Spécification Formelle des systèmes d'information

Dr. Dairi Abdelkader
Université des Sciences et de la Technologie
d'Oran Mohamed-Boudiaf USTOMB

23 octobre 2021

Table des figures

Table des matières

Table des figures

Preface

Les données prennent une grande importance au sein des entreprises, et leur volume ne cesse de s'accroître d'une façon exponentielle. De plus les données sont devenues le capital des entreprises et en même temps une ressource précieuse qu'il faut maintenir et protéger.
Les système d'information sont construit au tour des données, en proposant une architecture adéquate composée d'un ensemble de ressources matérielles et immatérielles, avec des objectifs précis et permettent de collecter, stocker , traiter et publier les données sous forme compréhensible et facile à interpréter.
Le présent polycopié est le support de cours : "***Spécification Formelle des Systèmes d'Information (SFSI)***" destiné pour la première année Master de la spécialité ***Système d'Information et Données (SID)*** au département informatique à l'Université des Sciences et de la Technologie d'Oran Mohamed-Boudiaf USTOMB .
Ce manuscrit se focalise sur les différentes méthodes de conception des SI qualifiées de formelles telles que : NIAM, Express-G et Z utilisant un formalisme mathématique et parfois logique. D'autre part d'autres méthodes sont explorées utilisant une notation visuelle (graphique) basée sur des diagrammes. En effet ces méthode sont qualifiées d'être *Semi-Formelle*, les méthodes utilisant le concept Objet font partie de cette catégorie telles que : OMT, OOD, OOSE, et UML.
Ce manuscrit est organisé en quatre chapitres :

Chapitre 1 : ce chapitre présente une introduction aux Systèmes d'information, leur définition , organisation , composition et propriétés ; nous aborderons aussi les méthodes de conception.

Chapitre 2 : ce chapitre décrit les spécifications, méthodes et langages formelles utilisés pour la conception des systèmes d'information suivant le volet données ainsi que le volet dynamique.

Chapitre 3 : Dans ce chapitre différentes méthodes et langages sont étudiés en l'occurrence : la méthode Merise , NIMA, Express , Z , OMT , UML.

Chapitre 4 : Ce chapitre présente un cas d'étude en utilisant la méthode NIAM, le symbolisme de cette méthode est étudiée avec plusieurs exemples.

Chapitre 1

Introduction Aux Systèmes d'information (SI)

1.1 Introduction

De nos jours, les systèmes d'information SI sont devenus un élément central du fonctionnement d'une organisation [Stair and Reynolds, 2015, Jessup and Valacich, 2008]. Un système d'information peut être vu comme un ensemble de ressources représentées par : un personnel, des processus, des données, du matériels (télécommunication, informatique ...).
L'objectif est d'effectuer une série d'opérations sur les données notamment : la collecte, le stockage, la manipulation, la gestion, l'analyse, le transport, la diffusion sous différents formats : textes et multimédias.
Malgré l'avancée technologique, l'unité principale contenant les données reste les fichiers, les SI ont débuté par des systèmes de gestion de fichiers ou un logiciel est élaboré sur mesure pour gérer un cas particulier a la fois les données et le codage métier, par exemple une application de gestion de stock ou calcul de la paie pour le personnel d'une organisation.
A chaque fois qu'un besoin se présente, un nouveau logiciel doit être conçu, les systèmes de gestion de bases de données (SGBD) ont été proposés afin d'unifier la façon de gérer et manipuler les données en utilisant un langage particulier nommé en anglais "*Structured Query Languge*" (SQL).
Les systèmes d'information basés sur les applications exploitant des bases de données ont évolué et se sont spécialisés. Nous pouvons citer à titre d'exemple les systèmes d'informations géographiques (SIG), conçus pour la gestion et la présentation des données spatiales. D'autre part, certains systèmes d'information de gestion particulière tels que la gestion de relation clientèle ou la gestion des ressources humaines, ou encore gestion de la chaîne logistique.

Le regroupement de plusieurs applications de gestion (métier), exploitant la même base de données a donné naissance aux progiciels de gestion intégrés (PGI), en anglais : Enterprise Resource Planning. L'avantage principal des ERP est d'avoir une seule base de données contenant toutes les informations nécessaires à l'organisation modulaire reflétant sa structure et représentant ces processus métier sous forme de workflow.
Afin de mieux gérer son système d'information l'entreprise ou l'organisation a besoin d'une infrastructure matérielle, un personnel qualifié et des procédures de travail clairement définies.
Les systèmes d'information délivrent l'information demandée aux personnes autorisées et sous la forme appropriée. Souvent, ils sont dotés aussi de tableau de bord qui servent à piloter l'entreprise en améliorant la prise de décision aux décideurs (Directeurs ou Managers). D'autres fonctionnalités ont été ajoutées telles que : la gestion du contenu , business intelligent et l'incorporation des modèles prédictifs appartenant au domaine de l'intelligence artificielle. Ces approches sont basées sur les données, nous pouvons citer : les réseaux de neurones ou d'autre modèles statistique en l'occurrence les Machines à vecteurs de support (en anglais SVM).
Aux fils des années, le volume de données stockées dans les bases de données, a rendu difficile voir impossible dans certains cas d'analyser et traiter ces données. Ce qui a pousser les concepteurs, ingénieurs et chercheurs de trouver d'autres mécanismes et technologies afin de mieux gérer Les nouvelles mégas bases de données, ce qui a donner naissance aux environnements : Big Data.

FIGURE 1.1 – Architecture d'un système d'information

L'infrastructure matérielle hébergeant les systèmes d'information a aussi évolué et a subit beaucoup changements au fils du temps. Au début de l'informatisation ou la numérisation des données qui ont été stockées sur des disques de machines souvent appelées Serveur, ensuite des baie de disques résidant dans un endroit spécial en l'occurrence "DATA CENTER" ont apparu.
Grace aux avancées technologiques surtout dans le domaine de la télécommunication et l'amélioration du débit internet, une nouvelle plateforme est apparue permettant aux organisations ou les entreprises de louer un espace de stockage d'une capacité importante, une puissance de calcul et mémoire vive afin d'héberger leur système d'information sur internet, ces plateformes sont connues sous le nom Nuages ou Cloud en anglais. L'infrastructure Cloud est une solution efficace , optimale et sécurisée pour les entreprises souhaitant minimiser le coût de gestion de leur système d'information.

1.2 Données et Information

"*La connaissance s'acquiert par l'expérience, tout le reste n'est que de l'information.*" (Albert Einstein)
Une connaissance est basée souvent sur des informations, qu'on trouve aussi dans le langage de communication entre les êtres humains. On peut dire aussi que l'information véhicule des données pré-traitées, ou le cumul d'information représente un réel support des connaissances.
Nous utilisant souvent indifféremment les termes : donnée et information. En fait, une donnée subit des traitements ou des transformations pour devenir une information qu'un être humain peut l'interpréter et attribuer par la suite une signification dans un contexte bien défini.
Un système d'information enregistre les données sur des support numérique tel que les disques dur, ou une donnée occupe un espace mémoire en bit ou octets tout dépend du volume de cette dernière.
Exemple : *La note d'examen de mathématique est une donnée pour un étudiant, cependant la moyenne de ces notes obtenues est une information qui reflète ces performances*.
En analysant l'information autour de la moyenne obtenue, nous pouvons en déduire d'autre information par rapport à l'étudiant, Ses points forts et faiblesse, afin de prendre quelques décisions concernant par exemple quels sont les cours de support dont l'étudiant a besoin.
Par ailleurs, une information peut être représentée de plusieurs façons comparée à la donnée source qu'elle a générée. Une information peut être communiquée sous plusieurs formes : texte, graphique, paroles ou multimédias ex : (images, vidéos). Le tableau 1.1 illustre quelques différences entre la donnée et l'information. La figure 1.2 représente la relation entre données, information et connaissance. Nous pouvons re-

Data	Information
Une donnée est une entité brute sans contexte et sans interprétation	Information est le résultat du traitement d'une donnée.
Donnée est indépendante de l'information.	Information dépendante des données.
Mesurée en bits et octets.	Mesuré en unités significatives comme le temps, la quantité, …..
La donnée peut être structurée sous forme : tabulaires, matricielle, arbre de données, graphique	Langage, idées et pensées basés sur les données fournies.

TABLE 1.1 – Tableau comparative : Données vs Information [Stair and Reynolds, 2015]

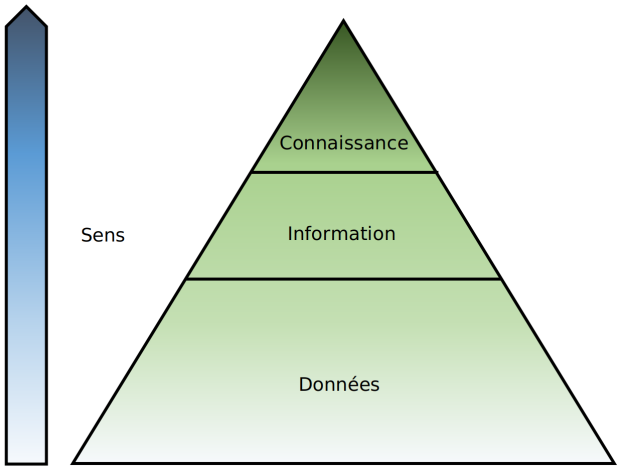

FIGURE 1.2 – Données vs Information

marquer qu'une donnée a peu de sens par contre la connaissance est un concentré d'informations proposant plus de sens et de signification.

1.3 Organisation et Entreprise

Une organisation est un ensemble formel de ressources de types et de natures différente y compris la ressource humaine, afin d'atteindre des objectifs courts, moyens et longs termes. Une entreprise ou une administration sont des occurrences d'une organisation.
Une organisation utilise constamment pour son fonctionnement un budget (de l'argent), un personnel, des matériaux, des machines et d'autres équipements, des données et d'informations.
Cependant dans une organisation on a toujours besoin de prendre des décisions sous formes d'actions qui impactent directement l'avenir de cette organisation. Ces décisions

sont souvent prises par des décideurs (directeurs , managers ...) en se basant sur des information qui proviennent d'un système d'information.
La figure 1.3 décrit la structure interne d'une organisation. On distingue le système opérant qui sert à appliquer les actions afin de produire des biens ou des services. Les systèmes opérant sont également appelés système opératoire ou encore un système de production. Le système de décision nommé également système de pilotage a une

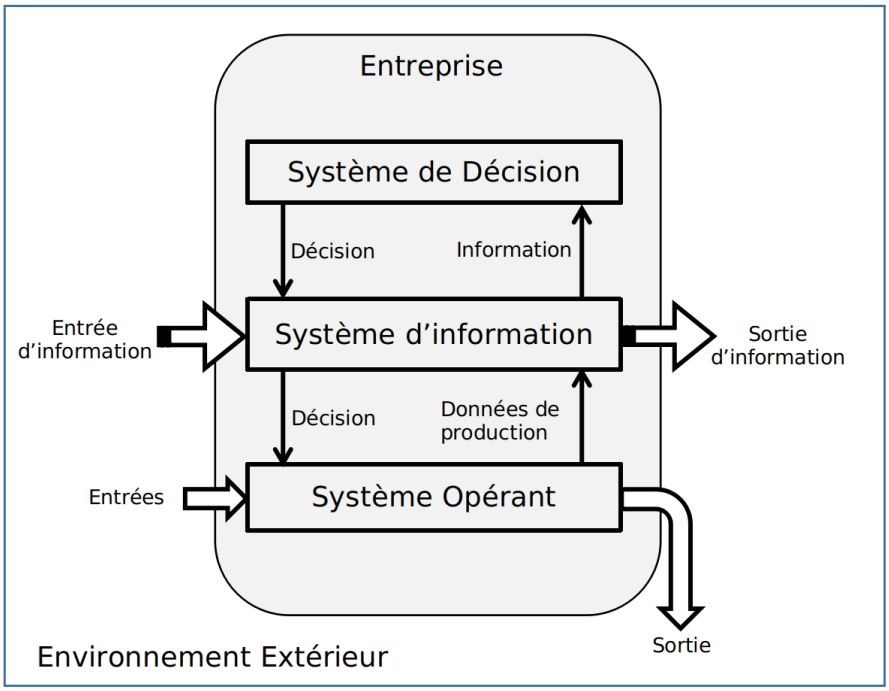

FIGURE 1.3 – Structure interne d'une organisation

lourde responsabilité d'assignant ses objectifs et veuille à ce qu'ils soient atteints. Les décideurs font une analyse croisée de l'environnement extérieur et la situation interne de l'organisation. Le système d'information joue un rôle très important qui orchestre le flux d'information et d'activités au sein d'une organisation. Il existe une synergie entre les différents systèmes composant une organisation et le monde externe, souvent on appelle ça une relation en anglais Business-2-Business (B2B).

1.4 Composantes des SI

Un système d'information est une collection de ressources : humaine , matérielle , logicielle ainsi qu'une séries de procédures de travail. Chaque ressource joue un rôle important pour le fonctionnement du SI.

Ressource Humaine :

La Ressource Humaine est également appelée : personnes , personnel ou des em-

ployés , dont rôle principal est d'utiliser ou manipuler les outils que propose le SI d'un coté et d'un autre coté veuille à le maintenir, le gérer , le protéger et le sauvegarder. On distingue deux catégories de ressource humaine l'utilisateur final et l'informaticien.
Les utilisateurs finaux appartiennent aux personnes du métier, par exemple les financiers , les comptables , les gestionnaires de stock,....
Par ailleurs, les individus qui veillent au bon fonctionnement du SI en l'occurrence l'équipe des informaticien ou aussi nommé en anglais : Information Technology (IT) Team, sont amené à effectuer des taches de maintenance sur le système et aussi d'optimiser ces paramètres afin de garantir un temps de réponse acceptable.

Ressource Matérielle :

Les ressources matérielles sont toutes équipements utilisé pour stocker, transformer, représenter, transporter ou véhiculer de l'information. Nous pouvons les classifier en plusieurs classes, les machines dédiées pour le stockage des données que nous appelons les serveurs de base données, une autre classe pour les équipements de transport (transmission) des données tel que les réseaux en l'occurrence : routeurs, switch , borne Wifi .... La dernière classe concerne les équipements qui consomment l'information tels que les PCs , terminaux mobiles et les équipements qui représentent (publient) l'information tels que les écrans d'affichage , imprimantes , ...

Ressource logicielle :

Parmi les logiciels largement utilisés dans tout système d'information les systèmes de gestion de base de données (SGBD) , dont le rôle est de stocker les données sous structure (forme) optimale permettant de définir et manipuler les données en toute sécurité via une façon standard en utilisant un langage standard et structuré. D'autre part, il existe une autre catégorie de logiciel avec un objectif bien défini et qui concerne une partie des données bien précise.
Il s'agit des applications métier de gestion , qui se focalisent sur un métier généralement, nous pouvons citer a titre d'exemple : application pour le calcul de la paie , gestion des ressources humaines , gestion des finances , gestion des stocks, gestion des clients ...
Par ailleurs, il existe des solutions qui intègrent toutes les applications de gestion métier dans un seul environnement nommé Progiciel de Gestion Intégré ou en anglais ERP (Enterprise Resource Planning).

Procédures :

Une procédure définit les étapes à suivre pour obtenir un résultat final spécifique tel que la saisie d'une commande client, le paiement d'une facture fournisseur ou la demande d'un rapport d'inventaire actuel.
Une description détaillée des procédures facilite, simplifie et optimise les processus de

travail afin d'atteindre les objectifs escomptés. Tout en précisant clairement les tâches à effectuer, qui doit les faire, comment, et le moment opportun pour le faire.
Lorsque les employés sont bien formés et suivent les procédures de travail qu'ils trouvent au niveau du Système d'information, ils deviennent plus efficaces, perdre moins de temps, réduisent les coûts et utilisent mieux les ressources de l'entreprise.

Données :

Les systèmes d'information sont conçus autour des données, qui sont le patrimoine et le capital d'une organisation. La mémorisation des données se fait à travers une conception logique et une structure physique, afin de stocker un maximum de données de qualité d'un côté et d'un autre côté de pouvoir consulter et manipuler un volume important des données en toute simplicité.
La figure 1.4, illustre la consommation de données annuelle ainsi que des prévisions pour les années qui viennent. Les sources de données d'un système d'information pro-

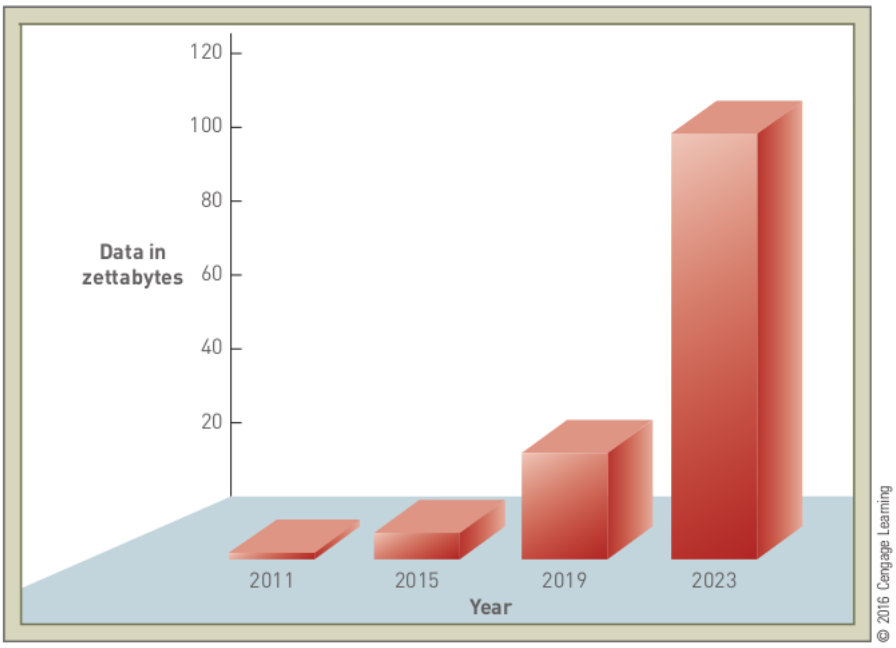

FIGURE 1.4 – la quantité d'informations numériques générées chaque année et son évolution [Stair and Reynolds, 2015]

viennent de l'intérieur de l'organisation et de l'Environnement Extérieur. Les données générées par les processus métier de l'organisation par exemple la production d'articles , l'état du stock , échange entre les différent départements ou de services composant l'organisation en question.
Par ailleurs, les données externes peuvent alimenter aussi le système d'information dans le cadre d'une relation B2B, nous pouvons citer par exemple l'échange de données avec les banques , le réseau de fournisseurs et de clients. Il faut noter qu'un système

d'information peut lui même exporter des données à d'autres SI, sous forme de e-mails , fichiers texte , fichiers de données ...

1.5 Les propriétés des SI :

L'objectif principal d'un système d'information est de fournir les informations nécessaires à des utilisateurs sous forme convenable afin de les accompagner à accomplir leurs tâches et responsabilités respectives au sein d'une organisation. Un système d'information doit assurer un ensemble de propriétés de différents types : statique , dynamique , sûreté et vivacité.

Les propriétés statiques :

Elles caractérisent les mécanismes qui doivent être mis en place par le système d'information afin de garantir la cohérence, consistance et intégrité des données tout au long de leur existence.

Les propriétés dynamiques :

Elles font en sorte de gérer les événements et les actions (tâches) correspondantes dès leur occurrence ainsi que leur ordonnancement, il s'agit d'orchestrer les flux de tâches et données souvent appelé (en anglais) : work-flow et data-flow.

Les propriétés de sûreté :

Cette propriété regroupent les méthodes , techniques et mécanismes permettant d'assurer un accès sécurisé vers les données hébergées au sein d'une organisation via le système d'information. En effet, la sécurité commence de l'authentification des utilisateurs , hachage et/ou cryptage des flux de données échangés , le contrôle d'accès : via les droits octroyés (autorisations) et les listes d'accès (en anglais access lists).
Un point important dans le volet sécurité est de toujours utiliser des firewalls avec antivirus afin de protéger le SI des attaques qui viennent de l'extérieur (internet)

Les propriétés de vivacité :

Un système d'information doit être doté de fonctionnalités et mécanismes permettant de le rendre toujours disponible aux utilisateur avec de bonnes performances notamment le temps de réponse lors des interrogations (demande d'information). Afin de garantir la vivacité, le système d'information doit être doté d'une infrastructure matérielle robuste (Serveurs performant et réseau à haut débit) et de couche logicielle

ou software performante (SGBD et applications).
Beaucoup de stratégies existent pour garantir cette propriété de vivacité comme d'avoir des sites miroirs en cas de problème sur le site principal hébergeant le SI les autres sites peuvent prendre le relais, cette technique est appelé en anglais "business continuity".

Les fonctions du système d'information sont des tâches appliquées sur les données , la liste suivante stipule les tâches les plus importantes effectuées par les outils disponibles sur une SI, en l'occurrence introduire (saisir), consulter , modifier, transformer , traiter et publier sous différents format des données. :

La collecte des données :

Les systèmes d'information offrent aux utilisateurs des outils sous forme d'interface graphique utilisateur (IGU ou en anglais : Graphic User Interface), ces interfaces sont fournies par des applications métier de gestion : finance , comptabilité , stock, ...
Par ailleurs, une collecte d'information est possible d'une façon automatique via internet ou à partir d'un système d'information externe, souvent les web services sont utilisés pour réaliser cette interaction.

Mémorisation des données :

En effet , le stockage et la sauvegarde des données sont à la charge des systèmes de gestion de données (SGBD) et les baies de disques (un ensemble des disques sur une machine dédiée). Avec cette configuration logicielle et matérielle, les données sont conservées, protégées et gérées d'une manière efficace et sécurisée.

Traitement des données :

Le traitement des données constitue une fonction active du SI, et permet de produire de l'information consommable d'une façon rapide, fiable et intelligente. Le traitement des données effectivement charge les données suivant la requête des utilisateurs, par la suite une série de transformations tels que des calculs, agrégation , représentation tabulaire ou graphique ...

Communication d'information :

L'information résulte d'un processus de transformation, Une fois l'information est prête pour la consommation , une autre fonctionnalité est sollicité notamment la diffusion vers l'utilisateur ou le système demandeur. Cette étape peut être appelée la publication.

1.6 Conception des SI

Concevoir un système d'information est toujours un défi, où le grand problème réside dans le passage du langage naturel à une modélisation , autrement dit : comment passer d'une description du besoin à une réalisation avec précision et en réduisant l'écart entre l'objectif escompté et l'implémentation du système en question.
L'expression de besoins reste un problème d'actualité car il s'agit des éxigences que doit respecter les architectes des systèmes d'information.
Les méthodes de conception des systèmes d'information sont catégorisées en trois classes :

1. Méthodes fonctionnelles
2. Méthodes systémiques
3. Méthodes orientées objet

Méthodes fonctionnelles :

Le terme fonctionnel vient du fait que ces méthodes représentent un SI en tant qu'une globale fonction de gestion. Elle-même se compose d'autres fonctions qu'il faut identifier par la suite. Cette catégorie de méthode se base sur des approches de décomposition cartésienne et sur la modélisation des flots de données. La fonction globale qui identifie le système d'information est décomposée en une série de fonctions basique , simple et facile à modéliser.
Les méthodes fonctionnelles utilisent une modélisation graphique basée sur des diagrammes qui représentent la structure d'un système ainsi que les flots de données, nous pouvons citer quelques méthodes appartenant à cette catégorie comme Gane et Sarson et SADT qui ne sont pas formelles , c'est à dire ne se base pas sur un formalisme mathématique et/ou logique.

Méthodes systémiques :

Les méthodes dites systémiques se concentrent sur la modélisation des données, ou l'information sur le système à concevoir ainsi que la dépendance existante entre ces composantes. Cependant, la conception est dite à multi-niveaux d'abstraction, où en distingue exactement trois niveaux : conceptuel, logique et physique.
Des schémas conceptuels de données et de traitements sont utilisés pour la modélisation abstraite.
Le modèle relationnel entité-association est généralement utilisé pour la modélisation conceptuelle des données. Tandis que la modélisation des traitements reste propre à chaque méthode.
Merise est parmi les méthodes systémiques les plus connues et largement utilisées pour la modélisation des système d'information (données et traitement).

Néanmoins les méthodes qui se basent sur la modélisation avec entités associations sont aussi considérées comme non formelles. Toutefois, certaines méthodes ont été étendues afin d'être adaptées aux spécifications formelles.

Méthodes orientées objet Les méthodes orientées objet se basent sur la notion de classe d'objets pour la modélisation des systèmes d'information. Les classes composées d'attributs et méthodes autrement dit représentent un état et qui définissent des opérations. Par ailleurs, les associations représentent les relations entre les classes. Une conception orientée objet comporte trois types de modèles en l'occurrence : statique, dynamique et interaction.

Modèle statique : Ce modèle permet de modéliser les classes ainsi que leurs attributs et de définir les associations entre classes.

Modèle dynamique : modélise le comportement dynamique de chaque type d'objet tel que les méthodes et opération des classes.

Modèle d'interaction : modélise les flux de messages échangés entre les objets, tel que les états , événements et actions.

Plusieurs méthodes de conception basées sur l'objet ont été proposées dans la littérature, Nous pouvons citer par exemple : Object-oriented design (OOD) , Object-Oriented Software Engineering (OOSE) ,Object-Modeling Technique (OMT) et Unified Modeling Language (UML).
Ces approches utilisent une notation graphique (diagrammes) qui est à la fois un avantage car elles offrent une meilleure compréhension , apporte plus de clarté et facilite la lisibilité.
Cependant l'inconvénient réside dans le fait que leur sémantique n'est pas précise permettant par fois d'introduire certaines ambiguïté, c'est pour cette raison que nous qualifiant les langages graphiques comme UML d'être considérés comme des outils de modélisation semi-formels.
Les langages formels ont été utilisés pour la modélisation des données, néanmoins la spécification du comportement d'un système d'information demeure une tâche difficile.

1.7 Conclusion

Dans ce chapitre nous avons montré l'importance des systèmes d'information pour une organisation, notamment pour l'augmentation de la productivité des employés ainsi que leur efficacité. Les applications métier de gestion disponible dans un SI améliorent les conditions , le cadre de travail et permet aux décideurs une meilleure gestion à travers une visibilité plus claire de l'existant ainsi qu'une possibilité de se projeter dans le

future afin de faire des prévisions d'investissement ou approvisionnement , ou achat etc....
Un système d'information permet globalement de mieux utiliser les ressources de l'entreprise afin de minimiser la dépense et maximiser le profit. La rapidité et facilité d'accès à une information fiable, cohérente et sûre sont les éléments clé d'un système d'information.
Le volet sécurité et la confidentialité des données est aussi un grand défi surtout dans le contexte actuel de la mondialisation et l'utilisation des technologies tels que internet et les environnement des Cloud Computing.

Chapitre 2

Les spécifications Formelles

La langue naturelle est connue d'être structurellement ambiguë, du fait qu'un mot ou une phrase peut avoir plusieurs significations (sens ou sémantique), tout dépend de son interprétation. Cependant un formalisme mathématique peut soulever l'ambiguïté , conflits ou malentendu avec une notation formelle.
En effet, les méthodes de spécification formelle peuvent remédier aux problèmes du sens et proposent une spécification plus claire et précise grâce à une notation et symbolisme mathématique et/ ou logique ainsi qu'une notation visuelle ou graphique plus compréhensible et lisible. Il existe plusieurs méthodes de spécification formelle, toutes utilisant des modèles graphiques (diagrammes) à cause de leur simplicité et efficacité [Gervais, 2006, Junior, 1997].

2.1 Introduction

Les approches adoptées pour la modélisation du comportement des systèmes d'informations telle que les modèles entités associations étendues, les réseaux de Petri, où les méthodes orientées objet ont démontré leurs limites, ceci vient du fait qu'il ne se base pas sur une notation purement formelle, autrement dit ces méthodes n'utilisent pas un langage formel.
D'autre part ces méthodes sont employées une fois le contexte du système en question est défini, de plus,elles ne permettent pas de valider et de vérifier si certaines propriétés et contraintes sont réellement satisfaites. La vérification des propriétés du système n'est pas possible dans toutes les situations, en particulier pour les propriétés dynamiques qui changent dans le temps.

2.2 Principes

Le terme *méthodes formelles* fait référence à diverses techniques mathématiques utilisées pour la spécification formelle et le développement de logiciels. Les méthodes formelles consistent en un langage de spécification formelle et utilisent une collection d'outils pour prendre en charge la vérification syntaxique d'une spécification, ainsi que la preuve de validité des propriétés.
L'utilisation de la notation mathématique évite les spéculations sur la signification des phrases dans une description en langage naturel d'un système mal formulée. Le langage naturel est par nature ambigu, alors que les mathématiques emploient une notation rigoureuse et précise.
La spécification formelle est l'utilisation de la notation mathématique pour décrire de manière précise les propriétés qu'un système d'information doit posséder, sans restreindre indûment la manière dont ces propriétés sont obtenues.
La spécification formelle devient ainsi le point de référence clé pour les différentes parties impliquées dans la conception du système. La spécification formelle peut être utilisée comme point de référence pour les exigences ; la mise en œuvre ; tests et documentation du code applicatifs

2.3 Les méthodes Formelles

La mention méthodes formelles est utilisée pour décrire un langage de spécification formelle, employé pour la modélisation, la conception et la mise en œuvre d'un système informatique.
Les exigences sont la base du système à modéliser, et quelles que soient les meilleures pratiques de conception et de développement. Dans la plupart des projets de réalisation, le produit final sera incomplet et ne satisfait pas les exigences, qui peuvent aussi être incomplètes ou incohérentes ou encore floues. Car le processus de conception et implémentation et ceci dépend essentiellement su les exigences incluses au niveau cahier des charges établi par le client.
L'objectif de la validation des exigences est de s'assurer que ces exigences reflètent ce qui est réellement requis par le client (afin de concevoir le bon système).
Des méthodes formelles peuvent être utilisées pour modéliser les exigences, et permettent la vérification de la conformité et la validation de certaines propriétés par rapport à la spécification initiale, ce qui est une tâche difficile mais cruciale pour l'aboutissement du produit final (en parle du SI à concevoir), en particulier dans les applications critiques pour la sécurité des usagés par exemple : les moyens de transport autonomes.
L'utilisation de méthodes formelles conduit généralement à un logiciel plus robuste et à une confiance accrue dans son exactitude. Des méthodes formelles ont été appliquées

à une large gamme d'applications, y compris les domaines critiques de la sûreté et de la sécurité pour développer des logiciels fiables.
Les applications comprennent le secteur ferroviaire, la vérification des microprocesseurs, la spécification de normes, la spécification et la vérification des programmes.
Les méthodes formelles sont potentiellement très utiles et raisonnablement faciles à utiliser. L'utilisation d'une méthode formelle telle que Z ou VDM oblige l'ingénieur logiciel à être précis et permet d'éviter les ambiguïtés présentes dans le langage naturel.
Il y a une forte motivation à utiliser les meilleures pratiques en génie logiciel afin de produire des logiciels conformes à des normes de haute qualité.
Les méthodes formelles sont une technologie de pointe permettant de réduire l'apparition de défauts dans les produits ou logiciels.
L'utilisation de méthodes formelles est obligatoire dans certaines circonstances, surtout là ou la santé , sécurité des humains est en jeu. En outre, l'utilisation de méthodes formelles permet des économies sur le coût du projet.
Des méthodes formelles ont été employées dans plusieurs domaines tels que le secteur des transports, le secteur nucléaire, le secteur spatial, le secteur de la défense, le secteur des semi-conducteurs, le secteur financier et le secteur des télécommunications, les applications en temps réel dans l'industrie nucléaire, l'industrie aérospatiale.
Ces secteurs sont soumis à des contrôles réglementaires stricts pour garantir que la sûreté et la sécurité sont correctement prises en compte.
Plusieurs organisations ont mis à l'essai des méthodes formelles avec plus ou moins de succès. IBM a développé le langage de spécification VDM dans son laboratoire de Vienne et a piloté les langages de spécification formelle B et Z.
Le langage de spécification est basé sur une logique d'ordre supérieur et le prouveur de théorème est guidé par l'utilisateur dans la conduite de la preuve. Il a été appliqué à la vérification du matériel et des logiciels.
Il existe principalement deux familles de méthodes formelles : Orientée Modèle et Axiomatique (ou Algébrique).

Orientée Modèle :

L'approche de la spécification orientée modèle est basée sur des modèles mathématiques, où un modèle est une simplification ou une abstraction du monde réel qui ne contient que les détails essentiels. Par exemple, le modèle d'un avion n'inclura pas la couleur de l'avion, et l'objectif peut être de modéliser l'aérodynamique de l'avion.
L'importance des modèles est qu'ils servent à expliquer le comportement d'une entité particulière et peuvent également être utilisés pour prédire le comportement futur.

Axiomatique :

L'approche axiomatique se concentre sur les propriétés que le système proposé doit

satisfaire, et il n'y a aucune intention de produire un modèle abstrait du système. Les propriétés et le comportement requis du système sont indiqués en notation mathématique. L'accent est mis sur la spécification des propriétés requises du système et les problèmes de mise en œuvre sont évités dans ce stade.
Les propriétés sont généralement énoncées en utilisant une logique mathématique ou des logiques d'ordre supérieur. Des techniques automatisées de démonstration de théorèmes peuvent être utilisées pour prouver les résultats.

Preuve et méthodes formelles :

Une preuve mathématique comprend généralement un langage naturel et des symboles mathématiques, le problème est décomposé en sous-problèmes et tenter par la suite de prouver chacun à part.
De nombreuses preuves dans les méthodes formelles concernent la vérification croisée des détails de la spécification, ou la vérification de la validité que certaines propriétés soient satisfaites par la spécification.
Une preuve mathématique formelle consiste en une séquence de formules, où chaque élément est soit un axiome, soit dérivé d'un élément précédent de la série en appliquant un ensemble fixe de règles automatiques.
L'application de méthodes formelles dans un environnement industriel nécessite l'utilisation de la preuve assistée par machine, car des milliers d'obligations de preuve découlent d'une spécification formelle, et les prouveurs de théorèmes sont essentiels pour les résoudre efficacement.
La démonstration automatisée du théorème est difficile, car souvent les mathématiciens prouvent un théorème avec un sentiment intuitif initial que le théorème est vrai. L'intervention humaine pour fournir des conseils ou de l'intuition améliore l'efficacité du prouveur de théorème.
Les prouveurs de théorèmes sont des programmes supposés fonctionner correctement. Effectivement, les méthodes formelles offrent une confiance accrue dans l'exactitude du logiciel, plutôt qu'une preuve absolue d'exactitude.

2.4 Les langages formels basés sur les états

Les méthodes basées sur les états représentent le système à travers deux modèles complémentaires : la partie statique permet de décrire les entités constituant le système et leurs états, tandis que la partie dynamique modélise les changements d'états que le système peut effectuer par l'intermédiaire d'opérations ou d'actions. Des propriétés d'invariance sont souvent définies sur le système pour assurer la cohérence du système.

Parmi les langages appartenant à cette catégorie de langages basés sur les états figure : VDM , B et Z.

2.5 Les langages formels basés sur les événements

Les approches basées sur les événements représentent le système à travers des processus ou des agents, qui sont des entités indépendantes qui communiquent entre elles ou avec l'environnement extérieur du système. Ces méthodes permettent de modéliser le comportement du système à l'aide de séquences ou d'arbres d'événements. Nous pouvons citer quelques approches appartenant à cette catégorie : les réseaux de Petri , LOTOS , CCS , CSP ,et EB3.

2.6 Exemple de spécification formelle

Nous présentons dans cette section un exemple de spécification formelle élaboré en langage *EXPRESS*, il s'agit de modéliser un établissement d'enseignement. La modélisation se base sur le concept des entités (*Entity* en anglais) pour modéliser les acteurs du système et la définition des types de données.

Figure 2.1 décrit une spécification formelle en langage *EXPRESS*. Une spécification commence toujours par le terme *SCHEMA* suivi d'un nom ou intitulé dans notre exemple établissement. En effet, l'entête doit comporter la définition de tous types avant d'être référencés dans la spécification.

On trouve une définition de deux types simple : "t_salaire" numérique avec une contrainte qui l'oblige d'être toujours positif et le type qui représente le numéro de la sécurité sociale "t_num_ss" il s'agit d'un type chaîne de caractères qui commencent par un '1' ou '2' avec une longueur fixe de 12 caractères.

Par ailleurs, un type de collection (tableau) est défini pour contenir la liste des cours enseignés. L'exemple comporte aussi un autre type simple "t_nom" chaîne de caractères avec une contrainte sur la longueur qui doit être supérieure à zéro.

Une entité caractérise un type composé de : type simple (numérique ou chaîne de caractères) et/ou d'autres types composés. L'entité "*PERSONNE*" est définie avec une information sur les entités qui peuvent être dérivées de cette dernière en l'occurrence : "*ETUDIANT*" et "*ENSEIGNANT*" qui ne sont pas encore définis, L'entité "*PERSONNE*" dispose de deux attributs : "num_ss" , et "nom" de type : "t_num_ss" et "t_nom" respectivement.

Figure 2.2, illustre la deuxième partie de cette spécification, où on trouve une déclaration d'une entité "*ETUDIANT*" un type composé qui en fait hérite de l'entité "*PERSONNE*" ses attributs (propriétés), avec la possibilité de définir d'autres attributs comme deux

tableaux (ARRAY) un pour les cours et le deuxième pour les notes. Il faut noté que les contraintes sur les types sont imposées en utilisant la clause "*WHERE*" qui se trouve juste après la définition du type en question, plus de détails sur cet exemple peut être trouvés dans la section 3.4.

2.7 Conclusion

Une investigation sur les méthodes formelles a été présentée dans ce chapitre, ou l'importance des ces approches dans la vérification , validation et optimisation des propriétés des systèmes d'information à concevoir est démontrée.
Nous avons vu qu'une conception basée sur une spécification formelle bien formulée permet d'assuré de bonne performance du système (sans défaut) et un fonctionnement sécurisé et optimale sans perte ni de coût (argent) ni de temps de maintenance. Les méthodes formelles servent principalement à minimiser les risques a travers une conception fondu sur les mathématique et la logique.
En qualifie une méthode de spécification formelle si et seulement-si elle emploie un ou plusieurs langages formels de spécification.
En outre, plusieurs langages formels ont été proposés dans la littérature afin de formuler une spécification formelle. Cependant nous avons vu que ces langages peuvent être classés en deux catégories : les langages formels basés sur les États et les langages formels basés les événements.

```
SCHEMA etablissement_schema;

TYPE t_salaire = REAL;
WHERE
   SELF > 0;
END_TYPE;

TYPE t_num_ss = STRING(13) FIXED;
WHERE
   wr: SELF[1] = '1'
      OR SELF[1] = '2';
END_TYPE;

TYPE t_cours = ENUMERATION OF(
   math, info, histoire, sport);
END_TYPE;

TYPE t_nom = STRING;
WHERE
   LENGHT(SELF) > 0;
END_TYPE;

ENTITY personne
ABSTRACT SUPERTYPE OF
   ONEOF (etudiant, enseignant);
   num_ss: t_num_ss;
   nom: t_nom;
DERIVE
   initiale:STRING :=
         calcule(SELF.nom);
UNIQUE
   ur: num_ss;
END_ENTITY;
```

FIGURE 2.1 – Exemple de spécification Formelle en langage *EXPRESS* (1)

```
ENTITY etudiant
SUBTYPE OF personne;
   notes: ARRAY [1:5] OF OPTIONAL REAL;
   suit: ARRAY [1:5] OF OPTIONAL cours;
WHERE
   wr: SIZEOF(QUERY(x <* notes |
       (x < 0) OR (x > 20))) = 0;
END_ENTITY;

ENTITY enseignant
SUBTYPE OF personne;
   salaire: OPTIONAL t_salaire;
   enseigne: SET[1:?] OF cours;
END_ENTITY;

ENTITY cours
   intitulé: t_cours;
INVERSE
   est_suivi_par: SET[0:30] OF
      étudiant FOR suit;
   est_enseigné_par: enseignant
      FOR enseigne;
END_ENTITY;

FUNCTION calcule(nom: t_nom): STRING;
   RETURN(nom[1]);
END_FUNCTION;

END_SCHEMA ;  -- etablissement_schema
```

FIGURE 2.2 – Exemple de spécification Formelle en langage *EXPRESS* (2)

Chapitre 3

Les méthodes de spécification formelle dans les SI

3.1 Introduction

Concevoir un système d'information n'est pas une tâche facile, la difficulté vient de la complexité structurelle, organisationnelle et comportementale (dynamique). La conception consiste à créer à travers une représentation visuelle (diagrammes) ou une expression sous forme d'un langage les propriétés statiques et dynamiques du système conçu.
Plusieurs méthodes ont été proposées dans la littérature afin de traduire un cahier des charges qui contient les besoins et les exigences devant être implémentés et intégrés au niveau d'un système d'information.
Les exigences sont en fait des fonctionnalités, des propriétés et des contraintes qu'un développeur doit impérativement recenser, comprendre et traduire sous forme d'une spécification. Cette spécification peut être formulée sous différentes formes et formats. En effet une représentation graphique ou visuelle peut être employée afin de décrire un système, d'autre part une représentation symbolique peut être utilisée aussi pour la spécification d'un SI.

3.2 Méthode MERISE

L'objectif de Merise [Diviné, 1989] est de concevoir un système d'information, en se basant sur la séparation des données et des traitements. En effet, MERISE s'appuie sur une vision systémique de l'entreprise et propose plusieurs modèles conceptuels et physiques. La séparation des données et des traitements permet une modélisation robuste. Merise repose sur quatre niveaux d'abstraction (voir figure 3.1) :

1. Niveau conceptuel ,

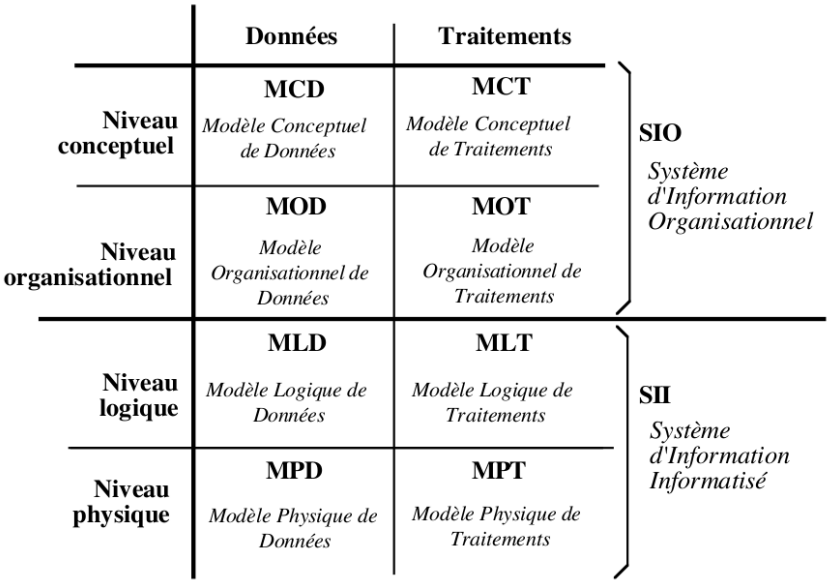

FIGURE 3.1 – Merise : Les différents modèles du système d'information.

2. Niveau organisationnel ,
3. Niveau logique ,
4. niveau physique.

Les systèmes d'information sont représentés par un modèle selon chaque niveau d'abstraction (conceptuel, organisationnel, logique, physique), et aussi par rapport aux volet : données et traitements. Chaque modèle est exprimé dans un formalisme utilisant des concepts adaptés au niveau d'abstraction.

3.3 Méthode NIAM

Nijssen's Information Analysis Method (NIAM) [Halpin, 1998, Wintraecken, 2012] est un langage graphique formel qui a été développé à l'origine pour l'analyse de phrases en langage naturel. Le langage formel de NIAM est largement utilisé en Europe pour la modélisation des données. Chaque contrainte affine la relation entre deux nœuds ou spécifie une restriction entre deux ou plusieurs relations. Les nœuds sont soit des types d'objets lexicaux (lots), soit des types d'objets non lexicaux (nolots). Le lexical (lots) fait référence à la langue ; les moyens de représentation et de communication, ou la dénomination des choses.
Non-lexical (nolots) fait référence aux choses physiques ou aux concepts mentaux ; les choses sur lesquelles nous communiquons. Nous représentons les lots par des encadrés en pointillés et les nolots par des cercles.
Un fait d'idée relie deux nolots et un fait de pont relie un nolot et plus.

Chaque fait contient deux rôles. Un rôle décrit la signification ou la nature de la relation entre son nœud adjacent et l'autre nœud de la relation. Les faits et les rôles sont représentés par les cases appariées. Dans la terminologie introduite précédemment, un nolot équivaut à une entité. Il n'y a pas d'idée équivalente à un attribut.
NIAM implique une méthodologie sous-jacente qui impose la cinquième forme normale, ainsi les modèles NIAM ont une forte sensation relationnelle. La méthodologie suggère que les choses qui ne peuvent pas être exprimées dans la langue doivent être décrites par un texte structuré. Le modèle graphique est normalement expliqué par des descriptions textuelles associées. Vous trouvez plus de détails au niveau du chapitre 4.

3.4 Méthode EXPRESS G

EXPRESS est un langage de spécification de modèle d'information à saveur d'objet qui a été initialement développé pour permettre l'écriture de modèles d'information formels [Schenck and Wilson, 1994]. La construction d'un modèle d'information est souvent un prélude à la construction d'un système d'information, comprenant une base d'informations.
La base d'informations traite du stockage et de l'accès aux (valeurs de) choses, et d'autres questions de cohérence, de comportement, etc. Le système d'information interagit avec les utilisateurs, exécute la mission du système, réagit et signale les problèmes, et traite d'autres questions de fonctionnement et d'environnement.
L'un des principaux principes d'EXPRESS est qu'une séparation claire doit être faite entre le modèle d'information et l'environnement du système d'information. Le raisonnement est simple : une information n'est pas liée à une seule application qui l'utilise. En inversant l'argument, nous pourrions dire : lorsqu'une information est liée à une seule application, il y a beaucoup moins de chances qu'elle puisse ou soit partagée.
Nous reprenons l'exemple 2.6, il s'agit d'une spécification formelle élaboré en langage *EXPRESS* d'un établissement d'enseignement. Une spécification commence toujours par le terme *SCHEMA* suivie d'un nom ou intitulé, comme c'est le cas des langages de programmation les types qui seront utilisés dans la spécification doivent figurer (déclarer) au niveau de l'entête.
En effet langage *EXPRESS* ressemble beaucoup aux langages utilisant le concept Objet car on peut trouver la notion typage et héritage. la notation entité "ENTITY" est en fait un objet ou un type composé type simple (numérique ou chaîne de caractères) et/ou un autre type composé.
Le type numérique est dénoté par la notation "*REAL*", et le type chaîne de caractères dénoté par "*STRING*".
La relation d'héritage figure au niveau de l'entité père avec la notation "*SUPERTYPE*" et les entités avec la notation "*SUBTYPE*".

L'exemple 3.2, il faut noter que la définition d'une entité est délimitée par "ENTITY"

```
SCHEMA etablissement_schema;

TYPE t_salaire = REAL;
WHERE
   SELF > 0;
END_TYPE;

TYPE t_num_ss = STRING(13) FIXED;
WHERE
   wr: SELF[1] = '1'
      OR SELF[1] = '2';
END_TYPE;

TYPE t_cours = ENUMERATION OF(
   math, info, histoire, sport);
END_TYPE;

TYPE t_nom = STRING;
WHERE
   LENGHT(SELF) > 0;
END_TYPE;

ENTITY personne
ABSTRACT SUPERTYPE OF
   ONEOF (etudiant, enseignant);
   num_ss: t_num_ss;
   nom: t_nom;
DERIVE
   initiale:STRING :=
         calcule(SELF.nom);
UNIQUE
   ur: num_ss;
END_ENTITY;

ENTITY etudiant
SUBTYPE OF personne;
   notes: ARRAY [1:5] OF OPTIONAL REAL;
   suit: ARRAY [1:5] OF OPTIONAL cours;
WHERE
   wr: SIZEOF(QUERY(x <* notes |
       (x < 0) OR (x > 20))) = 0;
END_ENTITY;

ENTITY enseignant
SUBTYPE OF personne;
   salaire: OPTIONAL t_salaire;
   enseigne: SET[1:?] OF cours;
END_ENTITY;

ENTITY cours
   intitulé: t_cours;
INVERSE
   est_suivi_par: SET[0:30] OF
      étudiant FOR suit;
   est_enseigné_par: enseignant
      FOR enseigne;
END_ENTITY;

FUNCTION calcule(nom: t_nom): STRING;
   RETURN(nom[1]);
END_FUNCTION;

END_SCHEMA ;  -- etablissement_schema
```

FIGURE 3.2 – Spécification Formelle en langage *EXPRESS*

et "END_ENTITY", de la même façon la définition d'un type débute par "TYPE" et se termine par "END_TYPE".
En outre, le langage EXPRESS permet de définir des fonctions avec la notation "FUNCTION" ce qui représente la modélisation de l'aspect dynamique. De même EXPRESS utilise la notation "ENUMERATION OF" afin de déclarer un vecteur (ou tableau) d'éléments. Il faut noter que "END_SCHEMA" marque la fin d'une spécification.
EXPRESS-G est une notation graphique standard pour les modèles d'information, il s'agit d'un outil complément utile du langage EXPRESS afin de représenter graphiquement les définitions d'entités et de types, les relations et la cardinalité.
L'exemple illustré sur la figure 3.3 est effectivement la traduction de l'exemple 3.2 en utilisant une notation graphique d'ou le nom "*EXPRESS-G*".

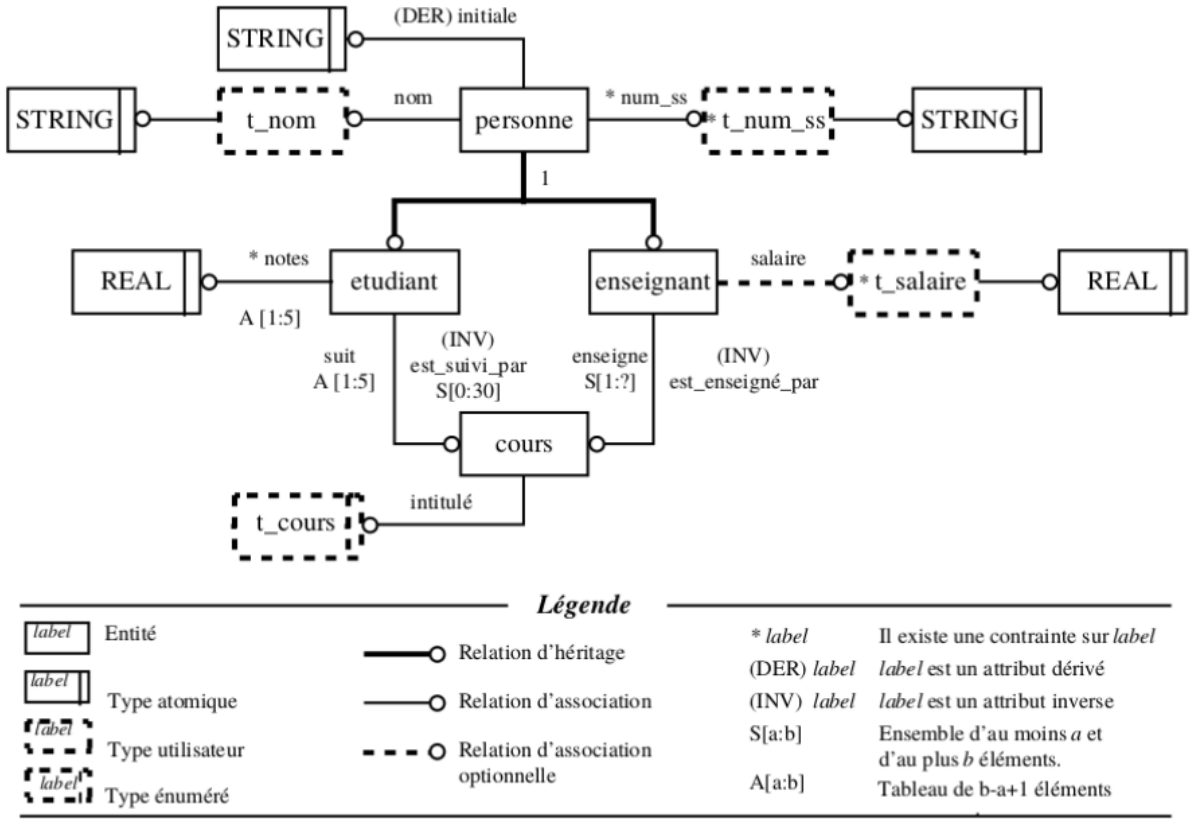

FIGURE 3.3 – Spécification Formelle en langage *EXPRESS-G*

Effectivement, *EXPRESS-G* est une notation de modélisation schématique destinée à la modélisation de l'information avec une approche qui ressemble aux méthodes orientées objet. La liste suivant stipule quelques propriétés importante du langage *EXPRESS-G* :

— Une description normative basé sur une norme internationale ;
— Apporte une meilleure lisibilité ;
— Se base sur un modèle proche du modèle orienté objet ;
— Offre une indépendance de la représentation physique des données ;
— Définit une structure hiérarchique extensible.

Suivant la notation du langage EXPRESS-G (Figure 3.3), une entité est représentée sous la forme d'un rectangle contenant le nom de l'entité ; les propriétés de l'entité sont définies par des attributs. Si une propriété est facultative pour cette entité, elle est désignée comme attribut facultatif. Les relations "entité-attribut" et "entité-entité" sont affichées par des lignes pleines, et seulement si une relation a des attributs facultatifs, une ligne en pointillés est utilisée. Le nom de l'attribut est écrit près d'une ligne de lien et le cercle à la fin de la ligne indique la direction du lien. La représentation du modèle sous forme graphique avec la notation EXPRESS-G a une vue plus visuelle, pratique et claire que lorsqu'elle est décrite sous forme textuelle.

3.5 Méthode Z

Z est un langage de spécification formel basé sur la théorie des ensembles de Zermelo [Spivey and Abrial, 1992]. Il a été développé au Programming Research Group à Oxford université au début des années 1980 et est devenue une norme ISO en 2002.
Les spécifications Z sont mathématiques et utilisent une logique classique à deux valeurs. L'utilisation des mathématiques garantit la précision et permet d'identifier les incohérences et les lacunes dans la spécification.
Des prouveurs de théorème peuvent être utilisés pour démontrer que l'implémentation du logiciel répond à ses spécifications.
Z est une approche «orientée modèle» avec un modèle explicite de l'état d'une machine abstraite donné, et les opérations sont définies en fonction de cet état. La notation mathématique de Z est utilisée pour la spécification formelle et le schéma est utilisé pour structurer les spécifications.
Un schéma Z se compose essentiellement de boîtes, qui décrivent les opérations et les états. Les schémas peuvent être utilisés comme blocs de construction et peuvent être combinés avec d'autres schémas.
La notion de schéma est un moyen puissant pour décomposer une spécification en morceaux ou sous-schémas plus petits. Cela permet de rendre les spécifications Z hautement lisibles, car chaque schéma individuel est de petite taille et autonome.
La gestion des exceptions est résolue en définissant des schémas pour les cas d'exception. Ceux-ci sont ensuite combinés avec le schéma d'opération d'origine. Les types de données mathématiques sont utilisés pour modéliser les données dans un système, et ces types de données obéissent aux lois mathématiques. Ces lois permettent de simplifier les expressions et sont utiles pour les preuves.
Les opérations sont définies dans un style de précondition postcondition , ou une condition préalable doit être vraie avant que l'opération ne soit exécutée et la postcondition doit être vraie après l'exécution de l'opération.
La condition préalable est définie implicitement dans l'opération, ou chaque opération a une obligation de preuve associée pour garantir que si la condition préalable est vraie, alors l'opération préserve l'invariant du système. L'invariant système est une propriété du système qui doit être vraie à tout moment. L'état initial lui-même est, bien entendu, nécessaire pour satisfaire l'invariant du système.
Les postconditions emploient un prédicat logique qui relie le pré-état au post-état, le post-état d'une variable étant distingué en déclarant la variable. Z est un langage typé et chaque fois qu'une variable est introduite, son type doit être indiqué. Un type est simplement une collection d'objets, et il existe plusieurs types standard dans Z. Ceux-ci incluent les nombres naturels N, les entiers Z et les nombres réels R.
La déclaration d'une variable **x** de type **X** s'écrit **x :X**. Il est également possible de créer

ses propres types dans Z.
Diverses conventions sont utilisées dans la spécification Z, par exemple, **v?** indique que **v** est une variable d'entrée ; **v!** indique que **v** est une variable de sortie. Dans un schéma l'opération *Op* peut affecter l'état, comme elle peut ne pas le faire, ça dépend de cas de figure. De nombreux types de données utilisés dans Z n'ont pas d'équivalent dans les langages de programmation standard. Il est donc important d'identifier et de décrire les structures de données concrètes qui représenteront finalement les structures mathématiques abstraites.
Comme les structures concrètes peuvent différencier de l'abstrait, les opérations sur les structures de données abstraites peuvent être affinées pour produire des opérations sur les données concrètes qui donnent des résultats équivalents.
Ce qui suit est une spécification Z (voir Figure 3.4) pour emprunter un livre à une bibliothèque. La bibliothèque est composée de livres qui sont sur l'étagère ; livres empruntés et livres manquants. L'état du système est défini dans le schéma de la bi-

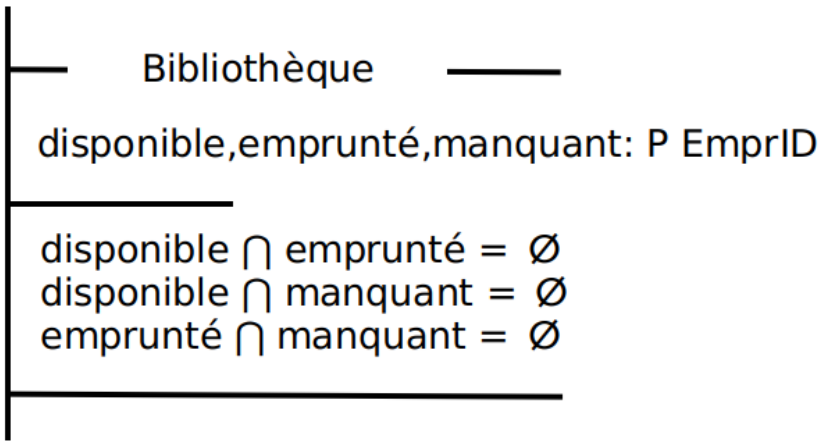

FIGURE 3.4 – Exemple d'une spécification de bibliothèque en langage Z

bliothèque ci-dessous et des opérations telles que Emprunter et Retourner affectent l'état. L'opération Emprunter est spécifiée (Figure 3.5).

Le cahier des charges modélise une bibliothèque avec des ensembles représentant des livres en rayon, prêtés ou manquants. Ce sont trois sous-ensembles mutuellement disjoints de l'ensemble de livres **EmprID**.
La notation **EmprID** est utilisée pour représenter l'ensemble de tous les sous-ensembles de **EmprID**. La condition de disjonction pour la bibliothèque est exprimée par l'exigence que l'intersection par paires des sous-ensembles sur étagère, empruntés, manquants soit l'ensemble vide.
La condition préalable à l'opération d'emprunt est que le livre doit être disponible en rayon pour emprunter. La condition postérieure est que le livre emprunté est ajouté à l'ensemble de livres empruntés et est retiré des livres sur l'étagère.

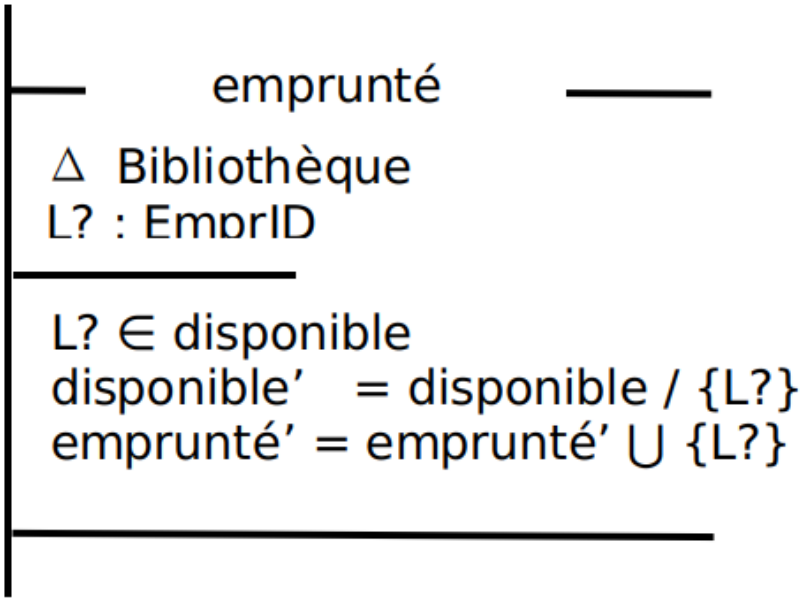

FIGURE 3.5 – Exemple d'une spécification de bibliothèque (opérations) en langage Z

3.6 Méthode Object Modeling Technique (OMT)

La technique de modélisation d'objets (Object Modeling Technique : OMT) est une méthodologie d'analyse, de conception et de mise en œuvre orientée objet qui se concentre sur la création d'un modèle d'objets du monde réel, puis sur l'utilisation de ce modèle pour développer un logiciel orienté objet.
La méthodologie OMT (Figure 3.6) utilise trois types de modèles pour décrire un système :

1. Modèle d'objet - décrit les objets du système et leurs relations ;
2. Modèle dynamique - décrit les interactions entre les objets du système ;
3. Modèle fonctionnel - décrit les transformations de données du système.

Il faut noter que la méthodologie OMT couvre le cycle de vie complet du développement logiciel. La méthodologie comprend les étapes suivantes :

1. Analyse : L'analyste construit un modèle de la situation du monde réel montrant ses propriétés importantes ;
2. Conception du système - Le concepteur du système prend des décisions de haut niveau sur l'architecture globale, y compris l'organisation du sous-système ;
3. Conception d'objets - Le concepteur d'objets construit un modèle de conception contenant des structures de données et des algorithmes pour les classes d'objets ;
4. Implémentation - Les classes d'objets et les relations développées lors de la conception d'objets sont finalement traduites dans un langage de programmation, une base de données ou une implémentation matérielle particulière

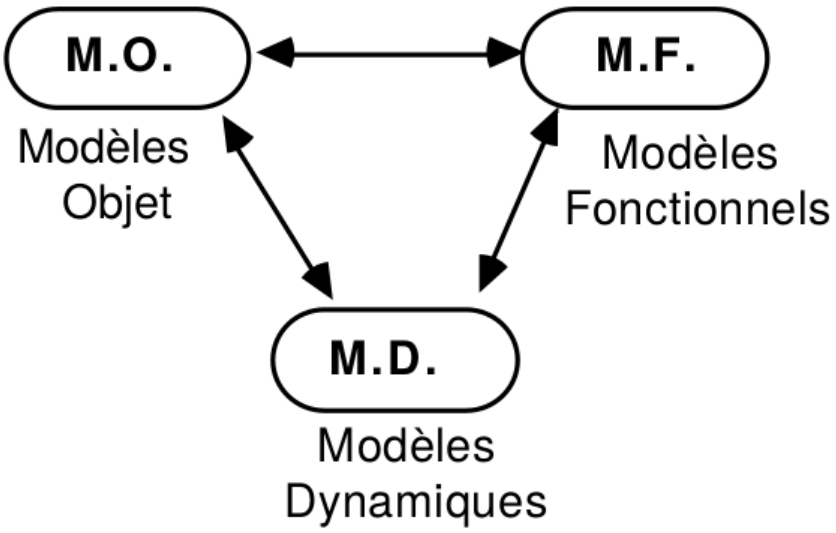

FIGURE 3.6 – Modèle Objet de *OMT*

L'exemple de la figure 3.7, illustre un exemple de gestion des commandes pour un client, ou une commande comporte plusieurs produits. Ce modèle impose des contraintes de cardinalité sur l'objet produit par rapport à une commande. Une commande doit avoir au moins un produit. Le cercle plein désigne qu'un client peut passer zéro ou plusieurs commandes.

L'exemple de la figure 3.8, présente une relation n-aires, reliant trois objet : Enseignant

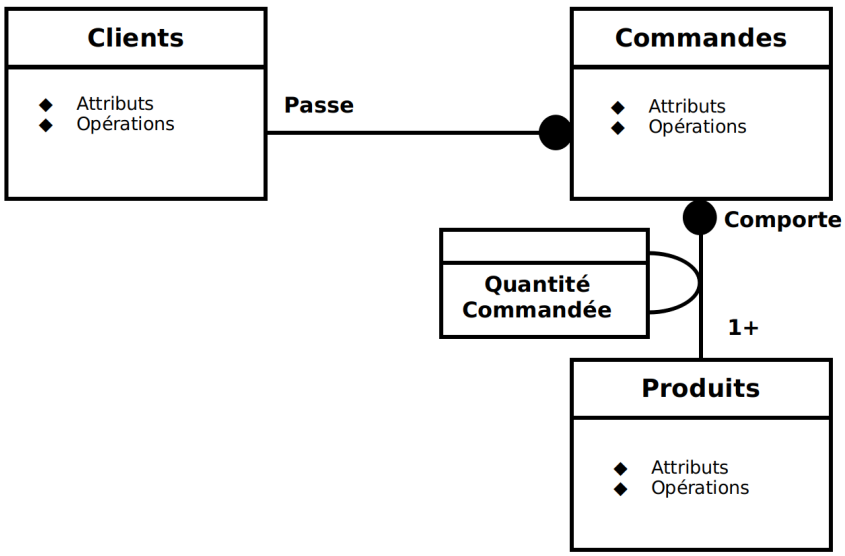

FIGURE 3.7 – *OMT* : un exemple de relation binaire

, Module et Classe.

La méthode OMT supporte aussi les relations d'héritage ainsi que la relation de composition ou agrégation entre classes.

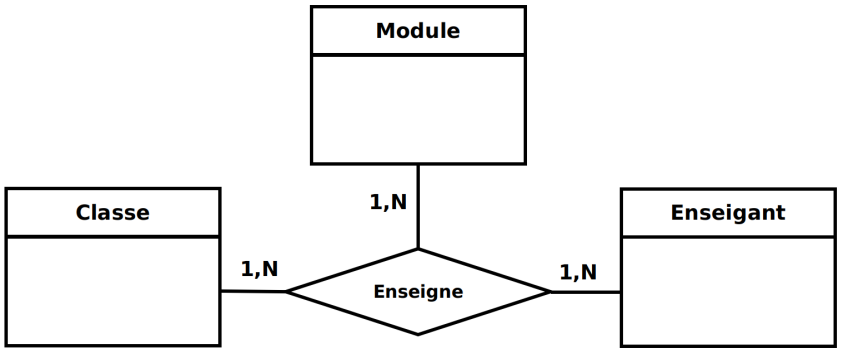

FIGURE 3.8 – *OMT* : un exemple d'une relation n-aires

3.7 Méthode UML

Le langage de modélisation unifié (UML) est un langage de modélisation visuelle pour les systèmes logiciels. Il a été développé par Jim Rumbaugh, Grady Booch et Ivar Jacobson chez Rational Corporation (qui fait maintenant partie d'IBM), en tant que notation pour la modélisation de systèmes orientés objet. Il fournit un moyen visuel pour spécifier, concevoir et même documenter les systèmes orientés objet et facilite, la compréhension de l'architecture du système à concevoir.
Le langage a été fortement influencé par trois méthodes existantes : la technique de modélisation d'objet (OMT) développée par Rumbaugh ; la méthode Booch développée par Booch et le génie logiciel orienté objet (OOSE) développé par Jacobson.
UML unifie et améliore ces méthodes, et il est devenu une approche formelle populaire de la modélisation des systèmes logiciels. Les modèles permettent de mieux comprendre le système à développer et un modèle UML permet de visualiser le système avant sa mise en œuvre. Les grands systèmes complexes sont difficiles à comprendre dans leur intégralité, et l'utilisation d'un modèle UML est un moyen de simplifier la réalité sous-jacente et de gérer la complexité. Le choix du modèle est fondamental, et un bon modèle fournira un bon aperçu du système. Les modèles doivent être explorés et testés pour garantir leur adéquation en tant que représentation du système.
Les modèles simplifient la réalité, mais il est important de s'assurer que la simplification n'exclut aucun détail important. Le modèle choisi affecte la vue du système et différents rôles nécessitent différents points de vue du système proposé.
Un développeur de base de données se concentrera souvent sur les modèles de relation entité-relation, tandis qu'un analyste de systèmes se concentrera souvent sur les modèles algorithmiques. Un développeur orienté objet se concentrera sur les classes et sur les interactions des classes. Souvent, il est nécessaire de visualiser le système à différents niveaux de détail, et aucun modèle ne suffit à lui seul pour cela. Cela conduit au développement d'un petit nombre de modèles interdépendants.
UML fournit un modèle semi-formel au système et permet de présenter les mêmes in-

formations de plusieurs manières et à différents niveaux de détail. Les exigences du système sont exprimées en termes de cas d'utilisation ; la vue de conception capture l'espace du problème et l'espace de la solution ; la vue processus modélise les processus système ; la vue d'implémentation traite de l'implémentation du système et la vue de déploiement modélise le déploiement physique du système.
UML est un langage de modélisation graphique expressif pour visualiser, spécifier, construire et documenter un système logiciel. Il fournit plusieurs vues de l'architecture du logiciel et possède une syntaxe et une sémantique clairement définies.
Les différents diagrammes UML fournissent une visualisation graphique du système à partir de différents points de vue.

Diagrammes structurels :

Diagramme de déploiement :
: Modélise l'architecture des nœuds matériels et des processeurs du système et offre la possibilité de montrer les nœuds sur lesquels les composants logiciels résideront.

Diagramme composite :
: Modélise le comportement du composant ou de l'objet lors de l'exécution en montrant la disposition, les relations et les instances des composants pendant l'exécution du système

Diagramme de classes :
: Représente les classes, leurs définitions et leurs relations. Les classes et les entités de l'espace problème sont également des entités techniques détaillées dans l'espace solution. Les attributs et opérations qui définissent les classes sont inclus dans ce diagramme de classes. Les relations dans un diagramme de classes illustrent comment les classes interagissent, collaborent et héritent des autres classes. Les classes peuvent également représenter des tables relationnelles, des interfaces utilisateur et des contrôleurs.

Diagramme de paquetage :
: Représente les sous-systèmes et les domaines de l'organisation du système. Il peut également modéliser les dépendances entre les packages et aider à séparer les entités métier des interfaces utilisateurs, des bases de données, de la sécurité et des packages administratifs.

Diagramme de profil :
: Permet la création de profils extensibles qui peuvent être appliqués aux éléments hérités des profils. Ces schémas ajoutent de la valeur en étendant les normes de manière contrôlée.

Diagramme d'objets :

Affiche les objets et leurs liens dans la mémoire lors de l'exécution. Par conséquent, ces diagrammes d'objets aident également à visualiser les multiplicités dans la pratique.

Diagramme de composants :

Modélise les composants et leurs relations de manière structurelle. Ces composants peuvent inclure, par exemple, des exécutables, bibliothèques, services Web et services mobiles. Ces schémas ajoutent de la valeur à la prise de décision architecturale du système.

Diagrammes de comportement :

Diagramme de cas d'utilisation :

Fournit une vue d'ensemble des fonctionnalités du système ou des processus métier du point de vue de l'utilisateur. La manière dont un utilisateur «utilise» le système est le point de départ pour créer un diagramme de cas d'utilisation.

Diagramme d'activité :

Modélise le flux n'importe où dans le système. En particulier, le flux dans un cas d'utilisation décrivant les interactions normales des utilisateurs et les alternatives et exceptions est très bien modélisé par ces diagrammes d'activité.

Diagramme de machine d'état :

Affiche le cycle de vie d'exécution d'un objet en mémoire. Un tel cycle de vie comprend tous les états d'un objet et les conditions dans lesquelles les états changent.

Diagramme de séquence :

Modélise les interactions entre les objets en fonction de leur chronologie. Les objets peuvent être spécifiquement affichés sur ces diagrammes ou ils peuvent être des objets anonymes appartenant à une classe.

Diagramme de communication :

Montre comment les objets communiquent (interagissent) entre eux dans la mémoire au moment de l'exécution. Ces diagrammes de communication sont similaires aux diagrammes de séquence en termes de leur objectif ; cependant, leur représentation est différente.

Diagrammes d'interaction :

Présente un aperçu des interactions au sein d'un système, cela permet également de comprendre comment les diagrammes UML (par exemple, un diagramme de séquence) dépendent les uns des autres et sont liés les uns aux autres.

Diagrammes de timing :

Modélise le concept de temps et la manière dont l'état d'un objet change au fil du temps. De plus, ces diagrammes permettent de comparer les états de plusieurs objets à la fois.

L'aspect structurel d'un diagramme illustre la manière dont un système est organisé, tandis que l'aspect comportemental modélise le flux du système. La structure, par exemple, montre comment les classes sont liées les unes aux autres dans un diagramme de classes. Le comportement montre la manière dont un utilisateur interagit avec le système via un cas d'utilisation ou un diagramme d'activité, par exemple.
L'aspect statique versus dynamique représente la dépendance temporelle du modèle. Un diagramme sans notion de temps ou de mouvement est statique, alors qu'un diagramme qui montre des changements dans le temps (ou même un instantané dans le temps) est considéré comme dynamique.

3.8 Conclusion

Dans ce chapitre différentes méthodes de conception de système d'information ont été présentées. Certains méthodes utilisent le concept objet pour la modélisation nous pouvons citer : OMT et UML, se basant sur une notation visuelle notamment les diagrammes. Ces méthodes sont considérées néanmoins comme Semi-Formelle.
D'autre part, il existe des approches se basent sur des langages formel tels que la méthode Z et Express avec une notation visuelle dite Express-G.

Chapitre 4

Étude de cas : la méthode NIAM

4.1 Introduction

La «méthode d'analyse de l'information de Nijssen» (NIAM) est une méthode formelle à l'appui de l'analyse de l'information qui a été développée depuis 1975 [Halpin, 1998, Wintraecken, 2012]. Cette méthode a une base théorique solide et a été appliquée avec succès dans le passé dans diverses situations pratiques. L'objectif de la méthode NIAM est de concevoir un schéma conceptuel de base de données en se basant sur un modèle relationnel binaire.

4.2 Concepts et idées

4.2.1 Les Concepts

Dans un schéma conceptuel, il n'existe qu'un seul concept, ou chaque concept n'admet qu'un seul identifiant, cette propriété ressemble a la clé primaire utilisé dans les contraintes d'intégrités des base de données relationnelle.
Un concept au sens NIAM est une classe d'objets, le concept *objet* est largement adopté dans la modélisation des systèmes d'information.
Les objets qui sont regroupés sous le même concept sont porteurs d'informations communes relatives à leurs propriétés (comportement) et descriptions (attributs) vis à vis du problème (du modèle) examiné. Par exemple, être humain, enseignant, étudiant.
La Figure 4.1 illustre un exemple d'un concept, il s'agit un étudiant de type *CONCEPT* doté d'un identifiant unique *NumEtud*.
Les concepts sont représentés graphiquement par des cercles pleins avec un trait solide, entourés par un cercle avec un trait discontinu définissant la clé d'identification. En fait avec ce mécanisme d'unicité, le concept *enseignant* peut contenir une collection ou ensemble d'instance d'objets *enseignant*.

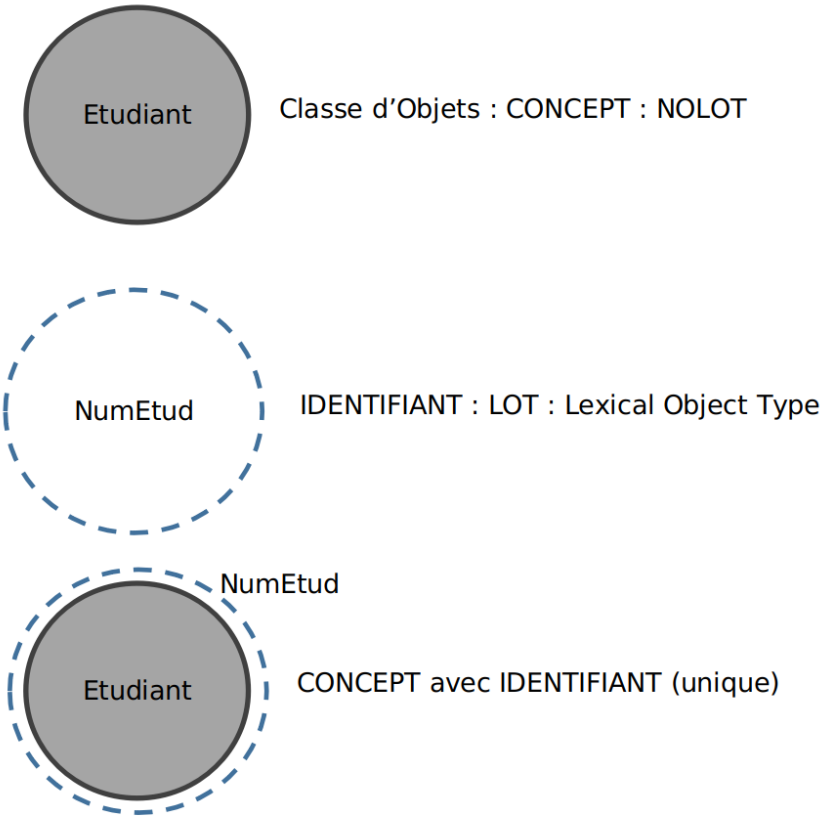

FIGURE 4.1 – Exemple de Concept et identifiant (1)

Un concept est qualifié de *NO LOT (non Lexical Object Type)*, ce qui veut dire qu'il est conteneur d'information complexe (un objet). Cependant *LOT (Lexical Object Type)* est une information atomique ou plate, comme un attribut *Nom* d'un étudiant.
Les propriétés descriptives des concepts peuvent être définies sur les concepts afin d'associer autant d'informations nécessaires pour la modélisation.
Pour le concept Étudiant, nous pouvons définir en premier son identifiant *NumEtud*, puis ajouter d'autres propriétés ou attributs comme le nom, prénom, date de naissance, ces informations caractérisent les détails personnels d'un étudiant donné.
Par ailleurs, d'autres informations complémentaires peuvent être ajoutées (Figure 4.2), tels que :modules en dette , exclu ...

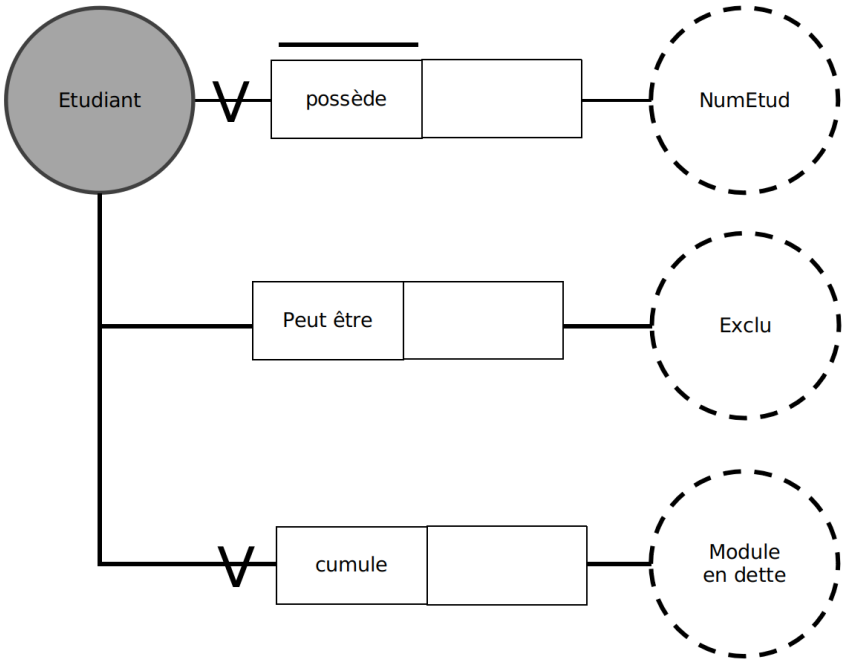

FIGURE 4.2 – Exemple de Concept et identifiant (2)

L'exemple ci-dessus peut être interprété comme suit :
— Tout Étudiant possède un seul *NumEtud*.
— Un étudiant peut cumuler zéro ou plusieurs modules en dette.
— Un étudiant peut être exclu.

4.2.2 Les idées

Une idée définit une relation entre deux concepts, il faut noter que relier deux concepts est équivalent à relier deux ensembles, donc nous pouvons qualifier une idée d'opérateur ensembliste.

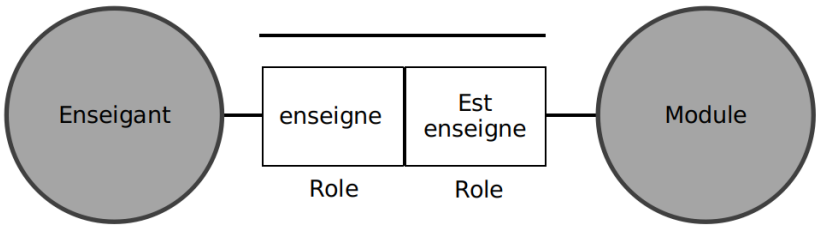

FIGURE 4.3 – Exemple d'idée : relation entre deux concepts

Un exemple d'idée (Figure 4.3) reliant deux concepts : Enseignant et Module.
Dans cet exemple la relation est bi-directionnelle avec un rôle et des contraintes de cardinalité pour chaque direction de lecture, ce qui est représenté par un trait épais

couvrant les deux rôles.
Suivant le sens *Enseignant* $\Longrightarrow$ *Module* le rôle de l'idée est *enseigne*, il faut noter qu'il n'y a pas de cardinalité spécifiée, cette relation peut être interprétée comme suit :
"$\mathcal{N}$ *Enseignants* enseignent $\mathcal{M}$ *Modules*, avec $\mathcal{N}, \mathcal{M} \in \{0, 1, 2, 3, ...\}$"
Ce pendant dans le sens inverse *Module* $\Longrightarrow$ *Enseignant* on aura *Est enseignée*, de même cette relation ne précise pas des contraintes de cardinalité. Elle est considérée par défaut $\in \{0, 1, 2, 3, ...\}$.

4.3 Symbolisme des contraintes entre deux concepts :

La méthode NIAM utilise un symbolisme pour imposer des contraintes entre deux concepts, afin de décrire les cardinalités dans les deux sens de la relation, c'est à dire pour chaque rôle.
L'exemple suivant (Figure 4.4), décrit une idée avec une contrainte de cardinalité appliquée sur un seul sens.

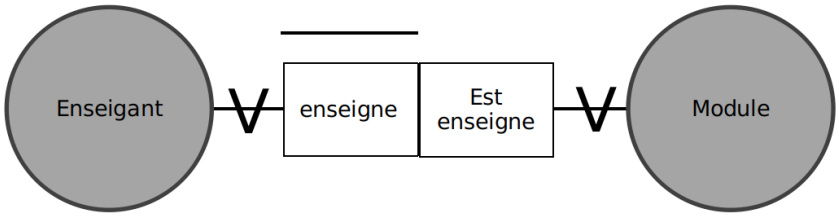

FIGURE 4.4 – Exemple d'idée : relation entre deux concepts avec contraintes de cardinalité

Le symbole **V** signifie "Quel que soit ou Tout" , et le trait épaisse s'applique uniquement sur le rôle "enseigne". La relation peut être interprétée comme suit :

Rôle enseigne : Tout enseignant doit enseigner obligatoirement un seul module.

Rôle Est enseigné Tout module est enseigné par $\mathcal{N}$ enseignants , avec $\mathcal{M} \in \{0, 1, 2, 3, ...\}$.

Cet exemple présente une contrainte « d'Unicité », qui s'applique sur la relation de l'ensemble destination, le trait court tracé sur le rôle dénote la contrainte sur la relation concerné.

Prenons l'exemple d'un hôpital, où nous avons des chirurgiens effectuant des opérations chirurgicales sur des patients. Si nous voulons imposer les contraintes suivantes :

— tout chirurgien effectue une ou plusieurs opérations
— toute opération est effectuée par au moins un chirurgien.

Le symbole différent de zéro $\neq$ est utilisé afin d'imposer la contrainte d'obligation d'avoir au moins une occurrence (Figure 4.5).

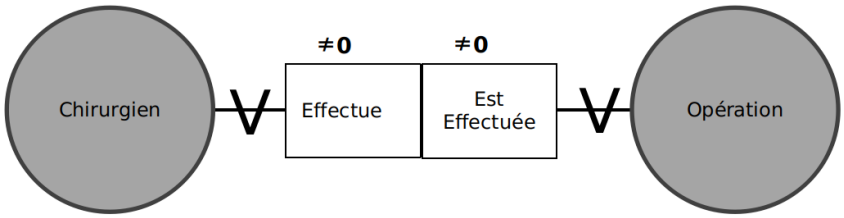

FIGURE 4.5 – Exemple d'idée : relation entre deux concepts avec contraintes de cardinalité différent de zéro $\neq$

Figure 4.6,illustre un exemple à deux idées reliant 03 concepts : Chirurgien, Service et Spécialité.

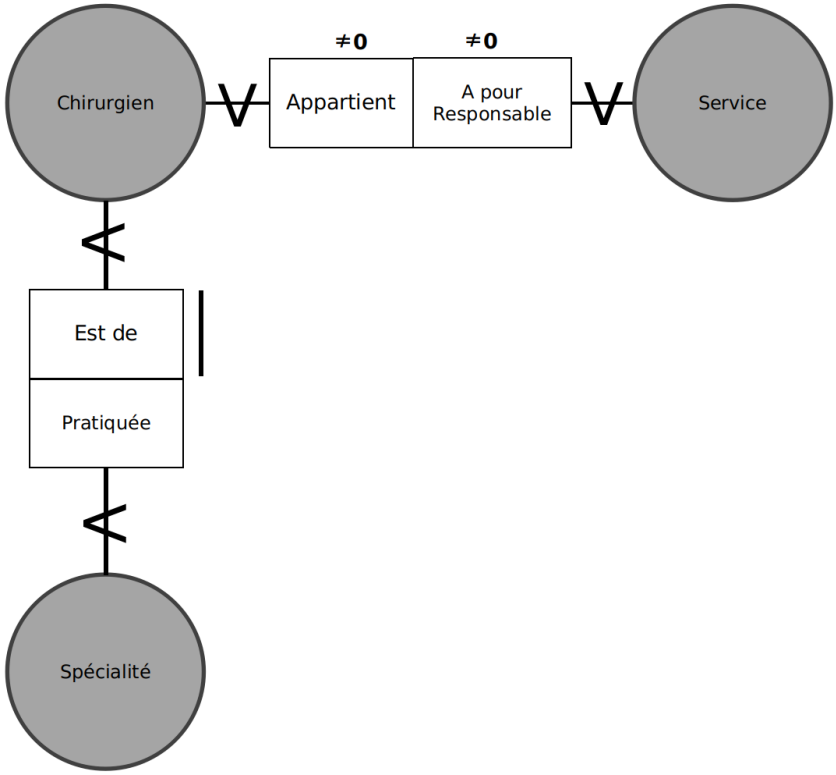

FIGURE 4.6 – Exemple de plusieurs idées : Avec 02 relations entre trois concepts avec contraintes de cardinalité différent de zéro $\neq$ (1)

— Tout Chirurgien appartient à un seul Service
— Tout Service à pour responsable au moins un Chirurgien.
— Un Chirurgien pratique obligatoirement une spécialité
— Une spécialité peut être pratiquée par zéro ou plusieurs Chirurgiens.

Un exemple de deux idées avec deux relation est présentée dans la figure 4.7. Cet exemple illustre un cas d'un match de foot dirigé par une équipe d'arbitres avec un superviseur.

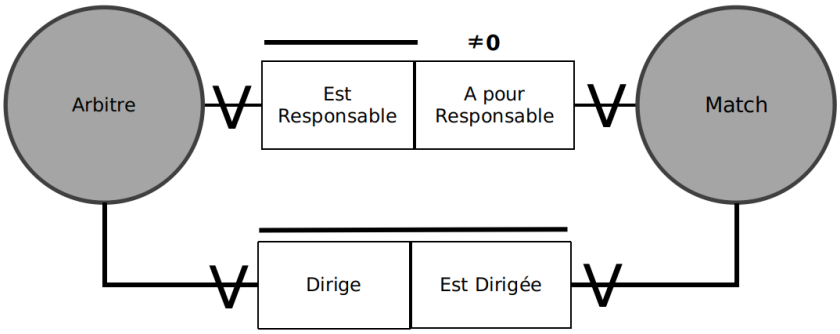

FIGURE 4.7 – Exemple de plusieurs idées : Avec 02 relations entre trois concepts avec contraintes de cardinalité différents de zéro $\neq$ (2)

Nous avons deux concepts : Arbitre et Match avec deux idées reliant ces deux derniers, une relation (idée) mettant l'accent sur la direction de match par des arbitres et la deuxième idée montre qu'un match a un seul responsable.

4.4 Contraintes entre deux idées

Nous reprenons l'exemple de la figure 4.7, avec une modification en imposant une contrainte assurant que l'arbitre responsable doit diriger le match qu'il supervise.

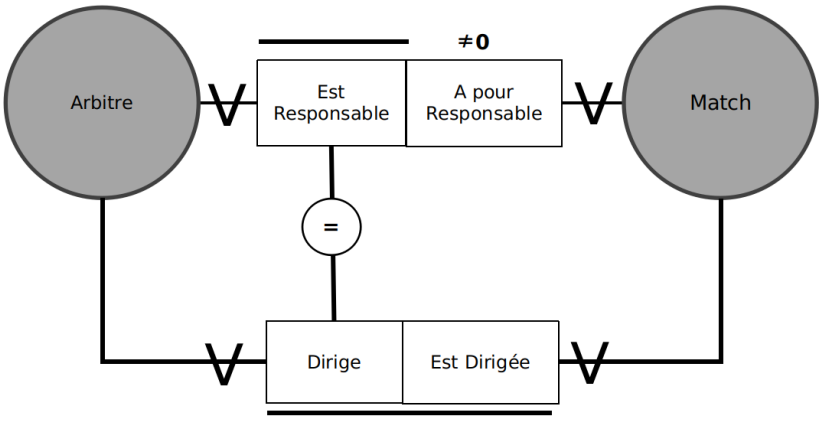

FIGURE 4.8 – Exemple de Contraintes entre deux idées (Contrainte d'égalité)

Dans l'exemple suivant (Figure 4.9),nous présentons une contrainte d'unicité entre deux idées. l'exemple est interprété comme suit :
Une occurrence de la combinaison (étudiant, examen) CORRESPOND 0 ou 1 note.
L'exemple (Figure 4.10) illustre une Contrainte de Totalité, permettant de limiter les

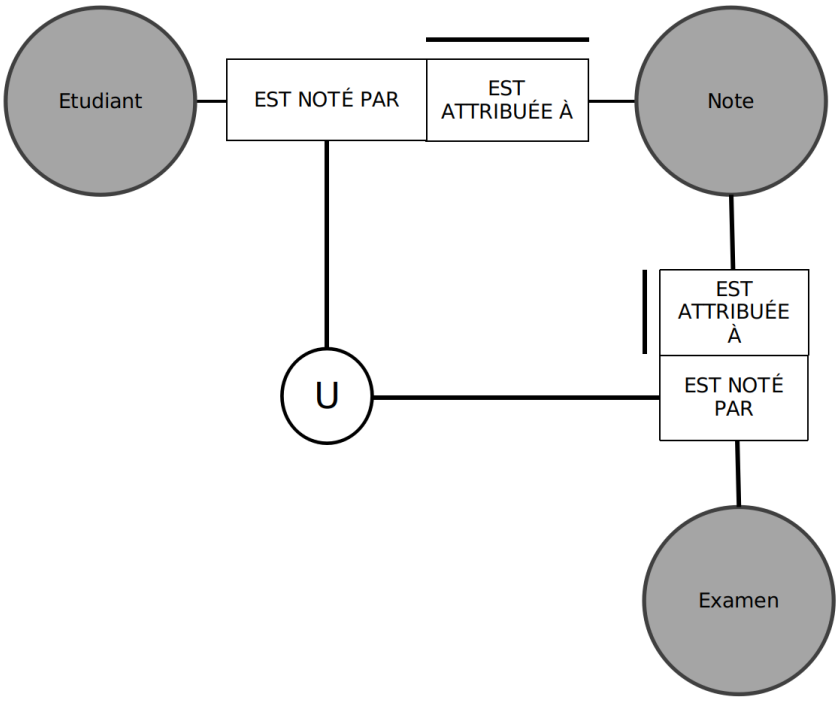

FIGURE 4.9 – Exemple de Contraintes entre deux idées (Contrainte d'unicité)

choix possibles, autrement dit nous pouvons voir cette contrainte comme un opérateur ensembliste UNION.

L'interprétation de cet exemple (Figure 4.10) est :

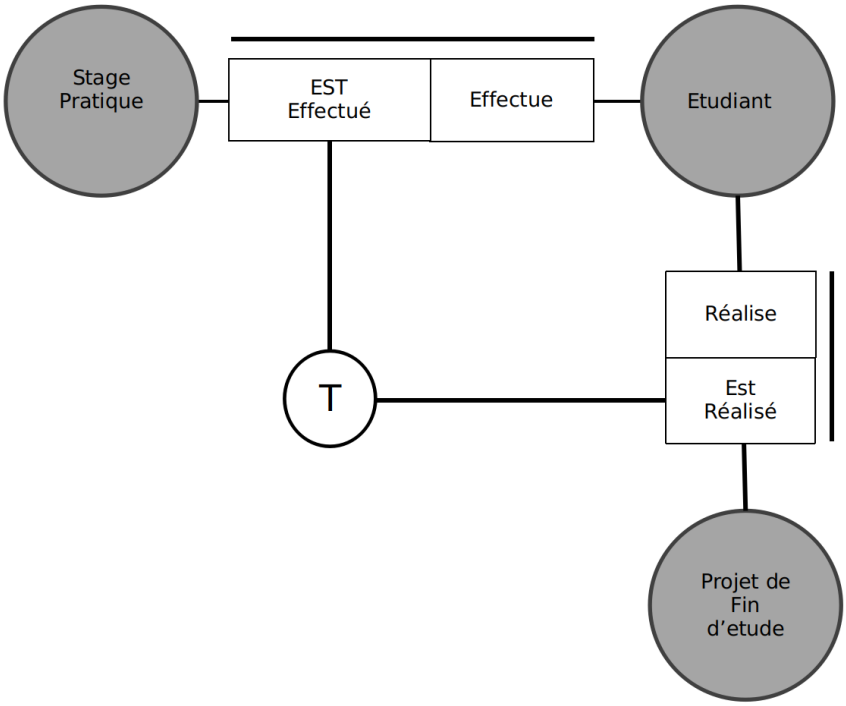

FIGURE 4.10 – Exemple de Contraintes entre deux idées (Contrainte De Totalité Entre Idées)

— UN étudiant prépare un (01) stage pratique OU un (01) projet de fin d'étude.
— {étudiants} ={ UN étudiant prépare un (01) stage pratique} ∪ { étudiant prépare un (01) projet de fin d'étude}.

4.5 Exemples

Dans cette section, nous présentons une série d'exemples, décrivant le symbolisme utilisé par la méthode NIAM afin de modéliser les concepts (avec la définition d'attributs) ainsi que les relations et les contraintes.
Dans l'exemple (Figure 4.11)

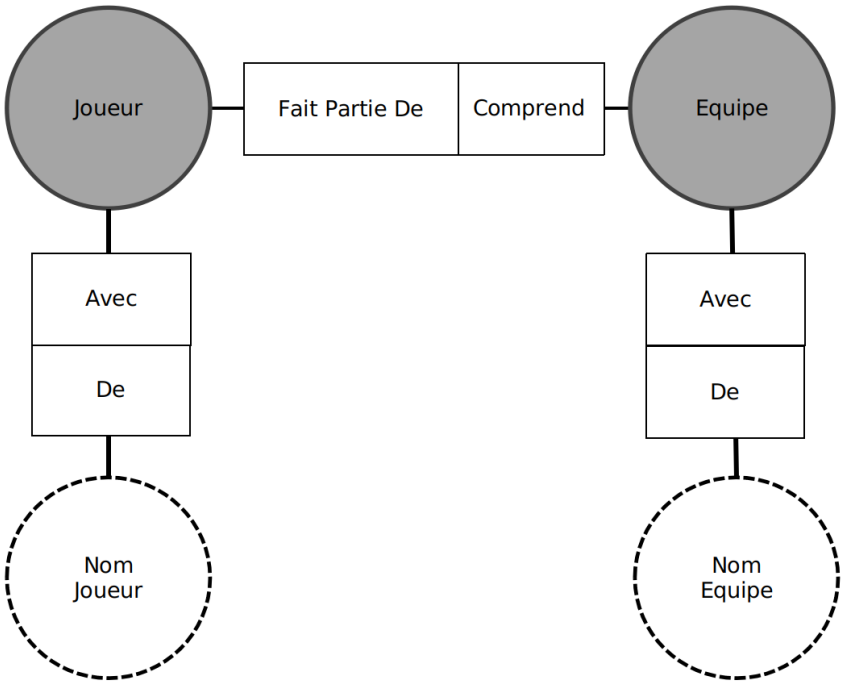

FIGURE 4.11 – Exemple de Contraintes entre deux idées avec définition d'attributs

Dans l'exemple illustré par la figure 4.11, deux concepts *Joueur* et *équipe* sont utilisés ou la relation (idée) les reliant comporte deux rôles : "Fait partie de" et "Comprend". Figure 4.12 illustre la manière d'utilisation des contraintes de cardinalités. La position du trait épais indique le rôle concerné par les contraintes, par exemple en absence du trait la cardinalité imposée dans ce cas est de 0 à N. Par contre, si le trait couvre un rôle donc la cardinalité est réduite à 0 ou 1, néanmoins que deux traits séparés couvrent chaque rôle à part donc la contrainte 0 ou 1 est imposée sur les deux rôle en même temps.
Figure 4.13, présente un exemple plus complet avec plus de détails. Il s'agit d'une modélisation détaillée de la gestion de livre et leur production, publication et toute informations relative aux livres.

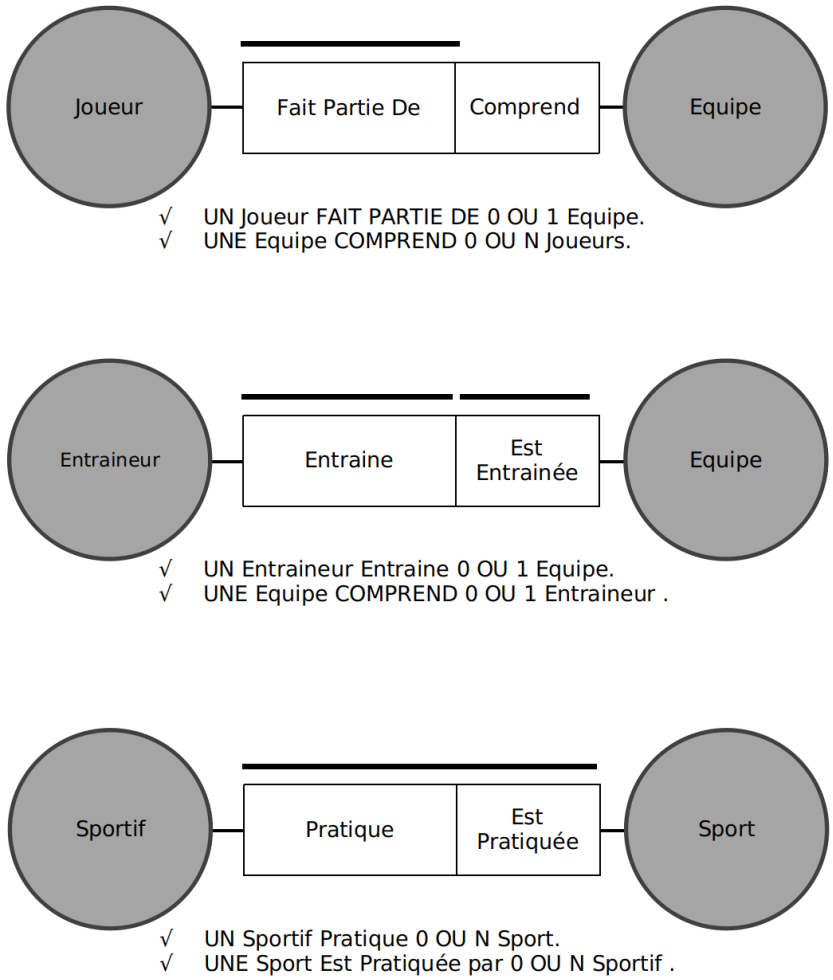

FIGURE 4.12 – Exemples Divers

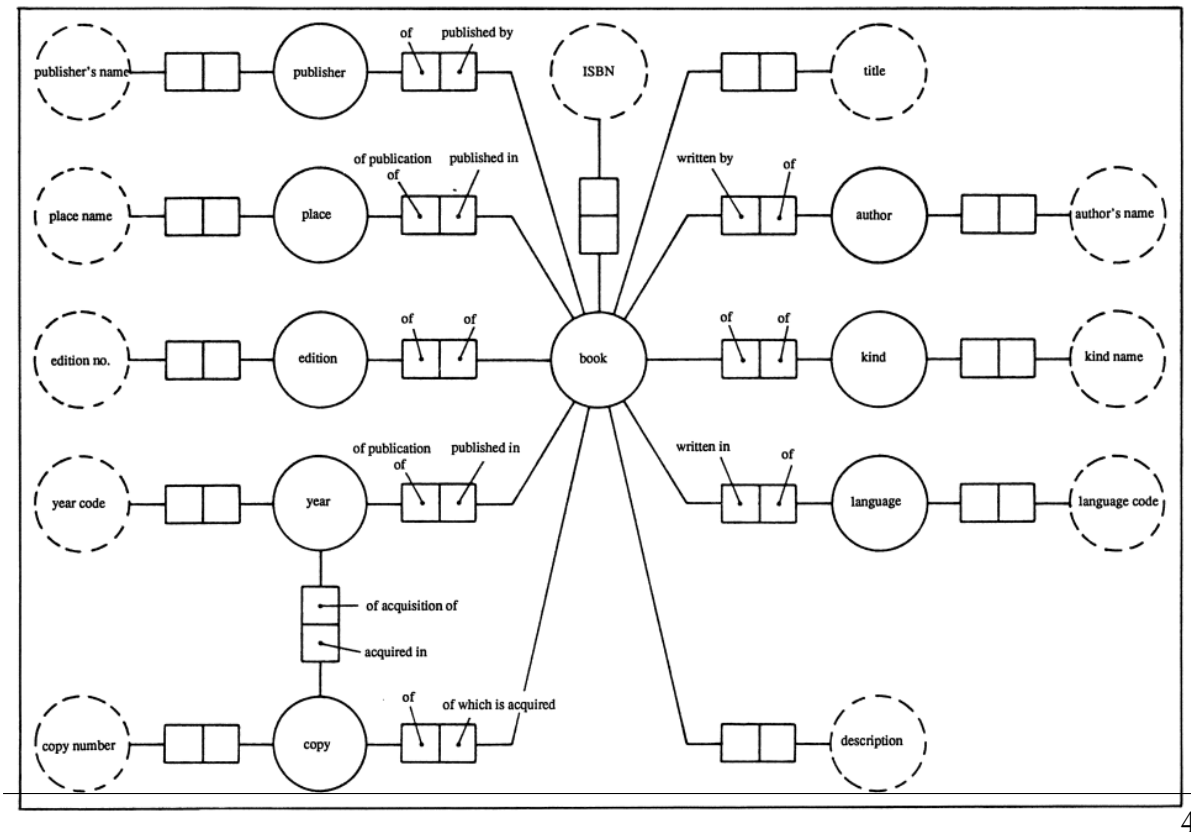

FIGURE 4.13 – Exemple de gestion de livres et leur publication

L'exemple (Figure 4.13) comporte plusieurs concepts reliés les uns aux autres (avec des idées) , en créant un schéma conceptuel NIAM, la liste suivant stipule ces derniers avec un description :

Book : Représente le concept livre, avec l'identifiant "ISBN" et des attributs titre et description, un livre est écrit par un auteur ou plusieurs. Un livre est écrit dans une langue et appartient à une catégorie, possède des éditions , année de sortie, nombre de copies , endroit de publication et un éditeur.

Author : désigne l'auteur de livre identifié par son nom, un auteur peut écrire plusieurs livres ou rien.

Kind : il s'agit des catégories possibles qu'un livre peut appartenir, où il est identifié par un nom ou un intitulé.

Publisher : est en effet l'éditeur du livre, identifiée par un nom.

Edition : il s'agit du numéro d'édition du livre.

Language : ce concept désigne les langues avec les quelles un livre peut être édité, ce concept est identifiée par un nom ou un intitulé.

Place : représente les endroits de publication du livre, ce concept est identifié par un son nom.

Year : ceci désigne l'année de publication ou de sortie du livre, avec plusieurs copies.

Copy : Modélise la relation copies du livre avec l'année de publication.

4.6 Classification des relations binaires entre concepts

Il existe plusieurs types de relation où tout dépend du cas échéant et de la nature sémantique du système à modéliser, la méthode NIAM se base sur des relations « binaires ».
Nous pouvons classifier les relations selon le contexte où elles sont utilisées : proximité, de voisinage, de décomposition, d'appartenance, d'affinité, de correspondance ... la liste est illimitée, ouverte.

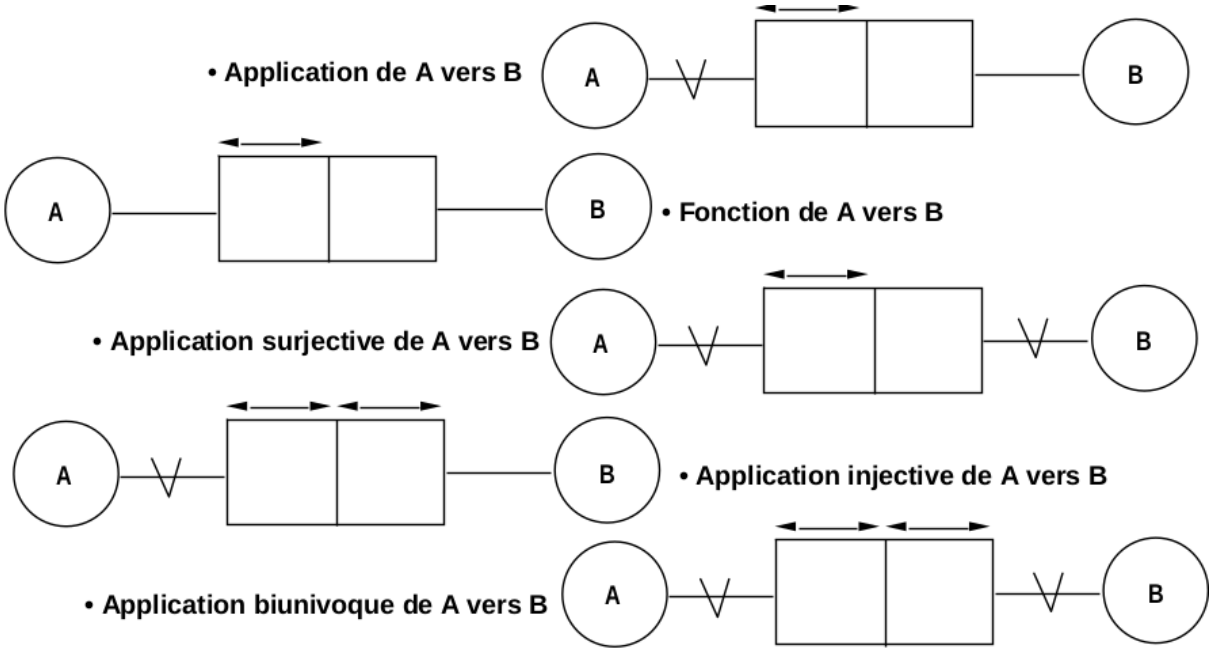

FIGURE 4.14 – Exemple de quelques relations binaires

Il faut noter que nous pouvons utiliser des traits ou des flèches à double sens afin de définir les cardinalités entre concepts.
Le principe des relations binaires est de créer une mise en correspondance (mapping en anglais) entre deux ensembles, en effet il s'agit de définir une liaison élément-élément, où on distingue une ensemble de départ est un autre un ensemble d'arriver.
Nous pouvons représenter cette liaison (ou relation) sous forme mathématique plus exactement, sous forme d'une fonction :

$$f : R_+ \mapsto R_+, x \mapsto f(x) = x^2 \tag{4.1}$$

Dans ce cas, l'ensemble de départ est R_+ pour la variable x, tandis que l'ensemble d'arrivée est l'espace de valeurs $f(x) = x^2$ qui est logiquement aussi R_+.
On se basant sur l'algèbre relationnel, nous pouvons distinguer environ huit relations :

— l'application simple
— la fonction
— l'injection
— la bijection
— la surjection
— la relation 1-N
— la relation 1-1
— la relation N-M

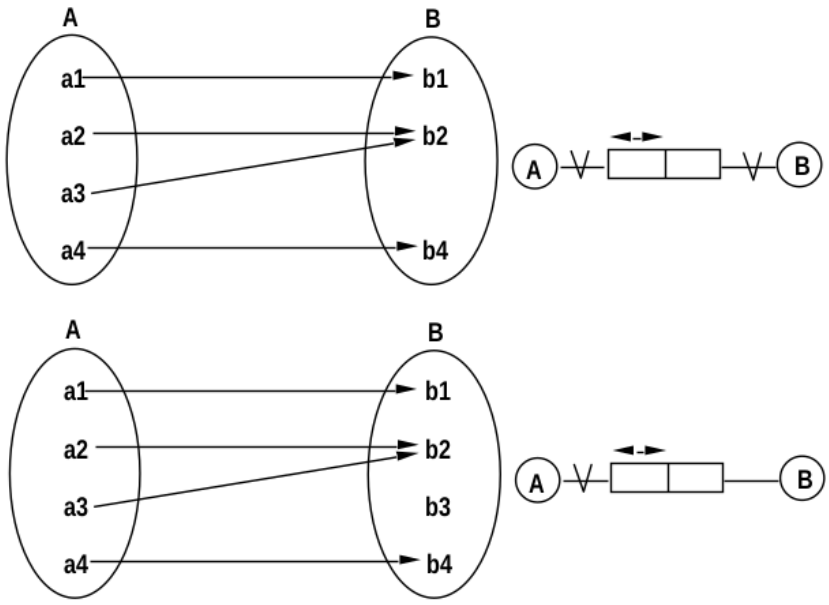

FIGURE 4.15 – Exemple d'une relation surjective.

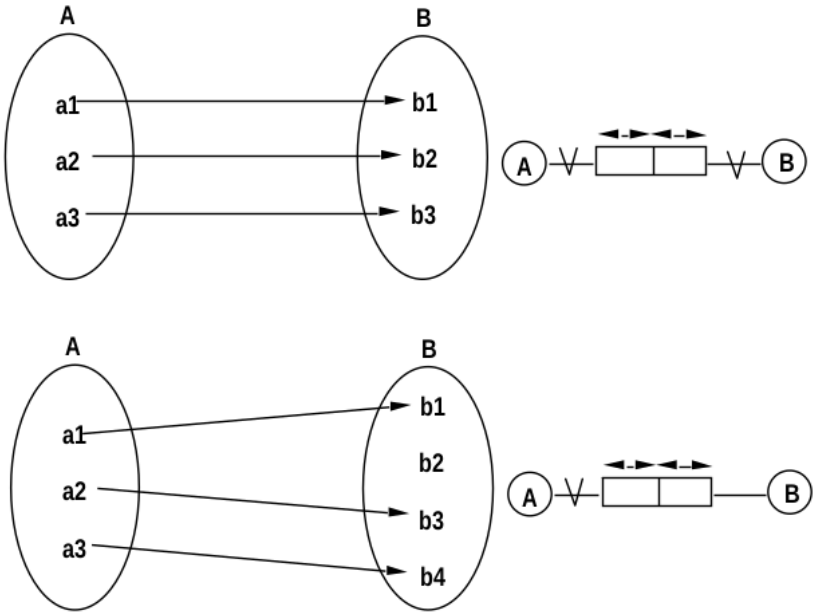

FIGURE 4.16 – Exemple d'une relation injective.

Chaque relation ensembliste peut être représentée par un schéma NIAM, et inversement chaque idée d'un schéma NIAM peut être considérée comme relation ensembliste. Une fonction ou application f est définie par un ensemble de départ A et un ensemble B. Tout éléments appartenant à l'ensemble A doit correspondre à un élément appartenant à l'ensemble B (voir Figure 4.17).

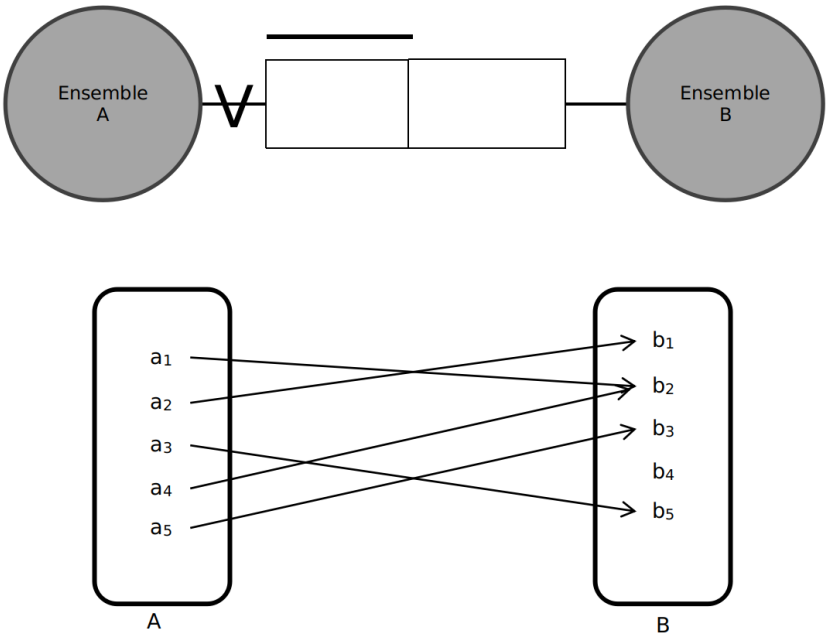

FIGURE 4.17 – Exemple d'une application simple.

4.7 Relation d'héritage

La relation héritage est souvent utilisée dans les méthodes objets pour définir la relation père-fils où le fils hérite les propriétés ainsi que la partie opérationnelle (méthodes). Figure 4.18, est un exemple de relation d'héritage entre fichiers, il existe plusieurs types de fichiers , avec quelques propriétés en commun tels que :
Un fichier possède : un nom, un lieu de stockage , une taille qui représente son contenus,...
Nous avons classés ces fichiers en deux classes :

Fichier Texte : cette classe se compose de plusieurs types : XML,CSV,JSON... ces fichiers partagent la même caractéristique de représenter les données sous forme textuelle, mais différent par rapport à la structure des données.

Fichier Binaire : Ils sont des fichiers spéciaux avec un contenu encodé sous différentes formes, ils peuvent varier des images, vidéos, PDF, ou fichiers exécutable.

Avec la méthode NIAM, nous pouvons véhiculer des informations sur les éléments qui héritent d'un autre concept ; par exemple une question peut être posée :
"Est-ce que sur le modèle d'héritage on trouve tous les fils de l'objet ?"
Si la réponse est oui, nous pouvons dire qu'il s'agit d'une relation héritage avec une contrainte de totalité.
La figure 4.19, illustre un exemple de relation d'héritage avec Contraintes de totalité, elle est représentée par la lettre "**T**" nous pouvons comparer cette dernière avec l'opérateur ensembliste union, limitant la liste des fils.

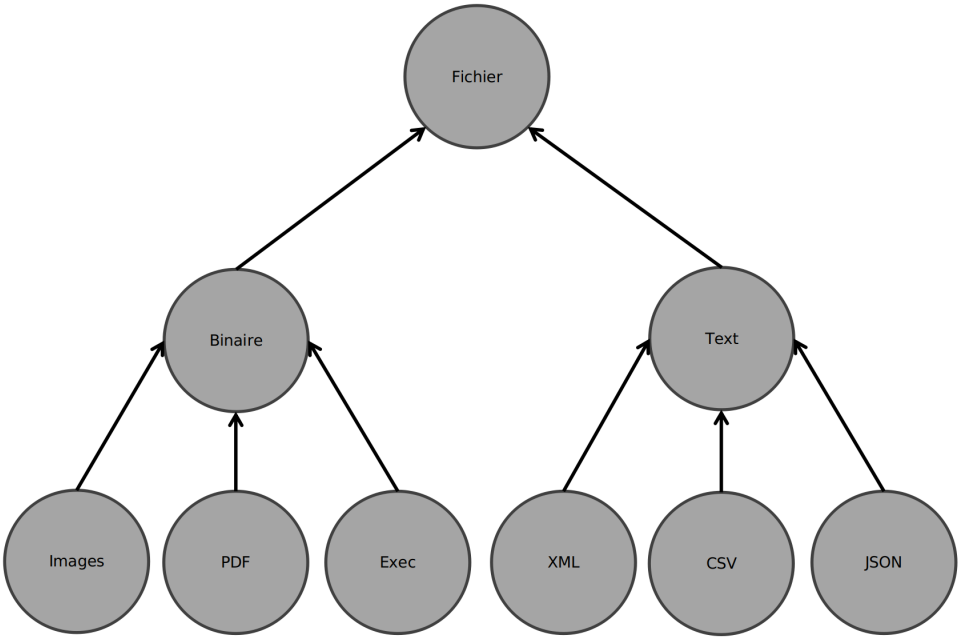

FIGURE 4.18 – Exemple de Relation d'héritage

Autrement dit par rapport à notre exemple nous pouvons dire que :

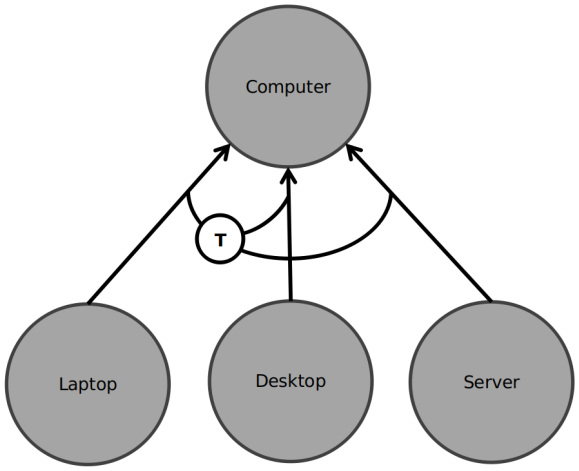

FIGURE 4.19 – Exemple de Relation d'héritage avec Contraintes de totalité

Un ordinateur peut incarner une des occurrences suivantes : Laptop, Desktop ou Serveur.
La figure 4.20, illustre un exemple de relation d'héritage avec Contraintes d'exclusion avec la lettre "**X**" reliant les arcs des relations. En effet l'exclusion permet de spécifier que les fils d'un concept sont porteurs de caractéristiques différentes mais partagent les mêmes propriétés héritées (avec possibilité de changer certaines propriétés).
Nous pouvons interpréter l'exemple 4.20, comme suit :

- Un PC portable (Laptop) est un ordinateur (Computer)
- Un PC de bureau (Desktop) est un ordinateur (Computer)
- Une machine serveur (Server) est un ordinateur (Computer)

— Un PC portable (Laptop) n'est pas PC de bureau (Desktop)
— Un PC de bureau (Desktop) n'est pas Un PC portable (Laptop)
— de même pour les autres...

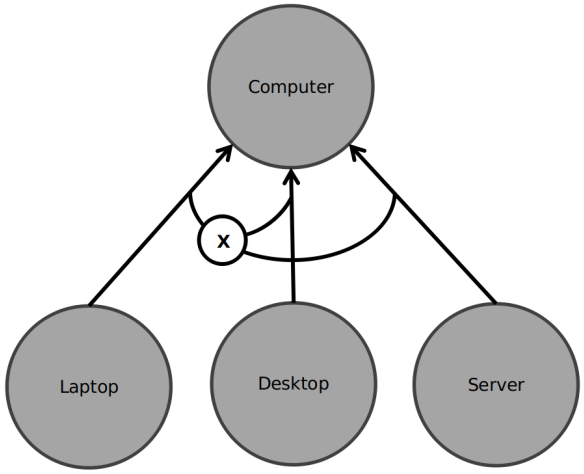

FIGURE 4.20 – Exemple de Relation d'héritage avec Contraintes d'exclusion (1)

Les fils sont différents les uns des autres mais ils partagent les même caractéristiques du père.
Un autre scénario d'exclusion, où nous reprenons l'exemple des fichiers, certes que PDF, image et texte sont des types de fichiers mais ils sont structurés de façon totalement différente et distincte.
La figure 4.21, illustre un autre exemple d'héritage avec contrainte d'exclusion. Il s'agit

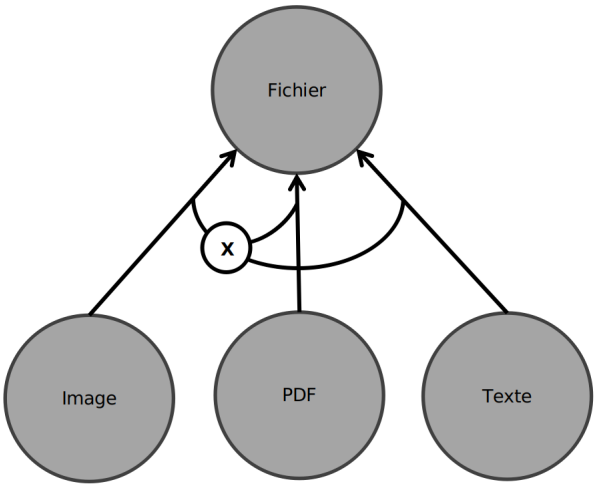

FIGURE 4.21 – Exemple de Relation d'héritage avec Contraintes d'exclusion (2).

de trois types de fichiers totalement différents (PDF , image , Texte) néanmoins ils sont tous des fichiers.
Il existe d'autre types de relations entre concepts, tels qu'une relation binaire de composition (d'appartenance, appelée aussi agrégation).

Cette relation modéliste la structure de composition ou d'organisation d'un concept.

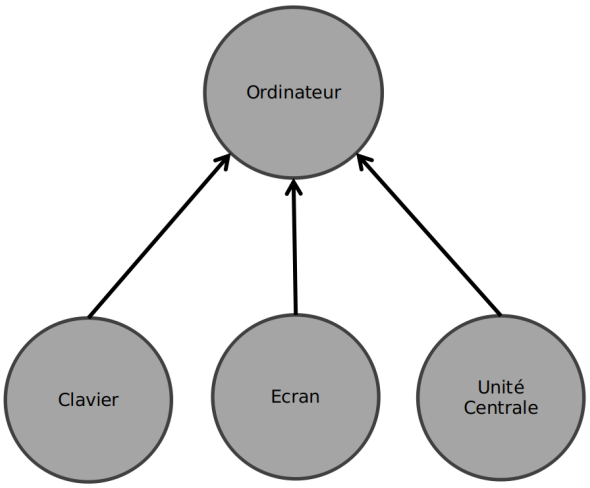

FIGURE 4.22 – Exemple de Relation d'Agrégation.

La figure 4.22 illustre un exemple d'agrégation, où un ordinateur est composé d'un ensemble d'éléments (clavier, souris, ...)

4.8 Conclusion

Ce chapitre présente une étude de cas en utilisant la méthode NIAM. Nous avons vu la terminologie utilisée afin de modéliser les acteurs dénommés les *CONCEPT* d'un système ainsi que les relations *IDÉE*.
NIAM permet d'imposer des contraintes d'intégrités sur les concepts et aussi entre les idées. NIAM permet de modéliser les relations d'héritage avec des options de totalisation et d'exclusion, ainsi que la relation de composition (agrégation) entre concept. De plus, NIAM permet la représentation des relations binaires telles que : les applications, les fonctions, injection, bijection, surjection.
La représentation visuelle des concepts , idées et contraintes simplifie et améliore la lisibilité du modèle proposé. Nous pouvons constater que NIAM offre pratiquement toutes les fonctionnalités et outils de modélisation offertes par les méthodes objet.

Acronymes

B2B Business to Business. 6

ERP Enterprise Resource Planning. 3

Merise Méthode d'Etude et de Réalisation Informatique pour les Systèmes d'Entreprise. 23

NIAM Nijssen's Information Analysis Method. 1, 24

OMT Object-Modeling Technique. 1, 29, 35

OOD Object-oriented design. 1

OOSE Object-Oriented Software Engineering. 1

SGBD Système de Gestion de Base de Données. 3

SI Système d'information. 3

SQL Structured Query Languge. 3

UML Unified Modeling Language. 1, 31, 35

Bibliographie

[Diviné, 1989] Diviné, M. (1989). *Parlez-vous Merise ?* Eyrolles.

[Gervais, 2006] Gervais, F. (2006). *Combinaison de spécifications formelles pour la modélisation des systèmes d'information*. PhD thesis, Conservatoire national des arts et metiers-CNAM ; Université de Sherbrooke.

[Halpin, 1998] Halpin, T. (1998). Object-role modeling (orm/niam). In *Handbook on architectures of information systems*, pages 81–103. Springer.

[Jessup and Valacich, 2008] Jessup, L. M. and Valacich, J. S. (2008). *Information systems today : managing in the digital world*, volume 3. Pearson Prentice Hall Upper Saddle River, NJ.

[Junior, 1997] Junior, J. C. F. (1997). *Ingénierie de systèmes d'information : une approche de multi-modélisation et de méta-modélisation*. PhD thesis, Université Joseph-Fourier-Grenoble I.

[Schenck and Wilson, 1994] Schenck, D. A. and Wilson, P. R. (1994). *Information modeling the EXPRESS way*. Oxford University Press.

[Spivey and Abrial, 1992] Spivey, J. M. and Abrial, J. (1992). *The Z notation*. Prentice Hall Hemel Hempstead.

[Stair and Reynolds, 2015] Stair, R. and Reynolds, G. (2015). *Fundamentals of information systems*. Cengage Learning.

[Wintraecken, 2012] Wintraecken, J.-J. V. (2012). *The NIAM information analysis method : theory and practice*. Springer Science & Business Media.