :
 - Action appeler (Call Operation).
 - Action comportement (Call Behaviour).
 - Action envoyer (Send).
 - Action accepter événement (Accept Event).
 - Action accepter appel (Accept Call).
 - Action répondre (Reply).
 - Action créer (Create).
 - Action détruire (destroy).
 - Action lever exception (raise exception).
- **Nœud de contrôle :** C'est un nœud d'activité abstrait utilisé pour coordonner les flots entre les nœuds d'une activité. Il existe plusieurs types de nœuds de contrôle, la figure 1.7 présente ces types :

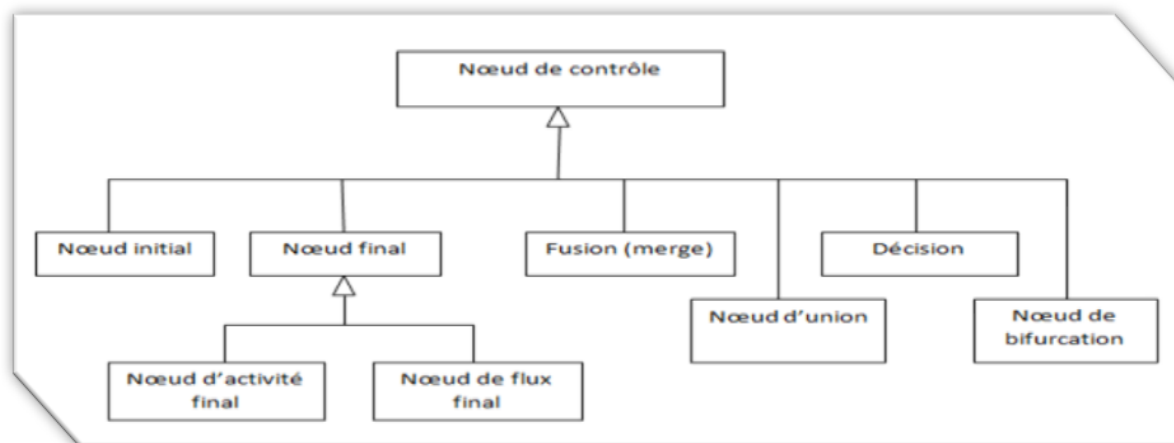



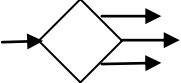


Figure 1.7 : Les types des nœuds de contrôle.

Dans le tableau 1.1 on a la description des nœuds de contrôle :

Elément	Notation	Description
Nœud initial		C'est un nœud de contrôle à partir le duquel le flot débute. Il possède un arc sortant et pas d'arc entrant. Dans une activité, on a seulement un nœud initial.
Nœud d'activité final		Dans un nœud final d'activité, lorsque l'un de ses arcs entrants est activé, l'exécution de l'activité en cours s'achève, et tout nœud ou flux actif au sein de cette activité est abandonné.
Nœud de flux final		L'arrivée du flux d'exécution à un nœud final de flux, implique la terminaison du flux de ce dernier. Mais cette fin n'a aucun effet sur les autres flux actifs de l'activité.
Décision		Il permet de faire le choix entre plusieurs flux sortants. Ces flux sont sélectionnés en fonction de la condition de garde qui est associé à chaque arc sortant. Si aucun arc en sortie n'est franchissable, le modèle est mal formé, et l'utilisation d'une [else] garde est recommandée.

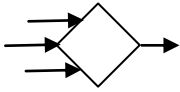
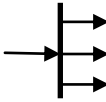
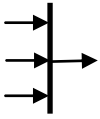
Fusion (Merge)		C'est un nœud de contrôle pour accepter un flot en sortie parmi plusieurs flots en entrées, l'utilité de ce nœud n'est pas pour synchroniser des flux concurrents.
Nœud de bifurcation (fork)		C'est le nœud de contrôle qui joue le rôle de séparation de flux d'entrée en plusieurs flots concurrents en sortie.
Nœud d'union (Join)		Il synchronise des flots multiples. Il possède plusieurs arcs entrants et un seul arc sortant. Ce dernier ne peut être activé que lorsque tous les arcs entrants sont activés.

Tableau 1.1 : Description des nœuds de contrôle.

1.10.3.2. Les arcs

Un arc d'activité est une connexion dirigée entre deux nœuds d'activités. On a trois types des arcs :

- ✓ **Flot de contrôle** : C'est l'arc qui ne transmet pas les données, il décrit le séquençement de deux nœuds d'activité.
- ✓ **Flot d'objet** : C'est l'arc qui permet de transmettre les données entre les nœuds d'objet.
- ✓ **Handler d'exception** : C'est une activité qui spécifie un organisme à exécuter. Il possède un pin d'entrée du type de l'exception qu'il gère, et il est lié à l'activité protégé par un arc.

1.10.3.3. Partitions

Partitions appelés aussi couloires ou bien ligne d'eau, elles sont utilisées pour organiser les nœuds d'activité disposés dans un diagramme d'activités par le biais de regroupements. Les partitions divisent le diagramme en colonnes comme contenus des actions qui sont menées par le groupe responsable.

1.10.4. Exemple

Dans cet exemple (voir la figure 1.8 ci-dessous), on présente un diagramme d'activité qui explique la méthode de résolution d'une équation mathématique de 2^{ème} degré à une seule variable.

D'abord, lire les coefficients d'équation de 2^{ème} degré (c.-à-d. les données : a, b, c), en suite on calcule la variable delta qui égale à $(b^2 - 4ac)$. Si delta inférieur à zéro ($\text{delta} < 0$) alors pas de solution, sinon ($\text{delta} \geq 0$) on a deux cas : soit delta égal à zéro ($\text{delta} = 0$) et la solution est : $X = b/2a$, sinon on calcule la racine carrée de delta (R) et dans ce cas-là on a deux solutions : $X_1 = (-b + r)/2a$ et $X_2 = (-b - r)/2a$. Enfin, on affiche le résultat final de cette solution.

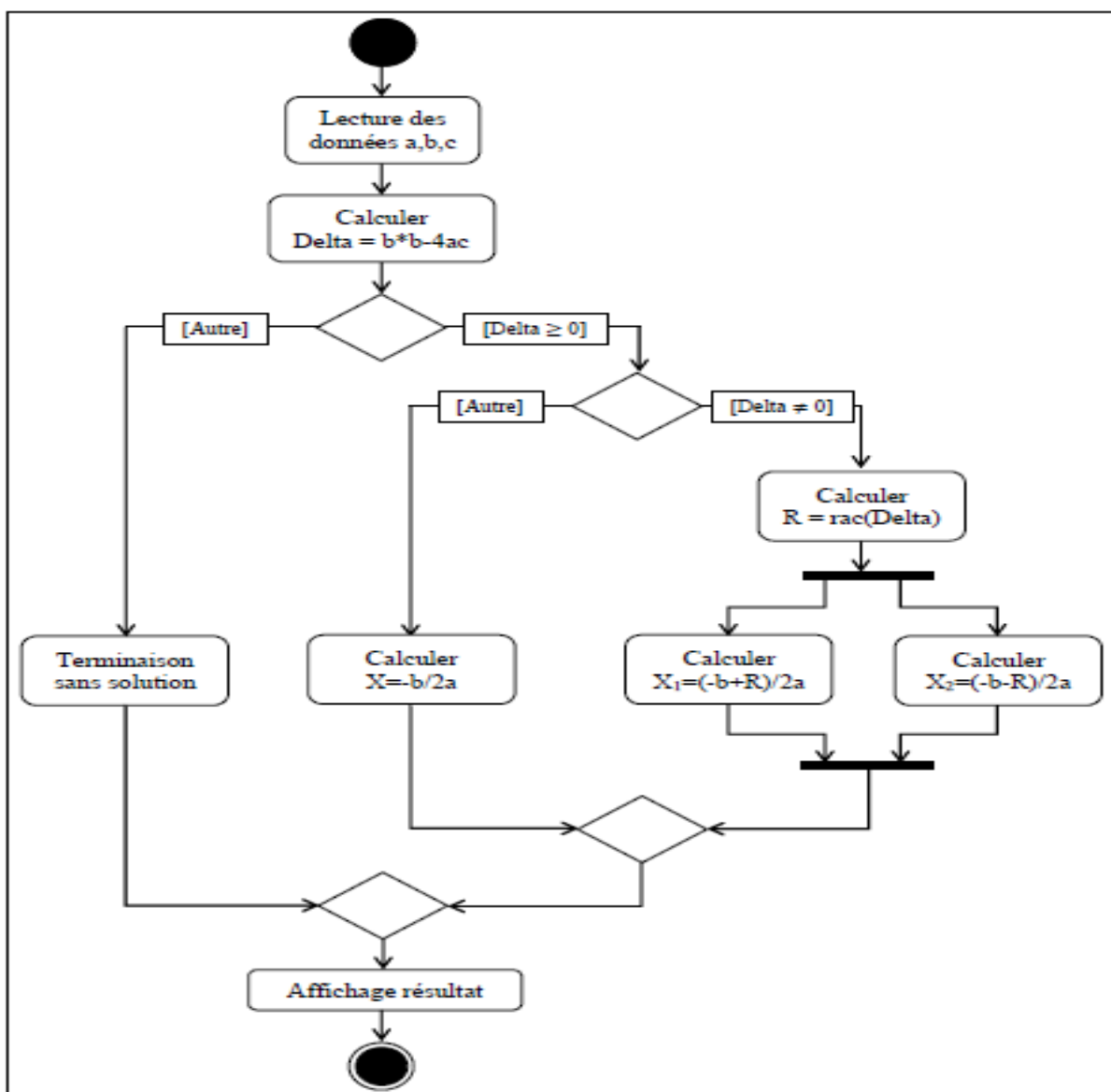


Figure 1.8 : exemple d'un diagramme d'activité [18].

1.11. Conclusion

Dans ce chapitre nous avons présenté le langage de modélisation unifié UML. Nous avons commencé par une introduction à la modélisation des systèmes, une définition de langage, son évolution et ses diagrammes. Ensuite nous avons détaillé les diagrammes d'activités d'UML et leur composition puisqu'ils constituent la base de notre travail.

Dans le chapitre suivant nous allons aborder l'ontologie OWL-S qui représente le modèle cible de notre approche.